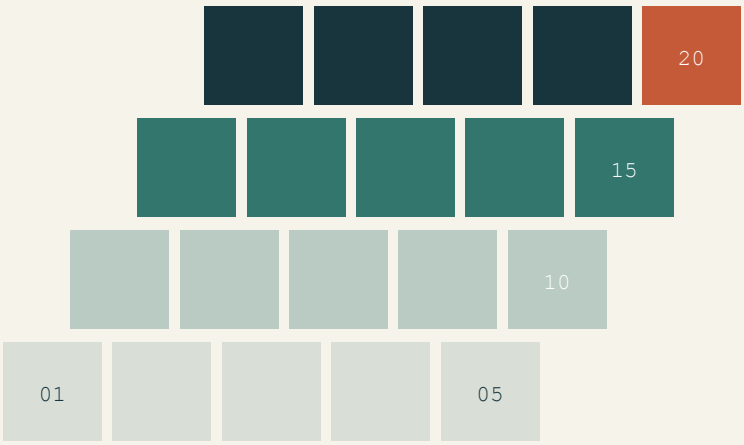
ΤΕΤΡΑΔΙΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

# Προγραμματισμός με μικρά βήματα

100 μαθήματα και ασκήσεις

Από τις πρώτες τιμές στους αλγορίθμους




**Ruby**

**ΑΣΚΗΣΕΙΣ 01-100**

ΕΛΛΗΝΙΚΗ ΕΚΔΟΣΗ

ΔΙΑΒΑΣΕ ΚΑΙ ΔΟΚΙΜΑΣΕ

# Η σειρά 1-20

Πριν ξεκινήσεις · Η γλώσσα · RSpec · Η διαδρομή



Οι τίτλοι και οι αριθμοί σελίδων είναι σύνδεσμοι.

ΔΙΑΒΑΣΕ ΚΑΙ ΔΟΚΙΜΑΣΕ

# Η σειρά 21-40

Πριν ξεκινήσεις · Η γλώσσα · RSpec · Η διαδρομή



Οι τίτλοι και οι αριθμοί σελίδων είναι σύνδεσμοι.

ΔΙΑΒΑΣΕ ΚΑΙ ΔΟΚΙΜΑΣΕ

# Η σειρά 41-60

Πριν ξεκινήσεις · Η γλώσσα · RSpec · Η διαδρομή



Οι τίτλοι και οι αριθμοί σελίδων είναι σύνδεσμοι.

ΔΙΑΒΑΣΕ ΚΑΙ ΔΟΚΙΜΑΣΕ

# Η σειρά 61-80

Πριν ξεκινήσεις · Η γλώσσα · RSpec · Η διαδρομή



Οι τίτλοι και οι αριθμοί σελίδων είναι σύνδεσμοι.

ΔΙΑΒΑΣΕ ΚΑΙ ΔΟΚΙΜΑΣΕ

# Η σειρά 81-100

Πριν ξεκινήσεις · Η γλώσσα · RSpec · Η διαδρομή



Οι τίτλοι και οι αριθμοί σελίδων είναι σύνδεσμοι.

ΑΠΟ ΤΟ ΜΗΔΕΝ

# Πριν ξεκινήσεις

Ξεκινάς από το μηδέν και λύνεις εκατό ασκήσεις, από τις πρώτες τιμές μέχρι αλγορίθμους σε Ruby. Σε κάθε βήμα διαβάζεις ένα μάθημα, γράφεις τη δική σου συνάρτηση και τη δοκιμάζεις με το εργαλείο testing της γλώσσας. Οι λύσεις βρίσκονται στο τελευταίο μέρος.

## Άνοιξε τον σωστό φάκελο

Άνοιξε τον φάκελο `ruby` στον editor σου. Στο Terminal γράψε `cd ` , σύρε τον ίδιο φάκελο από το Finder και πάτησε Enter. Έλεγξε πού βρίσκεσαι:

```sh
pwd
ls
```

Θα πρέπει να βλέπεις το `BOOK.md`, το PDF και τον φάκελο `exercises`. Οι εντολές του βιβλίου ξεκινούν από αυτόν τον φάκελο. Όταν μια εντολή χρειάζεται διαφορετική θέση, τα απαιτούμενα `cd` εμφανίζονται ρητά.

## Διάβασε και γράψε

Το PDF και το `BOOK.md` περιέχουν την ίδια ύλη σε δύο μορφές. Διάλεξε μία για διάβασμα. Στην πρώτη άσκηση γράφεις στο exercise.rb. Το exercise_spec.rb περιέχει τις δοκιμές της. Η εκφώνηση βρίσκεται και στην αρχή κάθε `exercise.rb`, μαζί με τα όρια και τα παραδείγματα. Γράφεις κάτω από τα σχόλια.

Αποθήκευε με `Command-S` πριν τρέξεις tests. Το `TODO` δείχνει το προσωρινό σώμα που θα αντικαταστήσεις. Κράτησε το όνομα της συνάρτησης και τις παραμέτρους της. Το βιβλίο εξηγεί ξεχωριστά πώς προσθέτεις δική σου δοκιμή.

## Τι είναι συνάρτηση και test

Μια συνάρτηση δέχεται τιμές και επιστρέφει αποτέλεσμα. Μια συνάρτηση που τριπλασιάζει το `4` επιστρέφει `12`. Η συγκεκριμένη τιμή που δίνεις λέγεται όρισμα. Το όνομα που τη δέχεται στον ορισμό της συνάρτησης λέγεται παράμετρος.

Ένα test καλεί τη συνάρτηση με γνωστή είσοδο και συγκρίνει την απάντηση με εκείνη που περιμένει. Η επιστροφή τιμής και η εμφάνιση στην οθόνη είναι διαφορετικές

πράξεις. Εδώ γράφεις συναρτήσεις που επιστρέφουν τιμές· το εργαλείο testing εμφανίζει αν οι δοκιμές πέτυχαν.

## Τι σημαίνει μια αποτυχία

Το αρχικό `TODO` προκαλεί επίτηδες αποτυχία. Πρώτα το αντικαθιστάς με δική σου υλοποίηση και μετά εξετάζεις τα αποτελέσματα των tests. Ένα σφάλμα σύνταξης εμφανίζεται πριν μπορέσει να ελεγχθεί η συμπεριφορά της συνάρτησης.

Όταν αποτυγχάνει μια σύγκριση, διάβασε το όνομα του test, την είσοδο, την αναμενόμενη απάντηση και την πραγματική. Διόρθωσε την προσπάθειά σου και ξανατρέξε την ίδια εντολή. Όταν όλα περάσουν, εξήγησε γιατί η λύση σου δουλεύει. Οι δοκιμές καλύπτουν συγκεκριμένες περιπτώσεις και όρια· η εκφώνηση περιγράφει τη συνολική απαίτηση.

## Πότε προχωράς

Προχώρα όταν μπορείς να εξηγήσεις τι δέχεται η συνάρτηση, τι επιστρέφει και γιατί περνούν τα παραδείγματα. Αν χρειαστεί να διαβάσεις την έτοιμη λύση, κλείσε την και ξαναγράψε την προσπάθειά σου από τη δική σου εξήγηση.

ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΣΥΝΤΑΞΗ

# Η Ruby με μια ματιά

Στη Ruby θα γράφεις κώδικα σε αρχεία που τελειώνουν σε `.rb`. Ο interpreter, δηλαδή το πρόγραμμα που εκτελεί τη Ruby, διαβάζει αυτά τα αρχεία με την εντολή `ruby`. Αυτό που στο βιβλίο λέμε συνάρτηση, στη Ruby ονομάζεται μέθοδος. Οι ασκήσεις χρησιμοποιούν τη μέθοδο `solve`.

## Εγκατάσταση στο macOS

Αν έχεις ήδη Ruby 3.2 ή νεότερη, μπορείς να ξεκινήσεις. Έλεγξε από το Terminal.

```sh
ruby --version
```

Αν χρειάζεσαι εγκατάσταση και έχεις Homebrew, ο επίσημος οδηγός της Ruby δίνει την παρακάτω εντολή.

```sh
brew install ruby
```

Βάλε τον φάκελο της εγκατάστασης πρώτα στο `PATH` του τρέχοντος Terminal και έλεγξε ποια Ruby εκτελείται.

```sh
export PATH="$(brew --prefix ruby)/bin:$PATH"
which ruby
ruby --version
```

Για να ισχύει και σε νέο Terminal με zsh, πρόσθεσε την ίδια γραμμή `export` στο αρχείο `~/.zshrc` με τον editor σου. Αν χρησιμοποιείς ήδη διαχειριστή εκδόσεων όπως το rbenv, επίλεξε τη Ruby μέσα από εκείνον. Οι τρέχουσες λεπτομέρειες βρίσκονται στη σελίδα του Homebrew για τη Ruby. Τα tests χρησιμοποιούν RSpec και Bundler. Το επόμενο κεφάλαιο εξηγεί την εγκατάσταση και τις εντολές. Δεν χρειάζεται Rails.

### Διάβασε μια μικρή συνάρτηση

RUBY

```ruby
def example()
  7
end
```

Το `def` ξεκινά τον ορισμό. Το `example` είναι το όνομα. Οι κενές παρενθέσεις σημαίνουν ότι δεν υπάρχουν παράμετροι. Το `end` κλείνει τον ορισμό. Η τελευταία έκφραση στο σώμα, εδώ το `7`, δίνει την τιμή που επιστρέφεται. Μπορείς επίσης να γράψεις `return 7` για άμεση επιστροφή. Οι λύσεις εδώ χρησιμοποιούν την τελευταία έκφραση.

Χρησιμοποίησε δύο κενά εσοχής για τις γραμμές του σώματος. Το `end` ορίζει πού τελειώνει η μέθοδος. Οι εσοχές βοηθούν να το διαβάζεις. Η κλήση `example()` επιστρέφει `7`. Ο ορισμός μόνος του δεν την καλεί. Ο επίσημος οδηγός μεθόδων περιγράφει αυτή τη σύνταξη.

Αργότερα θα δεις το `def solve(n)`. Το `n` είναι παράμετρος. Δεν γράφουμε τον τύπο της δίπλα στο όνομα. Οι πρώτες εκφωνήσεις δίνουν ακέραιες εισόδους μέσα σε συγκεκριμένα όρια. Αργότερα θα δέχεσαι και κείμενα ή πίνακες. Το `"7"` είναι κείμενο, ενώ το `7` είναι αριθμός. Οι λογικές τιμές γράφονται `true` και `false`, με πεζά γράμματα.

ΤΟ ΕΡΓΑΛΕΙΟ TESTING

# RSpec

Στο βιβλίο χρησιμοποιούμε **RSpec**. Τα `describe`, `it` και `expect` δίνουν όνομα στη συμπεριφορά που ελέγχουμε και ξεχωριστό παράδειγμα για κάθε είσοδο. Μαθαίνεις την εντολή `bundle exec rspec`, πώς διαβάζεις μια αποτυχία και πώς προσθέτεις δικό σου spec. Το RSpec λειτουργεί και σε απλά αρχεία Ruby. Το `rspec-rails` αφορά την ενσωμάτωση σε Rails και δεν χρειάζεται εδώ.

Το **Minitest** είναι επίσης κανονική επιλογή. Προσφέρει tests με `Minitest::Test` και assertions όπως `assert_equal`, αλλά και σύνταξη spec με `describe` και `it`. Είναι λογική επιλογή όταν το project το χρησιμοποιεί ή όταν προτιμάς tests με απλές κλάσεις και μεθόδους. Δεν χρειάζεται να μεταφέρεις ένα υπάρχον test suite σε RSpec μόνο επειδή αυτό χρησιμοποιεί το βιβλίο. Και τα δύο εργαλεία διαθέτουν δικές τους εξαρτήσεις και εκδόσεις.

Η επιλογή εδώ είναι διδακτική, όχι ισχυρισμός ότι ένα εργαλείο είναι πάντα καλύτερο ή έχει μεγαλύτερο μερίδιο χρήσης. Ξεκινάμε με πραγματικά specs για τις 100 ασκήσεις και κρατάμε το ίδιο εργαλείο μέχρι το τέλος.

Πηγές: RSpec Core, Minitest, RSpec Rails.

RSPEC

# Ετοίμασε το RSpec

Από τον φάκελο `ruby`, έλεγξε τα εργαλεία και εγκατάστησε τα gems του βιβλίου:

```sh
ruby --version
bundle --version
bundle install
```

Η έκδοση αυτή χρησιμοποιεί Ruby 3.2 ή νεότερη και RSpec 3.13.2. Δοκιμάστηκε με Ruby 3.4.7. Το `Gemfile` δηλώνει το RSpec και το `Gemfile.lock` κρατά τις συγκεκριμένες εκδόσεις των εξαρτήσεών του. Το `bundle install` χρειάζεται στην αρχή και όταν αλλάζουν οι εξαρτήσεις.

Αν η εντολή `bundle` δεν υπάρχει στην επιλεγμένη Ruby, εγκατέστησε το Bundler και ξαναδοκίμασε:

```sh
gem install bundler -v 4.0.14
bundle install
```

Το `bundle exec` τρέχει την επόμενη εντολή με τα gems αυτού του project. Γι' αυτό χρησιμοποιούμε `bundle exec rspec`, ακόμη κι αν το σκέτο `rspec` υπάρχει ήδη στο μηχάνημα. Δεν χρειάζεται Rails. Ο οδηγός του Bundler εξηγεί τη διαχείριση των εξαρτήσεων.

RSPEC

# Διάβασε το πρώτο spec

Γράφεις στο `exercises/01-hello/exercise.rb`. Στην αρχή του αρχείου υπάρχει η εκφώνηση ως σχόλιο. Οι γραμμές που αρχίζουν με `#` δεν εκτελούνται. Κάτω από αυτές συμπληρώνεις τη `solve`. Τα tests βρίσκονται δίπλα, στο `exercise_spec.rb`:

RUBY

```ruby
RSpec.describe "01. Μια σταθερή απάντηση" do
  source = File.join(__dir__, "exercise.rb")
  # Κάθε ομάδα tests φορτώνει τη δική της solve.
  class_eval(File.read(source), source, 1)

  it "exact greeting" do
    result = solve()
    expect(result).to be_an_instance_of(String)
    expect(result).to eql("Hello!")
  end
end
```

Το `RSpec.describe` ομαδοποιεί τα tests της άσκησης. Κάθε `it` περιγράφει μία περίπτωση. Το `result` κρατά την απάντηση της `solve`. Το `expect` δηλώνει τι περιμένεις: εδώ συγκεκριμένο τύπο `String` και τιμή `"Hello!"`.

Οι δύο γραμμές με `source` και `class_eval` φορτώνουν τη μέθοδο μέσα στη συγκεκριμένη ομάδα. Έτσι οι 100 διαφορετικές `solve` μπορούν να ελεγχθούν μαζί χωρίς να αντικαθιστά η μία την άλλη. Τις αφήνεις όπως είναι και γράφεις τα tests σου στα blocks `it`.

Η τεκμηρίωση του RSpec παρουσιάζει τις ομάδες και τα παραδείγματα.

RSPEC

# Τρέξε την πρώτη άσκηση

Αποθήκευσε με `Command-S`. Από τον φάκελο `ruby`:

```sh
bundle exec rspec \
  exercises/01-hello/exercise_spec.rb
```

Η διαδρομή επιλέγει μόνο τα specs της άσκησης 01. Το αρχείο `.rspec` ζητά αναλυτική εμφάνιση των ονομάτων. Είναι ρύθμιση του ίδιου του RSpec.

Στην αρχή θα εμφανιστεί `TODO`, επειδή η προσπάθεια περιέχει ακόμη `raise`. Αν επιστρέψεις λάθος τιμή, το μήνυμα δείχνει `expected` για την αναμενόμενη απάντηση και `got` για τη δική σου. Με σωστή λύση, η πρώτη άσκηση γράφει:

```text
1 example, 0 failures
```

Το `ruby exercise.rb` από μόνο του ορίζει τη μέθοδο. Δεν την καλεί και δεν εκτελεί τα specs. Για έλεγχο της συμπεριφοράς χρησιμοποιείς την εντολή RSpec που δίνει κάθε άσκηση.

RSPEC

# Πρόσθεσε δικό σου test

Στην άσκηση 02 η `solve` επιστρέφει τον αριθμό που δέχεται. Μέσα στο `RSpec.describe` του `exercises/02-echo/exercise_spec.rb`, πριν από το τελευταίο `end`, πρόσθεσε:

RUBY

```ruby
it "επιστρέφει το 23" do
  expect(solve(23)).to eql(23)
end
```

Τρέξε πρώτα όλο το αρχείο:

SH

```sh
bundle exec rspec \
  exercises/02-echo/exercise_spec.rb
```

Για να εκτελέσεις μόνο τη δική σου περίπτωση, πρόσθεσε φίλτρο ονόματος:

SH

```sh
bundle exec rspec \
  exercises/02-echo/exercise_spec.rb \
  --example "επιστρέφει το 23"
```

Το `eql` ελέγχει την ισότητα με τη Ruby `eql?`. Για παράδειγμα, ξεχωρίζει το `23` από το `23.0`. Το επίσης συνηθισμένο `eq` χρησιμοποιεί `==`, όπου αυτοί οι δύο αριθμοί συγκρίνονται ως ίσοι. Δες τους matchers ισότητας.

Αν το φίλτρο εμφανίσει `0 examples`, δεν έτρεξε καμία δοκιμή. Έλεγξε τη διαδρομή και το όνομα στο `--example`.

RSPEC

# Τρέξε περισσότερες ασκήσεις

Από τον φάκελο `ruby`, δύο συγκεκριμένες ασκήσεις:

```sh
bundle exec rspec \
  exercises/01-hello/exercise_spec.rb \
  exercises/02-echo/exercise_spec.rb
```

Για όλες τις ασκήσεις:

```sh
bundle exec rspec
```

Το `.rspec` ορίζει ως προεπιλεγμένο φάκελο το `exercises`. Οι άλυτες ασκήσεις θα αποτύχουν. Όσο μαθαίνεις, τρέχε το αρχείο της άσκησης που δουλεύεις. Το `bundle exec rspec --help` εμφανίζει τις επιλογές του εργαλείου.

Για να δεις μία τιμή χωρίς test, αφού λύσεις την άσκηση 02:

```sh
ruby -r ./exercises/02-echo/exercise.rb \
  -e 'p solve(23)'
```

Το `-r` φορτώνει το αρχείο, το `-e` εκτελεί την έκφραση και το `p` εμφανίζει την τιμή. Η `solve` συνεχίζει να επιστρέφει το αποτέλεσμα.

RSPEC

# Πίνακες και όρια των tests

Στις επόμενες ασκήσεις τα ορίσματα γίνονται πίνακες και κείμενα. Τα γράφεις απευθείας μέσα στο `it`, όπως `numbers = [2, 3, -1]`. Μετά καλείς `solve(numbers)` και συγκρίνεις το αποτέλεσμα.

Όπου υπάρχουν μεταβλητά δεδομένα, τα specs κρατούν ανεξάρτητο αντίγραφο με `Marshal.load(Marshal.dump(...))`. Μετά την κλήση ελέγχουν ότι τα ορίσματα έμειναν ίδια. Το βαθύ αντίγραφο περιλαμβάνει και τους εσωτερικούς πίνακες. Αν δεις «Η solve άλλαξε τα ορίσματα εισόδου», έλεγξε πράξεις όπως `sort!`, `shift` ή ανάθεση σε στοιχείο της εισόδου.

Τα 706 παραδείγματα ελέγχουν τιμές, τύπους και οριακές περιπτώσεις. Η επιτυχία τους δεν αποδεικνύει ότι μια λύση έχει την απαιτούμενη πολυπλοκότητα ή καλύπτει κάθε πιθανή είσοδο. Στις ασκήσεις 41-100 εξήγησε και τον στόχο απόδοσης που γράφει η εκφώνηση.

Οι έτοιμες λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου. Αν θέλεις να ελέγξεις μία, κράτησε πρώτα τη δική σου προσπάθεια, βάλε τον κώδικα της λύσης στο ίδιο `exercise.rb` κάτω από τα σχόλια και τρέξε το ίδιο spec. Έπειτα επανάφερε τη δική σου προσπάθεια.

ΠΡΩΤΑ ΚΑΤΑΛΑΒΑΙΝΟΥΜΕ, ΜΕΤΑ ΠΡΟΧΩΡΑΜΕ

# Η διαδρομή

Ένα μικρό βήμα κάθε φορά. Οι ασκήσεις εμπέδωσης δεν απαιτούν νέα έννοια. Οι είσοδοι είναι μικρές και έγκυρες. Δεν χρησιμοποιούμε ακόμη επαναλήψεις, πίνακες, αναδρομή, αρχεία ή δικτυακές υπηρεσίες.

| Βήμα | Η νέα ιδέα |
|---|---|
| 01 | Επιστροφή μιας σταθερής τιμής |
| 02 | Μία παράμετρος |
| 03 | Πρόσθεση με + |
| 04 | Δύο παράμετροι |
| 05 | Πολλαπλασιασμός με * |
| 06 | Δίνουμε ονόματα σε τιμές και αποτελέσματα |
| 07 | Αφαίρεση με `-` |
| 08 | Παρενθέσεις σε μια έκφραση |
| 09 | Υπόλοιπο διαίρεσης |
| 10 | Σύγκριση ισότητας και λογική τιμή |
| 11 | Σύγκριση με > |
| 12 | Σύγκριση που περιλαμβάνει την ισότητα |
| 13 | Εμπέδωση: υπόλοιπο και ισότητα |
| 14 | Πρώτη επιλογή με if / else |
| 15 | Εμπέδωση: επιλέγουμε ανάμεσα σε δύο παραμέτρους |
| 16 | Περισσότερα από δύο ενδεχόμενα |
| 17 | Συνδυάζουμε δύο ελέγχους με και |
| 18 | Εμπέδωση: ο διαιρέτης γίνεται παράμετρος |
| 19 | Εμπέδωση: μια συνθήκη επιλέγει κείμενο |
| 20 | Σειρά ελέγχων όταν οι περιπτώσεις συμπίπτουν |

Μετά την εικοστή μπορούν να ακολουθήσουν η επανάληψη, το μέτρημα από το 1 έως το n και η πλήρης ακολουθία FizzBuzz.

ΠΡΩΤΑ ΚΑΤΑΛΑΒΑΙΝΟΥΜΕ, ΜΕΤΑ ΠΡΟΧΩΡΑΜΕ

# Η διαδρομή 21-40

Οι ασκήσεις 21-40 συνεχίζουν από τις πρώτες είκοσι. Πρώτα μαθαίνουμε να επαναλαμβάνουμε βήματα. Μετά συγκεντρώνουμε τιμές σε πίνακες, τις μετατρέπουμε σε κείμενα και φτιάχνουμε το πλήρες FizzBuzz. Η νέα ύλη υπάρχει μόνο στη Ruby.

Όταν μια άσκηση δέχεται πίνακα, διάβασε τα στοιχεία χωρίς να αλλάξεις την είσοδο. Αν ζητά διαφορετικές τιμές, επέστρεψε νέο πίνακα.

| Βήμα | Η νέα ιδέα |
|---|---|
| 21 | Νέα τιμή σε μια υπάρχουσα μεταβλητή |
| 22 | Επανάληψη με while |
| 23 | Συγκεντρώνουμε αποτέλεσμα μέσα σε επανάληψη |
| 24 | Μια συνθήκη μέσα σε επανάληψη |
| 25 | Συγκεκριμένο πλήθος επαναλήψεων με times |
| 26 | Ο πίνακας Array και η σειρά των στοιχείων |
| 27 | Διαβάζουμε μια θέση πίνακα με δείκτη |
| 28 | Το πλήθος ενός πίνακα με length |
| 29 | Προσθήκη στο τέλος με << |
| 30 | Επισκεπτόμαστε κάθε στοιχείο με each |
| 31 | Εμπέδωση: each μαζί με if και μετρητή |
| 32 | Νέος πίνακας με map |
| 33 | Επιλογή στοιχείων με select |
| 34 | Μετατροπή με to_s |
| 35 | Ένωση κειμένων με + |
| 36 | Ένωση πίνακα κειμένων με join |
| 37 | Το Range με .. και η μετατροπή to_a |
| 38 | Εμπέδωση: map, if και μετατροπή σε κείμενο |
| 39 | Εμπέδωση: η σειρά των κανόνων μέσα σε map |
| 40 | Συνδυάζουμε διάστημα, κανόνες και γραμμές κειμένου |

Στο τέλος της δεύτερης εικοσάδας θα συνδυάζεις επαναλήψεις, πίνακες και κείμενα.

ΠΡΩΤΑ ΚΑΤΑΛΑΒΑΙΝΟΥΜΕ, ΜΕΤΑ ΠΡΟΧΩΡΑΜΕ

# Η διαδρομή 41-60

Από προθέματα και δύο δείκτες μέχρι την επιλογή του καλύτερου συνεχόμενου τμήματος. Κανονικοποίηση, καταμέτρηση και αναζήτηση πληροφοριών που έχουν ήδη εμφανιστεί.

| Βήμα | Η νέα ιδέα |
|---|---|
| 41 | Πρόθεμα και αμετάβλητη είσοδος |
| 42 | Προϋπολογισμός για πολλές ερωτήσεις |
| 43 | Προθέματα και επιθέματα χωρίς διαίρεση |
| 44 | Κυκλικοί δείκτες και κανονικοποίηση |
| 45 | Δύο δείκτες που προχωρούν |
| 46 | Σταθερή σειρά με έναν δείκτη εγγραφής |
| 47 | Τοπική κατάσταση και καλύτερη απάντηση |
| 48 | Το καλύτερο προηγούμενο στοιχείο |
| 49 | Επέκταση ή νέο ξεκίνημα |
| 50 | Συνολικό άθροισμα και αφαίρεση |
| 51 | Κανονικοποίηση και δύο άκρα |
| 52 | Ομάδες ίδιων γειτονικών τιμών |
| 53 | Ανάλυση ομάδων με scan |
| 54 | Hash με μετρητές και σαφής ισοβαθμία |
| 55 | Δύο περάσματα με διαφορετικό σκοπό |
| 56 | Ισότητα πολυσυνόλων |
| 57 | Hash ως σύνολο και κανονική έξοδος |
| 58 | Αναζήτηση μόνο από πραγματικές αρχές |
| 59 | Αποθήκευση του παρελθόντος |
| 60 | Συχνότητες αθροισμάτων προθεμάτων |

Αιτιολόγησε τη σωστή απάντηση και τον στόχο απόδοσης πριν προχωρήσεις.

ΠΡΩΤΑ ΚΑΤΑΛΑΒΑΙΝΟΥΜΕ, ΜΕΤΑ ΠΡΟΧΩΡΑΜΕ

# Η διαδρομή 61-80

Μονοτονικές συνθήκες, δυαδικά όρια και παράθυρα που ενημερώνονται χωρίς νέο πλήρη υπολογισμό. Εκκρεμότητες που λύνονται με συγκεκριμένη σειρά, γεγονότα στον χρόνο και δισδιάστατες δομές.

| Βήμα | Η νέα ιδέα |
|---|---|
| 61 | Κράτησε μόνο το μισό διάστημα |
| 62 | Κατώτερο όριο με διπλότυπα |
| 63 | Δύο διαφορετικά δυαδικά όρια |
| 64 | Δυαδική αναζήτηση στην απάντηση |
| 65 | Δυαδική αναζήτηση με έλεγχο εφικτότητας |
| 66 | Αφαίρεσε ό,τι βγαίνει, πρόσθεσε ό,τι μπαίνει |
| 67 | Παράθυρο που μεγαλώνει και μικραίνει |
| 68 | Παράθυρο και τελευταία θέση |
| 69 | Μετρητές μέσα στο τρέχον παράθυρο |
| 70 | Παράθυρο με σταθερό αλφάβητο |
| 71 | Στοίβα για εμφωλευμένες δομές |
| 72 | Επισκευή προθέματος με τον ελάχιστο αριθμό κινήσεων |
| 73 | Στοίβα τιμών και σειρά τελεστών |
| 74 | Μονοτονική στοίβα από δεξιά |
| 75 | Στοίβα εκκρεμών δεικτών |
| 76 | Το μικρότερο ύψος καθορίζει το πλάτος |
| 77 | Ταξινόμηση και επέκταση του τελευταίου |
| 78 | Σάρωση γεγονότων με ισοβαθμίες χρόνου |
| 79 | Δισδιάστατη δομή με ανεξάρτητες γραμμές |
| 80 | Τέσσερα όρια που συρρικνώνονται |

Αιτιολόγησε τη σωστή απάντηση και τον στόχο απόδοσης πριν προχωρήσεις.

ΠΡΩΤΑ ΚΑΤΑΛΑΒΑΙΝΟΥΜΕ, ΜΕΤΑ ΠΡΟΧΩΡΑΜΕ

# Η διαδρομή 81-100

Συνδεσιμότητα, αποστάσεις, εξαρτήσεις και δέντρα επιλογών με αναίρεση και κλάδεμα. Καταστάσεις, μεταβάσεις και βελτιστοποίηση, μέχρι συντομότερες διαδρομές και υποσύνολα πόλεων.

| Βήμα | Η νέα ιδέα |
|---|---|
| 81 | Εξερεύνηση συνδεδεμένης περιοχής |
| 82 | Ουρά και αποστάσεις ανά επίπεδο |
| 83 | Λίστες γειτόνων και BFS |
| 84 | Επανεκκίνηση αναζήτησης από κάθε άγνωστη κορυφή |
| 85 | Τοπολογική ταξινόμηση και αδιέξοδο |
| 86 | Χρωματισμός και σύγκρουση περιορισμών |
| 87 | Αναδρομή, αναίρεση και αποφυγή διπλότυπων |
| 88 | Αύξουσες επιλογές και κλάδεμα |
| 89 | Κατασκευή με έγκυρα προθέματα |
| 90 | Backtracking με τρία σύνολα περιορισμών |
| 91 | Δυναμικός προγραμματισμός με απαγορευμένες καταστάσεις |
| 92 | Βέλτιστη λύση ανά μικρότερο ποσό |
| 93 | Η σειρά των βρόχων αλλάζει την καταμέτρηση |
| 94 | Κάθε αντικείμενο χρησιμοποιείται το πολύ μία φορά |
| 95 | Δυναμικός προγραμματισμός δύο κειμένων |
| 96 | Ελάχιστο κόστος τριών μεταβάσεων |
| 97 | Ελάχιστη τελευταία τιμή ανά μήκος |
| 98 | Ταξινόμηση, δυαδικό όριο και δυναμικός προγραμματισμός |
| 99 | Οριστικοποίηση της μικρότερης απόστασης |
| 100 | Δυναμικός προγραμματισμός σε υποσύνολα |

Αιτιολόγησε τη σωστή απάντηση και τον στόχο απόδοσης πριν προχωρήσεις.

ΜΕΡΟΣ Α

# Οι ασκήσεις

Διάβασε το μάθημα, γράψε και τρέξε τα tests.

Κάθε άσκηση δίνει τα αρχεία και την ακριβή εντολή.

01 02 03 04 05 06 07 08 09 10

11 12 13 14 15 16 17 18 19 20

ΕΝΑ ΜΙΚΡΟ ΒΗΜΑ ΚΑΘΕ ΦΟΡΑ

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 01 Ο ορισμός και η πρώτη τιμή

Πριν ζητήσουμε από τον υπολογιστή να λύσει κάτι, του δίνουμε έναν κανόνα με όνομα. Το παράδειγμα επιστρέφει πάντα το κείμενο `"Ready!"`. Διάβασε πρώτα πού αρχίζει και πού τελειώνει η συνάρτηση.

RUBY

```ruby
def example()
  "Ready!"
end
```

## Διάβασε τις γραμμές

01 Το `def` αρχίζει τον ορισμό μιας μεθόδου. Το `example` είναι το όνομά της. Οι κενές παρενθέσεις `()` δείχνουν ότι δεν δίνουμε ορίσματα.

02 Το `"Ready!"` είναι κείμενο. Τα εισαγωγικά ορίζουν πού αρχίζει και πού τελειώνει. Δεν είναι μέρος της τιμής.

03 Η τελευταία έκφραση του σώματος δίνει την τιμή που επιστρέφεται. Εδώ είναι το `"Ready!"`, χωρίς να χρειάζεται `return`.

04 Το `end` κλείνει τον ορισμό. Τα δύο κενά εσοχής πριν από το κείμενο βοηθούν να δούμε ότι ανήκει στο σώμα.

05 Η `example()` είναι κλήση. Εκτελεί το σώμα και παίρνει την τιμή. Ο ορισμός μόνος του δεν καλεί τη μέθοδο ούτε εμφανίζει κείμενο.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 01 Από τον κώδικα στην τιμή


## Ακολούθησε μια κλήση

Η `example()` επιστρέφει `"Ready!"`. Η κλήση δεν δίνει είσοδο. Το σώμα δίνει τη σταθερή τιμή.

## Στάσου λίγο

Τα διπλά εισαγωγικά είναι μέρος του αποτελέσματος; Αρκεί να γράψεις τον ορισμό για να εμφανιστεί το μήνυμα;

**Έλεγξε τη σκέψη σου.** Όχι και στις δύο ερωτήσεις. Τα εισαγωγικά ορίζουν το κείμενο στον κώδικα. Χρειάζεται κλήση για να πάρουμε την τιμή και χωριστή εντολή εμφάνισης για να τη δούμε. Στις ασκήσεις, το test καλεί τη συνάρτηση της άσκησης και ελέγχει την τιμή της.

## Το αρχείο της προσπάθειάς σου

Κράτησε το `def solve()` και το τελικό `end`. Αντικατάστησε το σχόλιο `TODO` και τη γραμμή `raise` με τη δική σου τιμή που θα επιστρέφεται.

Συνέχισε στην εκφώνηση της άσκησης 01, στη σελίδα 26.

Η ΑΣΚΗΣΗ

# 01 Μια σταθερή απάντηση

## Τι θα φτιάξεις

Συμπλήρωσε τη συνάρτηση ώστε να επιστρέφει ακριβώς το κείμενο `"Hello!"`. Δεν δέχεται καμία παράμετρο.

**Όρια:** Δεν υπάρχει είσοδος. Η απάντηση έχει κεφαλαίο Η και ένα θαυμαστικό, χωρίς κενά ή αλλαγή γραμμής.


Μία κλήση. Η ίδια απάντηση κάθε φορά.

## Η ιδέα

Μια συνάρτηση μπορεί να δίνει πάντα την ίδια απάντηση. Το κείμενο γράφεται μέσα σε διπλά εισαγωγικά. Τα εισαγωγικά δείχνουν πού αρχίζει και πού τελειώνει το κείμενο. Δεν αποτελούν μέρος του αποτελέσματος.

ΜΙΑ ΣΤΑΘΕΡΗ ΑΠΑΝΤΗΣΗ

# 01 Γράψε και έλεγξε

Η συνάρτηση καλείται χωρίς ορίσματα.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve()` | `"Hello!"` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/01-hello/exercise.rb.

Tests: exercises/01-hello/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/01-hello/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το αρχείο έχει ήδη το περίβλημα της συνάρτησης. Χρειάζεσαι μόνο τη γραμμή που δίνει την απάντηση.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 355.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 02 Μια τιμή αποκτά όνομα

Η ίδια συνάρτηση μπορεί να δεχτεί διαφορετικούς αριθμούς σε διαφορετικές κλήσεις. Θα ονομάσουμε τον αριθμό `value` και θα τον επιστρέψουμε όπως τον παραλάβαμε.

RUBY

```ruby
def example(value)
  value
end
```

## Διάβασε τις γραμμές

01 Το `value` μέσα στις παρενθέσεις ορίζει μία παράμετρο. Δεν γράφουμε τύπο δίπλα της. Στο παράδειγμα δίνουμε ακέραιους.

02 Στον ορισμό το `value` είναι παράμετρος. Στην κλήση `example(6)`, το `6` είναι όρισμα. Η τιμή του είναι διαθέσιμη με το όνομα `value`.

03 Η τελευταία γραμμή του σώματος διαβάζει το `value` και το επιστρέφει. Το `"value"` με εισαγωγικά θα επέστρεφε κείμενο.

## Ακολούθησε μια κλήση

Η `example(6)` επιστρέφει `6`. Το `value` παίρνει την τιμή `6`.

Η `example(-3)` επιστρέφει `-3`. Σε νέα κλήση, το `value` παίρνει την τιμή `-3`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(0)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `0`. Επιστρέφουμε την τιμή της συγκεκριμένης κλήσης, όχι τον αριθμό που χρησιμοποιήσαμε στην προηγούμενη.

Συνέχισε στην εκφώνηση της άσκησης 02, στη σελίδα 29.

Η ΑΣΚΗΣΗ

# 02 Ο ίδιος αριθμός

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε τον ίδιο αριθμό.

**Όρια:** -1000 ≤ n ≤ 1000.


είσοδος αποτέλεσμα

Η τιμή περνά χωρίς να αλλάξει.

## Η ιδέα

Η παράμετρος `n` είναι το όνομα της τιμής που δίνεται στη συνάρτηση. Σε κάθε κλήση μπορεί να έχει άλλη τιμή. Το όνομα γράφεται χωρίς εισαγωγικά, γιατί θέλουμε την τιμή του.

Ο ΙΔΙΟΣ ΑΡΙΘΜΟΣ

# 02 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(8)` | `8` |
| `solve(-4)` | `-4` |
| `solve(0)` | `0` |
| `solve(-1000)` | `-1000` |
| `solve(1000)` | `1000` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/02-echo/exercise.rb.

Tests: exercises/02-echo/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/02-echo/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αν σου δώσουν 8, επιστρέφεις 8. Αν σου δώσουν -4, επιστρέφεις -4. Ποιο όνομα κρατά αυτή την τιμή;

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 356.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 03 Η πρόσθεση γράφεται με +

Μια έκφραση είναι ένα κομμάτι κώδικα που δίνει μια τιμή. Το `value + 3` είναι έκφραση. Υπολογίζουμε το άθροισμα πριν το επιστρέψουμε.

```ruby
def example(value)
  value + 3
end
```

## Διάβασε τις γραμμές

01 Το `+` προσθέτει δύο αριθμούς. Στο `value + 3` διαβάζεται η τιμή της παραμέτρου και προστίθεται το `3`.

02 Η έκφραση είναι η τελευταία του σώματος, οπότε επιστρέφεται το άθροισμα. Η παράμετρος `value` δεν αλλάζει.

03 Το `3` είναι αριθμός. Το `"3"` θα ήταν κείμενο και δεν μπορεί να προστεθεί σε έναν ακέραιο με αυτόν τον κώδικα.

## Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει `7`. Το `value + 3` γίνεται `4 + 3` και μετά `7`.

Η `example(8)` επιστρέφει `11`. Το `value + 3` γίνεται `8 + 3` και μετά `11`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(-3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `0`, επειδή το `-3 + 3` κάνει μηδέν.

Συνέχισε στην εκφώνηση της άσκησης 03, στη σελίδα 32.

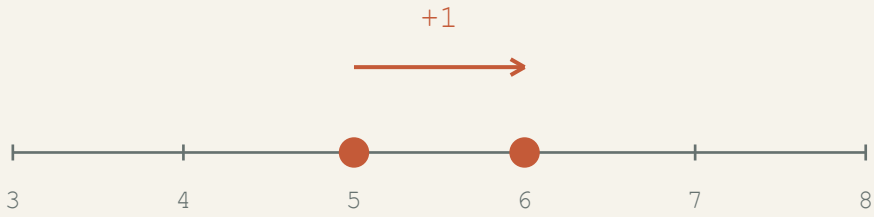
Η ΑΣΚΗΣΗ

# 03 Ο επόμενος αριθμός

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε τον αριθμό που είναι κατά 1 μεγαλύτερος.

**Όρια:** -1000 ≤ n ≤ 1000.




Ένα μόνο βήμα προς τα δεξιά.

## Η ιδέα

Μια έκφραση είναι ένα μικρό κομμάτι κώδικα που δίνει μια τιμή. Το σύμβολο `+` προσθέτει δύο αριθμούς. Μπορείς να επιστρέψεις κατευθείαν την τιμή μιας έκφρασης.

Ο ΕΠΟΜΕΝΟΣ ΑΡΙΘΜΟΣ

# 03 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(5)` | `6` |
| `solve(0)` | `1` |
| `solve(-1)` | `0` |
| `solve(-1000)` | `-999` |
| `solve(1000)` | `1001` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/03-next-number/exercise.rb.

Tests: exercises/03-next-number/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/03-next-number/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ξεκίνα από την τιμή που σου δόθηκε και πρόσθεσε μία μονάδα.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 357.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 04 Δύο παράμετροι, δύο θέσεις

Μετράμε μήλα και αχλάδια. Χρειαζόμαστε δύο τιμές στην ίδια κλήση. Κάθε τιμή παίρνει τη δική της θέση μέσα στις παρενθέσεις.

RUBY

```ruby
def example(apples, pears)
  apples + pears
end
```

## Διάβασε τις γραμμές

01 Το κόμμα στο `apples, pears` χωρίζει δύο παραμέτρους. Η καθεμία έχει δικό της όνομα.

02 Στην κλήση, το πρώτο όρισμα αντιστοιχεί στο `apples` και το δεύτερο στο `pears`. Το κόμμα χωρίζει και τα ορίσματα.

03 Το `apples + pears` διαβάζει τις δύο τιμές. Το άθροισμα είναι η τελευταία έκφραση και επιστρέφεται.

## Ακολούθησε μια κλήση

Η `example(2, 4)` επιστρέφει `6`. Το `apples` γίνεται `2` και το `pears` γίνεται `4`.

Η `example(3, 5)` επιστρέφει `8`. Το `3 + 5` δίνει `8`.

## Στάσου λίγο

Στην κλήση με ορίσματα `0, 7`, ποια τιμή παίρνει κάθε παράμετρος και τι επιστρέφεται;

**Έλεγξε τη σκέψη σου.** Το `apples` παίρνει `0` και το `pears` παίρνει `7`. Το άθροισμα είναι `7`.

Συνέχισε στην εκφώνηση της άσκησης 04, στη σελίδα 35.

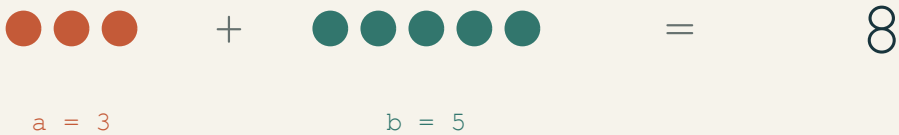
Η ΑΣΚΗΣΗ

# 04 Δύο αριθμοί μαζί

## Τι θα φτιάξεις

Δέξου δύο ακέραιους `a` και `b` και επέστρεψε το άθροισμά τους.

**Όρια:** -1000 ≤ a, b ≤ 1000.




`a = 3` `b = 5`

Δύο ποσότητες γίνονται ένα άθροισμα.

## Η ιδέα

Μια συνάρτηση μπορεί να δέχεται περισσότερες από μία τιμές. Τα ονόματα χωρίζονται με κόμμα. Στην κλήση `solve(3, 5)`, το 3 αντιστοιχεί στο `a` και το 5 στο `b`.

ΔΥΟ ΑΡΙΘΜΟΙ ΜΑΖΙ

# 04 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `a`, `b`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3, 5)` | `8` |
| `solve(0, 0)` | `0` |
| `solve(-3, 5)` | `2` |
| `solve(-4, -6)` | `-10` |
| `solve(1000, 1000)` | `2000` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/04-add/exercise.rb.

Tests: exercises/04-add/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/04-add/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η πρόσθεση είναι η ίδια που χρησιμοποίησες πριν. Τώρα και οι δύο αριθμοί δίνονται από την κλήση.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 358.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 05 Ο πολλαπλασιασμός γράφεται με *

Για να πάρουμε τρεις ίδιες ποσότητες, πολλαπλασιάζουμε την αρχική τιμή με το `3`. Στον κώδικα ο πολλαπλασιασμός γράφεται με αστερίσκο.

```ruby
def example(value)
  value * 3
end
```

## Διάβασε τις γραμμές

01 Το `*` είναι ο τελεστής του πολλαπλασιασμού. Χρειάζεται και ανάμεσα σε όνομα και αριθμό. Δεν γράφουμε `3value`.

02 Στο `value * 3`, η παράμετρος παίρνει τιμή από την κλήση και το `3` είναι σταθερό. Επιστρέφεται το γινόμενο.

03 Τα κενά γύρω από το `*` βοηθούν την ανάγνωση. Το μαθηματικό `×` και το γράμμα `x` δεν αντικαθιστούν τον αστερίσκο.

## Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει `12`. Το `4 * 3` είναι `4 + 4 + 4`.

Η `example(0)` επιστρέφει `0`. Το `0 * 3` παραμένει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(-2)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `-6`. Ο αστερίσκος λειτουργεί και με αρνητικό ακέραιο.

Συνέχισε στην εκφώνηση της άσκησης 05, στη σελίδα 38.

Η ΑΣΚΗΣΗ

# 05 Το διπλάσιο

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε το διπλάσιό του.

**Όρια:** -1000 ≤ n ≤ 1000.


Τέσσερα, δύο φορές.

## Η ιδέα

Ο πολλαπλασιασμός γράφεται με `*`. Η ιδέα είναι η ίδια με το να πάρεις μια ποσότητα δύο φορές. Δεν χρησιμοποιούμε το γράμμα x για πολλαπλασιασμό.

ΤΟ ΔΙΠΛΑΣΙΟ

# 05 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4)` | `8` |
| `solve(1)` | `2` |
| `solve(0)` | `0` |
| `solve(-3)` | `-6` |
| `solve(1000)` | `2000` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/05-double/exercise.rb.

Tests: exercises/05-double/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/05-double/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μπορείς πρώτα να το σκεφτείς ως πρόσθεση του αριθμού με τον εαυτό του.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 359.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 06 Κρατάμε ένα ενδιάμεσο αποτέλεσμα

Ένα κουτί έχει δέκα αντικείμενα. Δίνουμε όνομα σε αυτή την ποσότητα και μετά όνομα στο συνολικό πλήθος. Έτσι διαβάζουμε τον υπολογισμό σε μικρά βήματα.

RUBY

```ruby
def example(boxes)
  items_per_box = 10
  total = boxes * items_per_box
  total
end
```

## Διάβασε τις γραμμές

01 Το `items_per_box = 10` αποθηκεύει το `10` με όνομα `items_per_box`. Το `=` κάνει ανάθεση. Η κάτω παύλα χωρίζει λέξεις του ονόματος.

02 Το `total = boxes * items_per_box` υπολογίζει το γινόμενο και το αποθηκεύει στη μεταβλητή `total`. Και οι δύο πλευρές του `*` χρησιμοποιούν τιμές από ονόματα.

03 Το τελικό `total` διαβάζει και επιστρέφει την αποθηκευμένη τιμή. Οι τοπικές μεταβλητές ορίζονται ξανά σε κάθε κλήση.

## Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει `40`. Τέσσερα κουτιά επί δέκα αντικείμενα δίνουν `40`.

Η `example(0)` επιστρέφει `0`. Μηδέν κουτιά δίνουν `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `30`. Το όνομα της ενδιάμεσης τιμής δεν αλλάζει την πράξη που υπολογίζουμε.

Συνέχισε στην εκφώνηση της άσκησης 06, στη σελίδα 41.

Η ΑΣΚΗΣΗ

# 06 Από λεπτά σε δευτερόλεπτα

## Τι θα φτιάξεις

Δέξου έναν ακέραιο αριθμό λεπτών `minutes` και επέστρεψε πόσα δευτερόλεπτα είναι. Κάθε λεπτό έχει 60 δευτερόλεπτα. Δώσε πρώτα ένα όνομα στην τιμή 60. Πολλαπλασίασε το `minutes` με αυτό το όνομα, αποθήκευσε το αποτέλεσμα στο `seconds` και μετά επέστρεψέ το.

**Όρια:** 0 ≤ minutes ≤ 1000. Δουλεύουμε μόνο με ολόκληρα λεπτά.


3 λεπτά, 60 δευτερόλεπτα στο καθένα.

## Η ιδέα

Μπορούμε να δώσουμε ένα όνομα σε μια τιμή και να το χρησιμοποιήσουμε στην επόμενη γραμμή. Ο πολλαπλασιασμός λειτουργεί και όταν γράφουμε ονόματα και στις δύο πλευρές του `*`. Ο υπολογιστής χρησιμοποιεί τις τιμές τους. Το `seconds` λέει ποια ποσότητα έχουμε βρει.

ΑΠΟ ΛΕΠΤΑ ΣΕ ΔΕΥΤΕΡΟΛΕΠΤΑ

# 06 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `minutes`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3)` | `180` |
| `solve(1)` | `60` |
| `solve(0)` | `0` |
| `solve(60)` | `3600` |
| `solve(1000)` | `60000` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/06-minutes-to-seconds/exercise.rb.

Tests: exercises/06-minutes-to-seconds/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/06-minutes-to-seconds/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Για 3 λεπτά έχεις τρεις ομάδες των 60 δευτερολέπτων.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 360.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 07 Η αφαίρεση και η σειρά των ορισμάτων

Αφαιρούμε ένα κουπόνι από μια τιμή. Το `-` ανάμεσα σε δύο τιμές είναι αφαίρεση. Εδώ φαίνεται γιατί έχει σημασία η σειρά των ορισμάτων.

RUBY

```ruby
def example(price, coupon)
  price - coupon
end
```

## Διάβασε τις γραμμές

01 Το `price - coupon` αφαιρεί τη δεξιά τιμή από την αριστερή. Το πρώτο όρισμα πηγαίνει στο `price` και το δεύτερο στο `coupon`.

02 Αν ανταλλάξουμε τα ορίσματα, συνήθως αλλάζει το αποτέλεσμα. Η μέθοδος επιστρέφει τη διαφορά με τη σειρά που γράψαμε.

03 Στο `5 - 20`, το `-` αφαιρεί δύο τιμές. Στο αποτέλεσμα `-15` δείχνει ότι ο αριθμός είναι αρνητικός.

## Ακολούθησε μια κλήση

Η `example(20, 5)` επιστρέφει `15`. Το `price - coupon` γίνεται `20 - 5`.

Η `example(5, 20)` επιστρέφει `-15`. Με αντίστροφα ορίσματα γίνεται `5 - 20`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(12, 12)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `0`. Αφαιρούμε ίσες ποσότητες.

Συνέχισε στην εκφώνηση της άσκησης 07, στη σελίδα 44.

Η ΑΣΚΗΣΗ

# 07 Η διαφορά δύο αριθμών

## Τι θα φτιάξεις

Δέξου δύο ακέραιους `a` και `b` και επέστρεψε τη διαφορά τους, αφαιρώντας τον δεύτερο από τον πρώτο.

**Όρια:** -1000 ≤ a, b ≤ 1000.

8 − 3 = 5

Αφαιρούμε τρία. Απομένουν πέντε.

## Η ιδέα

Το − ανάμεσα σε δύο αριθμούς κάνει αφαίρεση. Η σειρά έχει σημασία. Το 8 μείον 3 δίνει 5, ενώ το 3 μείον 8 δίνει -5. Η συνάρτηση δέχεται δύο παραμέτρους όπως η άσκηση 4. Αλλάζει μόνο η πράξη.

Η ΔΙΑΦΟΡΑ ΔΥΟ ΑΡΙΘΜΩΝ

# 07 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `a`, `b`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(8, 3)` | `5` |
| `solve(3, 8)` | `-5` |
| `solve(0, 0)` | `0` |
| `solve(-4, -6)` | `2` |
| `solve(1000, -1000)` | `2000` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/07-difference/exercise.rb.

Tests: exercises/07-difference/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/07-difference/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Γράψε πρώτα τον αριθμό από τον οποίο αφαιρείς και μετά αυτόν που αφαιρείται.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 361.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 08 Οι παρενθέσεις αλλάζουν τη σειρά

Θέλουμε τρεις φορές το άθροισμα δύο αριθμών. Οι παρενθέσεις μας επιτρέπουν να πούμε ποια πράξη πρέπει να γίνει πρώτη.

RUBY

```ruby
def example(first, second)
  (first + second) * 3
end
```

## Διάβασε τις γραμμές

01 Οι παρενθέσεις στο `(first + second) * 3` ομαδοποιούν το άθροισμα. Πρώτα προσθέτουμε και μετά πολλαπλασιάζουμε.

02 Χωρίς αυτές, το `first + second * 3` θα έκανε πρώτα τον πολλαπλασιασμό. Με `2` και `4` θα έδινε `14` αντί για `18`.

03 Οι παρενθέσεις μέσα στην πράξη ορίζουν τη σειρά υπολογισμού. Εκείνες δίπλα στο όνομα της μεθόδου περιέχουν παραμέτρους ή ορίσματα.

## Ακολούθησε μια κλήση

Η `example(2, 4)` επιστρέφει `18`. Πρώτα `2 + 4 = 6`. Μετά `6 * 3 = 18`.

Η `example(1, 5)` επιστρέφει `18`. Πρώτα `1 + 5 = 6`. Μετά `6 * 3 = 18`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(0, 3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `9`. Πρώτα βρίσκουμε το `0 + 3`.

Συνέχισε στην εκφώνηση της άσκησης 08, στη σελίδα 47.

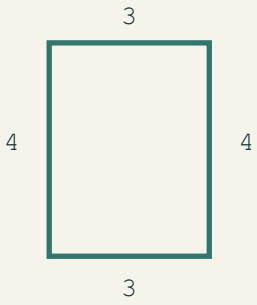
Η ΑΣΚΗΣΗ

# 08 Γύρω από το ορθογώνιο

## Τι θα φτιάξεις

Δέξου το πλάτος `width` και το ύψος `height` και επέστρεψε την περίμετρο: δύο φορές το άθροισμα πλάτους και ύψους.

**Όρια:** 0 ≤ width, height ≤ 1000. Εφαρμόζουμε τον ίδιο τύπο και για μηδενικές διαστάσεις.




```
2 × (3 + 4)
= 14
```

## Η ιδέα

Οι παρενθέσεις ομαδοποιούν μια πράξη. Το `2 * (3 + 4)` υπολογίζει πρώτα το άθροισμα. Χωρίς τις παρενθέσεις, ο πολλαπλασιασμός εκτελείται πριν από την πρόσθεση.

ΓΥΡΩ ΑΠΟ ΤΟ ΟΡΘΟΓΩΝΙΟ

# 08 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `width`, `height`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3, 4)` | `14` |
| `solve(1, 1)` | `4` |
| `solve(0, 5)` | `10` |
| `solve(5, 0)` | `10` |
| `solve(0, 0)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/08-rectangle-perimeter/exercise.rb.

Tests: exercises/08-rectangle-perimeter/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/08-rectangle-perimeter/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Βάλε μέσα σε παρενθέσεις την ποσότητα που θέλεις να πάρεις δύο φορές.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 362.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 09 Τι μένει από μια μοιρασιά

Μοιράζουμε έντεκα μάρκες σε τέσσερα κουτιά. Κάθε κουτί παίρνει δύο και μένουν τρεις. Το `11 = 4 * 2 + 3` δείχνει το πηλίκο `2` και το υπόλοιπο `3`. Εδώ ζητάμε μόνο το υπόλοιπο.

RUBY

```ruby
def example(tokens, boxes)
  tokens % boxes
end
```

## Διάβασε τις γραμμές

01 Το `%` δίνει το υπόλοιπο της διαίρεσης. Το `tokens % boxes` επιστρέφει όσες μάρκες περισσεύουν από τη μοιρασιά.

02 Το `11 % 4` είναι `3`. Δεν ζητάμε το πηλίκο `2` ούτε υπολογίζουμε ποσοστό.

03 Το `boxes` πρέπει να είναι θετικό, όπως ορίζουν οι είσοδοι του παραδείγματος. Διαίρεση με μηδέν προκαλεί σφάλμα.

## Ακολούθησε μια κλήση

Η `example(11, 4)` επιστρέφει `3`. Τέσσερα κουτιά επί δύο μάρκες χρησιμοποιούν οκτώ. Μένουν `3`.

Η `example(12, 4)` επιστρέφει `0`. Τέσσερα κουτιά επί τρεις μάρκες χρησιμοποιούν δώδεκα. Μένουν `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(14, 4)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Μένουν `2`, επειδή `14 = 4 * 3 + 2`.

Συνέχισε στην εκφώνηση της άσκησης 09, στη σελίδα 50.

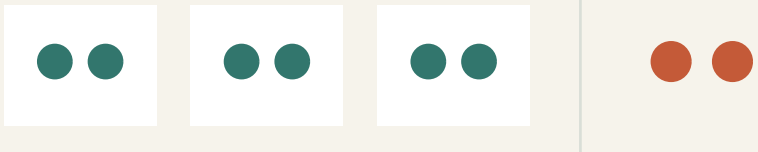
Η ΑΣΚΗΣΗ

# 09 Τι περισσεύει

## Τι θα φτιάξεις

Έχεις `candies` καραμέλες και `children` παιδιά. Δίνεις σε κάθε παιδί τον ίδιο ακέραιο αριθμό καραμελών, όσο περισσότερες γίνεται. Επέστρεψε πόσες περισσεύουν.

**Όρια:** 0 ≤ candies ≤ 1000 και 1 ≤ children ≤ 1000. Υπάρχει πάντα τουλάχιστον ένα παιδί.




8 καραμέλες, 3 παιδιά, 2 καραμέλες σε κάθε παιδί.

## Η ιδέα

Ας ξεκινήσουμε από τη μοιρασιά. Με 8 καραμέλες και 4 παιδιά, η διαίρεση 8 ÷ 4 δίνει 2 καραμέλες σε κάθε παιδί. Με 3 παιδιά δίνεις πάλι 2 στο καθένα, άρα μοιράζεις 3 × 2 = 6 και μένουν 8 - 6 = 2. Αυτό που μένει ονομάζεται υπόλοιπο. Το υπόλοιπο διαίρεσης δίνει ακριβώς αυτό που περισσεύει. Στις περισσότερες γλώσσες γράφεται με `%`. Στην Elixir γράφεται `rem(a, b)`.

ΤΙ ΠΕΡΙΣΣΕΥΕΙ

# 09 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `candies`, `children`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(8, 3)` | `2` |
| `solve(12, 4)` | `0` |
| `solve(2, 5)` | `2` |
| `solve(0, 3)` | `0` |
| `solve(7, 1)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/09-leftover/exercise.rb.

Tests: exercises/09-leftover/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/09-leftover/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Δεν ζητάμε πόσες παίρνει το κάθε παιδί. Ζητάμε ό,τι απομένει μετά τη μοιρασιά.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 363.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 10 Μια σύγκριση δίνει λογική τιμή

Μέχρι εδώ επιστρέφαμε αριθμό ή κείμενο. Τώρα ρωτάμε αν ένας αριθμός είναι ίσος με το `5`. Η απάντηση είναι μια λογική τιμή, αληθές ή ψευδές.

RUBY

```ruby
def example(value)
  value == 5
end
```

## Διάβασε τις γραμμές

01 Το `==` συγκρίνει δύο τιμές. Το `value == 5` ρωτά αν η παράμετρος είναι ίση με το `5`.

02 Το αποτέλεσμα είναι `true` ή `false`, χωρίς εισαγωγικά. Επιστρέφεται απευθείας ως τελευταία έκφραση.

03 Το μονό `=` κάνει ανάθεση. Αν γράψεις `value = 5`, αλλάζεις τη μεταβλητή αντί να ελέγξεις την τιμή της.

## Ακολούθησε μια κλήση

Η `example(5)` επιστρέφει `true`. Το `value` είναι ίσο με `5`.

Η `example(6)` επιστρέφει `false`. Το `6` δεν είναι ίσο με `5`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(0)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η σύγκριση είναι ψευδής. Το `0` δεν ισούται με το `5`.

Συνέχισε στην εκφώνηση της άσκησης 10, στη σελίδα 53.

Η ΑΣΚΗΣΗ

# 10 Είναι μηδέν;

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n`. Επέστρεψε αληθές αν είναι μηδέν και ψευδές σε κάθε άλλη περίπτωση.

**Όρια:** -1000 ≤ n ≤ 1000.


Μόνο το μηδέν απαντά «ναι».

## Η ιδέα

Μια σύγκριση απαντά σε μια ερώτηση. Το `n == 0` ελέγχει αν δύο τιμές είναι ίσες. Στην TypeScript χρησιμοποιούμε `===`. Το αποτέλεσμα είναι λογική τιμή, δηλαδή αληθές ή ψευδές.

ΕΙΝΑΙ ΜΗΔΕΝ;

# 10 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(0)` | `true` |
| `solve(1)` | `false` |
| `solve(-1)` | `false` |
| `solve(1000)` | `false` |
| `solve(-1000)` | `false` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/10-is-zero/exercise.rb.

Tests: exercises/10-is-zero/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/10-is-zero/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η ίδια η σύγκριση παράγει την απάντηση. Μπορείς να την επιστρέψεις απευθείας.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 364.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 11 Μεγαλύτερο σημαίνει αυστηρά μεγαλύτερο

Το σύμβολο `>` ρωτά αν η αριστερή τιμή είναι μεγαλύτερη από τη δεξιά. Η ισότητα δεν αρκεί.

RUBY

```ruby
def example(value)
  value > 10
end
```

## Διάβασε τις γραμμές

01 Το `>` σημαίνει «μεγαλύτερο από». Το `value > 10` επιστρέφει `true` μόνο όταν η τιμή ξεπερνά το `10`.

02 Η ισότητα δεν αρκεί. Το `10 > 10` δίνει `false`. Δεν χρειάζεται `if` για να επιστρέψουμε το αποτέλεσμα της σύγκρισης.

03 Οι τιμές `true` και `false` είναι λογικές τιμές. Το `"true"` θα ήταν κείμενο και θα αποτύγχανε στα tests λογικού αποτελέσματος.

## Ακολούθησε μια κλήση

Η `example(11)` επιστρέφει `true`. Το `11 > 10` είναι αληθές.

Η `example(10)` επιστρέφει `false`. Το `10 > 10` είναι ψευδές.

## Στάσου λίγο

Τι θα επιστρέψει η `example(9)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η σύγκριση είναι ψευδής, αφού το `9` είναι μικρότερο από το `10`.

Συνέχισε στην εκφώνηση της άσκησης 11, στη σελίδα 56.

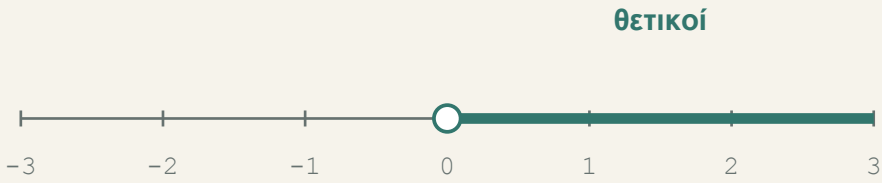
Η ΑΣΚΗΣΗ

# 11 Είναι θετικός;

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε αν είναι αυστηρά μεγαλύτερος από το μηδέν.

**Όρια:** -1000 ≤ n ≤ 1000.




Το μηδέν μένει έξω.

## Η ιδέα

Το `>` διαβάζεται «μεγαλύτερο από». Η λέξη αυστηρά σημαίνει ότι το ίδιο το όριο δεν περιλαμβάνεται. Το μηδέν δεν είναι θετικός αριθμός.

ΕΙΝΑΙ ΘΕΤΙΚΟΣ;

# 11 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3)` | `true` |
| `solve(1)` | `true` |
| `solve(0)` | `false` |
| `solve(-1)` | `false` |
| `solve(1000)` | `true` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/11-is-positive/exercise.rb.

Tests: exercises/11-is-positive/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/11-is-positive/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Δοκίμασε στο χαρτί τις τιμές -1, 0 και 1 πριν γράψεις τον έλεγχο.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 365.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 12 Όταν το όριο συμπεριλαμβάνεται

Έχουμε μια διαθέσιμη ποσότητα και μια απαιτούμενη. Ρωτάμε αν αρκεί. Ακριβώς όση χρειάζεται είναι αρκετή.

RUBY

```ruby
def example(available, needed)
  available >= needed
end
```

## Διάβασε τις γραμμές

01 Το `>=` σημαίνει «μεγαλύτερο ή ίσο». Τα δύο σύμβολα γράφονται ενωμένα, με αυτή τη σειρά.

02 Στο `available >= needed`, η αριστερή τιμή είναι ό,τι έχουμε και η δεξιά ό,τι χρειαζόμαστε. Η ισότητα δίνει `true`.

03 Η σύγκριση είναι η τελευταία έκφραση, άρα επιστρέφει τη λογική τιμή της. Δεν χρειάζεται ξεχωριστή περίπτωση για την ισότητα.

## Ακολούθησε μια κλήση

Η `example(4, 4)` επιστρέφει `true`. Η διαθέσιμη ποσότητα είναι ακριβώς η απαιτούμενη.

Η `example(3, 4)` επιστρέφει `false`. Λείπει μία μονάδα από την απαιτούμενη ποσότητα.

## Στάσου λίγο

Τι θα επιστρέψει η `example(5, 4)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι αληθής. Το `5` είναι μεγαλύτερο από το `4`.

Συνέχισε στην εκφώνηση της άσκησης 12, στη σελίδα 59.

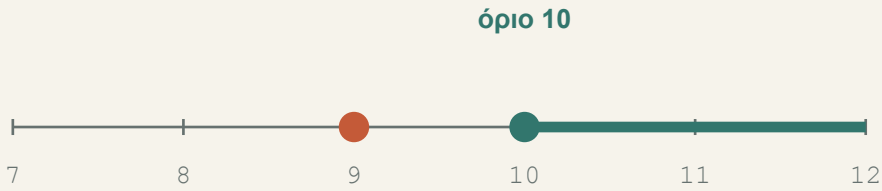
Η ΑΣΚΗΣΗ

# 12 Έφτασε το όριο;

## Τι θα φτιάξεις

Δέξου μια βαθμολογία `score` και ένα απαιτούμενο όριο `threshold`. Επέστρεψε αν η βαθμολογία έφτασε ή ξεπέρασε το όριο.

**Όρια:** 0 ≤ score, threshold ≤ 1000.




Το 9 είναι κάτω από το όριο. Το 10 το φτάνει.

## Η ιδέα

Το `>=` διαβάζεται «μεγαλύτερο ή ίσο». Όταν η εκφώνηση λέει «τουλάχιστον» ή «έφτασε το όριο», περιλαμβάνει και την ισότητα.

ΕΦΤΑΣΕ ΤΟ ΟΡΙΟ;

# 12 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `score`, `threshold`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(9, 10)` | `false` |
| `solve(10, 10)` | `true` |
| `solve(11, 10)` | `true` |
| `solve(0, 0)` | `true` |
| `solve(0, 1)` | `false` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/12-at-least/exercise.rb.

Tests: exercises/12-at-least/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/12-at-least/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αν το όριο είναι 10, σκέψου τι πρέπει να γίνει με βαθμολογίες 9, 10 και 11.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 366.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 13 Μια πράξη μέσα σε σύγκριση

Ενώνουμε δύο πράγματα που ήδη γνωρίζουμε. Για έναν μη αρνητικό ακέραιο, υπόλοιπο `1` στη διαίρεση με το `2` σημαίνει ότι περισσεύει μία μονάδα μετά τον σχηματισμό ζευγαριών.

RUBY

```ruby
def example(value)
  value % 2 == 1
end
```

## Διάβασε τις γραμμές

01 Πρώτα υπολογίζεται το `value % 2`. Μετά το `== 1` συγκρίνει το υπόλοιπο με το `1`.

02 Στο παράδειγμα αναγνωρίζουμε περιττούς αριθμούς. Το τελικό αποτέλεσμα είναι λογική τιμή, όχι το αριθμητικό υπόλοιπο.

03 Οι τελεστές είναι γνωστοί από τα προηγούμενα βήματα. Εδώ συνδυάζονται σε μία έκφραση που επιστρέφεται.

## Ακολούθησε μια κλήση

Η `example(7)` επιστρέφει `true`. Από το `7` μένει `1`. Συγκρίνουμε αυτό το `1` με το `1`.

Η `example(8)` επιστρέφει `false`. Από το `8` μένει `0`. Η σύγκριση με το `1` είναι ψευδής.

## Στάσου λίγο

Τι θα επιστρέψει η `example(9)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι αληθής. Το `9` αφήνει υπόλοιπο `1`.

Συνέχισε στην εκφώνηση της άσκησης 13, στη σελίδα 62.

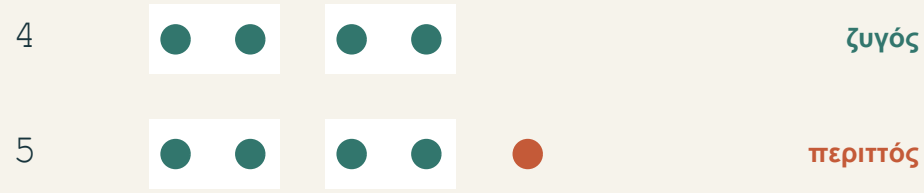
Η ΑΣΚΗΣΗ

# 13 Ζυγός αριθμός

## Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n`. Επέστρεψε αν είναι ζυγός, δηλαδή αν διαιρείται ακριβώς με το 2.

**Όρια:** 0 ≤ n ≤ 1000. Το 0 θεωρείται ζυγός αριθμός.




Χωρίζουμε σε ζευγάρια και κοιτάμε αν περισσεύει ένα.

## Η ιδέα

Στην άσκηση με τις καραμέλες έβρισκες ένα υπόλοιπο. Εδώ ελέγχεις αν το υπόλοιπο από τη διαίρεση με το 2 είναι μηδέν. Συνδυάζεις δύο πράγματα που έχεις ήδη χρησιμοποιήσει.

ΖΥΓΟΣ ΑΡΙΘΜΟΣ

# 13 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4)` | `true` |
| `solve(3)` | `false` |
| `solve(0)` | `true` |
| `solve(1)` | `false` |
| `solve(2)` | `true` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/13-is-even/exercise.rb.

Tests: exercises/13-is-even/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/13-is-even/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Πρώτα βρες τι μένει όταν χωρίσεις τον αριθμό σε ζευγάρια. Μετά έλεγξε αν έμεινε κάτι.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 367.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 14 Δύο δρόμοι με if και else

Μέχρι εδώ επιστρέφαμε τη σύγκριση. Τώρα τη χρησιμοποιούμε για να επιλέξουμε αριθμό. Πάνω από το `10` δίνουμε `1`. Σε κάθε άλλη περίπτωση δίνουμε `0`.

RUBY

```ruby
def example(value)
  if value > 10
    1
  else
    0
  end
end
```

## Διάβασε τις γραμμές

01 Το `if value > 10` ελέγχει τη συνθήκη. Αν είναι αληθής, επιλέγεται η γραμμή με το `1`. Το `else` καλύπτει την άλλη περίπτωση.

02 Στη Ruby το `if` έχει τιμή, την τελευταία έκφραση του κλάδου που εκτελέστηκε. Εδώ δίνει είτε `1` είτε `0` και η μέθοδος επιστρέφει αυτή την τιμή.

03 Το πρώτο `end` κλείνει το `if`. Το δεύτερο κλείνει τη μέθοδο. Οι εσοχές δείχνουν σε ποιο επίπεδο ανήκει κάθε γραμμή.

## Ακολούθησε μια κλήση

Η `example(12)` επιστρέφει `1`. Η συνθήκη είναι αληθής. Επιλέγεται ο πρώτος κλάδος.

Η `example(10)` επιστρέφει `0`. Η ισότητα δεν περνά το `>`. Επιλέγεται ο κλάδος `else`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `0`. Εκτελείται ο κλάδος `else`.

Συνέχισε στην εκφώνηση της άσκησης 14, στη σελίδα 65.

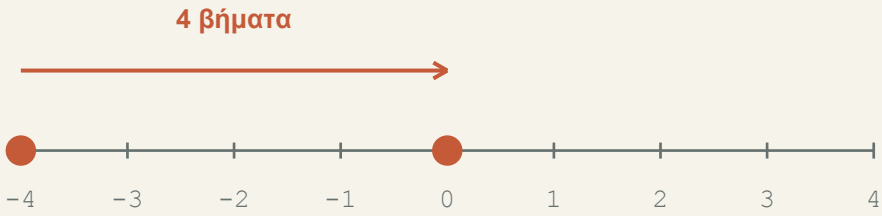
Η ΑΣΚΗΣΗ

# 14 Απόσταση από το μηδέν

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε την απόστασή του από το μηδέν. Για παράδειγμα, τόσο το -4 όσο και το 4 απέχουν 4. Χρησιμοποίησε μια συνθήκη, χωρίς έτοιμη συνάρτηση απόλυτης τιμής.

**Όρια:** -1000 ≤ n ≤ 1000.




Η απόσταση από το μηδέν δεν είναι αρνητική.

## Η ιδέα

Το `if` επιλέγει τι θα γίνει όταν μια συνθήκη είναι αληθής. Το `else` καλύπτει την άλλη περίπτωση. Για έναν αρνητικό αριθμό, το `-n` αλλάζει το πρόσημο. Για μη αρνητικό αριθμό κρατάμε την τιμή του.

ΑΠΟΣΤΑΣΗ ΑΠΟ ΤΟ ΜΗΔΕΝ

# 14 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(-4)` | `4` |
| `solve(4)` | `4` |
| `solve(0)` | `0` |
| `solve(-1)` | `1` |
| `solve(1)` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/14-absolute/exercise.rb.

Tests: exercises/14-absolute/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/14-absolute/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Χώρισε τις εισόδους σε δύο περιπτώσεις: κάτω από το μηδέν και όλες τις υπόλοιπες.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 368.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 15 Ένας κλάδος μπορεί να επιστρέφει παράμετρο

Θα κρατήσουμε τον μικρότερο από δύο αριθμούς. Χρησιμοποιούμε την ίδια επιλογή με πριν, αλλά κάθε κλάδος επιστρέφει μια από τις τιμές που πήρε η συνάρτηση.

RUBY

```ruby
def example(a, b)
  if a > b
    b
  else
    a
  end
end
```

## Διάβασε τις γραμμές

01 Το `if a > b` συγκρίνει τις δύο παραμέτρους. Αν η συνθήκη είναι αληθής, ο πρώτος κλάδος δίνει το `b`.

02 Το `else` δίνει το `a`. Το παράδειγμα επιλέγει τη μικρότερη τιμή. Η άσκηση που ακολουθεί θα ζητήσει τη μεγαλύτερη.

03 Όταν οι τιμές είναι ίσες, εκτελείται το `else`. Και οι δύο επιλογές τότε έχουν την ίδια τιμή. Τα δύο `end` κλείνουν το `if` και τη μέθοδο.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 15 Από τον κώδικα στην τιμή


## Ακολούθησε μια κλήση

Η `example(3, 8)` επιστρέφει `3`. Το `3 > 8` είναι ψευδές. Επιστρέφεται το `a`.

Η `example(9, 4)` επιστρέφει `4`. Το `9 > 4` είναι αληθές. Επιστρέφεται το `b`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(5, 5)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `5`. Η συνθήκη είναι ψευδής και ο άλλος κλάδος επιστρέφει το `a`. Με ίσες τιμές και οι δύο επιλογές δίνουν το ίδιο.

Συνέχισε στην εκφώνηση της άσκησης 15, στη σελίδα 69.

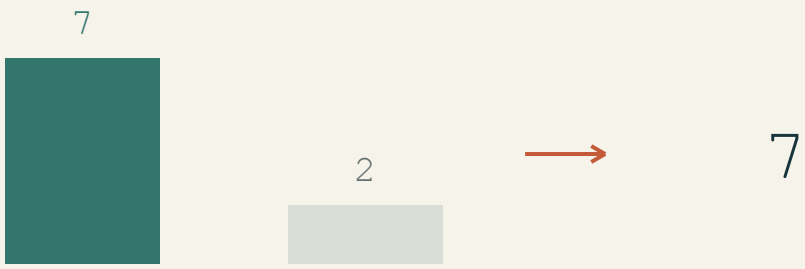
Η ΑΣΚΗΣΗ

# 15 Ο μεγαλύτερος από τους δύο

## Τι θα φτιάξεις

Δέξου δύο ακέραιους `a` και `b` και επέστρεψε τον μεγαλύτερο. Αν είναι ίσοι, επέστρεψε αυτή την κοινή τιμή. Χρησιμοποίησε `if`, χωρίς έτοιμη συνάρτηση μέγιστου.

**Όρια:** -1000 ≤ a, b ≤ 1000.




Κρατάμε τη μεγαλύτερη τιμή.

## Η ιδέα

Η συνθήκη δεν χρειάζεται να συγκρίνει μια παράμετρο με έναν σταθερό αριθμό. Μπορεί να συγκρίνει τις δύο παραμέτρους μεταξύ τους και να επιλέξει ποια τιμή θα επιστρέψει.

Ο ΜΕΓΑΛΥΤΕΡΟΣ ΑΠΟ ΤΟΥΣ ΔΥΟ

# 15 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `a`, `b`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(7, 2)` | `7` |
| `solve(2, 7)` | `7` |
| `solve(5, 5)` | `5` |
| `solve(-7, -2)` | `-2` |
| `solve(0, -1)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/15-larger/exercise.rb.

Tests: exercises/15-larger/exercise_spec.rb.

SH
```
bundle exec rspec \
  exercises/15-larger/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αν το `a` δεν είναι μεγαλύτερο από το `b`, ποια από τις δύο τιμές μπορείς να επιστρέψεις;

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 369.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 16 Μια τρίτη περίπτωση

Ξεχωρίζουμε τρεις περιπτώσεις γύρω από το `10`. Πάνω από δέκα δίνουμε `2`, ακριβώς δέκα δίνουμε `1` και κάτω από δέκα δίνουμε `0`. Ελέγχουμε με αυτή τη σειρά.

RUBY

```ruby
def example(value)
  if value > 10
    2
  elsif value == 10
    1
  else
    0
  end
end
```

## Διάβασε τις γραμμές

01 Το `elsif` σημαίνει «αλλιώς, αν». Γράφεται ως μία λέξη. Ελέγχεται μόνο όταν δεν πέρασε η προηγούμενη συνθήκη.

02 Η αλυσίδα επιλέγει έναν κλάδο. Για τιμή πάνω από `10` δίνει `2`, για ακριβώς `10` δίνει `1` και αλλιώς δίνει `0`.

03 Ένα `end` κλείνει ολόκληρη την αλυσίδα `if / elsif / else`. Το επόμενο κλείνει τη μέθοδο που επιστρέφει την επιλεγμένη τιμή.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 16 Από τον κώδικα στην τιμή


## Ακολούθησε μια κλήση

Η `example(12)` επιστρέφει `2`. Περνά η πρώτη συνθήκη. Δεν χρειάζεται ο δεύτερος έλεγχος.

Η `example(10)` επιστρέφει `1`. Αποτυγχάνει ο πρώτος έλεγχος. Περνά η ισότητα.

## Στάσου λίγο

Τι θα επιστρέψει η `example(6)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `0`. Αποτυγχάνουν και οι δύο έλεγχοι, οπότε επιλέγεται η τελευταία περίπτωση.

Συνέχισε στην εκφώνηση της άσκησης 16, στη σελίδα 73.

Η ΑΣΚΗΣΗ

# 16 Τρεις περιπτώσεις

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n`. Επέστρεψε -1 αν είναι αρνητικός, 0 αν είναι μηδέν και 1 αν είναι θετικός.

**Όρια:** -1000 ≤ n ≤ 1000.

| αρνητικός | μηδέν | θετικός |
|---|---|---|
| −1 | 0 | 1 |

Τρεις περιπτώσεις. Μία απάντηση για καθεμία.

## Η ιδέα

Μερικές αποφάσεις έχουν τρεις περιπτώσεις. Ελέγχουμε την πρώτη και, αν δεν ισχύει, συνεχίζουμε στη δεύτερη. Ό,τι απομένει καλύπτεται από την τελευταία περίπτωση.

ΤΡΕΙΣ ΠΕΡΙΠΤΩΣΕΙΣ

# 16 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(-7)` | `-1` |
| `solve(0)` | `0` |
| `solve(42)` | `1` |
| `solve(-1)` | `-1` |
| `solve(1)` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/16-sign/exercise.rb.

Tests: exercises/16-sign/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/16-sign/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Έλεγξε πρώτα αν ο αριθμός είναι κάτω από το μηδέν. Αν δεν είναι, τι χρειάζεται να ξεχωρίσεις ακόμη;

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 370.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 17 Δύο συνθήκες πρέπει να ισχύουν μαζί

Μια επιτρεπτή τιμή πρέπει να είναι τουλάχιστον `2` και το πολύ ίση με το `limit`. Χρειαζόμαστε τη λογική πράξη «και». Το κάτω όριο μένει σταθερό, άρα συνεχίζουμε με δύο παραμέτρους.

RUBY

```ruby
def example(value, limit)
  (value >= 2) && (value <= limit)
end
```

## Διάβασε τις γραμμές

01 Το `&&` συνδέει δύο συνθήκες που πρέπει να ισχύουν μαζί. Εδώ και οι δύο πλευρές είναι συγκρίσεις, άρα το αποτέλεσμα είναι `true` ή `false`.

02 Το `value >= 2` ελέγχει το κάτω όριο. Το `value <= limit` ελέγχει το πάνω όριο. Οι παρενθέσεις ξεχωρίζουν τις δύο συγκρίσεις.

03 Αν η πρώτη συνθήκη είναι ψευδής, η Ruby δεν εξετάζει τη δεύτερη. Μία αποτυχημένη σύγκριση αρκεί για `false`.

## Ακολούθησε μια κλήση

Η `example(5, 8)` επιστρέφει `true`. Ισχύουν και το κάτω και το πάνω όριο.

Η `example(1, 8)` επιστρέφει `false`. Η πρώτη συνθήκη είναι ψευδής. Αυτό αρκεί για ψευδές αποτέλεσμα.

## Στάσου λίγο

Τι θα επιστρέψει η `example(9, 8)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι ψευδής. Το `9` περνά το κάτω όριο, αλλά ξεπερνά το πάνω όριο `8`.

Συνέχισε στην εκφώνηση της άσκησης 17, στη σελίδα 76.

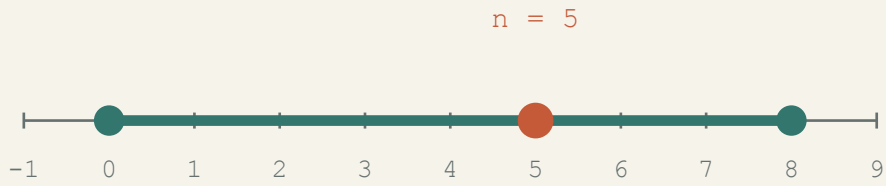
Η ΑΣΚΗΣΗ

# 17 Μέσα στα όρια

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και ένα μη αρνητικό πάνω όριο `high`. Επέστρεψε αν ο αριθμός βρίσκεται από το 0 έως το `high`, συμπεριλαμβανομένων και των δύο άκρων. Το κάτω όριο είναι πάντα 0.

**Όρια:** -1000 ≤ n ≤ 1000 και 0 ≤ high ≤ 1000.




Από το 0 έως το 8, μαζί με τα δύο άκρα.

## Η ιδέα

Για να βρίσκεται ένας αριθμός μέσα σε ένα διάστημα, πρέπει να ικανοποιούνται δύο έλεγχοι μαζί. Το λογικό «και» γράφεται `and` στην Python και στην Elixir και `&&` στις υπόλοιπες γλώσσες.

ΜΕΣΑ ΣΤΑ ΟΡΙΑ

# 17 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`, `high`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(5, 8)` | `true` |
| `solve(0, 8)` | `true` |
| `solve(8, 8)` | `true` |
| `solve(-1, 8)` | `false` |
| `solve(9, 8)` | `false` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/17-in-range/exercise.rb.

Tests: exercises/17-in-range/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/17-in-range/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Γράψε δύο ξεχωριστές προτάσεις: «ο αριθμός δεν είναι κάτω από το μηδέν» και «δεν είναι πάνω από το πάνω όριο».

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 371.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 18 Ο διαιρέτης μπορεί να αλλάζει

Θέλουμε να χωρίσουμε ένα πλήθος σε ίσες ομάδες, χωρίς περίσσευμα. Το μέγεθος της ομάδας δίνεται στη δεύτερη παράμετρο. Αυτό συνδυάζει γνωστή σύνταξη, δεν προσθέτει νέα εντολή.

RUBY

```ruby
def example(total, size)
  total % size == 0
end
```

## Διάβασε τις γραμμές

01 Το `total % size` υπολογίζει το υπόλοιπο από ομάδες μεγέθους `size`. Το `size` δίνεται ως παράμετρος και είναι θετικό.

02 Το `== 0` ελέγχει αν δεν περισσεύει τίποτα. Η έκφραση επιστρέφει λογική τιμή, όπως στην άσκηση με τους άρτιους.

03 Το μηδέν χωρίζεται χωρίς υπόλοιπο με κάθε θετικό μέγεθος. Το `% 0` όμως θα προκαλούσε σφάλμα και δεν είναι έγκυρη είσοδος εδώ.

## Ακολούθησε μια κλήση

Η `example(12, 4)` επιστρέφει `true`. Το `12` χωρίζεται σε ομάδες των `4` χωρίς υπόλοιπο.

Η `example(14, 4)` επιστρέφει `false`. Το `14` αφήνει υπόλοιπο `2`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(15, 5)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι αληθής. Το `15` χωρίζεται σε τρεις ομάδες των `5`.

Συνέχισε στην εκφώνηση της άσκησης 18, στη σελίδα 79.

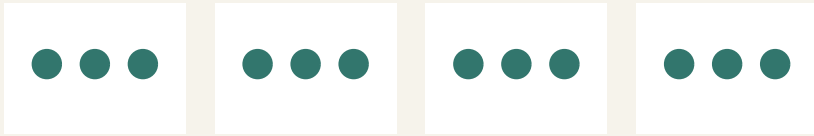
Η ΑΣΚΗΣΗ

# 18 Χωρίς υπόλοιπο

## Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n` και έναν θετικό ακέραιο `divisor`. Επέστρεψε αν ο πρώτος είναι πολλαπλάσιο του δεύτερου, δηλαδή αν η διαίρεσή τους δεν αφήνει υπόλοιπο.

**Όρια:** 0 ≤ n ≤ 1000 και 1 ≤ divisor ≤ 1000. Το 0 είναι πολλαπλάσιο κάθε επιτρεπτού διαιρέτη.



Το 12 χωρίζεται σε ομάδες των 3 χωρίς υπόλοιπο.

## Η ιδέα

Ο έλεγχος για ζυγό αριθμό ήταν μια ειδική περίπτωση με διαιρέτη το 2. Εδώ ο διαιρέτης δίνεται στη συνάρτηση. Η ιδέα παραμένει ίδια.

ΧΩΡΙΣ ΥΠΟΛΟΙΠΟ

# 18 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`, `divisor`. Η τιμή `true` σημαίνει αληθές και η `false` ψευδές.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(12, 3)` | `true` |
| `solve(13, 3)` | `false` |
| `solve(0, 7)` | `true` |
| `solve(7, 1)` | `true` |
| `solve(2, 5)` | `false` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/18-is-multiple/exercise.rb.

Tests: exercises/18-is-multiple/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/18-is-multiple/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ξαναδές την άσκηση 13 και σκέψου ποια σταθερά πρέπει να γίνει παράμετρος.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 372.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 19 Η απόφαση μπορεί να δίνει κείμενο

Για έναν αριθμό που σχηματίζει πλήρη ζευγάρια επιστρέφουμε το κείμενο `"Pair"`. Διαφορετικά επιστρέφουμε κενό κείμενο. Οι κλάδοι παραμένουν δύο, αλλά ο τύπος της απάντησης αλλάζει.

RUBY

```ruby
def example(value)
  if value % 2 == 0
    "Pair"
  else
    ""
  end
end
```

## Διάβασε τις γραμμές

01 Το `value % 2 == 0` ελέγχει αν ο αριθμός είναι άρτιος. Το `if` επιλέγει ποιο κείμενο θα δώσει.

02 Ο πρώτος κλάδος δίνει `"Pair"`. Ο κλάδος `else` δίνει `""`, κείμενο με μηδέν χαρακτήρες. Το `" "` θα περιείχε ένα κενό.

03 Η μέθοδος επιστρέφει το κείμενο του επιλεγμένου κλάδου. Δεν γράφουμε `puts` μέσα της, γιατί τα tests ελέγχουν την τιμή που επιστρέφει.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 19 Από τον κώδικα στην τιμή


## Ακολούθησε μια κλήση

Η `example(8)` επιστρέφει `"Pair"`. Δεν μένει υπόλοιπο. Επιλέγεται το πρώτο κείμενο.

Η `example(7)` επιστρέφει `""`. Μένει υπόλοιπο `1`. Επιλέγεται το κενό κείμενο.

## Στάσου λίγο

Τι θα επιστρέψει η `example(0)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `"Pair"`. Το μηδέν αφήνει μηδενικό υπόλοιπο στη διαίρεση με το `2`.

Συνέχισε στην εκφώνηση της άσκησης 19, στη σελίδα 83.

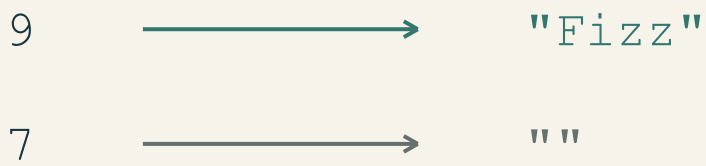
Η ΑΣΚΗΣΗ

# 19 Η πρώτη λέξη του FizzBuzz

## Τι θα φτιάξεις

Δέξου έναν θετικό ακέραιο `n`. Αν είναι πολλαπλάσιο του 3, επέστρεψε `"Fizz"`. Σε κάθε άλλη περίπτωση επέστρεψε το κενό κείμενο `""`.

**Όρια:** 1 ≤ n ≤ 1000. Επιστρέφουμε μόνο μια ετικέτα. Δεν μετατρέπουμε τον αριθμό σε κείμενο.




Μερικές φορές η απάντηση είναι κενό κείμενο.

## Η ιδέα

Έχεις ήδη επιστρέψει κείμενο στην πρώτη άσκηση και έχεις ελέγξει πολλαπλάσια. Τώρα τα συνδυάζεις. Το `if` μπορεί να επιλέξει κείμενο όπως επέλεγε αριθμό στην άσκηση 15.

Η ΠΡΩΤΗ ΛΕΞΗ ΤΟΥ FIZZBUZZ

# 19 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3)` | `"Fizz"` |
| `solve(1)` | `""` |
| `solve(5)` | `""` |
| `solve(6)` | `"Fizz"` |
| `solve(15)` | `"Fizz"` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/19-fizz/exercise.rb.

Tests: exercises/19-fizz/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/19-fizz/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Πρώτα απάντησε αν ο αριθμός διαιρείται ακριβώς με το 3. Μετά διάλεξε ένα από τα δύο κείμενα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 373.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 20 Η σειρά των κανόνων έχει σημασία

Δίνουμε τις ετικέτες `"Pair"` για πολλαπλάσια του `2` και `"Trio"` για πολλαπλάσια του `3`. Όταν ισχύουν και οι δύο κανόνες δίνουμε `"PairTrio"`. Συνδυάζουμε όσα ήδη μάθαμε.

RUBY

```ruby
def example(value)
  if (value % 2 == 0) && (value % 3 == 0)
    "PairTrio"
  elsif value % 2 == 0
    "Pair"
  elsif value % 3 == 0
    "Trio"
  else
    ""
  end
end
```

## Διάβασε τις γραμμές

01 Η πρώτη συνθήκη συνδέει με `&&` τους ελέγχους για πολλαπλάσιο του `2` και του `3`. Πρέπει να ισχύουν και οι δύο.

02 Τα δύο `elsif` εξετάζουν τις απλές περιπτώσεις με τη σειρά. Μόλις πετύχει μία συνθήκη, δεν εξετάζονται οι επόμενες.

03 Το `else` δίνει κενό κείμενο όταν κανένας έλεγχος δεν πετύχει. Όλοι οι κλάδοι δίνουν κείμενο, που επιστρέφεται από τη μέθοδο.

04 Ο κοινός έλεγχος μπαίνει πρώτος. Αν ο έλεγχος του `2` ήταν πρώτος, το `6` θα έπαιρνε `"Pair"` αντί για `"PairTrio"`.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 20 Από τον κώδικα στην τιμή


## Ακολούθησε μια κλήση

Η `example(6)` επιστρέφει `"PairTrio"`. Ισχύουν και οι δύο κανόνες. Επιλέγεται ο πρώτος κλάδος.

Η `example(4)` επιστρέφει `"Pair"`. Ισχύει μόνο ο κανόνας του `2`.

Η `example(9)` επιστρέφει `"Trio"`. Ισχύει μόνο ο κανόνας του `3`.

Η `example(5)` επιστρέφει `""`. Δεν ισχύει κανένας από τους δύο κανόνες.

## Στάσου λίγο

Τι θα επιστρέψει η `example(12)`; Βρες το αποτέλεσμα με το χέρι πριν διαβάσεις παρακάτω.

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `"PairTrio"`. Ο κοινός έλεγχος πετυχαίνει πρώτος. Αν βάζαμε πρώτο τον απλό έλεγχο του `2`, θα παίρναμε λάθος `"Pair"`.

Συνέχισε στην εκφώνηση της άσκησης 20, στη σελίδα 87.

Η ΑΣΚΗΣΗ

# 20 Οι κανόνες του FizzBuzz

## Τι θα φτιάξεις

Δέξου έναν θετικό ακέραιο `n` και επέστρεψε τη λέξη που του αντιστοιχεί: `"FizzBuzz"` αν είναι πολλαπλάσιο και του 3 και του 5, `"Fizz"` αν είναι μόνο του 3, `"Buzz"` αν είναι μόνο του 5 και `""` αν δεν είναι κανενός από τα δύο.

**Όρια:** 1 ≤ n ≤ 1000. Η συνάρτηση επιστρέφει μόνο την ετικέτα για έναν αριθμό. Για παράδειγμα, στο 7 επιστρέφει `""`, όχι `"7"`.

| n | με το 3; | με το 5; | απάντηση |
|---|---|---|---|
| 15 | ναι | ναι | `FizzBuzz` |
| 9 | ναι | όχι | `Fizz` |
| 10 | όχι | ναι | `Buzz` |
| 7 | όχι | όχι | `""` |

Η κοινή περίπτωση προηγείται.

## Η ιδέα

Ένας αριθμός μπορεί να περνά δύο ελέγχους ταυτόχρονα. Το 15 είναι πολλαπλάσιο και του 3 και του 5. Σε μια αλυσίδα αποφάσεων, η πρώτη περίπτωση που ταιριάζει καθορίζει την απάντηση.

ΟΙ ΚΑΝΟΝΕΣ ΤΟΥ FIZZBUZZ

# 20 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(15)` | `"FizzBuzz"` |
| `solve(9)` | `"Fizz"` |
| `solve(10)` | `"Buzz"` |
| `solve(7)` | `""` |
| `solve(1)` | `""` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/20-fizzbuzz-label/exercise.rb.

Tests: exercises/20-fizzbuzz-label/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/20-fizzbuzz-label/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Πριν ελέγξεις ξεχωριστά το 3 και το 5, ξεχώρισε την περίπτωση όπου ισχύουν και τα δύο.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 374.

ΑΣΚΗΣΕΙΣ 21-40

# Συνεχίζουμε στη Ruby

Από τις μεμονωμένες τιμές περνάμε στις επαναλήψεις, στους πίνακες και στα κείμενα. Συνέχισε να γράφεις στο `exercise.rb` και να τρέχεις το διπλανό `exercise_spec.rb` με RSpec.

## Δοκίμασε έναν πίνακα

Μέσα σε ένα spec δίνεις κανονική τιμή Ruby, όπως `solve([2, 4, 6])`. Για να εμφανίσεις μία απάντηση στο Terminal, αφού λύσεις την άσκηση 30, από τον φάκελο `ruby`:

```sh
ruby -r ./exercises/30-sum-an-array/exercise.rb \
  -e 'p solve([2, 4, 6])'
```

Το `p` εμφανίζει την τιμή με αγκύλες όταν είναι πίνακας. Για κείμενο με αλλαγές γραμμής, το `puts` μπορεί να εμφανίσει κάθε γραμμή χωριστά. Η μέθοδος επιστρέφει τιμή και η εντολή που την καλεί αναλαμβάνει την εμφάνιση.

## Πριν από το πλήρες FizzBuzz

Στην 38η άσκηση εφαρμόζεις έναν κανόνα σε κάθε θέση. Στην 39η συνδυάζεις τους κανόνες και στην 40ή ενώνεις τις απαντήσεις σε ένα κείμενο. Αν δυσκολευτείς, γύρισε στο μικρότερο βήμα που του αντιστοιχεί.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 21 Το ίδιο όνομα, μια νέα τιμή

Μέχρι τώρα δίναμε ονόματα σε τιμές. Τώρα θα ενημερώσουμε ένα από αυτά τα ονόματα, χρησιμοποιώντας πρώτα την τιμή που ήδη κρατά.

RUBY

```ruby
def example(start)
  value = start
  value = value + 4
  value
end
```

## Διάβασε τις γραμμές

01 Το `value = start` αντιγράφει την τιμή της παραμέτρου σε μια τοπική μεταβλητή.

02 Στο `value = value + 4`, πρώτα διαβάζεται το παλιό `value` και γίνεται η πρόσθεση. Μετά το αποτέλεσμα αποθηκεύεται στο ίδιο όνομα.

03 Το `=` είναι ανάθεση, όχι μαθηματική ισότητα. Η τελευταία γραμμή επιστρέφει τη νέα τιμή.

## Ακολούθησε μια κλήση

Η `example(2)` επιστρέφει `6`.

Η `example(-4)` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(0)`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `4`. Η δεξιά πλευρά υπολογίζει `0 + 4` πριν ενημερωθεί η μεταβλητή.

Συνέχισε στην εκφώνηση της άσκησης 21, στη σελίδα 91.

Η ΑΣΚΗΣΗ

# 21 Δύο αυξήσεις στην ίδια τιμή

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n`. Αποθήκευσέ τον στη μεταβλητή `value`, αύξησε την ίδια μεταβλητή κατά `1` δύο φορές και επέστρεψε την τελική τιμή. Χρησιμοποίησε δύο χωριστές αναθέσεις.

**Όρια:** -1000 ≤ n ≤ 1000.

**Η κλήση**

```
solve(3)
```

↓ Η τιμή που επιστρέφεται

```
5
```

## Η ιδέα

Μια μεταβλητή μπορεί να πάρει νέα τιμή. Στη δεξιά πλευρά διαβάζουμε την προηγούμενη τιμή και μετά αντικαθιστούμε εκείνη που κρατούσε το όνομα.

ΔΥΟ ΑΥΞΗΣΕΙΣ ΣΤΗΝ ΙΔΙΑ ΤΙΜΗ

# 21 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3)` | `5` |
| `solve(0)` | `2` |
| `solve(-2)` | `0` |
| `solve(-10)` | `-8` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/21-update-a-value/exercise.rb.

Tests: exercises/21-update-a-value/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/21-update-a-value/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ξεκίνα με `value = n`. Σε κάθε επόμενη ανάθεση, η δεξιά πλευρά χρησιμοποιεί την τιμή που υπάρχει εκείνη τη στιγμή.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 375.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 22 Επανέλαβε όσο ισχύει μια συνθήκη

Θα ανεβαίνουμε κατά τρία μέχρι να φτάσουμε σε ένα όριο. Δεν ξέρουμε από πριν πόσες επαναλήψεις θα χρειαστούν.

RUBY

```ruby
def example(limit)
  value = 0
  while value < limit
    value = value + 3
  end
  value
end
```

## Διάβασε τις γραμμές

01 Το `while value < limit` ελέγχεται πριν εκτελεστεί το σώμα και ξανά μετά από κάθε πέρασμα.

02 Το `value = value + 3` μετακινεί την τιμή προς το όριο. Με όριο `5`, οι τιμές γίνονται `0`, `3`, `6`.

03 Το πρώτο `end` κλείνει το `while`. Το τελικό `value` βρίσκεται έξω από τον βρόχο και επιστρέφει την τιμή με την οποία σταμάτησε.

## Ακολούθησε μια κλήση

Η `example(5)` επιστρέφει `6`.

Η `example(0)` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(7)`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `9`. Μετά το `6` γίνεται άλλο ένα πέρασμα και το `9 < 7` είναι ψευδές.

Συνέχισε στην εκφώνηση της άσκησης 22, στη σελίδα 94.

Η ΑΣΚΗΣΗ

# 22 Ανέβα με βήματα των δύο

## Τι θα φτιάξεις

Δέξου τους ακεραίους `start` και `limit`. Ξεκίνα από το `start` και πρόσθετε `2` όσο η τιμή είναι μικρότερη από το `limit`. Επέστρεψε την πρώτη τιμή που φτάνει ή ξεπερνά το όριο. Χρησιμοποίησε `while`.

**Όρια:** 0 ≤ start ≤ 100. 0 ≤ limit ≤ 100.

**Η κλήση**

```
solve(1, 6)
```

Η τιμή που επιστρέφεται

```
7
```

## Η ιδέα

Το `while` ελέγχει μια συνθήκη πριν από κάθε επανάληψη. Αν είναι ήδη ψευδής, το σώμα δεν εκτελείται ούτε μία φορά.

ΑΝΕΒΑ ΜΕ ΒΗΜΑΤΑ ΤΩΝ ΔΥΟ

# 22 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `start, limit`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(1, 6)` | `7` |
| `solve(2, 6)` | `6` |
| `solve(6, 6)` | `6` |
| `solve(8, 3)` | `8` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/22-step-to-a-limit/exercise.rb.

Tests: exercises/22-step-to-a-limit/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/22-step-to-a-limit/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κράτα την τρέχουσα τιμή σε μεταβλητή. Η αύξησή της πρέπει να βρίσκεται μέσα στο `while`.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 376.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 23 Ένα όνομα μετρά, ένα όνομα αθροίζει

Κάθε γύρος δίνει τέσσερις πόντους. Θα μετράμε τους γύρους και θα κρατάμε χωριστά τους πόντους που έχουμε συγκεντρώσει.

RUBY

```ruby
def example(rounds)
  count = 0
  total = 0
  while count < rounds
    total = total + 4
    count = count + 1
  end
  total
end
```

## Διάβασε τις γραμμές

01 Το `count` μετρά πόσα περάσματα έγιναν. Το `total` κρατά το άθροισμα των πόντων. Και τα δύο αρχίζουν πριν από τον βρόχο.

02 Σε κάθε πέρασμα προσθέτουμε `4` στο `total` και `1` στο `count`. Οι δύο μεταβλητές έχουν διαφορετική δουλειά.

03 Όταν το `count` φτάσει το `rounds`, σταματά η επανάληψη. Επιστρέφουμε το άθροισμα, όχι τον μετρητή.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 23 Από τον κώδικα στην τιμή

**Η κλήση**

```
example(3)
```

↓ Η τιμή που επιστρέφεται

```
12
```

## Ακολούθησε μια κλήση

Η `example(3)` επιστρέφει `12`.

Η `example(0)` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2)`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `8`. Τα δύο περάσματα προσθέτουν `4` και άλλη μία φορά `4`.

Συνέχισε στην εκφώνηση της άσκησης 23, στη σελίδα 98.

Η ΑΣΚΗΣΗ

# 23 Άθροισμα από το 1 έως το n

## Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n` και επέστρεψε το άθροισμα των ακεραίων από το `1` έως και το `n`. Για `n = 0`, επέστρεψε `0`. Χρησιμοποίησε `while` και μεταβλητή για το άθροισμα.

**Όρια:** 0 ≤ n ≤ 100.

**Η κλήση**

```
solve(4)
```

↓ Η τιμή που επιστρέφεται

```
10
```

## Η ιδέα

Η μία μεταβλητή δείχνει ποιον αριθμό εξετάζουμε. Η άλλη κρατά το αποτέλεσμα από όλα τα περάσματα που έχουν ήδη γίνει.

ΑΘΡΟΙΣΜΑ ΑΠΟ ΤΟ 1 ΕΩΣ ΤΟ N

# 23 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4)` | `10` |
| `solve(0)` | `0` |
| `solve(1)` | `1` |
| `solve(2)` | `3` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/23-sum-up-to/exercise.rb.

Tests: exercises/23-sum-up-to/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/23-sum-up-to/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το άθροισμα ξεκινά από `0` και ο πρώτος αριθμός από `1`. Πρόσθεσε τον αριθμό πριν προχωρήσεις στον επόμενο.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 377.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 24 Επισκέψου όλες τις τιμές, μέτρησε μερικές

Θα μετρήσουμε τα πολλαπλάσια του τρία μέχρι ένα όριο. Η συνθήκη είναι γνωστή. Το νέο βήμα είναι η θέση της μέσα στην επανάληψη.

RUBY

```ruby
def example(limit)
  count = 0
  value = 1
  while value <= limit
    if value % 3 == 0
      count = count + 1
    end
    value = value + 1
  end
  count
end
```

## Διάβασε τις γραμμές

01 Το `while` περνά από το `1` έως το όριο. Το `if` εξετάζει την τιμή του συγκεκριμένου περάσματος.

02 Αυξάνουμε το `count` μόνο για πολλαπλάσια του τρία. Για όριο `7`, αυτά είναι το `3` και το `6`.

03 Το `value = value + 1` βρίσκεται έξω από το `if`. Έτσι προχωράμε ακόμη και όταν δεν μετρήσαμε τον αριθμό.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 24 Από τον κώδικα στην τιμή

**Η κλήση**

```
example(7)
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Ακολούθησε μια κλήση

Η `example(7)` επιστρέφει `2`.

Η `example(2)` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(9)`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `3`. Μετράμε τα `3`, `6` και `9`.

Συνέχισε στην εκφώνηση της άσκησης 24, στη σελίδα 102.

Η ΑΣΚΗΣΗ

# 24 Πόσοι άρτιοι μέχρι το όριο;

## Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n`. Μέτρησε πόσοι άρτιοι υπάρχουν από το `1` έως και το `n` και επέστρεψε το πλήθος. Χρησιμοποίησε `while` και `if`.

**Όρια:** $0 \le n \le 100$.

**Η κλήση**

```
solve(7)
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Η ιδέα

Η επανάληψη επισκέπτεται κάθε αριθμό. Η συνθήκη αποφασίζει αν ο συγκεκριμένος αριθμός θα αυξήσει τον μετρητή.

ΠΟΣΟΙ ΑΡΤΙΟΙ ΜΕΧΡΙ ΤΟ ΟΡΙΟ;

# 24 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(7)` | `3` |
| `solve(0)` | `0` |
| `solve(1)` | `0` |
| `solve(2)` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/24-count-even-up-to/exercise.rb.

Tests: exercises/24-count-even-up-to/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/24-count-even-up-to/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ο αριθμός προχωρά σε κάθε πέρασμα. Ο μετρητής αυξάνεται μόνο όταν το υπόλοιπο με το `2` είναι μηδέν.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 378.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 25 Πες πόσες φορές θα γίνει το βήμα

Κάθε γύρος δίνει δύο πόντους. Θα γράψουμε τη γνωστή επανάληψη χωρίς ξεχωριστή μεταβλητή που μετρά τα περάσματα.

RUBY

```ruby
def example(rounds)
  points = 0
  rounds.times do
    points = points + 2
  end
  points
end
```

## Διάβασε τις γραμμές

01 Η τελεία καλεί τη μέθοδο `times` πάνω στην τιμή `rounds`. Με `3`, το σώμα εκτελείται τρεις φορές.

02 Το `do` και το αντίστοιχο `end` ορίζουν ένα block, δηλαδή το κομμάτι κώδικα που δίνουμε στη `times` για κάθε πέρασμα.

03 Το `points` ορίστηκε πριν από το block και κρατά το άθροισμα. Με μηδέν γύρους, το block δεν εκτελείται και επιστρέφεται `0`.

## Ακολούθησε μια κλήση

Η `example(3)` επιστρέφει `6`.

Η `example(0)` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(4)`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `8`. Το block προσθέτει δύο πόντους τέσσερις φορές.

Συνέχισε στην εκφώνηση της άσκησης 25, στη σελίδα 105.

Η ΑΣΚΗΣΗ

# 25 Η ίδια κατάθεση πολλές φορές

## Τι θα φτιάξεις

Δέξου το πλήθος `count` και το ποσό `amount`. Ξεκίνα με σύνολο `0` και πρόσθεσε το ποσό ακριβώς `count` φορές. Επέστρεψε το σύνολο. Χρησιμοποίησε `times`.

**Όρια:** 0 ≤ count ≤ 100. 0 ≤ amount ≤ 100.

**Η κλήση**

```
solve(3, 5)
```

Η τιμή που επιστρέφεται

```
15
```

## Η ιδέα

Όταν ξέρουμε το πλήθος των επαναλήψεων, η Ruby μπορεί να το μετρήσει για εμάς. Το σώμα περιγράφει μόνο τι γίνεται κάθε φορά.

Η ΙΔΙΑ ΚΑΤΑΘΕΣΗ ΠΟΛΛΕΣ ΦΟΡΕΣ

# 25 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `count, amount`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3, 5)` | `15` |
| `solve(0, 9)` | `0` |
| `solve(4, 0)` | `0` |
| `solve(1, 7)` | `7` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/25-repeat-a-deposit/exercise.rb.

Tests: exercises/25-repeat-a-deposit/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/25-repeat-a-deposit/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η μέθοδος `times` καλείται πάνω στο πλήθος. Το σύνολο πρέπει να οριστεί πριν από το `do`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 379.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 26 Πολλές τιμές, ένα αποτέλεσμα

Θα συγκεντρώσουμε δύο αριθμούς σε έναν πίνακα. Οι αγκύλες δείχνουν τα όριά του και το κόμμα χωρίζει τα στοιχεία.

RUBY

```ruby
def example(first, second)
  [first + 1, second + 1]
end
```

## Διάβασε τις γραμμές

01 Το `[first + 1, second + 1]` δημιουργεί έναν πίνακα. Κάθε έκφραση υπολογίζεται πριν τοποθετηθεί η τιμή της στον πίνακα.

02 Το κόμμα κρατά δύο ξεχωριστά στοιχεία. Η σειρά είναι πρώτα η αριστερή έκφραση και μετά η δεξιά.

03 Η τελευταία έκφραση της μεθόδου είναι ολόκληρος ο πίνακας. Ο τύπος του στη Ruby λέγεται `Array`.

## Ακολούθησε μια κλήση

Η `example(2, 4)` επιστρέφει `[3, 5]`.

Η `example(0, 0)` επιστρέφει `[1, 1]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(-1, 3)`;

**Έλεγξε τη σκέψη σου.** Ο πίνακας είναι `[0, 4]`. Κάθε παράμετρος αυξάνεται κατά μία μονάδα.

Συνέχισε στην εκφώνηση της άσκησης 26, στη σελίδα 108.

Η ΑΣΚΗΣΗ

# 26 Δύο τιμές σε έναν πίνακα

## Τι θα φτιάξεις

Δέξου δύο ακεραίους `a` και `b`. Επέστρεψε έναν πίνακα που περιέχει πρώτα το `a` και μετά το `b`.

**Όρια:** -1000 ≤ a ≤ 1000. -1000 ≤ b ≤ 1000.

**Η κλήση**

```
solve(2, 7)
```

↓ Η τιμή που επιστρέφεται

```
[2, 7]
```

## Η ιδέα

Ένας πίνακας είναι μία τιμή που κρατά πολλά στοιχεία με σειρά. Μπορούμε να τον επιστρέψουμε όπως επιστρέφαμε έναν αριθμό.

ΔΥΟ ΤΙΜΕΣ ΣΕ ΕΝΑΝ ΠΙΝΑΚΑ

# 26 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `a, b`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(2, 7)` | `[2, 7]` |
| `solve(7, 2)` | `[7, 2]` |
| `solve(0, 0)` | `[0, 0]` |
| `solve(-3, 4)` | `[-3, 4]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/26-make-a-pair/exercise.rb.

Tests: exercises/26-make-a-pair/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/26-make-a-pair/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Χρησιμοποίησε αγκύλες και κόμμα. Τα στοιχεία πρέπει να είναι οι τιμές των παραμέτρων.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 380.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 27 Οι θέσεις αρχίζουν από το μηδέν

Μια παράμετρος μπορεί να δεχτεί πίνακα. Θα διαβάσουμε το δεύτερο στοιχείο, χωρίς να αλλάξουμε τη σειρά του.

RUBY

```ruby
def example(values)
  values[1]
end
```

## Διάβασε τις γραμμές

01 Το `values` είναι η παράμετρος που κρατά τον πίνακα της κλήσης. Οι αγκύλες στην κλήση φτιάχνουν αυτή την είσοδο.

02 Το `values[1]` χρησιμοποιεί δείκτη `1`, άρα διαβάζει το δεύτερο στοιχείο. Ο δείκτης δεν είναι η τιμή που ψάχνουμε.

03 Οι είσοδοι του παραδείγματος έχουν τουλάχιστον δύο στοιχεία. Η άσκηση ζητά την πρώτη θέση και εγγυάται τουλάχιστον ένα.

## Ακολούθησε μια κλήση

Η `example([3, 8])` επιστρέφει `8`.

Η `example([9, 0, 5])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([-2, 6])`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `6`. Το `-2` βρίσκεται στη θέση `0` και το `6` στη θέση `1`.

Συνέχισε στην εκφώνηση της άσκησης 27, στη σελίδα 111.

Η ΑΣΚΗΣΗ

# 27 Το πρώτο στοιχείο

## Τι θα φτιάξεις

Δέξου έναν μη κενό πίνακα ακεραίων `numbers` και επέστρεψε το πρώτο στοιχείο του. Μην αλλάξεις τον πίνακα.

**Όρια:** Ο πίνακας `numbers` έχει από 1 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από -1000 έως 1000.

**Η κλήση**

```
solve([4, 8, 2])
```

↓ Η τιμή που επιστρέφεται

```
4
```

## Η ιδέα

Ο δείκτης είναι ο αριθμός μιας θέσης. Η πρώτη θέση έχει δείκτη `0`, η δεύτερη `1` και η τρίτη `2`.

ΤΟ ΠΡΩΤΟ ΣΤΟΙΧΕΙΟ

# 27 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([4, 8, 2])` | `4` |
| `solve([9])` | `9` |
| `solve([0, 7])` | `0` |
| `solve([-3, 10])` | `-3` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/27-first-item/exercise.rb.

Tests: exercises/27-first-item/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/27-first-item/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μετά το όνομα του πίνακα γράφουμε τον δείκτη μέσα σε αγκύλες.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 381.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 28 Μέτρησε θέσεις, όχι τιμές

Η `length` μάς δίνει το πλήθος των στοιχείων. Στο παράδειγμα θα υπολογίσουμε πόσες θέσεις θα είχαμε με ένα ακόμη στοιχείο.

```ruby
def example(values)
  values.length + 1
end
```

## Διάβασε τις γραμμές

01 Το `values.length` καλεί τη μέθοδο `length` στον πίνακα και δίνει ακέραιο πλήθος.

02 Το `+ 1` προσθέτει μία θέση στο πλήθος. Δεν προσθέτει στοιχείο στον πραγματικό πίνακα.

03 Το `[]` είναι κενός πίνακας και έχει μήκος `0`. Το `[0]` περιέχει έναν αριθμό και έχει μήκος `1`.

## Ακολούθησε μια κλήση

Η `example([4, 9])` επιστρέφει `3`.

Η `example([])` επιστρέφει `1`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([7, 7, 7])`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `4`. Οι τρεις ίδιες τιμές καταλαμβάνουν τρεις διαφορετικές θέσεις.

Συνέχισε στην εκφώνηση της άσκησης 28, στη σελίδα 114.

Η ΑΣΚΗΣΗ

# 28 Πόσα στοιχεία έχει;

## Τι θα φτιάξεις

Δέξου έναν πίνακα ακεραίων `numbers` και επέστρεψε πόσα στοιχεία περιέχει. Επιτρέπεται κενός πίνακας.

**Όρια:** Ο πίνακας `numbers` έχει από 0 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από -1000 έως 1000.

**Η κλήση**

```
solve([5, 2, 8])
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Η ιδέα

Το πλήθος των στοιχείων είναι διαφορετικό από τις τιμές τους. Ο πίνακας `[100]` έχει ένα στοιχείο.

ΠΟΣΑ ΣΤΟΙΧΕΙΑ ΕΧΕΙ;

# 28 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([5, 2, 8])` | `3` |
| `solve([])` | `0` |
| `solve([0])` | `1` |
| `solve([7, 7, 7, 7])` | `4` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/28-array-length/exercise.rb.

Tests: exercises/28-array-length/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/28-array-length/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κάλεσε τη μέθοδο `length` πάνω στον πίνακα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 382.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 29 Ένας πίνακας μεγαλώνει βήμα βήμα

Θα φτιάξουμε έναν πίνακα με τα διπλάσια των αριθμών από το ένα μέχρι ένα μικρό όριο. Η επανάληψη είναι γνωστή. Μαθαίνουμε πώς προσθέτουμε στοιχείο.

RUBY

```ruby
def example(limit)
  values = []
  number = 1
  while number <= limit
    values << number * 2
    number = number + 1
  end
  values
end
```

## Διάβασε τις γραμμές

01 Το `values = []` δημιουργεί έναν άδειο πίνακα. Η μεταβλητή ορίζεται πριν από την επανάληψη.

02 Το `values << number * 2` προσθέτει το γινόμενο στο τέλος. Με τρία περάσματα ο πίνακας γίνεται `[2]`, `[2, 4]`, `[2, 4, 6]`.

03 Το `<<` εδώ αλλάζει τον ίδιο πίνακα. Το τελικό `values` επιστρέφει όλα τα στοιχεία που συγκεντρώσαμε.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 29 Από τον κώδικα στην τιμή

**Η κλήση**

```
example(3)
```

↓ Η τιμή που επιστρέφεται

```
[2, 4, 6]
```

## Ακολούθησε μια κλήση

Η `example(3)` επιστρέφει `[2, 4, 6]`.

Η `example(0)` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2)`;

**Έλεγξε τη σκέψη σου.** Ο πίνακας είναι `[2, 4]`, με ένα στοιχείο από κάθε πέρασμα.

Συνέχισε στην εκφώνηση της άσκησης 29, στη σελίδα 118.

Η ΑΣΚΗΣΗ

# 29 Φτιάξε τους αριθμούς από το 1

## Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n`. Επέστρεψε έναν νέο πίνακα με τους ακεραίους από το `1` έως και το `n`, με αυτή τη σειρά. Για `0`, επέστρεψε `[]`. Χρησιμοποίησε `while` και `<<`.

**Όρια:** 0 ≤ n ≤ 20.

**Η κλήση**

```
solve(4)
```

↓ Η τιμή που επιστρέφεται

```
[1, 2, 3, 4]
```

## Η ιδέα

Ξεκινάμε με κενό πίνακα. Σε κάθε πέρασμα προσθέτουμε στο τέλος την επόμενη τιμή.

ΦΤΙΑΞΕ ΤΟΥΣ ΑΡΙΘΜΟΥΣ ΑΠΟ ΤΟ 1

# 29 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4)` | `[1, 2, 3, 4]` |
| `solve(0)` | `[]` |
| `solve(1)` | `[1]` |
| `solve(2)` | `[1, 2]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/29-build-a-sequence/exercise.rb.

Tests: exercises/29-build-a-sequence/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/29-build-a-sequence/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ο πίνακας ορίζεται πριν από την επανάληψη. Πρόσθεσε την τρέχουσα τιμή πριν αυξήσεις τον αριθμό.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 383.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 30 Το block παίρνει μία τιμή κάθε φορά

Θα πολλαπλασιάσουμε τις τιμές ενός πίνακα. Το block που γνωρίσαμε με τη `times` θα δέχεται τώρα την τιμή του κάθε στοιχείου.

RUBY

```ruby
def example(values)
  product = 1
  values.each do |value|
    product = product * value
  end
  product
end
```

## Διάβασε τις γραμμές

01 Το `values.each` επισκέπτεται κάθε στοιχείο με τη σειρά. Δεν χρειάζεται χειροκίνητη ενημέρωση δείκτη.

02 Το `|value|` ορίζει την παράμετρο του block. Σε κάθε πέρασμα παίρνει την επόμενη τιμή. Οι κάθετες γραμμές δεν σημαίνουν διαίρεση.

03 Το γινόμενο αρχίζει από `1`. Αν αρχίζαμε από `0`, κάθε επόμενο γινόμενο θα παρέμενε μηδέν.

## Ακολούθησε μια κλήση

Η `example([2, 3])` επιστρέφει `6`.

Η `example([4, 0])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([2, 5])`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `10`. Το γινόμενο γίνεται πρώτα `2` και μετά `10`.

Συνέχισε στην εκφώνηση της άσκησης 30, στη σελίδα 121.

Η ΑΣΚΗΣΗ

# 30 Το άθροισμα ενός πίνακα

## Τι θα φτιάξεις

Δέξου έναν πίνακα ακεραίων `numbers` και επέστρεψε το άθροισμά τους. Για κενό πίνακα, επέστρεψε `0`. Χρησιμοποίησε `each`.

**Όρια:** Ο πίνακας `numbers` έχει από 0 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από -100 έως 100.

**Η κλήση**

```
solve([2, 5, -1])
```

Η τιμή που επιστρέφεται

```
6
```

## Η ιδέα

Δεν χρειάζεται να κρατάμε δείκτη όταν θέλουμε να χρησιμοποιήσουμε κάθε στοιχείο. Η `each` δίνει τις τιμές με τη σειρά.

ΤΟ ΑΘΡΟΙΣΜΑ ΕΝΟΣ ΠΙΝΑΚΑ

# 30 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 5, -1])` | `6` |
| `solve([])` | `0` |
| `solve([7])` | `7` |
| `solve([-4, -6])` | `-10` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/30-sum-an-array/exercise.rb.

Tests: exercises/30-sum-an-array/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/30-sum-an-array/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κράτα το άθροισμα έξω από το block και πρόσθεσε την παράμετρο του block σε κάθε πέρασμα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 384.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 31 Η ίδια συνθήκη σε κάθε στοιχείο

Θα μετρήσουμε τα μηδενικά ενός πίνακα. Χρησιμοποιούμε μόνο σύνταξη που έχουμε ήδη συναντήσει.

RUBY

```ruby
def example(values)
  count = 0
  values.each do |value|
    if value == 0
      count = count + 1
    end
  end
  count
end
```

## Διάβασε τις γραμμές

01 Το `value` παίρνει διαδοχικά κάθε στοιχείο. Το `if value == 0` ελέγχεται από την αρχή σε κάθε πέρασμα.

02 Η αύξηση του `count` είναι μέσα στο `if`. Ένα μη μηδενικό στοιχείο αφήνει τον μετρητή όπως ήταν.

03 Το πρώτο `end` κλείνει το `if`, το δεύτερο το block της `each` και το τελευταίο τη μέθοδο.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 31 Από τον κώδικα στην τιμή

**Η κλήση**

```
example([0, 3, 0])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Ακολούθησε μια κλήση

Η `example([0, 3, 0])` επιστρέφει `2`.

Η `example([1, 2])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, 0, 0])`;

**Έλεγξε τη σκέψη σου.** Η τιμή είναι `3`. Κάθε μία από τις τρεις θέσεις περνά τον έλεγχο.

Συνέχισε στην εκφώνηση της άσκησης 31, στη σελίδα 125.

Η ΑΣΚΗΣΗ

# 31 Πόσα στοιχεία είναι θετικά;

## Τι θα φτιάξεις

Δέξου έναν πίνακα ακεραίων `numbers` και επέστρεψε πόσα στοιχεία είναι μεγαλύτερα από το μηδέν. Χρησιμοποίησε `each`, `if` και μετρητή.

**Όρια:** Ο πίνακας `numbers` έχει από 0 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από -1000 έως 1000.

**Η κλήση**

```
solve([-2, 0, 4, 7])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Η `each` επισκέπτεται όλες τις θέσεις. Το `if` αποφασίζει αν η συγκεκριμένη θέση θα μετρήσει στο αποτέλεσμα.

ΠΟΣΑ ΣΤΟΙΧΕΙΑ ΕΙΝΑΙ ΘΕΤΙΚΑ;

# 31 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([-2, 0, 4, 7])` | `2` |
| `solve([])` | `0` |
| `solve([0, 0])` | `0` |
| `solve([-1, -3])` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/31-count-positive-items/exercise.rb.

Tests: exercises/31-count-positive-items/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/31-count-positive-items/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αύξησε τον μετρητή κατά `1`, όχι κατά την τιμή του θετικού αριθμού.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 385.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 32 Μία νέα τιμή για κάθε παλιά

Θα προσθέσουμε τρία σε κάθε στοιχείο και θα πάρουμε νέο πίνακα. Η `map` συγκεντρώνει αυτόματα τις απαντήσεις του block.

RUBY

```ruby
def example(values)
  values.map do |value|
    value + 3
  end
end
```

## Διάβασε τις γραμμές

01 Η `map` δίνει κάθε στοιχείο στην παράμετρο `value`, όπως η `each` που ήδη γνωρίζεις.

02 Η τελευταία έκφραση του block, εδώ το `value + 3`, γίνεται ένα στοιχείο του νέου πίνακα.

03 Η `map` επιστρέφει τον νέο πίνακα. Δεν χρειάζεται μεταβλητή αποτελέσματος ή `<<` για να τον συγκεντρώσουμε.

## Ακολούθησε μια κλήση

Η `example([1, 4])` επιστρέφει `[4, 7]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([-3, 0])`;

**Έλεγξε τη σκέψη σου.** Ο πίνακας είναι `[0, 3]`. Παραμένουν δύο στοιχεία, με την ίδια σειρά.

Συνέχισε στην εκφώνηση της άσκησης 32, στη σελίδα 128.

Η ΑΣΚΗΣΗ

# 32 Διπλασίασε κάθε στοιχείο

## Τι θα φτιάξεις

Δέξου έναν πίνακα ακεραίων `numbers`. Επέστρεψε νέο πίνακα με το διπλάσιο κάθε στοιχείου, στην ίδια σειρά. Χρησιμοποίησε `map` και μην αλλάξεις την είσοδο.

**Όρια:** Ο πίνακας `numbers` έχει από 0 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από -1000 έως 1000.

**Η κλήση**

```
solve([2, -3, 0])
```

↓ Η τιμή που επιστρέφεται

```
[4, -6, 0]
```

## Η ιδέα

Η `map` φτιάχνει ένα αποτέλεσμα για κάθε αρχικό στοιχείο. Το πλήθος και η σειρά των θέσεων παραμένουν ίδια.

ΔΙΠΛΑΣΙΑΣΕ ΚΑΘΕ ΣΤΟΙΧΕΙΟ

# 32 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, -3, 0])` | `[4, -6, 0]` |
| `solve([])` | `[]` |
| `solve([1])` | `[2]` |
| `solve([5, 5])` | `[10, 10]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/32-double-each-item/exercise.rb.

Tests: exercises/32-double-each-item/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/32-double-each-item/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η τελευταία έκφραση του block είναι η τιμή που θα μπει στη νέα θέση.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 386.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 33 Το block λέει ποια στοιχεία θα μείνουν

Θα κρατήσουμε μόνο τις τιμές που ξεπερνούν το πέντε. Ο νέος πίνακας μπορεί να έχει λιγότερα στοιχεία ή να είναι κενός.

RUBY

```ruby
def example(values)
  values.select do |value|
    value > 5
  end
end
```

## Διάβασε τις γραμμές

01 Το `value > 5` είναι η τελευταία έκφραση του block και δίνει λογική τιμή.

02 Όταν η συνθήκη είναι αληθής, η `select` κρατά το αρχικό στοιχείο. Το `8` παραμένει `8`, δεν γίνεται `true`.

03 Η `select` επιστρέφει νέο πίνακα και διατηρεί τη σχετική σειρά των στοιχείων που κράτησε.

## Ακολούθησε μια κλήση

Η `example([2, 8, 5, 9])` επιστρέφει `[8, 9]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([6, 5, 7])`;

**Έλεγξε τη σκέψη σου.** Ο πίνακας είναι `[6, 7]`. Το `5` δεν περνά την αυστηρή σύγκριση `> 5`.

Συνέχισε στην εκφώνηση της άσκησης 33, στη σελίδα 131.

Η ΑΣΚΗΣΗ

# 33 Κράτα μόνο τους άρτιους

## Τι θα φτιάξεις

Δέξου έναν πίνακα ακεραίων `numbers`. Επέστρεψε νέο πίνακα με μόνο τα άρτια στοιχεία, στην αρχική τους σειρά. Χρησιμοποίησε `select` και μην αλλάξεις την είσοδο.

**Όρια:** Ο πίνακας `numbers` έχει από 0 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από -1000 έως 1000.

**Η κλήση**

```
solve([3, 2, 7, 4])
```

↓ Η τιμή που επιστρέφεται

```
[2, 4]
```

## Η ιδέα

Η `select` αποφασίζει αν θα κρατήσει κάθε στοιχείο. Κρατά την αρχική τιμή, όχι το αποτέλεσμα της σύγκρισης.

ΚΡΑΤΑ ΜΟΝΟ ΤΟΥΣ ΑΡΤΙΟΥΣ

# 33 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([3, 2, 7, 4])` | `[2, 4]` |
| `solve([])` | `[]` |
| `solve([1, 3])` | `[]` |
| `solve([0, -2, 6])` | `[0, -2, 6]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/33-keep-even-items/exercise.rb.

Tests: exercises/33-keep-even-items/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/33-keep-even-items/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το block πρέπει να δίνει μια συνθήκη που είναι αληθής για τα στοιχεία που θέλεις.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 387.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 34 Τα ψηφία μπορούν να είναι χαρακτήρες

Θα κάνουμε πρώτα μια αριθμητική πράξη και μετά θα μετατρέψουμε το αποτέλεσμά της σε κείμενο.

RUBY

```ruby
def example(value)
  (value + 2).to_s
end
```

## Διάβασε τις γραμμές

01 Οι παρενθέσεις υπολογίζουν πρώτα το `value + 2`. Το αποτέλεσμα είναι ακόμη αριθμός.

02 Το `.to_s` καλείται πάνω σε αυτό το αποτέλεσμα και δίνει ένα `String`, δηλαδή κείμενο.

03 Τα εισαγωγικά στις αναμενόμενες τιμές δείχνουν τον τύπο. Το `"7"` διαφέρει από τον ακέραιο `7`.

## Ακολούθησε μια κλήση

Η `example(5)` επιστρέφει `"7"`.

Η `example(-2)` επιστρέφει `"0"`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(0)`;

**Έλεγξε τη σκέψη σου.** Επιστρέφει `"2"`, ως κείμενο. Η πρόσθεση γίνεται πριν από τη μετατροπή.

Συνέχισε στην εκφώνηση της άσκησης 34, στη σελίδα 134.

Η ΑΣΚΗΣΗ

# 34 Ένας αριθμός γίνεται κείμενο

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε την αναπαράστασή του ως κείμενο. Για είσοδο `7`, το αποτέλεσμα είναι `"7"`.

**Όρια:** -1000 ≤ n ≤ 1000.

Η κλήση

```
solve(7)
```

↓ Η τιμή που επιστρέφεται

```
"7"
```

## Η ιδέα

Ο ίδιος αριθμός μπορεί να αναπαρασταθεί ως κείμενο για να συμμετέχει σε ένα μήνυμα. Η αριθμητική τιμή και το κείμενο είναι διαφορετικοί τύποι.

ΕΝΑΣ ΑΡΙΘΜΟΣ ΓΙΝΕΤΑΙ ΚΕΙΜΕΝΟ

# 34 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(7)` | `"7"` |
| `solve(0)` | `"0"` |
| `solve(-12)` | `"-12"` |
| `solve(1000)` | `"1000"` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/34-number-as-text/exercise.rb.

Tests: exercises/34-number-as-text/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/34-number-as-text/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κάλεσε τη μέθοδο `to_s` στον αριθμό.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 388.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 35 Δύο κείμενα γίνονται ένα

Θα προσθέσουμε μια μικρή ετικέτα μπροστά από έναν αριθμό. Η μετατροπή σε κείμενο είναι ήδη γνωστή.

RUBY

```ruby
def example(value)
  "Score: " + value.to_s
end
```

## Διάβασε τις γραμμές

01 Εδώ το `+` ενώνει κείμενα. Δεν κάνει αριθμητική πρόσθεση, επειδή οι δύο πλευρές είναι `String`.

02 Το κενό στο τέλος του `"Score: "` παραμένει στο αποτέλεσμα. Η Ruby δεν προσθέτει μόνη της κενό ανάμεσα στα κείμενα.

03 Το `value.to_s` μετατρέπει και αρνητικούς αριθμούς, κρατώντας το πρόσημό τους.

## Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει `"Score: 4"`.

Η `example(-1)` επιστρέφει `"Score: -1"`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(0)`;

**Έλεγξε τη σκέψη σου.** Επιστρέφει `"Score: 0"`, με ένα κενό πριν από το μηδέν.

Συνέχισε στην εκφώνηση της άσκησης 35, στη σελίδα 137.

Η ΑΣΚΗΣΗ

# 35 Ένα μήνυμα με αριθμό

## Τι θα φτιάξεις

Δέξου έναν ακέραιο `n` και επέστρεψε το κείμενο `"Number: "` ακολουθούμενο από την τιμή του. Μετά την άνω και κάτω τελεία υπάρχει ακριβώς ένα κενό.

**Όρια:** -1000 ≤ n ≤ 1000.

**Η κλήση**

```
solve(8)
```

↓ Η τιμή που επιστρέφεται

```
"Number: 8"
```

## Η ιδέα

Το `+` μπορεί να ενώνει δύο κείμενα. Πρώτα μετατρέπουμε τον αριθμό σε κείμενο, ώστε οι δύο πλευρές να έχουν κατάλληλο τύπο.

ΕΝΑ ΜΗΝΥΜΑ ΜΕ ΑΡΙΘΜΟ

# 35 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(8)` | `"Number: 8"` |
| `solve(0)` | `"Number: 0"` |
| `solve(-3)` | `"Number: -3"` |
| `solve(1000)` | `"Number: 1000"` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/35-number-label/exercise.rb.

Tests: exercises/35-number-label/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/35-number-label/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το κενό ανήκει στο σταθερό κείμενο. Χρησιμοποίησε την `to_s` που έμαθες πριν.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 389.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 36 Ένα διαχωριστικό ανάμεσα στις τιμές

Θα ενώσουμε κείμενα με ένα απλό σύμβολο ανάμεσά τους. Η ίδια μέθοδος μπορεί να χρησιμοποιήσει και αλλαγή γραμμής.

RUBY

```ruby
def example(labels)
  labels.join(" / ")
end
```

## Διάβασε τις γραμμές

01 Η `join` καλείται στον πίνακα. Το όρισμά της είναι το κείμενο που θα μπει ανάμεσα στα στοιχεία.

02 Το `" / "` έχει κενό, κάθετο και κενό. Δεν μπαίνει πριν από το πρώτο στοιχείο ούτε μετά το τελευταίο.

03 Στα διπλά εισαγωγικά το `"\n"` παριστάνει αλλαγή γραμμής. Θα το χρησιμοποιήσεις στην άσκηση στη θέση του `" / "`.

## Ακολούθησε μια κλήση

Η `example(["A", "B"])` επιστρέφει `"A / B"`.

Η `example([])` επιστρέφει `""`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(["X", "Y", "Z"])`;

**Έλεγξε τη σκέψη σου.** Το κείμενο είναι `"X / Y / Z"`. Τρία στοιχεία χρειάζονται δύο διαχωριστικά.

Συνέχισε στην εκφώνηση της άσκησης 36, στη σελίδα 140.

Η ΑΣΚΗΣΗ

# 36 Μία γραμμή για κάθε κείμενο

## Τι θα φτιάξεις

Δέξου έναν πίνακα κειμένων `words`. Επέστρεψε ένα κείμενο που ενώνει τα στοιχεία με αλλαγή γραμμής ανάμεσά τους. Χρησιμοποίησε `join`. Μην προσθέσεις επιπλέον αλλαγή γραμμής στο τέλος.

**Όρια:** Ο πίνακας `words` έχει από 0 έως 20 στοιχεία.

Η κλήση

```
solve(["red", "blue"])
```

Η τιμή που επιστρέφεται

```
"red\nblue"
```

## Η ιδέα

Η `join` βάζει το διαχωριστικό ανάμεσα στα στοιχεία. Για μηδέν στοιχεία δίνει κενό κείμενο και για ένα στοιχείο δεν χρειάζεται διαχωριστικό.

ΜΙΑ ΓΡΑΜΜΗ ΓΙΑ ΚΑΘΕ ΚΕΙΜΕΝΟ

# 36 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `words`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(["red", "blue"])` | `"red\nblue"` |
| `solve([])` | `""` |
| `solve(["hello"])` | `"hello"` |
| `solve(["a", "b", "c"])` | `"a\nb\nc"` |

Στον πίνακα το `\n` παριστάνει αλλαγή γραμμής μέσα στο κείμενο.

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/36-join-lines/exercise.rb.

Tests: exercises/36-join-lines/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/36-join-lines/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μέσα σε διπλά εισαγωγικά, το `"\n"` γράφει μία αλλαγή γραμμής.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 390.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 37 Περιέγραψε την αρχή και το τέλος

Θα φτιάξουμε τους ακεραίους από μια αρχή έως ένα τέλος. Έχουμε ήδη φτιάξει τέτοιους πίνακες με `while`. Τώρα γνωρίζουμε μια σύντομη περιγραφή.

RUBY

```ruby
def example(start, finish)
  (start..finish).to_a
end
```

## Διάβασε τις γραμμές

01 Το `start..finish` δημιουργεί ένα `Range`. Οι δύο τελείες περιλαμβάνουν και τις δύο άκρες.

02 Το `.to_a` μετατρέπει το διάστημα σε πίνακα. Οι παρενθέσεις δείχνουν ότι η μέθοδος καλείται σε ολόκληρο το διάστημα.

03 Εδώ περνάμε τους ακεραίους προς τα πάνω. Αν η αρχή είναι μεγαλύτερη από το τέλος, δεν υπάρχουν στοιχεία για να συγκεντρωθούν.

## Ακολούθησε μια κλήση

Η `example(2, 4)` επιστρέφει `[2, 3, 4]`.

Η `example(3, 3)` επιστρέφει `[3]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(5, 3)`;

**Έλεγξε τη σκέψη σου.** Ο πίνακας είναι `[]`. Η `to_a` δεν ξεκινά αντίστροφη μέτρηση.

Συνέχισε στην εκφώνηση της άσκησης 37, στη σελίδα 143.

Η ΑΣΚΗΣΗ

# 37 Ένα συνεχόμενο διάστημα

## Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n` και επέστρεψε τον πίνακα από το `1` έως και το `n`. Αυτή τη φορά χρησιμοποίησε ένα `Range` με `..` και την `to_a`.

**Όρια:** 0 ≤ n ≤ 20.

**Η κλήση**

```
solve(4)
```

↓ Η τιμή που επιστρέφεται

```
[1, 2, 3, 4]
```

## Η ιδέα

Το διάστημα περιγράφει την αρχή και το τέλος. Η `to_a` συγκεντρώνει τις ακέραιες τιμές του σε πίνακα.

ΕΝΑ ΣΥΝΕΧΟΜΕΝΟ ΔΙΑΣΤΗΜΑ

# 37 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4)` | `[1, 2, 3, 4]` |
| `solve(0)` | `[]` |
| `solve(1)` | `[1]` |
| `solve(2)` | `[1, 2]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/37-range-to-array/exercise.rb.

Tests: exercises/37-range-to-array/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/37-range-to-array/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Οι δύο τελείες περιλαμβάνουν και το δεξί άκρο. Βάλε παρενθέσεις γύρω από το διάστημα πριν καλέσεις την `to_a`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 391.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 38 Το block μπορεί να πάρει μια απόφαση

Θα αντικαταστήσουμε τους άρτιους με μια λέξη και θα μετατρέψουμε τους υπόλοιπους σε κείμενο. Η επιλογή γίνεται χωριστά σε κάθε θέση.

RUBY

```ruby
def example(values)
  values.map do |value|
    if value % 2 == 0
      "Pair"
    else
      value.to_s
    end
  end
end
```

## Διάβασε τις γραμμές

01 Το `if` είναι η τελευταία έκφραση του block. Η τιμή του επιλεγμένου κλάδου γίνεται το νέο στοιχείο.

02 Και οι δύο κλάδοι δίνουν κείμενο. Η `to_s` κρατά ορατούς τους αριθμούς που δεν παίρνουν ετικέτα.

03 Το εσωτερικό `end` κλείνει το `if`. Ακολουθούν το `end` της `map` και εκείνο της μεθόδου.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 38 Από τον κώδικα στην τιμή

**Η κλήση**

```
example([2, 3, 4])
```

↓ Η τιμή που επιστρέφεται

```
["Pair", "3", "Pair"]
```

## Ακολούθησε μια κλήση

Η `example([2, 3, 4])` επιστρέφει `["Pair", "3", "Pair"]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([5, 6])`;

**Έλεγξε τη σκέψη σου.** Ο πίνακας είναι `["5", "Pair"]`. Η πρώτη τιμή είναι κείμενο με το ψηφίο πέντε.

Συνέχισε στην εκφώνηση της άσκησης 38, στη σελίδα 147.

Η ΑΣΚΗΣΗ

# 38 Βάλε Fizz στις σωστές θέσεις

## Τι θα φτιάξεις

Δέξου έναν πίνακα θετικών ακεραίων `numbers`. Επέστρεψε νέο πίνακα κειμένων: στη θέση κάθε πολλαπλάσιου του `3` βάλε `"Fizz"`, ενώ στις άλλες θέσεις βάλε τον ίδιο τον αριθμό ως κείμενο. Μην αλλάξεις την είσοδο.

**Όρια:** Ο πίνακας `numbers` έχει από 0 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από 1 έως 1000.

**Η κλήση**

```
solve([1, 3, 4, 6])
```

↓ Η τιμή που επιστρέφεται

```
["1", "Fizz", "4", "Fizz"]
```

## Η ιδέα

Το block της `map` μπορεί να επιλέγει τιμή με `if`. Όποιος κλάδος εκτελεστεί δίνει το κείμενο της συγκεκριμένης θέσης.

ΒΑΛΕ FIZZ ΣΤΙΣ ΣΩΣΤΕΣ ΘΕΣΕΙΣ

# 38 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 3, 4, 6])` | `["1", "Fizz", "4", "Fizz"]` |
| `solve([])` | `[]` |
| `solve([3])` | `["Fizz"]` |
| `solve([2, 5])` | `["2", "5"]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/38-fizz-in-an-array/exercise.rb.

Tests: exercises/38-fizz-in-an-array/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/38-fizz-in-an-array/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Στην περίπτωση που δεν είναι πολλαπλάσιο του τρία, χρειάζεσαι `to_s`, όχι κενό κείμενο.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 392.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 39 Πολλοί κανόνες για κάθε θέση

Θα εφαρμόσουμε δύο κανόνες μαζί σε έναν πίνακα. Οι ετικέτες είναι `Pair` για το δύο και `Trio` για το τρία.

RUBY

```ruby
def example(values)
  values.map do |value|
    if (value % 2 == 0) && (value % 3 == 0)
      "PairTrio"
    elsif value % 2 == 0
      "Pair"
    elsif value % 3 == 0
      "Trio"
    else
      value.to_s
    end
  end
end
```

## Διάβασε τις γραμμές

01 Ο κοινός έλεγχος προηγείται των δύο απλών. Έτσι το `6` παίρνει `"PairTrio"`.

02 Τα `elsif` ελέγχονται μόνο αν δεν πέρασε προηγούμενη συνθήκη. Για κάθε στοιχείο επιλέγεται ένας κλάδος.

03 Το τελικό `else` δίνει τα ψηφία ως κείμενο. Η `map` κρατά μία απάντηση για κάθε θέση, ακόμη και για αριθμούς χωρίς ετικέτα.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 39 Από τον κώδικα στην τιμή

**Η κλήση**

```
example([2, 3, 6, 5])
```

↓ Η τιμή που επιστρέφεται

```
["Pair", "Trio", "PairTrio", "5"]
```

## Ακολούθησε μια κλήση

Η `example([2, 3, 6, 5])` επιστρέφει `["Pair", "Trio", "PairTrio", "5"]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([12, 7])`;

**Έλεγξε τη σκέψη σου.** Ο πίνακας είναι `["PairTrio", "7"]`. Το δώδεκα περνά τον κοινό έλεγχο.

Συνέχισε στην εκφώνηση της άσκησης 39, στη σελίδα 151.

Η ΑΣΚΗΣΗ

# 39 FizzBuzz για κάθε στοιχείο

## Τι θα φτιάξεις

Δέξου έναν πίνακα θετικών ακεραίων `numbers`. Επέστρεψε νέο πίνακα κειμένων με τους κανόνες FizzBuzz: `"FizzBuzz"` για πολλαπλάσιο του `3` και του `5`, `"Fizz"` μόνο του `3`, `"Buzz"` μόνο του `5` και τον αριθμό ως κείμενο στις άλλες θέσεις.

**Όρια:** Ο πίνακας `numbers` έχει από 0 έως 20 στοιχεία. Κάθε στοιχείο του `numbers` είναι από 1 έως 1000.

**Η κλήση**

```
solve([3, 5, 15, 7])
```

Η τιμή που επιστρέφεται

```
["Fizz", "Buzz", "FizzBuzz", "7"]
```

## Η ιδέα

Οι κανόνες της άσκησης 20 εφαρμόζονται τώρα σε κάθε στοιχείο. Η πρώτη επιτυχημένη συνθήκη καθορίζει την τιμή της συγκεκριμένης θέσης.

FIZZBUZZ ΓΙΑ ΚΑΘΕ ΣΤΟΙΧΕΙΟ

# 39 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([3, 5, 15, 7])` | `["Fizz", "Buzz", "FizzBuzz", "7"]` |
| `solve([])` | `[]` |
| `solve([1])` | `["1"]` |
| `solve([15, 15])` | `["FizzBuzz", "FizzBuzz"]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/39-fizzbuzz-in-an-array/exercise.rb.

Tests: exercises/39-fizzbuzz-in-an-array/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/39-fizzbuzz-in-an-array/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Βάλε τον κοινό έλεγχο πρώτο. Το τελικό `else` πρέπει να κρατά τον αριθμό ως κείμενο.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 393.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 40 Από μια ακολουθία σε πολλές γραμμές

Συνδυάζουμε τους κανόνες `Pair` και `Trio` με ένα διάστημα. Η τελική απάντηση θα είναι ένα κείμενο με πολλές γραμμές.

RUBY

```ruby
def example(limit)
  labels = (1..limit).map do |value|
    if (value % 2 == 0) && (value % 3 == 0)
      "PairTrio"
    elsif value % 2 == 0
      "Pair"
    elsif value % 3 == 0
      "Trio"
    else
      value.to_s
    end
  end
  labels.join("\n")
end
```

## Διάβασε τις γραμμές

01 Το `Range` δίνει τις ακέραιες τιμές στην `map`, όπως ένας πίνακας. Εδώ δεν χρειάζεται να καλέσουμε πρώτα `to_a`.

02 Το `labels = ...` κρατά τον πίνακα που επιστρέφει η `map`. Η αλυσίδα κανόνων μέσα στο block είναι εκείνη του προηγούμενου μαθήματος.

03 Το `labels.join("\n")` ενώνει τις απαντήσεις με αλλαγή γραμμής. Η μέθοδος επιστρέφει ένα κείμενο, όχι πίνακα.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 40 Από τον κώδικα στην τιμή

**Η κλήση**

```
example(3)
```

↓ Η τιμή που επιστρέφεται

```
"1\nPair\nTrio"
```

## Ακολούθησε μια κλήση

Η `example(3)` επιστρέφει `"1\nPair\nTrio"`.

Η `example(0)` επιστρέφει `""`.

## Στάσου λίγο

Ποια θα είναι η τελευταία γραμμή της `example(6)`;

**Έλεγξε τη σκέψη σου.** Θα είναι `PairTrio`. Το έξι περνά πρώτο τον κοινό κανόνα του δύο και του τρία.

Συνέχισε στην εκφώνηση της άσκησης 40, στη σελίδα 155.

Η ΑΣΚΗΣΗ

# 40 Το πλήρες FizzBuzz

## Τι θα φτιάξεις

Δέξου έναν μη αρνητικό ακέραιο `n`. Επέστρεψε το πλήρες FizzBuzz από το `1` έως και το `n`, με μία απάντηση σε κάθε γραμμή και χωρίς επιπλέον αλλαγή γραμμής στο τέλος. Για `0`, επέστρεψε `""`. Για την κλασική εκδοχή δίνουμε `n = 100`.

**Όρια:** 0 ≤ n ≤ 100.

**Η κλήση**

```
solve(3)
```

↓ Η τιμή που επιστρέφεται

```
"1\n2\nFizz"
```

## Η ιδέα

Χτίζουμε την ακολουθία των αριθμών, δίνουμε μία ετικέτα σε κάθε θέση και ενώνουμε τα κείμενα. Στους αριθμούς χωρίς λέξη κρατάμε τα ψηφία τους.

ΤΟ ΠΛΗΡΕΣ FIZZBUZZ

# 40 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `n`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3)` | `"1\n2\nFizz"` |
| `solve(0)` | `""` |
| `solve(1)` | `"1"` |
| `solve(2)` | `"1\n2"` |

Στον πίνακα το `\n` παριστάνει αλλαγή γραμμής μέσα στο κείμενο.

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/40-full-fizzbuzz/exercise.rb.

Tests: exercises/40-full-fizzbuzz/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/40-full-fizzbuzz/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το `Range` μπορεί να χρησιμοποιήσει απευθείας `map`. Κράτα τον πίνακα των ετικετών σε μεταβλητή και μετά κάλεσε `join`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 394.

ΠΡΙΝ ΑΠΟ ΤΙΣ ΑΣΚΗΣΕΙΣ 41-100

# Πώς αλλάζει ο ρυθμός

Δώσε περισσότερο χρόνο σε κάθε άσκηση. Πριν γράψεις κώδικα, φτιάξε μια μικρή είσοδο στο χαρτί και εξήγησε ποια απάντηση περιμένεις. Μια απλή λύση που δοκιμάζει όλες τις επιλογές είναι χρήσιμη αφετηρία. Μετά αναζήτησε ποια δουλειά επαναλαμβάνει και ποια πληροφορία μπορεί να κρατήσει.

Κάθε άσκηση δίνει στόχο απόδοσης. Το `O(n)` σημαίνει ότι η δουλειά αυξάνεται το πολύ ανάλογα με το μήκος εισόδου. Το `O(n²)` περιγράφει τετραγωνική αύξηση, το `O(log n)` διαδοχικό περιορισμό της αναζήτησης και το `O(2ⁿ)` εκθετικό πλήθος καταστάσεων. Οι συμβολισμοί είναι ασυμπτωτικά όρια, όχι μετρημένοι χρόνοι. Οι πράξεις Hash έχουν αναμενόμενο σταθερό κόστος. Όταν μεγαλώνουν πολύ οι ακέραιοι, μετρά και το πλήθος ψηφίων τους.

Τα tests ελέγχουν τιμές, τύπους και ότι δεν άλλαξαν τα ορίσματα. Ο στόχος απόδοσης και περιορισμοί όπως «χωρίς ταξινόμηση» χρειάζονται και ανάγνωση κώδικα. Εξήγησε τι σημαίνει κάθε μεταβλητή στη μέση του βρόχου και γιατί αυτή η ιδιότητα παραμένει σωστή. Αυτό είναι το αναλλοίωτο του αλγορίθμου.

ΠΡΙΝ ΑΠΟ ΤΙΣ ΑΣΚΗΣΕΙΣ 41-100

# Τα νέα ορίσματα

Στα specs γράφεις κείμενα, πίνακες και δισδιάστατους πίνακες ως κανονικές τιμές Ruby. Για μία κλήση στο Terminal, αφού λύσεις την αντίστοιχη άσκηση:

```sh
ruby -r ./exercises/51-normalized-palindrome/exercise.rb \
  -e 'p solve("A man, a plan, a canal: Panama!")'
ruby -r ./exercises/79-matrix-transpose/exercise.rb \
  -e 'p solve([[1, 2], [3, 4]])'
```

Τα μονά εισαγωγικά κρατούν ολόκληρη την έκφραση Ruby σε ένα όρισμα της εντολής. Οι είσοδοι που προσθέτεις πρέπει να τηρούν την εκφώνηση: όρια μεγεθών, ταξινομημένους πίνακες, έγκυρους δείκτες ή τετραγωνικούς πίνακες όπου ζητούνται. Η απευθείας κλήση δεν προσθέτει δικό της έλεγχο εγκυρότητας.

ΠΡΙΝ ΑΠΟ ΤΙΣ ΑΣΚΗΣΕΙΣ 41-100

# Πριν κοιτάξεις τη λύση

Δοκίμασε την κανονική περίπτωση, ένα όριο και μια περίπτωση που θα χαλούσε μια βιαστική ιδέα. Για γράφους, σκέψου κύκλο και μη προσβάσιμη κορυφή. Για αναδρομή, έλεγξε τη βασική περίπτωση και την αναίρεση. Για δυναμικό προγραμματισμό, γράψε με μία πρόταση τι αποθηκεύει κάθε κατάσταση.

Οι λύσεις εξηγούν τον κανόνα ενημέρωσης, το κόστος και ένα κρίσιμο λάθος. Αν χρειαστεί να τις διαβάσεις, κλείσε τις μετά και ξαναγράψε την υλοποίηση από τη δική σου εξήγηση.

ΑΣΚΗΣΕΙΣ 41-50

# Πίνακες με λιγότερα περάσματα

Από προθέματα και δύο δείκτες μέχρι την επιλογή του καλύτερου συνεχόμενου τμήματος.



Δώσε χρόνο στη σκέψη, στις οριακές περιπτώσεις και στην εξήγηση του κόστους. Οι λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 41 Ένα όνομα ζει και μέσα στο block

Θα βρούμε τα διαδοχικά υπόλοιπα ενός λογαριασμού, ξεκινώντας από δέκα. Η ίδια μεταβλητή ενημερώνεται σε κάθε πέρασμα.

RUBY

```ruby
def example(changes)
  balance = 10
  changes.map do |change|
    balance += change
  end
end
```

## Διάβασε τις γραμμές

01 Το `+=` προσθέτει στη σημερινή τιμή και αποθηκεύει το αποτέλεσμα στο ίδιο όνομα.

02 Το `balance` ορίζεται πριν από το block, άρα κρατά τη νέα τιμή για το επόμενο πέρασμα.

03 Η `map` επιστρέφει νέο πίνακα. Το κενό input δίνει κενό output, ακόμη κι αν υπάρχει αρχικό υπόλοιπο.

## Ακολούθησε μια κλήση

Η `example([3, -2, 5])` επιστρέφει `[13, 11, 16]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([-10, 4])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 4]`. Το δεύτερο βήμα ξεκινά από το μηδέν που άφησε το πρώτο.

Συνέχισε στην εκφώνηση της άσκησης 41, στη σελίδα 162.

Η ΑΣΚΗΣΗ

# 41 Όλα τα ενδιάμεσα αθροίσματα

## Τι θα φτιάξεις

Επέστρεψε πίνακα ίδιου μήκους με το `numbers`. Στη θέση `i` θέλουμε το άθροισμα από τη θέση `0` έως και την `i`. Για κενή είσοδο επέστρεψε `[]`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

Η κλήση

```
solve([2, 3, -1])
```

↓ Η τιμή που επιστρέφεται

```
[2, 5, 4]
```

## Η ιδέα

Κράτα το άθροισμα όσων στοιχείων έχουν ήδη περάσει. Κάθε νέο στοιχείο το ενημερώνει μία φορά και δημιουργεί μία νέα απάντηση.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` χώρος για την έξοδο, με `O(1)` πρόσθετο χώρο.

ΟΛΑ ΤΑ ΕΝΔΙΑΜΕΣΑ ΑΘΡΟΙΣΜΑΤΑ

# 41 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 3, -1])` | `[2, 5, 4]` |
| `solve([])` | `[]` |
| `solve([-3])` | `[-3]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/41-prefix-sums/exercise.rb.

Tests: exercises/41-prefix-sums/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/41-prefix-sums/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το άθροισμα αρχίζει από μηδέν. Το block της `map` μπορεί να ενημερώνει ένα όνομα που ορίστηκε έξω από αυτό.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 395.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 42 Ένα αρχικό μηδέν απλοποιεί τα όρια

Θα φτιάξουμε πίνακα προθεμάτων που περιλαμβάνει και το κενό πρόθεμα. Το μήκος του θα είναι κατά ένα μεγαλύτερο από την είσοδο.

RUBY

```ruby
def example(values)
  prefix = [0]
  values.each do |value|
    prefix << prefix[-1] + value
  end
  prefix
end
```

## Διάβασε τις γραμμές

01 Η θέση `-1` διαβάζει το τελευταίο στοιχείο ενός πίνακα Ruby.

02 Η `<<` προσθέτει νέο στοιχείο μόνο στον τοπικό πίνακα `prefix`.

03 Το `prefix[2]` περιγράφει δύο στοιχεία της εισόδου, όχι την τιμή στη δεύτερη θέση της.

## Ακολούθησε μια κλήση

Η `example([4, -1, 2])` επιστρέφει `[0, 4, 3, 5]`.

Η `example([])` επιστρέφει `[0]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([7, 0])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 7, 7]`. Το τελευταίο μηδέν της εισόδου δεν αλλάζει το άθροισμα.

Συνέχισε στην εκφώνηση της άσκησης 42, στη σελίδα 165.

Η ΑΣΚΗΣΗ

# 42 Πολλά αθροίσματα διαστημάτων

## Τι θα φτιάξεις

Κάθε γραμμή του `queries` είναι `[left, right]` με `0 <= left <= right < numbers.length`. Επέστρεψε το άθροισμα του κλειστού διαστήματος για κάθε ερώτηση, στην ίδια σειρά. Όταν το `numbers` είναι κενό, το `queries` είναι επίσης κενό.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `queries` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 2 έως 2 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 199.

Η κλήση

```
solve([2, 4, -1], [[0, 1], [1, 2]])
```

↓ Η τιμή που επιστρέφεται

```
[6, 3]
```

## Η ιδέα

Αποθήκευσε μία φορά τα αθροίσματα προθεμάτων. Το ζητούμενο διάστημα είναι η διαφορά ανάμεσα σε δύο τέτοια αθροίσματα.

**Στόχος απόδοσης:** `O(n + q)` χρόνος και `O(n + q)` χώρος μαζί με την έξοδο, όπου `q` οι ερωτήσεις.

ΠΟΛΛΑ ΑΘΡΟΙΣΜΑΤΑ ΔΙΑΣΤΗΜΑΤΩΝ

# 42 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `queries`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 4, -1], [[0, 1], [1, 2]])` | `[6, 3]` |
| `solve([], [])` | `[]` |
| `solve([5], [[0, 0]])` | `[5]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/42-range-sum-queries/exercise.rb.

Tests: exercises/42-range-sum-queries/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/42-range-sum-queries/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Βάλε ένα επιπλέον μηδέν στην αρχή. Τότε η ερώτηση που αρχίζει στη θέση μηδέν δεν χρειάζεται ειδικό κλάδο.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 396.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 43 Διάβασε τον πίνακα από δεξιά

Θα κρατήσουμε τα γινόμενα από κάθε θέση μέχρι το τέλος. Η Ruby μπορεί να κατεβάζει έναν ακέραιο δείκτη με τη `downto`.

RUBY

```ruby
def example(values)
  result = Array.new(values.length, 1)
  product = 1
  (values.length - 1).downto(0) do |i|
    product *= values[i]
    result[i] = product
  end
  result
end
```

## Διάβασε τις γραμμές

01 Το `Array.new(length, 1)` δημιουργεί τις θέσεις της εξόδου. Οι ακέραιοι δεν μεταβάλλονται εσωτερικά.

02 Η `downto(0)` περιλαμβάνει το μηδέν. Αν αρχίσει από `-1`, δεν εκτελεί κανένα πέρασμα.

03 Εδώ πολλαπλασιάζουμε πριν από την αποθήκευση, ώστε να περιλαμβάνεται και η τρέχουσα θέση.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 43 Από τον κώδικα στην τιμή

**Η κλήση**

```
example([2, 3, 4])
```

↓ Η τιμή που επιστρέφεται

```
[24, 12, 4]
```

## Ακολούθησε μια κλήση

Η `example([2, 3, 4])` επιστρέφει `[24, 12, 4]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([5, 0, 2])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 0, 2]`. Το μηδέν επηρεάζει και όλα τα γινόμενα που βρίσκονται αριστερά του.

Συνέχισε στην εκφώνηση της άσκησης 43, στη σελίδα 169.

Η ΑΣΚΗΣΗ

# 43 Το γινόμενο όλων των άλλων

## Τι θα φτιάξεις

Στη θέση `i` της εξόδου βάλε το γινόμενο όλων των στοιχείων εκτός από το `numbers[i]`. Μην χρησιμοποιήσεις διαίρεση. Για κενό πίνακα επέστρεψε `[]` και για ένα μόνο στοιχείο `[1]`, το γινόμενο του κενού συνόλου παραγόντων.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 20 στοιχεία. Οι ακέραιες τιμές του είναι από -10 έως 10.

**Η κλήση**

```
solve([1, 2, 3, 4])
```

↓ Η τιμή που επιστρέφεται

```
[24, 12, 8, 6]
```

## Η ιδέα

Κάθε απάντηση χωρίζεται σε ένα γινόμενο αριστερά και ένα δεξιά. Δύο περάσματα αρκούν για να τα συνδυάσουν.

**Στόχος απόδοσης:** `O(n)` χρόνος, `O(n)` χώρος εξόδου και `O(1)` πρόσθετος χώρος.

ΤΟ ΓΙΝΟΜΕΝΟ ΟΛΩΝ ΤΩΝ ΑΛΛΩΝ

# 43 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 3, 4])` | `[24, 12, 8, 6]` |
| `solve([])` | `[]` |
| `solve([7])` | `[1]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/43-product-except-self/exercise.rb.

Tests: exercises/43-product-except-self/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/43-product-except-self/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Γράψε πρώτα το αριστερό γινόμενο στην έξοδο. Έπειτα πέρασε από δεξιά με έναν μόνο επιπλέον συσσωρευτή.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 397.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 44 Το modulo κλείνει έναν κύκλο

Θα παράγουμε τις θέσεις στις οποίες φτάνει μια μετατόπιση κατά δύο, πάνω σε έναν κύκλο δοσμένου μήκους.

RUBY

```ruby
def example(size)
  return [] if size == 0
  (0...size).map do |i|
    (i + 2) % size
  end
end
```

## Διάβασε τις γραμμές

01 Το `0...size` αποκλείει το δεξί άκρο και δίνει όλους τους έγκυρους δείκτες.

02 Το σύντομο `return [] if ...` σταματά τη μέθοδο πριν από τη διαίρεση με μηδέν.

03 Με θετικό διαιρέτη, το `%` κρατά το αποτέλεσμα μέσα στα όρια του κύκλου.

## Ακολούθησε μια κλήση

Η `example(5)` επιστρέφει `[2, 3, 4, 0, 1]`.

Η `example(0)` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 1]`. Δύο βήματα σε κύκλο δύο θέσεων είναι μία πλήρης περιστροφή.

Συνέχισε στην εκφώνηση της άσκησης 44, στη σελίδα 172.

Η ΑΣΚΗΣΗ

# 44 Περιστροφή χωρίς χαμένες τιμές

## Τι θα φτιάξεις

Περίστρεψε το `numbers` προς τα δεξιά κατά `steps` θέσεις. Οι τιμές που περνούν το τέλος εμφανίζονται στην αρχή. Επέστρεψε νέο πίνακα. Για κενή είσοδο επέστρεψε `[]`. Μην χρησιμοποιήσεις τη `rotate`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `steps` είναι από 0 έως 1000000.

**Η κλήση**

```
solve([1, 2, 3, 4], 1)
```

↓ Η τιμή που επιστρέφεται

```
[4, 1, 2, 3]
```

## Η ιδέα

Μια πλήρης περιστροφή δεν αλλάζει τίποτα. Αφού μικρύνει το πλήθος βημάτων, κάθε αρχική θέση έχει μία ακριβή τελική θέση.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` χώρος για την έξοδο, ανεξάρτητα από το μέγεθος του `steps`.

ΠΕΡΙΣΤΡΟΦΗ ΧΩΡΙΣ ΧΑΜΕΝΕΣ ΤΙΜΕΣ

# 44 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `steps`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 3, 4], 1)` | `[4, 1, 2, 3]` |
| `solve([], 5)` | `[]` |
| `solve([7], 100)` | `[7]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/44-rotate-array/exercise.rb.

Tests: exercises/44-rotate-array/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/44-rotate-array/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η θέση προορισμού είναι το υπόλοιπο της διαίρεσης του `i + steps` με το μήκος.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 398.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 45 Κάθε θέση έχει και έναν δείκτη

Θα συγκεντρώσουμε τις θέσεις των αρνητικών τιμών. Η `each_with_index` δίνει ταυτόχρονα την τιμή και τον δείκτη της.

RUBY

```ruby
def example(values)
  positions = []
  values.each_with_index do |value, i|
    positions << i if value < 0
  end
  positions
end
```

## Διάβασε τις γραμμές

01 Τα δύο ονόματα στο block είναι η τιμή και ο δείκτης, με αυτή τη σειρά.

02 Το `<< i if ...` προσθέτει θέση μόνο όταν η συνθήκη είναι αληθής.

03 Ο δείκτης ξεκινά από το μηδέν. Είναι χρήσιμος όταν θέλουμε να παρακολουθούμε πού βρισκόμαστε.

## Ακολούθησε μια κλήση

Η `example([2, -4, 0, -1])` επιστρέφει `[1, 3]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([-2, -2])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 1]`. Ίδιες τιμές σε διαφορετικές θέσεις δίνουν διαφορετικούς δείκτες.

Συνέχισε στην εκφώνηση της άσκησης 45, στη σελίδα 175.

Η ΑΣΚΗΣΗ

# 45 Συγχώνευση δύο ταξινομημένων πινάκων

## Τι θα φτιάξεις

Οι πίνακες `left` και `right` είναι ήδη ταξινομημένοι σε μη φθίνουσα σειρά. Επέστρεψε όλα τα στοιχεία τους σε έναν νέο ταξινομημένο πίνακα, διατηρώντας τα διπλότυπα. Μην χρησιμοποιήσεις `sort` ή `sort!`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `left` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `right` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

Η κλήση

```
solve([1, 4], [2, 3])
```

↓ Η τιμή που επιστρέφεται

```
[1, 2, 3, 4]
```

## Η ιδέα

Το μικρότερο διαθέσιμο στοιχείο βρίσκεται πάντα σε ένα από τα δύο μέτωπα. Αφού επιλεγεί, προχωρά μόνο ο δικός του δείκτης.

**Στόχος απόδοσης:** `O(n + m)` χρόνος και `O(n + m)` χώρος εξόδου.

ΣΥΓΧΩΝΕΥΣΗ ΔΥΟ ΤΑΞΙΝΟΜΗΜΕΝΩΝ ΠΙΝΑΚΩΝ

# 45 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `left`, `right`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 4], [2, 3])` | `[1, 2, 3, 4]` |
| `solve([], [])` | `[]` |
| `solve([], [1, 2])` | `[1, 2]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/45-merge-sorted-arrays/exercise.rb.

Tests: exercises/45-merge-sorted-arrays/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/45-merge-sorted-arrays/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Συνέχισε όσο απομένει στοιχείο σε οποιονδήποτε πίνακα. Ξεχώρισε την περίπτωση που έχει τελειώσει ο δεξιός.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 399.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 46 Παράλειψε ένα πέρασμα με next

Θα κρατήσουμε μόνο τα στοιχεία που βρίσκονται σε άρτιες θέσεις. Η επιλογή γίνεται με βάση τον δείκτη, όχι την τιμή.

RUBY

```ruby
def example(values)
  result = []
  values.each_with_index do |value, i|
    next if i.odd?
    result << value
  end
  result
end
```

## Διάβασε τις γραμμές

01 Η `odd?` επιστρέφει λογική τιμή. Το ερωτηματικό είναι μέρος του ονόματος της μεθόδου.

02 Το `next` συνεχίζει με το επόμενο στοιχείο. Δεν επιστρέφει από όλη τη μέθοδο όπως το `return`.

03 Η θέση μηδέν είναι άρτια, άρα το πρώτο στοιχείο κρατιέται.

## Ακολούθησε μια κλήση

Η `example([9, 8, 7, 6])` επιστρέφει `[9, 7]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, 5, 0])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 0]`. Τα μηδενικά κρατιούνται επειδή βρίσκονται στις σωστές θέσεις.

Συνέχισε στην εκφώνηση της άσκησης 46, στη σελίδα 178.

Η ΑΣΚΗΣΗ

# 46 Τα μηδενικά στο τέλος

## Τι θα φτιάξεις

Επέστρεψε νέο πίνακα με όλες τις μη μηδενικές τιμές του `numbers` στην αρχική σειρά τους και όλα τα μηδενικά στο τέλος. Το μήκος πρέπει να παραμείνει ίδιο. Μην ταξινομήσεις τον πίνακα.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

**Η κλήση**

```
solve([0, 3, 0, 1])
```

↓ Η τιμή που επιστρέφεται

```
[3, 1, 0, 0]
```

## Η ιδέα

Διαβάζουμε όλες τις θέσεις, αλλά γράφουμε μπροστά μόνο τις μη μηδενικές τιμές. Οι υπόλοιπες θέσεις της νέας εξόδου μένουν μηδενικές.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` χώρος εξόδου, με `O(1)` πρόσθετο χώρο.

ΤΑ ΜΗΔΕΝΙΚΑ ΣΤΟ ΤΕΛΟΣ

# 46 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([0, 3, 0, 1])` | `[3, 1, 0, 0]` |
| `solve([])` | `[]` |
| `solve([0])` | `[0]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/46-move-zeros/exercise.rb.

Tests: exercises/46-move-zeros/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/46-move-zeros/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Χρειάζεσαι έναν δείκτη για την επόμενη ελεύθερη θέση, όχι για τη θέση που διαβάζεις.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 400.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 47 Ένα μικρό if ως έκφραση

Θα μετατρέψουμε κάθε αριθμό σε έναν από δύο κωδικούς: έναν για θετικό και έναν για μη θετικό. Η επιλογή γράφεται σε μία έκφραση.

RUBY

```ruby
def example(values)
  values.map do |value|
    value > 0 ? 1 : -1
  end
end
```

## Διάβασε τις γραμμές

01 Το `condition ? a : b` επιστρέφει το `a` όταν ισχύει η συνθήκη και το `b` διαφορετικά.

02 Χρησιμοποίησέ το για δύο σύντομες επιλογές. Για πολλούς κλάδους προτίμησε ένα κανονικό `if`.

03 Η μηδενική τιμή ανήκει εδώ στον δεύτερο κλάδο. Δεν εξετάζουμε αν η ίδια η τιμή είναι αληθής.

## Ακολούθησε μια κλήση

Η `example([-2, 0, 3])` επιστρέφει `[-1, -1, 1]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 1])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[1, 1]`. Κάθε θέση εξετάζεται ανεξάρτητα.

Συνέχισε στην εκφώνηση της άσκησης 47, στη σελίδα 181.

Η ΑΣΚΗΣΗ

# 47 Η μεγαλύτερη ανοδική διαδρομή

## Τι θα φτιάξεις

Βρες το μήκος του μεγαλύτερου συνεχόμενου τμήματος του `numbers` στο οποίο κάθε τιμή είναι αυστηρά μεγαλύτερη από την προηγούμενη. Δεν επιτρέπεται να παραλείπεις θέσεις. Για κενό πίνακα επέστρεψε `0`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

**Η κλήση**

```
solve([1, 2, 2, 3, 4])
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Η ιδέα

Κράτα χωριστά το μήκος της διαδρομής που τελειώνει τώρα και το καλύτερο μήκος που έχει εμφανιστεί οπουδήποτε.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

Η ΜΕΓΑΛΥΤΕΡΗ ΑΝΟΔΙΚΗ ΔΙΑΔΡΟΜΗ

# 47 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 2, 3, 4])` | `3` |
| `solve([])` | `0` |
| `solve([7])` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/47-increasing-run/exercise.rb.

Tests: exercises/47-increasing-run/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/47-increasing-run/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μια ίση ή μικρότερη τιμή αρχίζει νέα διαδρομή μήκους ένα. Δεν μηδενίζει την απάντηση που έχεις ήδη βρει.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 401.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 48 Κράτησε το καλύτερο που έχεις δει

Θα επιστρέψουμε το μικρότερο στοιχείο κάθε προθέματος. Αυτό είναι χρήσιμο όταν η επόμενη απόφαση εξαρτάται από το παρελθόν.

RUBY

```ruby
def example(values)
  return [] if values.empty?
  smallest = values[0]
  values.map do |value|
    smallest = [smallest, value].min
  end
end
```

## Διάβασε τις γραμμές

01 Η `min` επιστρέφει το μικρότερο από τα δύο διαθέσιμα στοιχεία.

02 Η αρχικοποίηση από πραγματική τιμή λειτουργεί και όταν όλοι οι αριθμοί είναι θετικοί ή αρνητικοί.

03 Η μνήμη ενός προθέματος μπορεί να είναι μόνο ένας αριθμός, χωρίς να κρατάμε όλο το πρόθεμα.

## Ακολούθησε μια κλήση

Η `example([7, 3, 5, 2])` επιστρέφει `[7, 3, 3, 2]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([-2, -1, -4])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[-2, -2, -4]`. Η τιμή -1 δεν βελτιώνει το προηγούμενο ελάχιστο.

Συνέχισε στην εκφώνηση της άσκησης 48, στη σελίδα 184.

Η ΑΣΚΗΣΗ

# 48 Μία αγορά και μία πώληση

## Τι θα φτιάξεις

Το `prices[i]` είναι η τιμή μιας μετοχής την ημέρα `i`. Βρες το μέγιστο κέρδος από το πολύ μία αγορά και μία μεταγενέστερη πώληση. Αν δεν υπάρχει κερδοφόρα συναλλαγή, επέστρεψε `0`. Πρόκειται για αριθμητική άσκηση πάνω σε δεδομένα τιμών.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `prices` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 10000.

**Η κλήση**

```
solve([7, 1, 5, 3, 6, 4])
```

↓ Η τιμή που επιστρέφεται

```
5
```

## Η ιδέα

Για κάθε πιθανή ημέρα πώλησης αρκεί να ξέρουμε τη μικρότερη τιμή που προηγήθηκε. Δεν χρειάζεται να δοκιμάσουμε κάθε ζεύγος ημερών.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΜΙΑ ΑΓΟΡΑ ΚΑΙ ΜΙΑ ΠΩΛΗΣΗ

# 48 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `prices`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([7, 1, 5, 3, 6, 4])` | `5` |
| `solve([])` | `0` |
| `solve([8])` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/48-stock-profit/exercise.rb.

Tests: exercises/48-stock-profit/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/48-stock-profit/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Υπολόγισε το κέρδος πριν ενημερώσεις την ελάχιστη τιμή της αγοράς.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 402.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 49 Η καλύτερη τιμή μπορεί να είναι αρνητική

Θα βρούμε το μεγαλύτερο στοιχείο χωρίς να χρησιμοποιήσουμε τη `max` σε ολόκληρο τον πίνακα. Το παράδειγμα δέχεται μη κενή είσοδο.

RUBY

```ruby
def example(values)
  best = values[0]
  values.each do |value|
    best = value if value > best
  end
  best
end
```

## Διάβασε τις γραμμές

01 Η αρχική τιμή είναι υποψήφια λύση του ίδιου του προβλήματος.

02 Αν ξεκινούσαμε από μηδέν, οι είσοδοι μόνο με αρνητικούς αριθμούς θα έδιναν λάθος αποτέλεσμα.

03 Η `best` ποτέ δεν μειώνεται. Μετά από κάθε πέρασμα είναι το μέγιστο του προθέματος που διαβάσαμε.

## Ακολούθησε μια κλήση

Η `example([-8, -2, -5])` επιστρέφει `-2`.

Η `example([4])` επιστρέφει `4`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, -3, 2])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `2`. Το 2 είναι η μεγαλύτερη πραγματική τιμή της εισόδου.

Συνέχισε στην εκφώνηση της άσκησης 49, στη σελίδα 187.

Η ΑΣΚΗΣΗ

# 49 Το πιο κερδοφόρο συνεχόμενο τμήμα

## Τι θα φτιάξεις

Επέστρεψε το μεγαλύτερο άθροισμα ενός μη κενού συνεχόμενου τμήματος του `numbers`. Ο πίνακας έχει τουλάχιστον ένα στοιχείο. Το αποτέλεσμα μπορεί να είναι αρνητικό.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 1 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

**Η κλήση**

```
solve([-2, 1, -3, 4, -1, 2, 1, -5, 4])
```

↓ Η τιμή που επιστρέφεται

```
6
```

## Η ιδέα

Το καλύτερο τμήμα που τελειώνει σε μία θέση είτε αρχίζει εκεί είτε συνεχίζει το καλύτερο τμήμα που τελείωνε αμέσως πριν.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΤΟ ΠΙΟ ΚΕΡΔΟΦΟΡΟ ΣΥΝΕΧΟΜΕΝΟ ΤΜΗΜΑ

# 49 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([-2, 1, -3, 4, -1, 2, 1, -5, 4])` | `6` |
| `solve([-8, -3, -6])` | `-3` |
| `solve([5])` | `5` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/49-maximum-subarray/exercise.rb.

Tests: exercises/49-maximum-subarray/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/49-maximum-subarray/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κράτα δύο τιμές: καλύτερο άθροισμα που τελειώνει εδώ και καλύτερο άθροισμα συνολικά.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 403.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 50 Μια έγκαιρη επιστροφή κρατά την πρώτη απάντηση

Θα βρούμε την πρώτη θέση με μηδενική τιμή. Αν τελειώσει ο βρόχος χωρίς απάντηση, χρειάζεται ρητή εναλλακτική τιμή.

RUBY

```ruby
def example(values)
  values.each_with_index do |value, i|
    return i if value == 0
  end
  -1
end
```

## Διάβασε τις γραμμές

01 Το `return` μέσα στο block της `each_with_index` επιστρέφει από τη μέθοδο `example`.

02 Οι επόμενες θέσεις δεν χρειάζονται έλεγχο όταν έχει βρεθεί η πρώτη σωστή.

03 Το τελικό `-1` εκτελείται μόνο όταν δεν υπήρξε έγκαιρη επιστροφή.

## Ακολούθησε μια κλήση

Η `example([4, 0, 0])` επιστρέφει `1`.

Η `example([])` επιστρέφει `-1`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, 2])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `0`. Ο δείκτης μηδέν είναι κανονική επιτυχής απάντηση.

Συνέχισε στην εκφώνηση της άσκησης 50, στη σελίδα 190.

Η ΑΣΚΗΣΗ

# 50 Μια θέση με ίσα βάρη

## Τι θα φτιάξεις

Βρες τον μικρότερο δείκτη `i` για τον οποίο το άθροισμα αυστηρά αριστερά ισούται με το άθροισμα αυστηρά δεξιά. Το στοιχείο στη θέση `i` δεν συμμετέχει σε κανένα άθροισμα. Αν δεν υπάρχει τέτοια θέση, επέστρεψε `-1`. Το άθροισμα μιας κενής πλευράς είναι μηδέν.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

Η κλήση

```
solve([1, 7, 3, 6, 5, 6])
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Η ιδέα

Αφού γνωρίζουμε το συνολικό άθροισμα και την αριστερή πλευρά, η δεξιά προκύπτει με δύο αφαιρέσεις.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΜΙΑ ΘΕΣΗ ΜΕ ΙΣΑ ΒΑΡΗ

# 50 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 7, 3, 6, 5, 6])` | `3` |
| `solve([])` | `-1` |
| `solve([9])` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/50-equilibrium-index/exercise.rb.

Tests: exercises/50-equilibrium-index/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/50-equilibrium-index/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Έλεγξε τη θέση πριν προσθέσεις την τρέχουσα τιμή στο αριστερό άθροισμα.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 404.

ΑΣΚΗΣΕΙΣ 51-60

# Κείμενα, συχνότητες και hashes

Κανονικοποίηση, καταμέτρηση και αναζήτηση πληροφοριών που έχουν ήδη εμφανιστεί.



Δώσε χρόνο στη σκέψη, στις οριακές περιπτώσεις και στην εξήγηση του κόστους. Οι λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος αποτέλεσμα

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 51 Κράτα μόνο τους χαρακτήρες που μετράνε

Μια κανονική έκφραση περιγράφει σύνολα χαρακτήρων. Θα κρατήσουμε μόνο πεζά λατινικά γράμματα, αγνοώντας ακόμη και τα ψηφία.

RUBY

```ruby
def example(text)
  text.downcase.gsub(/[^a-z]/, "")
end
```

## Διάβασε τις γραμμές

01 Η `downcase` δημιουργεί πεζή μορφή του κειμένου.

02 Το `[^a-z]` μέσα στην κανονική έκφραση σημαίνει κάθε χαρακτήρα εκτός από λατινικό πεζό γράμμα.

03 Η `gsub` αντικαθιστά όλες τις αντιστοιχίες. Με κενό κείμενο αντικατάστασης τις αφαιρεί.

## Ακολούθησε μια κλήση

Η `example("Ruby 3!")` επιστρέφει `"ruby"`.

Η `example("123")` επιστρέφει `""`.

## Στάσου λίγο

Τι θα επιστρέψει η `example("A-B_C")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `"abc"`. Παύλα και underscore αφαιρούνται όπως και τα ψηφία.

Συνέχισε στην εκφώνηση της άσκησης 51, στη σελίδα 194.

Η ΑΣΚΗΣΗ

# 51 Παλίνδρομο μέσα στον θόρυβο

## Τι θα φτιάξεις

Το `text` περιέχει εκτυπώσιμους χαρακτήρες ASCII. Αγνόησε ό,τι δεν είναι λατινικό γράμμα ή ψηφίο και αντιμετώπισε κεφαλαία και πεζά ως ίδια. Επέστρεψε αν το υπόλοιπο διαβάζεται ίδιο από τις δύο κατευθύνσεις. Το κενό αποτέλεσμα είναι παλίνδρομο. Μην χρησιμοποιήσεις `reverse`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `text` έχει από 0 έως 200 χαρακτήρες.

Η κλήση

```
solve("A man, a plan, a canal: Panama!")
```

↓ Η τιμή που επιστρέφεται

```
true
```

## Η ιδέα

Πρώτα απομάκρυνε τις διαφορές που η εκφώνηση θεωρεί αδιάφορες. Μετά σύγκρινε συμμετρικές θέσεις από τα άκρα προς το κέντρο.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` πρόσθετος χώρος για το καθαρό κείμενο.

ΠΑΛΙΝΔΡΟΜΟ ΜΕΣΑ ΣΤΟΝ ΘΟΡΥΒΟ

# 51 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `text`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("A man, a plan, a canal: Panama!")` | `true` |
| `solve("")` | `true` |
| `solve("!!!")` | `true` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/51-normalized-palindrome/exercise.rb.

Tests: exercises/51-normalized-palindrome/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/51-normalized-palindrome/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ένας αριστερός δείκτης αυξάνεται και ένας δεξιός μειώνεται. Η πρώτη διαφορά αρκεί για αποτυχία.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 405.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 52 Μέτρησε την πρώτη ομάδα

Το παράδειγμα βρίσκει μόνο πόσοι αρχικοί χαρακτήρες είναι ίδιοι. Αυτό το μικρό βήμα μπορεί να επαναληφθεί από κάθε επόμενη αρχή ομάδας.

RUBY

```ruby
def example(text)
  return 0 if text.empty?
  finish = 1
  while finish < text.length && text[finish] == text[0]
    finish += 1
  end
  finish
end
```

## Διάβασε τις γραμμές

01 Ο δείκτης `finish` δείχνει την πρώτη θέση που δεν έχει επιβεβαιωθεί ως μέλος της ομάδας.

02 Ο έλεγχος ορίου προηγείται της ανάγνωσης χαρακτήρα.

03 Αν τελειώσει όλο το κείμενο, ο δείκτης είναι ίσος με το μήκος του.

## Ακολούθησε μια κλήση

Η `example("aaabb")` επιστρέφει `3`.

Η `example("")` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example("zzzz")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `4`. Η πρώτη ομάδα μπορεί να είναι ολόκληρη η είσοδος.

Συνέχισε στην εκφώνηση της άσκησης 52, στη σελίδα 197.

Η ΑΣΚΗΣΗ

# 52 Συμπίεση διαδοχικών χαρακτήρων

## Τι θα φτιάξεις

Το `text` περιέχει μόνο πεζά λατινικά γράμματα. Αντικατάστησε κάθε μέγιστη συνεχόμενη ομάδα ίδιων γραμμάτων με το γράμμα και το πλήθος της σε δεκαδική μορφή. Γράφουμε και το πλήθος `1`. Για κενό κείμενο επέστρεψε `""`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `text` έχει από 0 έως 500 χαρακτήρες.

Η κλήση

```
solve("aaabbc")
```

↓ Η τιμή που επιστρέφεται

```
"a3b2c1"
```

## Η ιδέα

Για κάθε αρχή ομάδας βρες την πρώτη διαφορετική θέση. Η διαφορά των δύο δεικτών δίνει το πλήθος χωρίς ξεχωριστό μετρητή.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` χώρος μαζί με την έξοδο.

ΣΥΜΠΙΕΣΗ ΔΙΑΔΟΧΙΚΩΝ ΧΑΡΑΚΤΗΡΩΝ

# 52 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `text`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("aaabbc")` | `"a3b2c1"` |
| `solve("")` | `""` |
| `solve("a")` | `"a1"` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/52-run-length-encode/exercise.rb.

Tests: exercises/52-run-length-encode/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/52-run-length-encode/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μετά την καταγραφή μιας ομάδας, η αναζήτηση συνεχίζει από το τέλος της, όχι από το επόμενο μόνο γράμμα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 406.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 53 Διάβασε ολόκληρους αριθμούς

Θα εξαγάγουμε όλες τις ομάδες ψηφίων από ένα κείμενο. Τα ψηφία της ίδιας ομάδας σχηματίζουν έναν αριθμό.

RUBY

```ruby
def example(text)
  text.scan(/[0-9]+/).map do |digits|
    digits.to_i
  end
end
```

## Διάβασε τις γραμμές

01 Το `+` στην έκφραση σημαίνει μία ή περισσότερες εμφανίσεις του προηγούμενου μοτίβου.

02 Η `scan` επιστρέφει πίνακα, ακόμη κι όταν δεν βρεθεί καμία αντιστοιχία.

03 Η `to_i` μετατρέπει κάθε ήδη απομονωμένη ομάδα ψηφίων σε ακέραιο.

## Ακολούθησε μια κλήση

Η `example("a12b3")` επιστρέφει `[12, 3]`.

Η `example("abc")` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example("x7y100")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[7, 100]`. Τα τρία ψηφία του 100 ανήκουν στην ίδια ομάδα.

Συνέχισε στην εκφώνηση της άσκησης 53, στη σελίδα 200.

Η ΑΣΚΗΣΗ

# 53 Ξαναφτιάξε το συμπιεσμένο κείμενο

## Τι θα φτιάξεις

Η είσοδος είναι κενή ή έγκυρη ακολουθία ομάδων: ένα πεζό λατινικό γράμμα και αμέσως ένας θετικός δεκαδικός αριθμός χωρίς αρχικό μηδέν. Επέστρεψε το αναπτυγμένο κείμενο. Κάθε πλήθος είναι από 1 έως 100 και το συνολικό αποτέλεσμα δεν ξεπερνά τους 1000 χαρακτήρες.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `encoded` έχει από 0 έως 500 χαρακτήρες.

Η κλήση

```
solve("a3b2c1")
```

↓ Η τιμή που επιστρέφεται

```
"aaabbc"
```

## Η ιδέα

Χώρισε την είσοδο σε ζεύγη γράμματος και πλήθους. Κάθε ζεύγος παράγει ανεξάρτητα ένα κομμάτι του αποτελέσματος.

**Στόχος απόδοσης:** `O(n + r)` χρόνος και χώρος, όπου `r` το μήκος του αναπτυγμένου κειμένου.

ΞΑΝΑΦΤΙΑΞΕ ΤΟ ΣΥΜΠΙΕΣΜΕΝΟ ΚΕΙΜΕΝΟ

# 53 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `encoded`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("a3b2c1")` | `"aaabbc"` |
| `solve("")` | `""` |
| `solve("z1")` | `"z"` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/53-run-length-decode/exercise.rb.

Tests: exercises/53-run-length-decode/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/53-run-length-decode/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Οι παρενθέσεις μιας κανονικής έκφρασης επιτρέπουν στη `scan` να επιστρέφει χωριστά τα τμήματα που σε ενδιαφέρουν.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 407.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 54 Κάθε λέξη έχει τον δικό της μετρητή

Θα μετρήσουμε τις εμφανίσεις των λέξεων και θα επιστρέψουμε τη συχνότητα στη θέση κάθε αρχικής λέξης.

RUBY

```ruby
def example(words)
  counts = Hash.new(0)
  words.each { |word| counts[word] += 1 }
  words.map { |word| counts[word] }
end
```

## Διάβασε τις γραμμές

01 Ένα `Hash` συνδέει κλειδιά με τιμές. Εδώ το κλειδί είναι λέξη και η τιμή μετρητής.

02 Η προεπιλογή μηδέν επιτρέπει το πρώτο `+= 1` χωρίς ξεχωριστό έλεγχο ύπαρξης.

03 Το δεύτερο πέρασμα γίνεται αφού έχουν ολοκληρωθεί όλοι οι μετρητές.

## Ακολούθησε μια κλήση

Η `example(["red", "blue", "red"])` επιστρέφει `[2, 1, 2]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(["a", "a", "a"])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[3, 3, 3]`. Όλες οι θέσεις διαβάζουν τον ίδιο τελικό μετρητή του a.

Συνέχισε στην εκφώνηση της άσκησης 54, στη σελίδα 203.

Η ΑΣΚΗΣΗ

# 54 Η συχνότερη λέξη

## Τι θα φτιάξεις

Κάθε στοιχείο του `words` είναι μη κενή λέξη με έως 20 πεζά λατινικά γράμματα. Επέστρεψε τη συχνότερη λέξη. Σε ισοβαθμία διάλεξε τη λεξικογραφικά μικρότερη. Για κενό πίνακα επέστρεψε `""`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `words` έχει από 0 έως 100 στοιχεία.

Η κλήση

```
solve(["red", "blue", "red"])
```

↓ Η τιμή που επιστρέφεται

```
"red"
```

## Η ιδέα

Αποθήκευσε έναν μετρητή ανά διαφορετική λέξη. Η τελική επιλογή έχει δύο κριτήρια: μεγαλύτερη συχνότητα και μικρότερη λέξη.

**Στόχος απόδοσης:** `O(n)` αναμενόμενος χρόνος και `O(u)` χώρος για `u` διαφορετικές λέξεις, με φραγμένο μήκος λέξης.

Η ΣΥΧΝΟΤΕΡΗ ΛΕΞΗ

# 54 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `words`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(["red", "blue", "red"])` | `"red"` |
| `solve([])` | `""` |
| `solve(["z", "a"])` | `"a"` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/54-most-frequent-word/exercise.rb.

Tests: exercises/54-most-frequent-word/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/54-most-frequent-word/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το `Hash.new(0)` επιστρέφει μηδέν για λέξη που δεν έχει εμφανιστεί ακόμη. Για ταξινόμηση δύο κριτηρίων μπορείς να σχηματίσεις πίνακα.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 408.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 55 Χαρακτήρες και bytes δεν είναι το ίδιο

Θα επιστρέψουμε κάθε δεύτερο χαρακτήρα ενός κειμένου. Η μετατροπή σε `chars` κάνει ρητή τη μονάδα των δεικτών μας.

RUBY

```ruby
def example(text)
  chars = text.chars
  result = []
  chars.each_with_index do |char, i|
    result << char if i.even?
  end
  result.join
end
```

## Διάβασε τις γραμμές

01 Η `chars` επιστρέφει πίνακα Unicode code points ως μικρά κείμενα.

02 Ένα ελληνικό γράμμα μπορεί να πιάνει πολλά bytes αλλά εδώ καταλαμβάνει μία θέση.

03 Η `join` χωρίς όρισμα ενώνει τα κομμάτια χωρίς διαχωριστικό.

## Ακολούθησε μια κλήση

Η `example("abcd")` επιστρέφει `"ac"`.

Η `example("")` επιστρέφει `""`.

## Στάσου λίγο

Τι θα επιστρέψει η `example("αβγδ")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `"αγ"`. Οι άρτιες θέσεις είναι η μηδενική και η δεύτερη.

Συνέχισε στην εκφώνηση της άσκησης 55, στη σελίδα 206.

Η ΑΣΚΗΣΗ

# 55 Ο πρώτος μοναδικός χαρακτήρας

## Τι θα φτιάξεις

Βρες τη θέση του πρώτου χαρακτήρα που εμφανίζεται ακριβώς μία φορά στο `text`. Επέστρεψε `-1` αν δεν υπάρχει. Κεφαλαία, πεζά και κενά μετρούν ως διαφορετικοί χαρακτήρες. Οι θέσεις ακολουθούν τη `String#chars` της Ruby, δηλαδή Unicode code points, όχι bytes ή οπτικά συμπλέγματα.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `text` έχει από 0 έως 200 χαρακτήρες.

Η κλήση

```
solve("swiss")
```

↓ Η τιμή που επιστρέφεται

```
1
```

## Η ιδέα

Το πρώτο πέρασμα μαθαίνει όλες τις συχνότητες. Το δεύτερο διατηρεί την αρχική σειρά και βρίσκει την πρώτη μοναδική θέση.

**Στόχος απόδοσης:** `O(n)` αναμενόμενος χρόνος και `O(n)` πρόσθετος χώρος.

Ο ΠΡΩΤΟΣ ΜΟΝΑΔΙΚΟΣ ΧΑΡΑΚΤΗΡΑΣ

# 55 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `text`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("swiss")` | `1` |
| `solve("")` | `-1` |
| `solve("aabb")` | `-1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/55-first-unique-character/exercise.rb.

Tests: exercises/55-first-unique-character/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/55-first-unique-character/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μην αποφασίσεις ότι ένας χαρακτήρας είναι μοναδικός μόλις τον δεις πρώτη φορά. Μπορεί να επανεμφανιστεί αργότερα.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 409.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 56 Μηδενικό υπόλοιπο για κάθε κλειδί

Θα μετρήσουμε τη συνολική διαφορά πλήθους εμφανίσεων ανάμεσα σε δύο πίνακες αριθμών. Η επιστροφή είναι το άθροισμα των απόλυτων διαφορών.

RUBY

```ruby
def example(first, second)
  counts = Hash.new(0)
  first.each { |value| counts[value] += 1 }
  second.each { |value| counts[value] -= 1 }
  counts.values.sum { |count| count.abs }
end
```

## Διάβασε τις γραμμές

01 Τα κλειδιά ενός Hash μπορούν να είναι και ακέραιοι.

02 Η `values` δίνει τις αποθηκευμένες τιμές χωρίς τα κλειδιά.

03 Η `sum` με block προσθέτει τις τιμές που επιστρέφει το block, εδώ τις απόλυτες διαφορές.

## Ακολούθησε μια κλήση

Η `example([1, 1, 2], [1, 3])` επιστρέφει `3`.

Η `example([], [])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([4, 4], [4, 4])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `0`. Η σειρά δεν έχει σημασία όταν όλοι οι μετρητές μηδενίζονται.

Συνέχισε στην εκφώνηση της άσκησης 56, στη σελίδα 209.

Η ΑΣΚΗΣΗ

# 56 Ίδιοι χαρακτήρες, άλλη σειρά

## Τι θα φτιάξεις

Επέστρεψε αν τα `left` και `right` περιέχουν ακριβώς τους ίδιους χαρακτήρες με τα ίδια πλήθη. Η σειρά δεν μετρά. Κενά και σημεία στίξης μετρούν κανονικά και η σύγκριση είναι ευαίσθητη σε κεφαλαία και πεζά. Χρησιμοποίησε χαρακτήρες Ruby, χωρίς Unicode κανονικοποίηση και χωρίς ταξινόμηση.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `left` έχει από 0 έως 200 χαρακτήρες. Το `right` έχει από 0 έως 200 χαρακτήρες.

Η κλήση

```
solve("listen", "silent")
```

↓ Η τιμή που επιστρέφεται

```
true
```

## Η ιδέα

Δες το ένα κείμενο ως προσθήκες σε μετρητές και το άλλο ως αφαιρέσεις. Η ισότητα σημαίνει ότι δεν περισσεύει τίποτα.

**Στόχος απόδοσης:** `O(n + m)` αναμενόμενος χρόνος και `O(u)` χώρος για τους διαφορετικούς χαρακτήρες.

ΙΔΙΟΙ ΧΑΡΑΚΤΗΡΕΣ, ΑΛΛΗ ΣΕΙΡΑ

# 56 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `left`, `right`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("listen", "silent")` | `true` |
| `solve("", "")` | `true` |
| `solve("aab", "abb")` | `false` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/56-anagrams/exercise.rb.

Tests: exercises/56-anagrams/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/56-anagrams/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ένας χαρακτήρας που εμφανίζεται μόνο στο δεύτερο κείμενο πρέπει να δημιουργήσει αρνητικό μετρητή.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 410.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 57 Ένα κλειδί υπάρχει μόνο μία φορά

Θα επιστρέψουμε όλες τις διαφορετικές τιμές σε αύξουσα σειρά. Η εισαγωγή μιας τιμής δεύτερη φορά δεν αυξάνει το πλήθος κλειδιών.

RUBY

```ruby
def example(values)
  seen = {}
  values.each { |value| seen[value] = true }
  seen.keys.sort
end
```

## Διάβασε τις γραμμές

01 Το `{}` δημιουργεί κενό Hash.

02 Η `keys` επιστρέφει πίνακα με τα κλειδιά. Κάθε κλειδί εμφανίζεται μία φορά.

03 Η `sort` δημιουργεί ταξινομημένο πίνακα και αφήνει την αρχική είσοδο ήσυχη.

## Ακολούθησε μια κλήση

Η `example([3, 1, 3, 2])` επιστρέφει `[1, 2, 3]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, -1, 0])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[-1, 0]`. Η ταξινόμηση ακολουθεί τις αριθμητικές τιμές.

Συνέχισε στην εκφώνηση της άσκησης 57, στη σελίδα 212.

Η ΑΣΚΗΣΗ

# 57 Κοινά στοιχεία χωρίς διπλότυπα

## Τι θα φτιάξεις

Επέστρεψε τους διαφορετικούς ακεραίους που υπάρχουν και στο `left` και στο `right`, σε αύξουσα σειρά. Κάθε κοινή τιμή εμφανίζεται μία φορά στην έξοδο, ανεξάρτητα από τα πλήθη εισόδου.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `left` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `right` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

Η κλήση

```
solve([3, 1, 3, 2], [2, 3, 3])
```

↓ Η τιμή που επιστρέφεται

```
[2, 3]
```

## Η ιδέα

Χρειάζεται έλεγχος παρουσίας, όχι συχνότητας. Κράτα ένα σύνολο τιμών από τον πρώτο πίνακα και ένα δεύτερο για τις κοινές τιμές.

**Στόχος απόδοσης:** `O(n + m + u log u)` αναμενόμενος χρόνος και `O(n + u)` χώρος, όπου `u` οι κοινές διαφορετικές τιμές.

ΚΟΙΝΑ ΣΤΟΙΧΕΙΑ ΧΩΡΙΣ ΔΙΠΛΟΤΥΠΑ

# 57 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `left`, `right`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([3, 1, 3, 2], [2, 3, 3])` | `[2, 3]` |
| `solve([], [1])` | `[]` |
| `solve([1], [])` | `[]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/57-unique-intersection/exercise.rb.

Tests: exercises/57-unique-intersection/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/57-unique-intersection/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ένα Hash με τιμή `true` για κάθε κλειδί μπορεί να χρησιμοποιηθεί ως σύνολο χωρίς πρόσθετη βιβλιοθήκη.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 411.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 58 Βρες τα σημεία από όπου αρχίζει κάτι

Θα επιστρέψουμε μόνο τις τιμές που δεν έχουν προκάτοχο μέσα στην είσοδο. Αυτές είναι οι πιθανές αρχές διαδοχικών ακολουθιών.

RUBY

```ruby
def example(values)
  present = {}
  values.each { |value| present[value] = true }
  present.keys.select do |value|
    !present.key?(value - 1)
  end.sort
end
```

## Διάβασε τις γραμμές

01 Το `!` αντιστρέφει τη λογική τιμή του ελέγχου παρουσίας.

02 Η `select` κρατά τις τιμές που ικανοποιούν τη συνθήκη.

03 Η τελική ταξινόμηση είναι μόνο για τη σειρά αυτού του παραδείγματος. Η άσκηση ζητά μήκος χωρίς ταξινόμηση.

## Ακολούθησε μια κλήση

Η `example([5, 2, 3, 9, 6])` επιστρέφει `[2, 5, 9]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, -1, 1])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[-1]`. Μόνο το -1 δεν έχει προηγούμενη τιμή στην είσοδο.

Συνέχισε στην εκφώνηση της άσκησης 58, στη σελίδα 215.

Η ΑΣΚΗΣΗ

# 58 Η μεγαλύτερη ακολουθία διαδοχικών τιμών

## Τι θα φτιάξεις

Βρες πόσοι διαφορετικοί διαδοχικοί ακέραιοι μπορούν να σχηματίσουν τη μεγαλύτερη ακολουθία από τιμές του `numbers`. Η θέση τους στην είσοδο δεν μετρά και τα διπλότυπα αγνοούνται. Για κενό πίνακα επέστρεψε `0`. Μην ταξινομήσεις.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

**Η κλήση**

```
solve([100, 4, 200, 1, 3, 2])
```

↓ Η τιμή που επιστρέφεται

```
4
```

## Η ιδέα

Μια τιμή είναι αρχή ακολουθίας μόνο αν λείπει η αμέσως προηγούμενη. Ξεκινώντας μόνο από τέτοιες αρχές, αποφεύγεις να ξαναμετράς το ίδιο τμήμα.

**Στόχος απόδοσης:** `O(n)` αναμενόμενος χρόνος και `O(n)` χώρος.

Η ΜΕΓΑΛΥΤΕΡΗ ΑΚΟΛΟΥΘΙΑ ΔΙΑΔΟΧΙΚΩΝ ΤΙΜΩΝ

# 58 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([100, 4, 200, 1, 3, 2])` | `4` |
| `solve([])` | `0` |
| `solve([8])` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/58-longest-consecutive-set/exercise.rb.

Tests: exercises/58-longest-consecutive-set/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/58-longest-consecutive-set/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κράτα παρουσία τιμών σε Hash. Πριν επεκτείνεις από το `value`, έλεγξε αν υπάρχει το `value - 1`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 412.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 59 Κράτα την πρώτη εμφάνιση

Θα σημειώσουμε σε ποια θέση εμφανίστηκε πρώτη φορά κάθε τιμή. Η απάντηση του παραδείγματος δίνει αυτή τη θέση για κάθε αρχικό στοιχείο.

RUBY

```ruby
def example(values)
  first = {}
  values.each_with_index do |value, i|
    first[value] = i unless first.key?(value)
  end
  values.map { |value| first[value] }
end
```

## Διάβασε τις γραμμές

01 Το `unless condition` εκτελεί την εντολή όταν η συνθήκη είναι ψευδής.

02 Η δεύτερη εμφάνιση βρίσκει ήδη αποθηκευμένο κλειδί και δεν αλλάζει την πρώτη θέση.

03 Η `key?` διαχωρίζει την απουσία από οποιαδήποτε αποθηκευμένη τιμή, ακόμη και τη θέση μηδέν.

## Ακολούθησε μια κλήση

Η `example([4, 2, 4, 2])` επιστρέφει `[0, 1, 0, 1]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, 0, 1])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 0, 2]`. Και τα δύο μηδενικά αναφέρονται στην πρώτη θέση.

Συνέχισε στην εκφώνηση της άσκησης 59, στη σελίδα 218.

Η ΑΣΚΗΣΗ

# 59 Δύο θέσεις με δοσμένο άθροισμα

## Τι θα φτιάξεις

Βρες δείκτες `i < j` με `numbers[i] + numbers[j] == target`. Επέστρεψε `[i, j]` ή `[]` αν δεν υπάρχει ζεύγος. Αν υπάρχουν πολλά, επίλεξε το μικρότερο `j` και, γι' αυτό, το μικρότερο `i`. Δεν επιτρέπεται να χρησιμοποιήσεις την ίδια θέση δύο φορές.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `target` είναι από -2000 έως 2000.

Η κλήση

```
solve([2, 7, 11, 15], 9)
```

↓ Η τιμή που επιστρέφεται

```
[0, 1]
```

## Η ιδέα

Όταν εξετάζεις μια νέα τιμή, ξέρεις ακριβώς ποια προηγούμενη τιμή θα συμπλήρωνε τον στόχο. Αποθήκευσε την πρώτη θέση κάθε τιμής.

**Στόχος απόδοσης:** `O(n)` αναμενόμενος χρόνος και `O(n)` χώρος.

ΔΥΟ ΘΕΣΕΙΣ ΜΕ ΔΟΣΜΕΝΟ ΑΘΡΟΙΣΜΑ

# 59 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `target`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 7, 11, 15], 9)` | `[0, 1]` |
| `solve([], 1)` | `[]` |
| `solve([3], 6)` | `[]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/59-two-sum-indices/exercise.rb.

Tests: exercises/59-two-sum-indices/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/59-two-sum-indices/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ο έλεγχος του συμπληρώματος πρέπει να προηγείται της αποθήκευσης της τρέχουσας θέσης.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 413.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 60 Πόσες προηγούμενες τιμές είναι ίδιες;

Θα μετρήσουμε ζεύγη ίσων τιμών με διαφορετικές θέσεις. Κάθε νέα εμφάνιση σχηματίζει ζεύγος με όλες τις προηγούμενες ίδιες.

RUBY

```ruby
def example(values)
  counts = Hash.new(0)
  pairs = 0
  values.each do |value|
    pairs += counts[value]
    counts[value] += 1
  end
  pairs
end
```

## Διάβασε τις γραμμές

01 Ο μετρητής διαβάζεται πριν αυξηθεί, ώστε η τιμή να μη σχηματίσει ζεύγος με τον εαυτό της.

02 Η τρίτη εμφάνιση προσθέτει δύο νέα ζεύγη, όχι μόνο ένα.

03 Το συνολικό πλήθος μπορεί να είναι τετραγωνικό, παρότι η καταμέτρηση χρειάζεται ένα μόνο πέρασμα.

## Ακολούθησε μια κλήση

Η `example([4, 4, 4])` επιστρέφει `3`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 2, 1, 2])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `2`. Υπάρχει ένα ζεύγος για τα 1 και ένα για τα 2.

Συνέχισε στην εκφώνηση της άσκησης 60, στη σελίδα 221.

Η ΑΣΚΗΣΗ

# 60 Πόσα τμήματα έχουν το σωστό άθροισμα;

## Τι θα φτιάξεις

Μέτρησε όλα τα μη κενά συνεχόμενα τμήματα του `numbers` που έχουν άθροισμα `target`. Διαφορετικά άκρα μετρούν ως διαφορετικά τμήματα. Επιτρέπονται αρνητικές τιμές και μηδενικά.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `target` είναι από -10000 έως 10000.

Η κλήση

```
solve([1, 1, 1], 2)
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Ένα τμήμα έχει το ζητούμενο άθροισμα όταν δύο προθέματα διαφέρουν κατά `target`. Για κάθε νέο πρόθεμα χρειάζεσαι πόσα κατάλληλα παλιότερα προθέματα υπάρχουν.

**Στόχος απόδοσης:** `O(n)` αναμενόμενος χρόνος και `O(n)` χώρος.

ΠΟΣΑ ΤΜΗΜΑΤΑ ΕΧΟΥΝ ΤΟ ΣΩΣΤΟ ΑΘΡΟΙΣΜΑ;

# 60 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `target`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 1, 1], 2)` | `2` |
| `solve([], 0)` | `0` |
| `solve([0, 0, 0], 0)` | `6` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/60-count-target-subarrays/exercise.rb.

Tests: exercises/60-count-target-subarrays/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/60-count-target-subarrays/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Το άθροισμα μηδέν έχει ήδη μία εμφάνιση πριν από το πρώτο στοιχείο: το κενό πρόθεμα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 414.

ΑΣΚΗΣΕΙΣ 61-70

# Αναζήτηση και κινούμενα παράθυρα

Μονοτονικές συνθήκες, δυαδικά όρια και παράθυρα που ενημερώνονται χωρίς νέο πλήρη υπολογισμό.



Δώσε χρόνο στη σκέψη, στις οριακές περιπτώσεις και στην εξήγηση του κόστους. Οι λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος αποτέλεσμα

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 61 Ένα όριο μπορεί να βρίσκεται μετά το τέλος

Θα βρούμε την πρώτη θέση με τιμή τουλάχιστον δέκα σε ταξινομημένο πίνακα. Αν δεν υπάρχει, θα επιστρέψουμε το μήκος του.

RUBY

```ruby
def example(values)
  left = 0
  right = values.length
  while left < right
    middle = (left + right) / 2
    if values[middle] < 10
      left = middle + 1
    else
      right = middle
    end
  end
  left
end
```

## Διάβασε τις γραμμές

01 Η διαίρεση δύο θετικών ακεραίων δίνει ακέραιο μέσο δείκτη.

02 Το `middle + 1` αποκλείει τη θέση που αποδείχθηκε πολύ μικρή. Το `right = middle` κρατά το μέσο ως πιθανό όριο.

03 Κάθε βήμα μικραίνει το διάστημα. Το όριο μπορεί νόμιμα να ισούται με το μήκος.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 61 Από τον κώδικα στην τιμή

**Η κλήση**

```
example([2, 10, 10, 15])
```

↓ Η τιμή που επιστρέφεται

```
1
```

## Ακολούθησε μια κλήση

Η `example([2, 10, 10, 15])` επιστρέφει `1`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 4, 9])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `3`. Όλες οι τιμές είναι μικρότερες από το όριο δέκα.

Συνέχισε στην εκφώνηση της άσκησης 61, στη σελίδα 226.

Η ΑΣΚΗΣΗ

# 61 Η πρώτη εμφάνιση με δυαδική αναζήτηση

## Τι θα φτιάξεις

Το `numbers` είναι ταξινομημένο σε μη φθίνουσα σειρά. Επέστρεψε τον πρώτο δείκτη του `target` ή `-1` αν λείπει. Υλοποίησε δυαδική αναζήτηση χωρίς `index`, `find_index`, `bsearch` ή `bsearch_index`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `target` είναι από -1000 έως 1000.

Η κλήση

```
solve([1, 2, 2, 2, 5], 2)
```

↓ Η τιμή που επιστρέφεται

```
1
```

## Η ιδέα

Κράτα ένα διάστημα πιθανών θέσεων. Όταν το μέσο είναι αρκετά μεγάλο, η πρώτη εμφάνιση δεν μπορεί να βρίσκεται δεξιότερα από αυτό.

**Στόχος απόδοσης:** `O(log n)` χρόνος και `O(1)` πρόσθετος χώρος.

Η ΠΡΩΤΗ ΕΜΦΑΝΙΣΗ ΜΕ ΔΥΑΔΙΚΗ ΑΝΑΖΗΤΗΣΗ

# 61 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `target`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 2, 2, 5], 2)` | `1` |
| `solve([], 1)` | `-1` |
| `solve([4], 4)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/61-binary-search-first/exercise.rb.

Tests: exercises/61-binary-search-first/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/61-binary-search-first/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Με ημι-ανοιχτό διάστημα `[left, right)`, το αρχικό δεξί όριο είναι το μήκος του πίνακα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 415.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 62 Σκέψου την αναζήτηση ως σειρά true και false

Θα μετατρέψουμε έναν ταξινομημένο πίνακα σε ελέγχους «τουλάχιστον πέντε». Οι σωστές συνθήκες σχηματίζουν ένα συνεχόμενο τελευταίο τμήμα.

```ruby
def example(values)
  values.map { |value| value >= 5 ? 1 : 0 }
end
```

## Διάβασε τις γραμμές

01 Η μονοτονία της εισόδου σημαίνει ότι μετά το πρώτο `1` δεν μπορεί να εμφανιστεί `0`.

02 Η δυαδική αναζήτηση εντοπίζει αυτή τη μετάβαση χωρίς να κατασκευάσει ολόκληρο τον πίνακα ελέγχων.

03 Επιτρέπεται να λείπουν εντελώς τα `0` ή τα `1`. Τα δύο άκρα είναι κανονικές περιπτώσεις.

## Ακολούθησε μια κλήση

Η `example([1, 4, 5, 9])` επιστρέφει `[0, 0, 1, 1]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([5, 5])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[1, 1]`. Η πρώτη σωστή θέση είναι ήδη στην αρχή.

Συνέχισε στην εκφώνηση της άσκησης 62, στη σελίδα 229.

Η ΑΣΚΗΣΗ

# 62 Πού θα μπει η νέα τιμή;

## Τι θα φτιάξεις

Ο πίνακας είναι ταξινομημένος σε μη φθίνουσα σειρά. Επέστρεψε τη θέση όπου θα εισαγόταν το `target` πριν από όλες τις ήδη ίσες τιμές. Αν είναι μεγαλύτερο από όλες, επέστρεψε `numbers.length`. Μην κάνεις την εισαγωγή. Υλοποίησε την αναζήτηση χωρίς έτοιμη μέθοδο δυαδικής αναζήτησης.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `target` είναι από -1000 έως 1000.

Η κλήση

```
solve([1, 3, 5], 4)
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Εδώ μας ενδιαφέρει το όριο ανάμεσα στις μικρότερες και τις όχι μικρότερες τιμές. Το όριο παραμένει χρήσιμο ακόμη και όταν ο στόχος απουσιάζει.

**Στόχος απόδοσης:** `O(log n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΠΟΥ ΘΑ ΜΠΕΙ Η ΝΕΑ ΤΙΜΗ;

# 62 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `target`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 3, 5], 4)` | `2` |
| `solve([], 5)` | `0` |
| `solve([2, 2, 2], 2)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/62-insertion-position/exercise.rb.

Tests: exercises/62-insertion-position/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/62-insertion-position/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η ίδια αναζήτηση της προηγούμενης άσκησης δουλεύει, αλλά ο τελικός έλεγχος ισότητας δεν χρειάζεται.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 416.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 63 Η Ruby έχει έτοιμη αναζήτηση ορίου

Θα βρούμε την πρώτη αυστηρά θετική τιμή ενός ταξινομημένου πίνακα. Το αποτέλεσμα θα είναι δείκτης ή το μήκος όταν δεν υπάρχει τέτοια τιμή.

RUBY

```ruby
def example(values)
  position = values.bsearch_index { |value| value > 0 }
  position || values.length
end
```

## Διάβασε τις γραμμές

01 Το block πρέπει να είναι ψευδές πριν από το όριο και αληθές μετά από αυτό.

02 Η `bsearch_index` επιστρέφει δείκτη, όχι την τιμή του στοιχείου.

03 Το `||` αντικαθιστά το `nil`. Η θέση μηδέν είναι αληθής στη Ruby και διατηρείται.

## Ακολούθησε μια κλήση

Η `example([-3, 0, 2, 9])` επιστρέφει `2`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 1])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `0`. Το πρώτο στοιχείο είναι ήδη αυστηρά θετικό.

Συνέχισε στην εκφώνηση της άσκησης 63, στη σελίδα 232.

Η ΑΣΚΗΣΗ

# 63 Πού αρχίζουν και πού τελειώνουν τα ίσα;

## Τι θα φτιάξεις

Σε ταξινομημένο πίνακα, επέστρεψε `[first, last]` με την πρώτη και την τελευταία θέση του `target`. Αν λείπει, επέστρεψε `[]`. Η αναζήτηση πρέπει να παραμένει λογαριθμική ακόμη κι όταν όλες οι τιμές είναι ίδιες. Επιτρέπεται η `bsearch_index`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `target` είναι από -1000 έως 1000.

Η κλήση

```
solve([1, 2, 2, 2, 4], 2)
```

↓ Η τιμή που επιστρέφεται

```
[1, 3]
```

## Η ιδέα

Το πρώτο όριο βρίσκει τιμές τουλάχιστον ίσες με τον στόχο. Το δεύτερο βρίσκει τιμές αυστηρά μεγαλύτερες. Ανάμεσά τους βρίσκονται όλες οι ίσες.

**Στόχος απόδοσης:** `O(log n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΠΟΥ ΑΡΧΙΖΟΥΝ ΚΑΙ ΠΟΥ ΤΕΛΕΙΩΝΟΥΝ ΤΑ ΙΣΑ;

# 63 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `target`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 2, 2, 4], 2)` | `[1, 3]` |
| `solve([], 1)` | `[]` |
| `solve([5], 5)` | `[0, 0]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/63-equal-range/exercise.rb.

Tests: exercises/63-equal-range/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/63-equal-range/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η `bsearch_index` μπορεί να βρει το πρώτο `true` μιας μονοτονικής συνθήκης. Αν δεν βρει, δίνει `nil`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 417.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 64 Μια συνθήκη χωρίζει και αριθμούς

Θα βρούμε με απλό βρόχο το πρώτο μη αρνητικό `x` του οποίου το διπλάσιο ξεπερνά έναν μη αρνητικό στόχο. Στην άσκηση θα αντικαταστήσουμε το γραμμικό ψάξιμο με δυαδικό.

RUBY

```ruby
def example(limit)
  x = 0
  x += 1 while 2 * x <= limit
  x
end
```

## Διάβασε τις γραμμές

01 Το σύντομο `while` επαναλαμβάνει τη μία εντολή όσο ισχύει η συνθήκη.

02 Η επιστροφή είναι η πρώτη αποτυχία, όχι η τελευταία επιτυχία.

03 Η ίδια διάκριση καθορίζει αν στο τέλος μιας δυαδικής αναζήτησης χρειάζεται αφαίρεση ενός.

## Ακολούθησε μια κλήση

Η `example(6)` επιστρέφει `4`.

Η `example(0)` επιστρέφει `1`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(7)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `4`. Τα 6 χωρούν στον στόχο, αλλά τα 8 τον ξεπερνούν.

Συνέχισε στην εκφώνηση της άσκησης 64, στη σελίδα 235.

Η ΑΣΚΗΣΗ

# 64 Τετραγωνική ρίζα χωρίς δεκαδικά

## Τι θα φτιάξεις

Επέστρεψε τον μεγαλύτερο μη αρνητικό ακέραιο `x` για τον οποίο `x * x <= number`. Μην χρησιμοποιήσεις `Math.sqrt`, `Integer.sqrt`, δεκαδικούς αριθμούς ή ύψωση στη δύναμη `0.5`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `number` είναι από 0 έως 1000000000.

**Η κλήση**

```
solve(8)
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Όταν ένα τετράγωνο είναι ήδη πολύ μεγάλο, όλα τα μεγαλύτερα είναι επίσης πολύ μεγάλα. Αναζητούμε την πρώτη αποτυχία αυτής της μονοτονικής συνθήκης.

**Στόχος απόδοσης:** `O(log(number + 1))` χρόνος και `O(1)` πρόσθετος χώρος.

ΤΕΤΡΑΓΩΝΙΚΗ ΡΙΖΑ ΧΩΡΙΣ ΔΕΚΑΔΙΚΑ

# 64 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `number`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(8)` | `2` |
| `solve(0)` | `0` |
| `solve(1)` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/64-integer-square-root/exercise.rb.

Tests: exercises/64-integer-square-root/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/64-integer-square-root/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Χρησιμοποίησε το διάστημα από μηδέν μέχρι `number + 1`. Η ζητούμενη τιμή είναι μία θέση πριν από το πρώτο υπερβολικό τετράγωνο.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 418.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 65 Μέτρα πόσες ομάδες χρειάζονται

Με σταθερό όριο δέκα, θα χωρίσουμε θετικά βάρη από 1 έως 10 σε διαδοχικές ομάδες. Κάθε ομάδα παίρνει όσα επόμενα βάρη χωρούν.

RUBY

```ruby
def example(weights)
  return 0 if weights.empty?
  groups = 1
  load = 0
  weights.each do |weight|
    if load + weight > 10
      groups += 1
      load = 0
    end
    load += weight
  end
  groups
end
```

## Διάβασε τις γραμμές

01 Το επόμενο βάρος ελέγχεται πριν προστεθεί στο τρέχον φορτίο.

02 Όταν δεν χωρά, αρχίζει νέα ομάδα και το ίδιο βάρος γίνεται το πρώτο της.

03 Ισότητα με το όριο επιτρέπεται. Νέα ομάδα χρειάζεται μόνο για αυστηρή υπέρβαση.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 65 Από τον κώδικα στην τιμή

**Η κλήση**

```
example([6, 4, 5])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Ακολούθησε μια κλήση

Η `example([6, 4, 5])` επιστρέφει `2`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([5, 5, 5, 5])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `2`. Κάθε ζεύγος πενταριών γεμίζει ακριβώς μία ομάδα.

Συνέχισε στην εκφώνηση της άσκησης 65, στη σελίδα 239.

Η ΑΣΚΗΣΗ

# 65 Η μικρότερη χωρητικότητα

## Τι θα φτιάξεις

Τα δέματα έχουν θετικά βάρη και πρέπει να μεταφερθούν με τη σειρά του `weights`, χωρίς να χωριστεί δέμα. Κάθε ημέρα φορτώνεις ένα συνεχόμενο επόμενο τμήμα μέχρι τη διαθέσιμη χωρητικότητα. Βρες τη μικρότερη ακέραιη χωρητικότητα που αρκεί για το πολύ `days` ημέρες.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `weights` έχει από 1 έως 100 στοιχεία. Οι ακέραιες τιμές του είναι από 1 έως 1000. Το `days` είναι από 1 έως 100.

**Η κλήση**

```
solve([1, 2, 3, 4, 5], 3)
```

↓ Η τιμή που επιστρέφεται

```
6
```

## Η ιδέα

Για δεδομένη χωρητικότητα μπορείς να μετρήσεις τις απαιτούμενες ημέρες με ένα πέρασμα. Αν μια χωρητικότητα αρκεί, κάθε μεγαλύτερη αρκεί επίσης.

**Στόχος απόδοσης:** `O(n log S)` χρόνος και `O(1)` πρόσθετος χώρος, όπου `S` το συνολικό βάρος.

Η ΜΙΚΡΟΤΕΡΗ ΧΩΡΗΤΙΚΟΤΗΤΑ

# 65 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `weights`, `days`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 3, 4, 5], 3)` | `6` |
| `solve([7], 10)` | `7` |
| `solve([3, 2, 2, 4, 1, 4], 3)` | `6` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/65-shipping-capacity/exercise.rb.

Tests: exercises/65-shipping-capacity/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/65-shipping-capacity/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κάτω όριο είναι το μεγαλύτερο δέμα. Πάνω όριο είναι το συνολικό βάρος, που μεταφέρεται σε μία ημέρα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 419.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 66 Ενημέρωσε μια σύνοψη με δύο κινήσεις

Θα επιστρέψουμε τα αθροίσματα όλων των γειτονικών ζευγών. Μετά το πρώτο, κάθε ζεύγος μοιράζεται ένα στοιχείο με το προηγούμενο.

RUBY

```ruby
def example(values)
  return [] if values.length < 2
  total = values[0] + values[1]
  result = [total]
  (2...values.length).each do |i|
    total += values[i] - values[i - 2]
    result << total
  end
  result
end
```

## Διάβασε τις γραμμές

01 Το στοιχείο δύο θέσεις πίσω είναι αυτό που εγκαταλείπει το ζεύγος.

02 Η ενημέρωση λειτουργεί και για αρνητικούς αριθμούς, επειδή είναι ακριβής αλγεβρική διαφορά.

03 Με λιγότερες από δύο τιμές δεν υπάρχει κανένα πλήρες παράθυρο.

## Ακολούθησε μια κλήση

Η `example([1, 4, 2, 5])` επιστρέφει `[5, 6, 7]`.

Η `example([9])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([-3, 1, 2])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[-2, 3]`. Το δεύτερο ζεύγος δεν περιέχει πια το -3.

Συνέχισε στην εκφώνηση της άσκησης 66, στη σελίδα 242.

Η ΑΣΚΗΣΗ

# 66 Το καλύτερο παράθυρο σταθερού μήκους

## Τι θα φτιάξεις

Ισχύει `1 <= width <= numbers.length`. Βρες το μεγαλύτερο άθροισμα συνεχόμενου τμήματος με ακριβώς `width` στοιχεία. Επιτρέπονται αρνητικές τιμές. Μην ξαναπροσθέτεις όλα τα στοιχεία για κάθε νέο τμήμα.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 1 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `width` είναι από 1 έως 200.

**Η κλήση**

```
solve([1, 4, 2, 10, 2, 3], 3)
```

↓ Η τιμή που επιστρέφεται

```
16
```

## Η ιδέα

Δύο γειτονικά παράθυρα διαφέρουν σε δύο μόνο θέσεις. Η υπόλοιπη δουλειά του προηγούμενου αθροίσματος μπορεί να διατηρηθεί.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΤΟ ΚΑΛΥΤΕΡΟ ΠΑΡΑΘΥΡΟ ΣΤΑΘΕΡΟΥ ΜΗΚΟΥΣ

# 66 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `width`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 4, 2, 10, 2, 3], 3)` | `16` |
| `solve([-5, -2, -4], 2)` | `-6` |
| `solve([7], 1)` | `7` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/66-maximum-fixed-window/exercise.rb.

Tests: exercises/66-maximum-fixed-window/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/66-maximum-fixed-window/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αφού υπολογίσεις το πρώτο παράθυρο, σε κάθε μετακίνηση αφαίρεσε τη θέση `i - width` και πρόσθεσε τη θέση `i`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 421.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 67 Ένα στοιχείο μπαίνει και βγαίνει μία φορά

Θα μετρήσουμε πόσα αρχικά θετικά στοιχεία πρέπει να αφαιρεθούν για να γίνει το συνολικό άθροισμα το πολύ πέντε.

RUBY

```ruby
def example(values)
  total = values.sum
  left = 0
  while total > 5
    total -= values[left]
    left += 1
  end
  left
end
```

## Διάβασε τις γραμμές

01 Η αφαίρεση θετικού στοιχείου μειώνει πάντα το άθροισμα.

02 Ο δείκτης `left` είναι ταυτόχρονα η επόμενη θέση αφαίρεσης και το πλήθος όσων έχουν φύγει.

03 Το κενό τμήμα έχει άθροισμα μηδέν, οπότε η διαδικασία σίγουρα θα σταματήσει.

## Ακολούθησε μια κλήση

Η `example([2, 3, 4])` επιστρέφει `2`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 1, 3])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `0`. Το αρχικό άθροισμα είναι ήδη μέσα στο όριο.

Συνέχισε στην εκφώνηση της άσκησης 67, στη σελίδα 245.

Η ΑΣΚΗΣΗ

# 67 Το συντομότερο αρκετά μεγάλο τμήμα

## Τι θα φτιάξεις

Οι τιμές του `numbers` είναι αυστηρά θετικές. Βρες το μικρότερο μήκος συνεχόμενου τμήματος με άθροισμα τουλάχιστον `target`. Αν δεν υπάρχει, επέστρεψε `0`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από 1 έως 1000. Το `target` είναι από 1 έως 10000.

Η κλήση

```
solve([2, 3, 1, 2, 4, 3], 7)
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Πρόσθεσε στοιχεία από δεξιά ώσπου το παράθυρο να αρκεί. Τότε αφαίρεσε από αριστερά όσο συνεχίζει να αρκεί, καταγράφοντας τα μικρότερα μήκη.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΤΟ ΣΥΝΤΟΜΟΤΕΡΟ ΑΡΚΕΤΑ ΜΕΓΑΛΟ ΤΜΗΜΑ

# 67 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `target`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 3, 1, 2, 4, 3], 7)` | `2` |
| `solve([], 5)` | `0` |
| `solve([1, 1, 1], 5)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/67-shortest-positive-window/exercise.rb.

Tests: exercises/67-shortest-positive-window/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/67-shortest-positive-window/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ο αριστερός δείκτης δεν χρειάζεται ποτέ να επιστρέψει πίσω, επειδή όλα τα στοιχεία είναι θετικά.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 422.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 68 Η τελευταία εμφάνιση αντικαθιστά την προηγούμενη

Θα επιστρέψουμε την απόσταση κάθε χαρακτήρα από την προηγούμενη εμφάνισή του. Για πρώτη εμφάνιση γράφουμε μηδέν.

RUBY

```ruby
def example(text)
  last = {}
  text.chars.each_with_index.map do |char, i|
    distance = last.key?(char) ? i - last[char] : 0
    last[char] = i
    distance
  end
end
```

## Διάβασε τις γραμμές

01 Χωρίς block, η `each_with_index` δίνει Enumerator που μπορεί να δεχτεί `map`.

02 Η απόσταση υπολογίζεται πριν αντικατασταθεί η προηγούμενη θέση.

03 Εδώ θέλουμε την τελευταία εμφάνιση, σε αντίθεση με την άσκηση 59 που κρατούσε την πρώτη.

## Ακολούθησε μια κλήση

Η `example("abac")` επιστρέφει `[0, 0, 2, 0]`.

Η `example("")` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example("aaa")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 1, 1]`. Η τρίτη εμφάνιση συγκρίνεται με τη δεύτερη, όχι με την πρώτη.

Συνέχισε στην εκφώνηση της άσκησης 68, στη σελίδα 248.

Η ΑΣΚΗΣΗ

# 68 Κείμενο χωρίς επαναλαμβανόμενο χαρακτήρα

## Τι θα φτιάξεις

Βρες το μήκος του μεγαλύτερου συνεχόμενου τμήματος του `text` χωρίς επανάληψη χαρακτήρα. Οι χαρακτήρες και τα μήκη ακολουθούν τη `chars` της Ruby. Κενά και διαφορές κεφαλαίων μετρούν κανονικά.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `text` έχει από 0 έως 200 χαρακτήρες.

Η κλήση

```
solve("abcabcbb")
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Η ιδέα

Η τελευταία θέση ενός χαρακτήρα δείχνει πόσο πρέπει να μετακινηθεί το αριστερό άκρο όταν τον ξανασυναντάς.

**Στόχος απόδοσης:** `O(n)` αναμενόμενος χρόνος και `O(n)` πρόσθετος χώρος.

ΚΕΙΜΕΝΟ ΧΩΡΙΣ ΕΠΑΝΑΛΑΜΒΑΝΟΜΕΝΟ ΧΑΡΑΚΤΗΡΑ

# 68 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `text`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("abcabcbb")` | `3` |
| `solve("")` | `0` |
| `solve("bbbbb")` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/68-longest-unique-substring/exercise.rb.

Tests: exercises/68-longest-unique-substring/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/68-longest-unique-substring/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μια προηγούμενη εμφάνιση μπορεί να βρίσκεται ήδη έξω από το παράθυρο. Το αριστερό άκρο δεν πρέπει να κινηθεί προς τα πίσω.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 423.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 69 Ένα μηδενικό πλήθος πρέπει να φύγει

Θα προσθέσουμε όλες τις τιμές και μετά θα αφαιρέσουμε μία εμφάνιση της πρώτης. Η επιστροφή δίνει πόσες διαφορετικές τιμές απέμειναν.

RUBY

```ruby
def example(values)
  return 0 if values.empty?
  counts = Hash.new(0)
  values.each { |value| counts[value] += 1 }
  first = values[0]
  counts[first] -= 1
  counts.delete(first) if counts[first] == 0
  counts.length
end
```

## Διάβασε τις γραμμές

01 Η `delete` αφαιρεί το ζεύγος κλειδιού και τιμής από το Hash.

02 Αν υπάρχουν άλλες εμφανίσεις, το κλειδί πρέπει να παραμείνει.

03 Ένα αποθηκευμένο μηδέν εξακολουθεί να μετρά ως κλειδί μέχρι να διαγραφεί.

## Ακολούθησε μια κλήση

Η `example([1, 2, 2])` επιστρέφει `1`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 1, 2])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι 2. Η αφαίρεση ενός 1 αφήνει ακόμη ένα 1.

Συνέχισε στην εκφώνηση της άσκησης 69, στη σελίδα 251.

Η ΑΣΚΗΣΗ

# 69 Παράθυρο με λίγες διαφορετικές τιμές

## Τι θα φτιάξεις

Βρες το μεγαλύτερο μήκος συνεχόμενου τμήματος που περιέχει το πολύ `limit` διαφορετικές τιμές. Τα διπλότυπα επιτρέπονται. Για μηδενικό όριο η απάντηση είναι `0`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000. Το `limit` είναι από 0 έως 200.

**Η κλήση**

```
solve([1, 2, 1, 2, 3], 2)
```

↓ Η τιμή που επιστρέφεται

```
4
```

## Η ιδέα

Το Hash πρέπει να περιγράφει μόνο το τρέχον παράθυρο. Όταν προστεθεί παραπάνω διαφορετική τιμή, αφαίρεσε από αριστερά ώσπου να αποκατασταθεί το όριο.

**Στόχος απόδοσης:** `O(n)` αναμενόμενος χρόνος και `O(min(n, limit + 1))` πρόσθετος χώρος.

ΠΑΡΑΘΥΡΟ ΜΕ ΛΙΓΕΣ ΔΙΑΦΟΡΕΤΙΚΕΣ ΤΙΜΕΣ

# 69 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`, `limit`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 1, 2, 3], 2)` | `4` |
| `solve([], 2)` | `0` |
| `solve([1, 1], 0)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/69-at-most-k-distinct/exercise.rb.

Tests: exercises/69-at-most-k-distinct/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/69-at-most-k-distinct/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Διέγραψε το κλειδί όταν ο μετρητής του γίνει μηδέν. Τότε το μήκος του Hash είναι το πλήθος διαφορετικών τιμών.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 424.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 70 Ένα μικρό αλφάβητο χωρά σε πίνακα

Το παράδειγμα δέχεται μόνο τα γράμματα a, b και c. Θα επιστρέψουμε τρεις μετρητές, έναν για κάθε γράμμα.

RUBY

```ruby
def example(text)
  counts = Array.new(3, 0)
  text.each_byte do |byte|
    counts[byte - 97] += 1
  end
  counts
end
```

## Διάβασε τις γραμμές

01 Οι ASCII κώδικες των a, b και c είναι διαδοχικοί, ξεκινώντας από το 97.

02 Η αφαίρεση του 97 μετατρέπει τον χαρακτήρα σε δείκτη πίνακα.

03 Η τεχνική αυτή εξαρτάται από το ρητά περιορισμένο αλφάβητο. Δεν εφαρμόζεται έτσι στα ελληνικά.

## Ακολούθησε μια κλήση

Η `example("cabba")` επιστρέφει `[2, 2, 1]`.

Η `example("")` επιστρέφει `[0, 0, 0]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example("ccc")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 0, 3]`. Μόνο ο τρίτος μετρητής αυξάνεται.

Συνέχισε στην εκφώνηση της άσκησης 70, στη σελίδα 254.

Η ΑΣΚΗΣΗ

# 70 Πόσα αναγράμματα κρύβονται στο κείμενο;

## Τι θα φτιάξεις

Τα `text` και `pattern` περιέχουν μόνο πεζά λατινικά γράμματα και το `pattern` δεν είναι κενό. Μέτρησε τα συνεχόμενα τμήματα του `text` που είναι αναγράμματα του `pattern`. Τα επικαλυπτόμενα τμήματα μετρούν χωριστά.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `text` έχει από 0 έως 500 χαρακτήρες. Το `pattern` έχει από 1 έως 100 χαρακτήρες.

**Η κλήση**

```
solve("cbaebabacd", "abc")
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Κάθε υποψήφιο παράθυρο έχει μήκος ίσο με το pattern. Ενημέρωσε τις 26 συχνότητές του όταν μπαίνει και βγαίνει ένας χαρακτήρας.

**Στόχος απόδοσης:** `O(n + m)` χρόνος για σταθερό αλφάβητο 26 γραμμάτων και `O(1)` πρόσθετος χώρος.

ΠΟΣΑ ΑΝΑΓΡΑΜΜΑΤΑ ΚΡΥΒΟΝΤΑΙ ΣΤΟ ΚΕΙΜΕΝΟ;

# 70 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `text`, `pattern`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("cbaebabacd", "abc")` | `2` |
| `solve("", "a")` | `0` |
| `solve("ab", "abc")` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/70-anagram-windows/exercise.rb.

Tests: exercises/70-anagram-windows/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/70-anagram-windows/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Για ASCII πεζά γράμματα, ο κώδικας του χαρακτήρα μείον 97 δίνει θέση από 0 έως 25.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

είσοδος αποτέλεσμα

Η λύση βρίσκεται στη σελίδα 425.

ΑΣΚΗΣΕΙΣ 71-80

# Στοίβες, διαστήματα και πίνακες

Εκκρεμότητες που λύνονται με συγκεκριμένη σειρά, γεγονότα στον χρόνο και δισδιάστατες δομές.



Δώσε χρόνο στη σκέψη, στις οριακές περιπτώσεις και στην εξήγηση του κόστους. Οι λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |
| | |

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 71 Το τελευταίο που μπαίνει βγαίνει πρώτο

Θα αδειάσουμε μια στοίβα αριθμών. Αντιγράφουμε πρώτα την είσοδο, ώστε οι αφαιρέσεις να μην αλλάξουν τον πίνακα του καλούντος.

RUBY

```ruby
def example(values)
  stack = values.dup
  result = []
  result << stack.pop until stack.empty?
  result
end
```

## Διάβασε τις γραμμές

01 Η `dup` δημιουργεί νέο εξωτερικό πίνακα. Για επίπεδο πίνακα ακεραίων αυτό αρκεί εδώ.

02 Η `pop` αφαιρεί και επιστρέφει το τελευταίο στοιχείο.

03 Το `until` επαναλαμβάνει όσο η συνθήκη παραμένει ψευδής. Σταματά πριν γίνει αφαίρεση από κενή στοίβα.

## Ακολούθησε μια κλήση

Η `example([1, 2, 3])` επιστρέφει `[3, 2, 1]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([4, 4])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[4, 4]`. Οι ίδιες τιμές παραμένουν δύο χωριστά στοιχεία.

Συνέχισε στην εκφώνηση της άσκησης 71, στη σελίδα 258.

Η ΑΣΚΗΣΗ

# 71 Σωστά ζευγάρια παρενθέσεων

## Τι θα φτιάξεις

Το `text` περιέχει μόνο `(`, `)`, `[`, `]`, `{` και `}`. Επέστρεψε αν κάθε άνοιγμα κλείνει με τον ίδιο τύπο και με σωστή εμφώλευση. Το κενό κείμενο είναι έγκυρο.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `text` έχει από 0 έως 500 χαρακτήρες.

Η κλήση

```
solve("([]){}")
```

↓ Η τιμή που επιστρέφεται

```
true
```

## Η ιδέα

Το επόμενο κλείσιμο πρέπει να ταιριάζει με το πιο πρόσφατο άνοιγμα που δεν έχει ακόμη κλείσει. Αυτή είναι ακριβώς η σειρά μιας στοίβας.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` πρόσθετος χώρος.

ΣΩΣΤΑ ΖΕΥΓΑΡΙΑ ΠΑΡΕΝΘΕΣΕΩΝ

# 71 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `text`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("([]){}")` | `true` |
| `solve("")` | `true` |
| `solve("([)]")` | `false` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/71-balanced-brackets/exercise.rb.

Tests: exercises/71-balanced-brackets/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/71-balanced-brackets/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Χρησιμοποίησε πίνακα: `<<` για ώθηση και `pop` για αφαίρεση από την κορυφή.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 426.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 72 Το υπόλοιπο κάθε προθέματος έχει σημασία

Θα επιστρέψουμε την ισορροπία μετά από κάθε παρένθεση: συν ένα για άνοιγμα και μείον ένα για κλείσιμο.

RUBY

```ruby
def example(text)
  balance = 0
  text.each_char.map do |char|
    balance += char == "(" ? 1 : -1
  end
end
```

## Διάβασε τις γραμμές

01 Ένα αρνητικό πρόθεμα σημαίνει ότι κάποιο κλείσιμο εμφανίστηκε πολύ νωρίς.

02 Η τελική ισορροπία μηδέν δεν εγγυάται ότι όλα τα προηγούμενα προθέματα ήταν νόμιμα.

03 Το παράδειγμα καταγράφει το πρόβλημα. Η άσκηση ζητά να υπολογίσεις πόσες προσθήκες το διορθώνουν.

## Ακολούθησε μια κλήση

Η `example("(()")` επιστρέφει `[1, 2, 1]`.

Η `example("")` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(")(")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[-1, 0]`. Η τελική ισορροπία κρύβει το αρνητικό πρώτο πρόθεμα.

Συνέχισε στην εκφώνηση της άσκησης 72, στη σελίδα 261.

Η ΑΣΚΗΣΗ

# 72 Πόσες παρενθέσεις λείπουν;

## Τι θα φτιάξεις

Το `text` περιέχει μόνο `(` και `)`. Βρες το μικρότερο πλήθος παρενθέσεων που πρέπει να προστεθούν οπουδήποτε ώστε να γίνει σωστά ισορροπημένο. Δεν επιτρέπεται αφαίρεση ή μετακίνηση υπαρχόντων χαρακτήρων.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `text` έχει από 0 έως 500 χαρακτήρες.

Η κλήση

```
solve("())")
```

↓ Η τιμή που επιστρέφεται

```
1
```

## Η ιδέα

Ένα κλείσιμο χωρίς διαθέσιμο άνοιγμα απαιτεί ένα νέο άνοιγμα πριν από αυτό. Όσα ανοίγματα περισσέψουν στο τέλος απαιτούν ισάριθμα κλεισίματα.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

ΠΟΣΕΣ ΠΑΡΕΝΘΕΣΕΙΣ ΛΕΙΠΟΥΝ;

# 72 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `text`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("())")` | `1` |
| `solve("")` | `0` |
| `solve("(((")` | `3` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/72-minimum-parenthesis-additions/exercise.rb.

Tests: exercises/72-minimum-parenthesis-additions/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/72-minimum-parenthesis-additions/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κράτα χωριστά τα ανοιχτά ζεύγη και τις προσθήκες που έχουν ήδη γίνει αναπόφευκτες.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 427.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 73 Επίλεξε συμπεριφορά με case

Θα εφαρμόσουμε έναν ονομαστικό κανόνα σε δύο σταθερούς αριθμούς. Η `case` είναι χρήσιμη όταν συγκρίνουμε την ίδια τιμή με πολλές συγκεκριμένες επιλογές.

RUBY

```ruby
def example(operation)
  case operation
  when "add"
    4 + 2
  when "subtract"
    4 - 2
  else
    0
  end
end
```

## Διάβασε τις γραμμές

01 Κάθε `when` περιγράφει μία αναγνωρισμένη τιμή της εισόδου.

02 Η τελευταία έκφραση του επιλεγμένου κλάδου γίνεται η τιμή ολόκληρης της `case`.

03 Το `else` καλύπτει όσα δεν αναγνωρίστηκαν. Η άσκηση δίνει ήδη έγκυρους τελεστές και αριθμούς.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 73 Από τον κώδικα στην τιμή

**Η κλήση**

```
example("add")
```

Η τιμή που επιστρέφεται

```
6
```

## Ακολούθησε μια κλήση

Η `example("add")` επιστρέφει `6`.

Η `example("subtract")` επιστρέφει `2`.

## Στάσου λίγο

Τι θα επιστρέψει η `example("unknown")`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `0`. Εκτελείται μόνο ο τελικός κλάδος.

Συνέχισε στην εκφώνηση της άσκησης 73, στη σελίδα 265.

Η ΑΣΚΗΣΗ

# 73 Υπολόγισε μια έκφραση με στοίβα

## Τι θα φτιάξεις

Το `tokens` είναι έγκυρη έκφραση αντίστροφης πολωνικής γραφής: ακέραιοι ως κείμενα και τελεστές `+`, `-`, `*`, `/`. Κάθε τελεστής παίρνει τους δύο τελευταίους διαθέσιμους αριθμούς. Η διαίρεση είναι ακέραιη προς τα κάτω, όπως η Ruby `div`: `-3 / 2` δίνει `-2`. Δεν υπάρχει διαίρεση με μηδέν. Οι αριθμοί εισόδου είναι από -1000 έως 1000.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `tokens` έχει από 1 έως 100 στοιχεία.

Η κλήση

```
solve(["2", "3", "+", "4", "*"])
```

↓ Η τιμή που επιστρέφεται

```
20
```

## Η ιδέα

Κάθε αριθμός περιμένει στη στοίβα μέχρι να εμφανιστεί ο τελεστής που τον χρησιμοποιεί. Το αποτέλεσμα της πράξης γίνεται νέα διαθέσιμη τιμή.

**Στόχος απόδοσης:** `O(n)` πράξεις στοίβας και `O(n)` χώρος. Το κόστος αριθμητικής μεγάλων ακεραίων εξαρτάται από τα ψηφία τους.

ΥΠΟΛΟΓΙΣΕ ΜΙΑ ΕΚΦΡΑΣΗ ΜΕ ΣΤΟΙΒΑ

# 73 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `tokens`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(["2", "3", "+", "4", "*"])` | `20` |
| `solve(["7"])` | `7` |
| `solve(["5", "2", "-"])` | `3` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/73-postfix-evaluator/exercise.rb.

Tests: exercises/73-postfix-evaluator/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/73-postfix-evaluator/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Πρώτα βγαίνει ο δεξιός τελεστέος και μετά ο αριστερός. Αυτό είναι κρίσιμο στην αφαίρεση και στη διαίρεση.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 428.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 74 Αφαίρεσε υποψήφιες τιμές που δεν βοηθούν

Θα διατηρήσουμε μια μη φθίνουσα στοίβα καθώς διαβάζουμε τιμές. Μια μικρότερη νέα τιμή αφαιρεί τις μεγαλύτερες από την κορυφή.

RUBY

```ruby
def example(values)
  stack = []
  values.each do |value|
    stack.pop while !stack.empty? && stack[-1] > value
    stack << value
  end
  stack
end
```

## Διάβασε τις γραμμές

01 Ο εσωτερικός βρόχος συγκρίνει πάντα με την τρέχουσα κορυφή μετά από κάθε αφαίρεση.

02 Η ισότητα διατηρείται εδώ, επειδή ο έλεγχος είναι αυστηρός.

03 Παρότι υπάρχουν δύο βρόχοι, κάθε εισαγωγή επιτρέπει το πολύ μία μελλοντική αφαίρεση.

## Ακολούθησε μια κλήση

Η `example([3, 1, 2])` επιστρέφει `[1, 2]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([2, 2, 1])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[1]`. Το 1 αφαιρεί και τα δύο μεγαλύτερα στοιχεία.

Συνέχισε στην εκφώνηση της άσκησης 74, στη σελίδα 268.

Η ΑΣΚΗΣΗ

# 74 Η επόμενη αυστηρά μεγαλύτερη τιμή

## Τι θα φτιάξεις

Για κάθε θέση βρες την πρώτη τιμή στα δεξιά που είναι αυστηρά μεγαλύτερη από την τρέχουσα. Αν δεν υπάρχει, γράψε `-1`. Επέστρεψε πίνακα ίδιου μήκους. Το `-1` μπορεί επίσης να είναι πραγματική τιμή της εισόδου.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

**Η κλήση**

```
solve([2, 1, 2, 4, 3])
```

↓ Η τιμή που επιστρέφεται

```
[4, 2, 4, -1, -1]
```

## Η ιδέα

Από δεξιά προς τα αριστερά, ορισμένες τιμές παύουν να είναι χρήσιμες υποψήφιες όταν συναντήσουν μια πιο κοντινή, τουλάχιστον ίση τιμή.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` χώρος.

Η ΕΠΟΜΕΝΗ ΑΥΣΤΗΡΑ ΜΕΓΑΛΥΤΕΡΗ ΤΙΜΗ

# 74 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 1, 2, 4, 3])` | `[4, 2, 4, -1, -1]` |
| `solve([])` | `[]` |
| `solve([5])` | `[-1]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/74-next-greater-value/exercise.rb.

Tests: exercises/74-next-greater-value/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/74-next-greater-value/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Πριν απαντήσεις για μια θέση, αφαίρεσε από την κορυφή όσα είναι μικρότερα ή ίσα με αυτή.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 430.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 75 Αποθήκευσε θέση αντί για τιμή

Θα κρατήσουμε τους δείκτες των διαδοχικών νέων μεγίστων ενός πίνακα. Από κάθε δείκτη μπορούμε αργότερα να ξαναδιαβάσουμε την τιμή.

RUBY

```ruby
def example(values)
  records = []
  values.each_with_index do |value, i|
    if records.empty? || value > values[records[-1]]
      records << i
    end
  end
  records
end
```

## Διάβασε τις γραμμές

01 Το `records[-1]` είναι δείκτης. Χρειάζεται δεύτερη πρόσβαση στο `values` για την αντίστοιχη τιμή.

02 Ο πρώτος δείκτης εισάγεται χωρίς σύγκριση, επειδή δεν υπάρχει προηγούμενο μέγιστο.

03 Μια ίση τιμή δεν δημιουργεί νέο αυστηρό ρεκόρ.

## Ακολούθησε μια κλήση

Η `example([2, 1, 3, 3, 5])` επιστρέφει `[0, 2, 4]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([4, 4])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0]`. Το δεύτερο 4 δεν ξεπερνά το πρώτο.

Συνέχισε στην εκφώνηση της άσκησης 75, στη σελίδα 271.

Η ΑΣΚΗΣΗ

# 75 Πόσες ημέρες μέχρι να ζεστάνει;

## Τι θα φτιάξεις

Για κάθε ημέρα επέστρεψε πόσες ημέρες πρέπει να περάσουν μέχρι την πρώτη αυστηρά υψηλότερη θερμοκρασία. Αν δεν υπάρξει θερμότερη ημέρα, γράψε `0`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `temperatures` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από -100 έως 100.

Η κλήση

```
solve([73, 74, 75, 71, 69, 72, 76, 73])
```

↓ Η τιμή που επιστρέφεται

```
[1, 1, 4, 2, 1, 1, 0, 0]
```

## Η ιδέα

Κράτα τους δείκτες ημερών που περιμένουν ακόμη απάντηση. Μια νέα υψηλότερη θερμοκρασία μπορεί να λύσει πολλές εκκρεμότητες μαζί.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` χώρος.

ΠΟΣΕΣ ΗΜΕΡΕΣ ΜΕΧΡΙ ΝΑ ΖΕΣΤΑΝΕΙ;

# 75 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `temperatures`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([73, 74, 75, 71, 69, 72, 76, 73])` | `[1, 1, 4, 2, 1, 1, 0, 0]` |
| `solve([])` | `[]` |
| `solve([20])` | `[0]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/75-warmer-day-distance/exercise.rb.

Tests: exercises/75-warmer-day-distance/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/75-warmer-day-distance/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αποθήκευσε δείκτες στη στοίβα, ώστε όταν έρθει η απάντηση να υπολογίσεις απόσταση και να γράψεις στη σωστή θέση.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 431.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 76 Για σταθερό διάστημα το ύψος είναι το ελάχιστο

Θα υπολογίσουμε το μεγαλύτερο ορθογώνιο που απλώνεται σε όλο το δοσμένο διάστημα στηλών. Η άσκηση θα επιλέγει και το διάστημα.

RUBY

```ruby
def example(heights)
  return 0 if heights.empty?
  heights.min * heights.length
end
```

## Διάβασε τις γραμμές

01 Το ορθογώνιο πρέπει να χωρά κάτω από κάθε στήλη, άρα περιορίζεται από τη χαμηλότερη.

02 Το πλάτος είναι το πλήθος στηλών, επειδή κάθε στήλη έχει πλάτος ένα.

03 Ένα μηδενικό ύψος μηδενίζει μόνο τα ορθογώνια που το διασχίζουν. Μικρότερα γειτονικά διαστήματα μπορεί να είναι καλύτερα.

## Ακολούθησε μια κλήση

Η `example([3, 2, 4])` επιστρέφει `6`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([5, 0, 5])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `0`. Το ορθογώνιο του παραδείγματος υποχρεώνεται να περάσει πάνω από το μηδέν.

Συνέχισε στην εκφώνηση της άσκησης 76, στη σελίδα 274.

Η ΑΣΚΗΣΗ

# 76 Το μεγαλύτερο ορθογώνιο στο ιστόγραμμα

## Τι θα φτιάξεις

Κάθε στοιχείο του `heights` είναι ύψος στήλης πλάτους ένα. Βρες το μεγαλύτερο εμβαδόν ορθογωνίου που χωρά κάτω από συνεχόμενες στήλες. Τα ύψη είναι μη αρνητικά. Για κενό ιστόγραμμα επέστρεψε `0`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `heights` έχει από 0 έως 200 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 1000.

**Η κλήση**

```
solve([2, 1, 5, 6, 2, 3])
```

↓ Η τιμή που επιστρέφεται

```
10
```

## Η ιδέα

Όταν εμφανιστεί χαμηλότερη στήλη, ορισμένα προηγούμενα ύψη δεν μπορούν να επεκταθούν άλλο δεξιά. Η στοίβα διατηρεί τα αριστερά όρια που χρειαζόμαστε.

**Στόχος απόδοσης:** `O(n)` χρόνος και `O(n)` πρόσθετος χώρος.

ΤΟ ΜΕΓΑΛΥΤΕΡΟ ΟΡΘΟΓΩΝΙΟ ΣΤΟ ΙΣΤΟΓΡΑΜΜΑ

# 76 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `heights`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 1, 5, 6, 2, 3])` | `10` |
| `solve([])` | `0` |
| `solve([7])` | `7` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/76-histogram-rectangle/exercise.rb.

Tests: exercises/76-histogram-rectangle/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/76-histogram-rectangle/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Πρόσθεσε νοητά μία στήλη ύψους μηδέν στο τέλος για να ολοκληρωθούν οι εκκρεμείς υπολογισμοί.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 432.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 77 Αντέγραψε τις γραμμές που θα αλλάξεις

Θα αυξήσουμε κατά ένα το δεύτερο στοιχείο κάθε ζεύγους. Η δημιουργία νέων ζευγών προστατεύει και τις εσωτερικές γραμμές της εισόδου.

RUBY

```ruby
def example(pairs)
  pairs.map do |first, second|
    [first, second + 1]
  end
end
```

## Διάβασε τις γραμμές

01 Το block αποσυσκευάζει κάθε ζεύγος σε δύο ονόματα.

02 Το `[first, second + 1]` είναι καινούργιος εσωτερικός πίνακας.

03 Μόνη της η `pairs.dup` θα αντέγραφε μόνο τον εξωτερικό πίνακα και θα μοιραζόταν τις παλιές γραμμές.

## Ακολούθησε μια κλήση

Η `example([[1, 3], [5, 7]])` επιστρέφει `[[1, 4], [5, 8]]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([[-2, 0]])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[[-2, 1]]`. Η αλλαγή αφορά τη νέα γραμμή, όχι το αρχικό ζεύγος.

Συνέχισε στην εκφώνηση της άσκησης 77, στη σελίδα 277.

Η ΑΣΚΗΣΗ

# 77 Ένωση επικαλυπτόμενων διαστημάτων

## Τι θα φτιάξεις

Κάθε γραμμή είναι κλειστό διάστημα `[start, finish]` με `start <= finish`. Ένωσε όσα επικαλύπτονται ή μοιράζονται άκρο. Επέστρεψε τα τελικά ξένα διαστήματα σε αύξουσα σειρά αρχής. Διαστήματα που απλώς απέχουν μία ακέραιη μονάδα δεν ενώνονται.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `intervals` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 2 έως 2 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

Η κλήση

```
solve([[1, 3], [2, 5], [8, 9]])
```

↓ Η τιμή που επιστρέφεται

```
[[1, 5], [8, 9]]
```

## Η ιδέα

Μετά την ταξινόμηση κατά αρχή, το επόμενο διάστημα μπορεί να αλληλεπιδράσει μόνο με το τελευταίο ήδη κατασκευασμένο διάστημα.

**Στόχος απόδοσης:** `O(n log n)` χρόνος και `O(n)` χώρος μαζί με την έξοδο.

ΕΝΩΣΗ ΕΠΙΚΑΛΥΠΤΟΜΕΝΩΝ ΔΙΑΣΤΗΜΑΤΩΝ

# 77 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `intervals`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[1, 3], [2, 5], [8, 9]])` | `[[1, 5], [8, 9]]` |
| `solve([])` | `[]` |
| `solve([[1, 2], [2, 3]])` | `[[1, 3]]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/77-merge-intervals/exercise.rb.

Tests: exercises/77-merge-intervals/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/77-merge-intervals/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αν η νέα αρχή είναι το πολύ ίση με το τελευταίο τέλος, επέκτεινε μόνο το τέλος.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 433.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 78 Η σειρά ισόχρονων γεγονότων αλλάζει το αποτέλεσμα

Θα ταξινομήσουμε ζεύγη `[time, change]`. Ο δεύτερος αριθμός λύνει την ισοβαθμία όταν ο χρόνος είναι ίδιος.

```ruby
def example(events)
  events.sort
end
```

## Διάβασε τις γραμμές

01 Οι πίνακες συγκρίνονται λεξικογραφικά: πρώτο στοιχείο, μετά δεύτερο αν χρειαστεί.

02 Σε ίδιο χρόνο, μια μείωση `-1` προηγείται μιας αύξησης `1`.

03 Η `sort` δημιουργεί νέο εξωτερικό πίνακα. Εδώ δεν μεταβάλλουμε τις εσωτερικές γραμμές.

## Ακολούθησε μια κλήση

Η `example([[2, 1], [2, -1], [1, 1]])` επιστρέφει `[[1, 1], [2, -1], [2, 1]]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([[0, 1], [0, -1]])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[[0, -1], [0, 1]]`. Η ελευθέρωση θα προηγηθεί της νέας χρήσης.

Συνέχισε στην εκφώνηση της άσκησης 78, στη σελίδα 280.

Η ΑΣΚΗΣΗ

# 78 Πόσες αίθουσες χρειάζονται;

## Τι θα φτιάξεις

Κάθε γραμμή `[start, finish]` είναι συνάντηση με `start < finish`. Βρες τον ελάχιστο αριθμό αιθουσών ώστε να γίνουν όλες. Μια αίθουσα που ελευθερώνεται τη στιγμή `t` μπορεί να χρησιμοποιηθεί αμέσως από συνάντηση που αρχίζει στο `t`. Για κενή είσοδο επέστρεψε `0`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `meetings` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 2 έως 2 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

**Η κλήση**

```
solve([[0, 30], [5, 10], [15, 20]])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Μετέτρεψε τις αρχές και τα τέλη σε γεγονότα αύξησης και μείωσης του πλήθους ενεργών συναντήσεων. Το μέγιστο πλήθος είναι η ζητούμενη χωρητικότητα.

**Στόχος απόδοσης:** `O(n log n)` χρόνος και `O(n)` πρόσθετος χώρος.

ΠΟΣΕΣ ΑΙΘΟΥΣΕΣ ΧΡΕΙΑΖΟΝΤΑΙ;

# 78 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `meetings`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[0, 30], [5, 10], [15, 20]])` | `2` |
| `solve([])` | `0` |
| `solve([[1, 2], [2, 3]])` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/78-meeting-rooms/exercise.rb.

Tests: exercises/78-meeting-rooms/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/78-meeting-rooms/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Σε ίδιο χρόνο, επεξεργάσου το τέλος πριν από την αρχή.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 434.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 79 Κάθε γραμμή δημιουργείται ξεχωριστά

Θα φτιάξουμε πίνακα όπου κάθε τιμή είναι το άθροισμα του δείκτη γραμμής και του δείκτη στήλης.

RUBY

```ruby
def example(rows, columns)
  Array.new(rows) do |row|
    Array.new(columns) do |column|
      row + column
    end
  end
end
```

## Διάβασε τις γραμμές

01 Η `Array.new` με block εκτελεί το block μία φορά για κάθε θέση.

02 Το εξωτερικό block δημιουργεί νέα γραμμή. Το εσωτερικό υπολογίζει τις τιμές της.

03 Οι δύο δείκτες ξεκινούν από το μηδέν και έχουν ανεξάρτητα όρια.

## Ακολούθησε μια κλήση

Η `example(2, 3)` επιστρέφει `[[0, 1, 2], [1, 2, 3]]`.

Η `example(0, 3)` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3, 1)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[[0], [1], [2]]`. Με μία στήλη ο δείκτης στήλης είναι πάντα μηδέν.

Συνέχισε στην εκφώνηση της άσκησης 79, στη σελίδα 283.

Η ΑΣΚΗΣΗ

# 79 Οι γραμμές γίνονται στήλες

## Τι θα φτιάξεις

Ο πίνακας είναι ορθογώνιος. Επέστρεψε την ανάστροφή του ως προς την κύρια διαγώνιο: η τιμή στη γραμμή `r`, στήλη `c` πηγαίνει στη γραμμή `c`, στήλη `r`. Μην χρησιμοποιήσεις την έτοιμη `transpose`. Για μηδενικές γραμμές ή στήλες επέστρεψε `[]`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `matrix` έχει από 0 έως 30 γραμμές. Κάθε γραμμή έχει από 0 έως 30 στοιχεία. Οι γραμμές έχουν ίδιο μήκος. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

**Η κλήση**

```
solve([[1, 2, 3], [4, 5, 6]])
```

↓ Η τιμή που επιστρέφεται

```
[[1, 4], [2, 5], [3, 6]]
```

## Η ιδέα

Η έξοδος έχει τόσες γραμμές όσες ήταν οι στήλες της εισόδου. Κάθε νέα γραμμή συλλέγει μία συγκεκριμένη στήλη.

**Στόχος απόδοσης:** `O(r c)` χρόνος και χώρος εξόδου για `r` γραμμές και `c` στήλες.

ΟΙ ΓΡΑΜΜΕΣ ΓΙΝΟΝΤΑΙ ΣΤΗΛΕΣ

# 79 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `matrix`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[1, 2, 3], [4, 5, 6]])` | `[[1, 4], [2, 5], [3, 6]]` |
| `solve([])` | `[]` |
| `solve([[], []])` | `[]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/79-matrix-transpose/exercise.rb.

Tests: exercises/79-matrix-transpose/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/79-matrix-transpose/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Δημιούργησε κάθε εσωτερικό πίνακα μέσα σε block της `Array.new`.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 435.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 80 Δύο δείκτες περιγράφουν κάθε θέση

Θα παράγουμε τις συντεταγμένες μιας ορθογώνιας περιοχής σε συνηθισμένη σειρά γραμμών. Η άσκηση αλλάζει τη σειρά επίσκεψης, όχι το σύνολο θέσεων.

RUBY

```ruby
def example(rows, columns)
  (0...rows).flat_map do |row|
    (0...columns).map { |column| [row, column] }
  end
end
```

## Διάβασε τις γραμμές

01 Η `map` της εσωτερικής περιοχής δίνει τα ζεύγη μιας γραμμής.

02 Η `flat_map` ενώνει ένα επίπεδο των αποτελεσμάτων και αφήνει κάθε ζεύγος ακέραιο ως πίνακα δύο στοιχείων.

03 Με μηδενικές γραμμές ή στήλες δεν υπάρχει καμία θέση για επίσκεψη.

## Ακολούθησε μια κλήση

Η `example(2, 2)` επιστρέφει `[[0, 0], [0, 1], [1, 0], [1, 1]]`.

Η `example(0, 3)` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(1, 3)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[[0, 0], [0, 1], [0, 2]]`. Ο δείκτης γραμμής παραμένει μηδέν σε όλα τα ζεύγη.

Συνέχισε στην εκφώνηση της άσκησης 80, στη σελίδα 286.

Η ΑΣΚΗΣΗ

# 80 Διάβασε τον πίνακα σπειροειδώς

## Τι θα φτιάξεις

Διάβασε τον ορθογώνιο πίνακα δεξιόστροφα σε σπείρα, ξεκινώντας από πάνω αριστερά: πάνω γραμμή, δεξιά στήλη, κάτω γραμμή, αριστερή στήλη και μετά προς το εσωτερικό. Επέστρεψε επίπεδο πίνακα με κάθε στοιχείο ακριβώς μία φορά.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `matrix` έχει από 0 έως 30 γραμμές. Κάθε γραμμή έχει από 0 έως 30 στοιχεία. Οι γραμμές έχουν ίδιο μήκος. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

Η κλήση

```
solve([[1, 2], [3, 4]])
```

↓ Η τιμή που επιστρέφεται

```
[1, 2, 4, 3]
```

## Η ιδέα

Περιέγραψε την αδιάβαστη περιοχή με πάνω, κάτω, αριστερό και δεξί όριο. Κάθε ολοκληρωμένη πλευρά αφαιρεί ένα από αυτά τα όρια.

**Στόχος απόδοσης:** `O(r c)` χρόνος, `O(r c)` χώρος εξόδου και `O(1)` πρόσθετος χώρος.

ΔΙΑΒΑΣΕ ΤΟΝ ΠΙΝΑΚΑ ΣΠΕΙΡΟΕΙΔΩΣ

# 80 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `matrix`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[1, 2], [3, 4]])` | `[1, 2, 4, 3]` |
| `solve([])` | `[]` |
| `solve([[], []])` | `[]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/80-spiral-matrix/exercise.rb.

Tests: exercises/80-spiral-matrix/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/80-spiral-matrix/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Πριν από την τρίτη και την τέταρτη πλευρά, έλεγξε ξανά αν απομένει περιοχή.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 436.

ΑΣΚΗΣΕΙΣ 81-90

# Γράφοι και αναδρομική αναζήτηση

Συνδεσιμότητα, αποστάσεις, εξαρτήσεις και δέντρα επιλογών με αναίρεση και κλάδεμα.



Δώσε χρόνο στη σκέψη, στις οριακές περιπτώσεις και στην εξήγηση του κόστους. Οι λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |
| | |

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 81 Οι γείτονες χρειάζονται έλεγχο ορίων

Θα επιστρέψουμε τους ορθογώνιους γείτονες μιας θέσης σε σταθερό πλέγμα τριών γραμμών και τριών στηλών.

RUBY

```ruby
def example(row, column)
  candidates = [
    [row-1, column], [row+1, column],
    [row, column-1], [row, column+1]
  ]
  candidates.select do |r, c|
    r.between?(0, 2) && c.between?(0, 2)
  end
end
```

## Διάβασε τις γραμμές

01 Κάθε γείτονας αλλάζει ακριβώς έναν από τους δύο δείκτες κατά ένα.

02 Η `between?` περιλαμβάνει και τα δύο όρια.

03 Οι θέσεις γωνίας έχουν δύο έγκυρους γείτονες, ενώ οι εσωτερικές τέσσερις.

## Ακολούθησε μια κλήση

Η `example(0, 0)` επιστρέφει `[[1, 0], [0, 1]]`.

Η `example(1, 1)` επιστρέφει `[[0, 1], [2, 1], [1, 0], [1, 2]]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2, 2)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[[1, 2], [2, 1]]`. Οι θέσεις με δείκτη τρία απορρίπτονται.

Συνέχισε στην εκφώνηση της άσκησης 81, στη σελίδα 290.

Η ΑΣΚΗΣΗ

# 81 Πόσα νησιά υπάρχουν στον χάρτη;

## Τι θα φτιάξεις

Το `grid` περιέχει γη `1` και νερό `0`. Ένα νησί είναι μέγιστη ομάδα κελιών γης που συνδέονται πάνω, κάτω, αριστερά ή δεξιά. Οι διαγώνιες επαφές δεν ενώνουν νησιά. Μέτρησε τα νησιά χωρίς να αλλάξεις τον χάρτη.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `grid` έχει από 0 έως 20 γραμμές. Κάθε γραμμή έχει από 1 έως 20 στοιχεία. Οι γραμμές έχουν ίδιο μήκος. Οι ακέραιες τιμές του είναι από 0 έως 1.

**Η κλήση**

```
solve([[1, 0], [0, 1]])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Κάθε άγνωστο κελί γης αρχίζει ένα νέο νησί. Πριν συνεχίσεις τη γενική σάρωση, εξερεύνησε όλη τη συνδεδεμένη περιοχή του.

**Στόχος απόδοσης:** `O(r c)` χρόνος και `O(r c)` πρόσθετος χώρος.

ΠΟΣΑ ΝΗΣΙΑ ΥΠΑΡΧΟΥΝ ΣΤΟΝ ΧΑΡΤΗ;

# 81 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `grid`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[1, 0], [0, 1]])` | `2` |
| `solve([])` | `0` |
| `solve([[0, 0], [0, 0]])` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/81-count-islands/exercise.rb.

Tests: exercises/81-count-islands/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/81-count-islands/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Σημείωσε ένα κελί ως επισκέψιμο πριν το βάλεις στη στοίβα, ώστε να μην προστεθεί από πολλούς γείτονες.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 438.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 82 Πρώτα ολοκληρώνεται το κοντινότερο επίπεδο

Από το μηδέν επιτρέπονται κινήσεις συν ένα ή συν δύο μέχρι έναν μη αρνητικό στόχο. Θα βρούμε τις ελάχιστες κινήσεις προς κάθε τιμή.

RUBY

```ruby
def example(limit)
  distance = Array.new(limit + 1, -1)
  distance[0] = 0
  queue = [0]
  head = 0
  while head < queue.length
    value = queue[head]
    head += 1
    [value + 1, value + 2].each do |next_value|
      next if next_value > limit || distance[next_value] != -1
      distance[next_value] = distance[value] + 1
      queue << next_value
    end
  end
  distance
end
```

## Διάβασε τις γραμμές

01 Νέες καταστάσεις προστίθενται στο τέλος και παλιότερες διαβάζονται από το `head`.

02 Η απόσταση ορίζεται πριν από την εισαγωγή, ώστε η κατάσταση να μην προστεθεί ξανά.

03 Η ορθότητα βασίζεται στο ίδιο κόστος για κάθε κίνηση. Διαφορετικά βάρη απαιτούν άλλη σειρά εξέτασης.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 82 Από τον κώδικα στην τιμή

**Η κλήση**

```
example(4)
```

↓ Η τιμή που επιστρέφεται

```
[0, 1, 1, 2, 2]
```

## Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει `[0, 1, 1, 2, 2]`.

Η `example(0)` επιστρέφει `[0]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(5)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 1, 1, 2, 2, 3]`. Το πέντε χρειάζεται τρεις κινήσεις, όπως 2, 2 και 1.

Συνέχισε στην εκφώνηση της άσκησης 82, στη σελίδα 294.

Η ΑΣΚΗΣΗ

# 82 Η συντομότερη διαδρομή στο πλέγμα

## Τι θα φτιάξεις

Εδώ το `0` είναι ελεύθερο κελί και το `1` εμπόδιο. Ξεκίνα πάνω αριστερά και φτάσε κάτω δεξιά με κινήσεις πάνω, κάτω, αριστερά ή δεξιά. Επέστρεψε το μικρότερο πλήθος κινήσεων ή `-1` όταν δεν υπάρχει διαδρομή. Ο κενός χάρτης δεν έχει διαδρομή.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `grid` έχει από 0 έως 20 γραμμές. Κάθε γραμμή έχει από 1 έως 20 στοιχεία. Οι γραμμές έχουν ίδιο μήκος. Οι ακέραιες τιμές του είναι από 0 έως 1.

**Η κλήση**

```
solve([[0, 0], [1, 0]])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Όταν κάθε κίνηση κοστίζει ένα, μια ουρά εξετάζει πρώτα όλες τις θέσεις απόστασης ένα, μετά δύο και ούτω καθεξής.

**Στόχος απόδοσης:** `O(r c)` χρόνος και `O(r c)` πρόσθετος χώρος.

Η ΣΥΝΤΟΜΟΤΕΡΗ ΔΙΑΔΡΟΜΗ ΣΤΟ ΠΛΕΓΜΑ

# 82 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `grid`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[0, 0], [1, 0]])` | `2` |
| `solve([])` | `-1` |
| `solve([[0]])` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/82-shortest-grid-path/exercise.rb.

Tests: exercises/82-shortest-grid-path/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/82-shortest-grid-path/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κράτα πίνακα αποστάσεων με αρχική τιμή `-1`. Μια θέση επισκέπτεται πρώτη φορά όταν εισάγεται στην ουρά.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 440.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 83 Από ζεύγη ακμών σε λίστα γειτόνων

Το παράδειγμα φτιάχνει κατευθυνόμενο γράφο. Κάθε ζεύγος δηλώνει μόνο κίνηση από την πρώτη κορυφή στη δεύτερη.

RUBY

```ruby
def example(size, edges)
  graph = Array.new(size) { [] }
  edges.each do |from, to|
    graph[from] << to
  end
  graph
end
```

## Διάβασε τις γραμμές

01 Η θέση `graph[v]` περιέχει τις κορυφές στις οποίες μπορούμε να πάμε από το `v`.

02 Οι άδειες λίστες είναι απαραίτητες για κορυφές χωρίς εξερχόμενη ακμή.

03 Στην αμφίδρομη άσκηση χρειάζεται επιπλέον και η αντίστροφη εισαγωγή.

## Ακολούθησε μια κλήση

Η `example(3, [[0, 1], [0, 2]])` επιστρέφει `[[1, 2], [], []]`.

Η `example(0, [])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2, [[1, 0]])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[[], [0]]`. Η ακμή ξεκινά από το ένα, οπότε η λίστα του μηδενός μένει κενή.

Συνέχισε στην εκφώνηση της άσκησης 83, στη σελίδα 297.

Η ΑΣΚΗΣΗ

# 83 Αποστάσεις σε έναν γράφο

## Τι θα φτιάξεις

Οι κορυφές είναι `0...size`. Κάθε ζεύγος `[a, b]` του `edges` είναι αμφίδρομη ακμή και όλοι οι δείκτες, μαζί με το `start`, είναι έγκυροι. Επέστρεψε την ελάχιστη απόσταση σε πλήθος ακμών από το `start` προς κάθε κορυφή ή `-1` αν δεν είναι προσβάσιμη. Διπλές ακμές και βρόχοι επιτρέπονται.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `size` είναι από 1 έως 30. Το `edges` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 2 έως 2 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 29. Το `start` είναι από 0 έως 29.

**Η κλήση**

```
solve(4, [[0, 1], [1, 2]], 0)
```

↓ Η τιμή που επιστρέφεται

```
[0, 1, 2, -1]
```

## Η ιδέα

Η αναζήτηση στο πλέγμα γενικεύεται όταν οι γείτονες δίνονται από λίστα αντί να υπολογίζονται από συντεταγμένες.

**Στόχος απόδοσης:** `O(v + e)` χρόνος και χώρος, με `v` κορυφές και `e` ακμές.

ΑΠΟΣΤΑΣΕΙΣ ΣΕ ΕΝΑΝ ΓΡΑΦΟ

# 83 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `size`, `edges`, `start`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4, [[0, 1], [1, 2]], 0)` | `[0, 1, 2, -1]` |
| `solve(1, [], 0)` | `[0]` |
| `solve(3, [], 1)` | `[-1, 0, -1]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/83-graph-distances/exercise.rb.

Tests: exercises/83-graph-distances/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/83-graph-distances/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Για αμφίδρομη ακμή πρόσθεσε και τις δύο κατευθύνσεις στη λίστα γειτόνων.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 442.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 84 Μέτρα τις κορυφές που δεν αναφέρονται πουθενά

Θα βρούμε το πλήθος κορυφών που δεν εμφανίζονται σε κανένα άκρο ακμής. Αυτό δεν λύνει όλες τις συνιστώσες, αλλά αναδεικνύει τις απομονωμένες.

RUBY

```ruby
def example(size, edges)
  mentioned = Array.new(size, false)
  edges.each do |a, b|
    mentioned[a] = true
    mentioned[b] = true
  end
  mentioned.count(false)
end
```

## Διάβασε τις γραμμές

01 Ο πίνακας αρχικοποιείται για όλες τις κορυφές, ακόμη και για όσες λείπουν από τις ακμές.

02 Η `count(false)` μετρά θέσεις με συγκεκριμένη τιμή.

03 Οι συνδεδεμένες ομάδες περισσότερων κορυφών χρειάζονται κανονική εξερεύνηση στην άσκηση.

## Ακολούθησε μια κλήση

Η `example(4, [[0, 1]])` επιστρέφει `2`.

Η `example(0, [])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3, [])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `3`. Χωρίς ακμές, καμία από τις τρεις κορυφές δεν αναφέρεται.

Συνέχισε στην εκφώνηση της άσκησης 84, στη σελίδα 300.

Η ΑΣΚΗΣΗ

# 84 Πόσα χωριστά δίκτυα υπάρχουν;

## Τι θα φτιάξεις

Οι κορυφές είναι `0...size` και τα έγκυρα ζεύγη του `edges` είναι αμφίδρομες ακμές. Μέτρησε τις συνδεδεμένες συνιστώσες. Κάθε απομονωμένη κορυφή μετρά ως μία. Με μηδενικές κορυφές οι ακμές είναι κενές και η απάντηση μηδέν. Διπλές ακμές και βρόχοι επιτρέπονται.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `size` είναι από 0 έως 30. Το `edges` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 2 έως 2 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 29.

Η κλήση

```
solve(5, [[0, 1], [1, 2], [3, 4]])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Κάθε νέα αναζήτηση από άγνωστη κορυφή ανακαλύπτει μία ολόκληρη συνιστώσα. Ήδη γνωστές κορυφές δεν αρχίζουν νέα αναζήτηση.

**Στόχος απόδοσης:** `O(v + e)` χρόνος και χώρος.

ΠΟΣΑ ΧΩΡΙΣΤΑ ΔΙΚΤΥΑ ΥΠΑΡΧΟΥΝ;

# 84 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `size`, `edges`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(5, [[0, 1], [1, 2], [3, 4]])` | `2` |
| `solve(0, [])` | `0` |
| `solve(4, [])` | `4` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/84-connected-components/exercise.rb.

Tests: exercises/84-connected-components/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/84-connected-components/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η εξωτερική σάρωση πρέπει να εξετάζει όλες τις κορυφές, όχι μόνο όσες εμφανίζονται στις ακμές.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 444.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 85 Μέτρησε τις εισερχόμενες εξαρτήσεις

Θα φτιάξουμε τον πίνακα εισερχόμενων βαθμών ενός κατευθυνόμενου γράφου. Αυτό είναι το σημείο εκκίνησης της τοπολογικής ταξινόμησης.

RUBY

```ruby
def example(size, edges)
  degree = Array.new(size, 0)
  edges.each do |_from, to|
    degree[to] += 1
  end
  degree
end
```

## Διάβασε τις γραμμές

01 Κάθε ακμή αυξάνει τον βαθμό του προορισμού, όχι της αφετηρίας.

02 Το `_from` δείχνει ότι η πρώτη τιμή δεν χρησιμοποιείται σε αυτό το μικρό βήμα.

03 Μηδενικός εισερχόμενος βαθμός σημαίνει ότι δεν υπάρχει προαπαιτούμενη εργασία στην τρέχουσα δομή.

## Ακολούθησε μια κλήση

Η `example(3, [[0, 1], [0, 2]])` επιστρέφει `[0, 1, 1]`.

Η `example(0, [])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2, [[0, 1], [0, 1]])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 2]`. Οι δύο καταγραφές της ίδιας ακμής μετρώνται και οι δύο.

Συνέχισε στην εκφώνηση της άσκησης 85, στη σελίδα 303.

Η ΑΣΚΗΣΗ

# 85 Μια σειρά που σέβεται τις εξαρτήσεις

## Τι θα φτιάξεις

Οι εργασίες είναι `0...size`. Κάθε έγκυρη ακμή `[a, b]` απαιτεί να γίνει η `a` πριν από τη `b`. Επέστρεψε τη λεξικογραφικά μικρότερη έγκυρη σειρά όλων των εργασιών. Αν υπάρχει κύκλος, επέστρεψε `[]`. Για μηδενικές εργασίες οι ακμές είναι κενές. Επαναλαμβανόμενες ακμές επιτρέπονται.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `size` είναι από 0 έως 30. Το `edges` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 2 έως 2 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 29.

Η κλήση

```
solve(4, [[0, 2], [1, 2], [2, 3]])
```

↓ Η τιμή που επιστρέφεται

```
[0, 1, 2, 3]
```

## Η ιδέα

Διαθέσιμη είναι μια εργασία χωρίς εκκρεμή προαπαιτούμενα. Εκτέλεσε τη μικρότερη διαθέσιμη και αφαίρεσε τη συμβολή της στις εξαρτήσεις των επόμενων.

**Στόχος απόδοσης:** `O(v² + e)` χρόνος και `O(v + e)` χώρος. Η γραμμική επιλογή είναι σκόπιμη για έως 30 κορυφές.

ΜΙΑ ΣΕΙΡΑ ΠΟΥ ΣΕΒΕΤΑΙ ΤΙΣ ΕΞΑΡΤΗΣΕΙΣ

# 85 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `size`, `edges`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4, [[0, 2], [1, 2], [2, 3]])` | `[0, 1, 2, 3]` |
| `solve(0, [])` | `[]` |
| `solve(3, [])` | `[0, 1, 2]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/85-topological-order/exercise.rb.

Tests: exercises/85-topological-order/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/85-topological-order/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κράτα τον αριθμό εισερχόμενων ακμών κάθε κορυφής. Για τα μικρά όρια της άσκησης μπορείς να σαρώσεις τους δείκτες για την επόμενη επιλογή.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 446.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 86 Ένας κύκλος πρέπει να επιστρέψει στο αρχικό χρώμα

Θα εναλλάξουμε δύο ομάδες πάνω σε διαδρομή δοσμένου μήκους. Η επιστροφή δείχνει το χρώμα στο τέλος, ξεκινώντας από μηδέν.

RUBY

```ruby
def example(length)
  color = 0
  length.times { color = 1 - color }
  color
end
```

## Διάβασε τις γραμμές

01 Η έκφραση `1 - color` ανταλλάσσει το μηδέν και το ένα.

02 Με άρτιο πλήθος κινήσεων επιστρέφουμε στο αρχικό χρώμα.

03 Ένας περιττός κύκλος θα απαιτούσε από την ίδια κορυφή δύο διαφορετικά χρώματα.

## Ακολούθησε μια κλήση

Η `example(4)` επιστρέφει `0`.

Η `example(0)` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `1`. Τρεις εναλλαγές δίνουν το αντίθετο χρώμα.

Συνέχισε στην εκφώνηση της άσκησης 86, στη σελίδα 306.

Η ΑΣΚΗΣΗ

# 86 Χωρίζεται ο γράφος σε δύο ομάδες;

## Τι θα φτιάξεις

Οι κορυφές είναι `0...size` και οι έγκυρες ακμές αμφίδρομες. Επέστρεψε αν μπορούν να χωριστούν σε δύο ομάδες ώστε κάθε ακμή να ενώνει διαφορετικές ομάδες. Απομονωμένες κορυφές επιτρέπονται. Ο κενός γράφος είναι αποδεκτός και βρόχος σε μία κορυφή προκαλεί αποτυχία.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `size` είναι από 0 έως 30. Το `edges` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 2 έως 2 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 29.

Η κλήση

```
solve(4, [[0, 1], [1, 2], [2, 3], [3, 0]])
```

↓ Η τιμή που επιστρέφεται

```
true
```

## Η ιδέα

Δώσε μια αυθαίρετη ομάδα στην πρώτη κορυφή κάθε συνιστώσας. Κάθε γείτονας πρέπει να πάρει την αντίθετη. Σύγκρουση σε ήδη χρωματισμένη κορυφή σημαίνει αποτυχία.

**Στόχος απόδοσης:** `O(v + e)` χρόνος και χώρος.

ΧΩΡΙΖΕΤΑΙ Ο ΓΡΑΦΟΣ ΣΕ ΔΥΟ ΟΜΑΔΕΣ;

# 86 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `size`, `edges`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4, [[0, 1], [1, 2], [2, 3], [3, 0]])` | `true` |
| `solve(0, [])` | `true` |
| `solve(3, [[0, 1], [1, 2], [2, 0]])` | `false` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/86-bipartite-graph/exercise.rb.

Tests: exercises/86-bipartite-graph/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/86-bipartite-graph/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Οι ομάδες μπορούν να είναι `0` και `1`, οπότε η αντίθετη είναι `1 - color`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 448.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 87 Μια lambda μπορεί να καλέσει τον εαυτό της

Θα δημιουργήσουμε την ακολουθία από έναν μη αρνητικό αριθμό μέχρι το μηδέν με αναδρομική κλήση.

RUBY

```ruby
def example(number)
  descend = lambda do |current|
    return [0] if current == 0
    [current] + descend.call(current - 1)
  end
  descend.call(number)
end
```

## Διάβασε τις γραμμές

01 Η `lambda` είναι αντικείμενο κώδικα που εκτελείται με `call`.

02 Το `return` μέσα σε lambda επιστρέφει από αυτή την κλήση της lambda, όχι από τη μέθοδο που την περιέχει.

03 Η βασική περίπτωση μηδέν σταματά την αναδρομή. Κάθε άλλη κλήση πλησιάζει σε αυτή.

## Ακολούθησε μια κλήση

Η `example(3)` επιστρέφει `[3, 2, 1, 0]`.

Η `example(0)` επιστρέφει `[0]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[2, 1, 0]`. Η κλήση με δύο περιμένει την κλήση με ένα, που τελικά φτάνει στο μηδέν.

Συνέχισε στην εκφώνηση της άσκησης 87, στη σελίδα 309.

Η ΑΣΚΗΣΗ

# 87 Όλες οι διαφορετικές μεταθέσεις

## Τι θα φτιάξεις

Επέστρεψε όλες τις διαφορετικές μεταθέσεις του `numbers` σε λεξικογραφική αύξουσα σειρά. Τα διπλότυπα της εισόδου πρέπει να συμμετέχουν με το αρχικό πλήθος τους, αλλά ίδιες τελικές μεταθέσεις εμφανίζονται μόνο μία φορά. Ο κενός πίνακας έχει μία μετάθεση: `[[]]`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 8 στοιχεία. Οι ακέραιες τιμές του είναι από -9 έως 9.

Η κλήση

```
solve([1, 1, 2])
```

↓ Η τιμή που επιστρέφεται

```
[[1, 1, 2], [1, 2, 1], [2, 1, 1]]
```

## Η ιδέα

Χτίσε την απάντηση μία θέση τη φορά. Δοκίμασε κάθε διαθέσιμη τιμή και αναίρεσε την επιλογή πριν δοκιμάσεις την επόμενη.

**Στόχος απόδοσης:** `O(n · n!)` άνω φράγμα χρόνου και χώρου εξόδου· `O(n)` χώρος αναζήτησης πέρα από την έξοδο.

ΟΛΕΣ ΟΙ ΔΙΑΦΟΡΕΤΙΚΕΣ ΜΕΤΑΘΕΣΕΙΣ

# 87 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 1, 2])` | `[[1, 1, 2], [1, 2, 1], [2, 1, 1]]` |
| `solve([])` | `[[]]` |
| `solve([4])` | `[[4]]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/87-unique-permutations/exercise.rb.

Tests: exercises/87-unique-permutations/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/87-unique-permutations/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ταξινόμησε ένα αντίγραφο. Αν δύο ίσες τιμές είναι ακόμη διαθέσιμες, διάλεξε πρώτα την αριστερότερη.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 450.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 88 Μια απόφαση μπορεί να αποκλείσει άχρηστο κλάδο

Θα βρούμε ποιες πρώτες τιμές επιτρέπουν επιλογή τριών αύξουσων αριθμών μέχρι ένα δοσμένο όριο.

RUBY

```ruby
def example(size)
  (1..(size - 2)).to_a
end
```

## Διάβασε τις γραμμές

01 Για τρεις επιλογές, μετά την πρώτη πρέπει να χωρούν ακόμη δύο μεγαλύτερες τιμές.

02 Αν το πάνω όριο είναι μικρότερο από το ένα, το αύξον Range δεν παράγει τιμές.

03 Το κλάδεμα απορρίπτει έναν κλάδο επειδή αποδεικνύεται αδύνατος, όχι επειδή φαίνεται απλώς λιγότερο πιθανός.

## Ακολούθησε μια κλήση

Η `example(5)` επιστρέφει `[1, 2, 3]`.

Η `example(2)` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[1]`. Μόνο η αρχή ένα αφήνει χώρο για άλλες δύο επιλογές.

Συνέχισε στην εκφώνηση της άσκησης 88, στη σελίδα 312.

Η ΑΣΚΗΣΗ

# 88 Διάλεξε k τιμές χωρίς επαναλήψεις

## Τι θα φτιάξεις

Επέστρεψε όλους τους συνδυασμούς `count` διαφορετικών τιμών από το `1..size`. Κάθε συνδυασμός είναι αύξων και οι συνδυασμοί ταξινομημένοι λεξικογραφικά. Για `count == 0` επέστρεψε `[[]]`. Αν `count > size`, επέστρεψε `[]`. Μην χρησιμοποιήσεις τη `combination`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `size` είναι από 0 έως 12. Το `count` είναι από 0 έως 12.

Η κλήση

```
solve(3, 2)
```

↓ Η τιμή που επιστρέφεται

```
[[1, 2], [1, 3], [2, 3]]
```

## Η ιδέα

Αφού επιλέξεις μία τιμή, οι επόμενες πρέπει να είναι μεγαλύτερες. Έτσι κάθε σύνολο επιλογών παράγεται μία φορά και δεν χρειάζεται τελική αφαίρεση διπλότυπων.

**Στόχος απόδοσης:** `O(k · C(n,k))` για τις έγκυρες εξόδους, με `O(k)` χώρο αναζήτησης πέρα από την αποθήκευσή τους.

ΔΙΑΛΕΞΕ Κ ΤΙΜΕΣ ΧΩΡΙΣ ΕΠΑΝΑΛΗΨΕΙΣ

# 88 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `size`, `count`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(3, 2)` | `[[1, 2], [1, 3], [2, 3]]` |
| `solve(0, 0)` | `[[]]` |
| `solve(3, 0)` | `[[]]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/88-combinations/exercise.rb.

Tests: exercises/88-combinations/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/88-combinations/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Αν απομένουν `needed` επιλογές, δεν έχει νόημα να αρχίσεις πέρα από το `size - needed + 1`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 452.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 89 Η αναζήτηση σχηματίζει δέντρο επιλογών

Θα παράγουμε όλα τα δυαδικά κείμενα μικρού δοσμένου μήκους. Η άσκηση θα περιορίσει τους επιτρεπτούς κλάδους.

RUBY

```ruby
def example(length)
  result = []
  visit = lambda do |text|
    return result << text if text.length == length
    visit.call(text + "0")
    visit.call(text + "1")
  end
  visit.call("")
  result
end
```

## Διάβασε τις γραμμές

01 Κάθε εσωτερικός κόμβος δοκιμάζει δύο επιλογές και κάθε φύλλο είναι μία ολοκληρωμένη απάντηση.

02 Το `text + ...` δημιουργεί νέο κείμενο, οπότε οι κλάδοι δεν μοιράζονται μεταβαλλόμενη διαδρομή.

03 Το πλήθος φύλλων είναι εκθετικό. Το κλάδεμα της άσκησης αποφεύγει όσα είναι ήδη άκυρα.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 89 Από τον κώδικα στην τιμή

**Η κλήση**

```
example(2)
```

↓ Η τιμή που επιστρέφεται

```
["00", "01", "10", "11"]
```

## Ακολούθησε μια κλήση

Η `example(2)` επιστρέφει `["00", "01", "10", "11"]`.

Η `example(0)` επιστρέφει `[""]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(1)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `["0", "1"]`. Υπάρχουν μόνο οι δύο επιλογές του πρώτου επιπέδου.

Συνέχισε στην εκφώνηση της άσκησης 89, στη σελίδα 316.

Η ΑΣΚΗΣΗ

# 89 Παρήγαγε μόνο σωστές παρενθέσεις

## Τι θα φτιάξεις

Επέστρεψε όλες τις σωστά ισορροπημένες ακολουθίες από `pairs` ζεύγη παρενθέσεων, σε λεξικογραφική σειρά. Για μηδενικά ζεύγη επέστρεψε `[""]`. Μην δημιουργήσεις πρώτα όλες τις δυαδικές ακολουθίες για να τις φιλτράρεις μετά.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `pairs` είναι από 0 έως 8.

Η κλήση

```
solve(2)
```

↓ Η τιμή που επιστρέφεται

```
["(())", "()()"]
```

## Η ιδέα

Ένα σωστό τελικό κείμενο έχει σωστά προθέματα: ποτέ περισσότερα κλεισίματα από ανοίγματα. Κατασκεύασε μόνο κλάδους που μπορούν ακόμη να οδηγήσουν σε λύση.

**Στόχος απόδοσης:** `O(n Cn)` χρόνος και χώρος εξόδου, όπου `Cn` το πλήθος Catalan· το βάθος αναδρομής είναι `O(n)`.

ΠΑΡΗΓΑΓΕ ΜΟΝΟ ΣΩΣΤΕΣ ΠΑΡΕΝΘΕΣΕΙΣ

# 89 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `pairs`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(2)` | `["(())", "()()"]` |
| `solve(0)` | `[""]` |
| `solve(1)` | `["()"]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/89-generate-parentheses/exercise.rb.

Tests: exercises/89-generate-parentheses/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/89-generate-parentheses/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Άνοιγμα επιτρέπεται όσο δεν έχεις χρησιμοποιήσει όλα τα ζεύγη. Κλείσιμο επιτρέπεται μόνο όταν υπάρχουν περισσότερα ανοίγματα από κλεισίματα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 453.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 90 Οι διαγώνιοι ελέγχονται αριθμητικά

Θα ελέγξουμε αν μια δοσμένη μικρή ομάδα θέσεων βασιλισσών έχει σύγκρουση. Κάθε θέση γράφεται ως ζεύγος γραμμής και στήλης.

RUBY

```ruby
def example(queens)
  queens.combination(2).none? do |first, second|
    r1, c1 = first
    r2, c2 = second
    r1 == r2 || c1 == c2 || (r1-r2).abs == (c1-c2).abs
  end
end
```

## Διάβασε τις γραμμές

01 Η `combination(2)` παράγει κάθε ζεύγος διαφορετικών θέσεων μία φορά.

02 Η `none?` είναι αληθής όταν κανένα ζεύγος δεν ικανοποιεί τη συνθήκη σύγκρουσης.

03 Στην ίδια διαγώνιο, η απόλυτη διαφορά γραμμών ισούται με την απόλυτη διαφορά στηλών.

## Ακολούθησε μια κλήση

Η `example([[0, 1], [1, 3]])` επιστρέφει `true`.

Η `example([])` επιστρέφει `true`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([[0, 0], [1, 1]])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `false`. Οι δύο διαφορές είναι και οι δύο ίσες με ένα.

Συνέχισε στην εκφώνηση της άσκησης 90, στη σελίδα 319.

Η ΑΣΚΗΣΗ

# 90 Πόσες τοποθετήσεις βασιλισσών υπάρχουν;

## Τι θα φτιάξεις

Σε σκακιέρα `size` επί `size`, μέτρησε τις τοποθετήσεις `size` βασιλισσών ώστε καμία να μην απειλεί άλλη. Απαγορεύεται κοινή γραμμή, στήλη ή διαγώνιος. Συμμετρικές τοποθετήσεις μετρούν χωριστά. Η άδεια σκακιέρα έχει μία κενή τοποθέτηση.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `size` είναι από 0 έως 10.

Η κλήση

```
solve(4)
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Τοποθέτησε μία βασίλισσα ανά γραμμή. Για κάθε υποψήφια στήλη αρκούν τρεις έλεγχοι: κατειλημμένη στήλη και οι δύο οικογένειες διαγωνίων.

**Στόχος απόδοσης:** `O(n · n!)` άνω φράγμα χρόνου και `O(n)` χώρος αναζήτησης. Δεν αποθηκεύονται όλες οι τοποθετήσεις.

ΠΟΣΕΣ ΤΟΠΟΘΕΤΗΣΕΙΣ ΒΑΣΙΛΙΣΣΩΝ ΥΠΑΡΧΟΥΝ;

# 90 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `size`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4)` | `2` |
| `solve(0)` | `1` |
| `solve(1)` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/90-n-queens-count/exercise.rb.

Tests: exercises/90-n-queens-count/exercise_spec.rb.

SH

```
bundle exec rspec \
  exercises/90-n-queens-count/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Οι θέσεις της ίδιας διαγωνίου έχουν σταθερό `row - column` ή σταθερό `row + column`.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 454.

ΑΣΚΗΣΕΙΣ 91-100

# Δυναμικός προγραμματισμός

Καταστάσεις, μεταβάσεις και βελτιστοποίηση, μέχρι συντομότερες διαδρομές και υποσύνολα πόλεων.



Δώσε χρόνο στη σκέψη, στις οριακές περιπτώσεις και στην εξήγηση του κόστους. Οι λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 91 Κάθε υποπρόβλημα λύνεται μία φορά

Θα βρούμε τους τρόπους άφιξης σε κάθε θέση μιας σκάλας χωρίς εμπόδια. Ο πίνακας αποθηκεύει όσα θα ξαναχρειαζόμασταν σε μια απλή αναδρομή.

RUBY

```ruby
def example(steps)
  ways = Array.new(steps + 1, 0)
  ways[0] = 1
  (1..steps).each do |step|
    ways[step] = ways[step - 1]
    ways[step] += ways[step - 2] if step >= 2
  end
  ways
end
```

## Διάβασε τις γραμμές

01 Η θέση του πίνακα είναι η κατάσταση του υποπροβλήματος και η τιμή της είναι η απάντησή του.

02 Η βασική περίπτωση μηδέν έχει μία δυνατότητα: να μην κάνουμε καμία κίνηση.

03 Όταν φτάνουμε σε μια νέα θέση, οι δύο προηγούμενες έχουν ήδη υπολογιστεί.

## Ακολούθησε μια κλήση

Η `example(3)` επιστρέφει `[1, 1, 2, 3]`.

Η `example(0)` επιστρέφει `[1]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(4)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[1, 1, 2, 3, 5]`. Οι πέντε τρόποι προκύπτουν από τρεις και δύο προηγούμενους.

Συνέχισε στην εκφώνηση της άσκησης 91, στη σελίδα 323.

Η ΑΣΚΗΣΗ

# 91 Ανέβα τη σκάλα με κλειστά σκαλοπάτια

## Τι θα φτιάξεις

Ξεκινάς στο μηδέν και θέλεις να φτάσεις ακριβώς στο `steps`, με άλματα ενός ή δύο σκαλοπατιών. Τα στοιχεία του `blocked` είναι έγκυροι δείκτες από 1 έως `steps` όπου δεν επιτρέπεται να πατήσεις. Διπλότυπα αγνοούνται. Μέτρησε τις διαφορετικές ακολουθίες αλμάτων. Για μηδενική σκάλα υπάρχει μία κενή διαδρομή.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `steps` είναι από 0 έως 60. Το `blocked` έχει από 0 έως 60 στοιχεία. Οι ακέραιες τιμές του είναι από 1 έως 60.

Η κλήση

```
solve(5, [2])
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Οι τρόποι άφιξης σε ένα ελεύθερο σκαλοπάτι προέρχονται από τα δύο αμέσως προηγούμενα. Μια απαγορευμένη θέση έχει μηδέν τρόπους, αλλά μπορεί να υπερπηδηθεί.

**Στόχος απόδοσης:** `O(n + b)` αναμενόμενος χρόνος και χώρος, για `n` σκαλοπάτια και `b` απαγορευμένες καταγραφές.

ΑΝΕΒΑ ΤΗ ΣΚΑΛΑ ΜΕ ΚΛΕΙΣΤΑ ΣΚΑΛΟΠΑΤΙΑ

# 91 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `steps`, `blocked`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(5, [2])` | `2` |
| `solve(0, [])` | `1` |
| `solve(3, [])` | `3` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/91-stairs-with-blocked-steps/exercise.rb.

Tests: exercises/91-stairs-with-blocked-steps/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/91-stairs-with-blocked-steps/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η θέση μηδέν πρέπει να αρχίσει με μία δυνατότητα, ώστε να δημιουργηθούν οι πρώτες πραγματικές κινήσεις.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 456.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 92 Μια άγνωστη κατάσταση δεν είναι μηδενικό κόστος

Οι τιμές είναι γνωστά κόστη ή -1 για αδύνατη κατάσταση. Θα προσθέσουμε ένα βήμα και θα κρατήσουμε το μικρότερο εφικτό κόστος.

RUBY

```ruby
def example(costs)
  best = nil
  costs.each do |cost|
    next if cost == -1
    candidate = cost + 1
    best = candidate if best.nil? || candidate < best
  end
  best || -1
end
```

## Διάβασε τις γραμμές

01 Το `nil` δηλώνει εδώ ότι δεν έχει βρεθεί ακόμη υποψήφια λύση.

02 Οι αδύνατες καταστάσεις παραλείπονται πριν προστεθεί το νέο βήμα.

03 Το μηδενικό κόστος είναι κανονική εφικτή τιμή και πρέπει να συμμετέχει.

## Ακολούθησε μια κλήση

Η `example([-1, 2, 5])` επιστρέφει `3`.

Η `example([])` επιστρέφει `-1`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, 3])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `1`. Η μηδενική προηγούμενη απόσταση δίνει τη φθηνότερη επέκταση.

Συνέχισε στην εκφώνηση της άσκησης 92, στη σελίδα 326.

Η ΑΣΚΗΣΗ

# 92 Το ποσό με τα λιγότερα νομίσματα

## Τι θα φτιάξεις

Κάθε τιμή του `coins` είναι διαθέσιμη ονομαστική αξία με απεριόριστο πλήθος νομισμάτων. Βρες το μικρότερο πλήθος για να σχηματίσεις ακριβώς το `amount`, ή `-1` αν είναι αδύνατο. Για ποσό μηδέν επέστρεψε `0`. Διπλές αξίες δεν αλλάζουν το πρόβλημα.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `coins` έχει από 0 έως 20 στοιχεία. Οι ακέραιες τιμές του είναι από 1 έως 50. Το `amount` είναι από 0 έως 500.

Η κλήση

```
solve([1, 3, 4], 6)
```

↓ Η τιμή που επιστρέφεται

```
2
```

## Η ιδέα

Αν το τελευταίο νόμισμα έχει αξία `coin`, πριν από αυτό πρέπει να έχει σχηματιστεί το υπόλοιπο ποσό. Δοκίμασε κάθε επιτρεπτή τελευταία αξία και κράτα την καλύτερη.

**Στόχος απόδοσης:** `O(a m)` χρόνος και `O(a)` χώρος, για ποσό `a` και `m` καταγεγραμμένες αξίες.

ΤΟ ΠΟΣΟ ΜΕ ΤΑ ΛΙΓΟΤΕΡΑ ΝΟΜΙΣΜΑΤΑ

# 92 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `coins`, `amount`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 3, 4], 6)` | `2` |
| `solve([], 0)` | `0` |
| `solve([], 7)` | `-1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/92-minimum-coins/exercise.rb.

Tests: exercises/92-minimum-coins/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/92-minimum-coins/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Χρησιμοποίησε τιμή μεγαλύτερη από κάθε εφικτό πλήθος ως αρχικό «άγνωστο». Με θετικές ακέραιες αξίες, το `amount + 1` αρκεί.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 457.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 93 Πρόσθεσε μία νέα επιτρεπτή αξία

Υποθέτουμε ότι έχουμε ήδη μόνο νομίσματα αξίας ένα και προσθέτουμε την αξία τρία. Θα επιστρέψουμε τα πλήθη για όλα τα ποσά μέχρι το όριο.

RUBY

```ruby
def example(amount)
  ways = Array.new(amount + 1, 1)
  (3..amount).each do |value|
    ways[value] += ways[value - 3]
  end
  ways
end
```

## Διάβασε τις γραμμές

01 Με μόνα νομίσματα του ενός, κάθε ποσό έχει ακριβώς έναν συνδυασμό.

02 Η ενημέρωση από μικρά προς μεγάλα επιτρέπει στη νέα απάντηση να χρησιμοποιήσει ξανά το τρία.

03 Δεν αλλάζουμε τη σειρά των ήδη επιλεγμένων αξιών, άρα δεν μετράμε μεταθέσεις νομισμάτων.

## Ακολούθησε μια κλήση

Η `example(6)` επιστρέφει `[1, 1, 1, 2, 2, 2, 3]`.

Η `example(0)` επιστρέφει `[1]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[1, 1, 1, 2]`. Το ποσό τρία φτιάχνεται με τρία μονά ή ένα τριάρι.

Συνέχισε στην εκφώνηση της άσκησης 93, στη σελίδα 329.

Η ΑΣΚΗΣΗ

# 93 Πόσοι συνδυασμοί φτιάχνουν το ποσό;

## Τι θα φτιάξεις

Με απεριόριστα νομίσματα των αξιών `coins`, μέτρησε τους διαφορετικούς συνδυασμούς που σχηματίζουν ακριβώς το `amount`. Η σειρά των νομισμάτων δεν μετρά. Διπλές καταγραφές της ίδιας αξίας αγνοούνται. Το μηδενικό ποσό έχει έναν συνδυασμό: κανένα νόμισμα.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `coins` έχει από 0 έως 20 στοιχεία. Οι ακέραιες τιμές του είναι από 1 έως 50. Το `amount` είναι από 0 έως 100.

Η κλήση

```
solve([1, 2, 5], 5)
```

↓ Η τιμή που επιστρέφεται

```
4
```

## Η ιδέα

Επεξεργάσου μία αξία τη φορά. Ο πίνακας μετρά όσα ποσά φτιάχνονται χρησιμοποιώντας μόνο τις αξίες που έχουν ήδη εξεταστεί.

**Στόχος απόδοσης:** `O(a u + m)` αναμενόμενος χρόνος και `O(a + u)` χώρος, όπου `u` οι διαφορετικές αξίες.

ΠΟΣΟΙ ΣΥΝΔΥΑΣΜΟΙ ΦΤΙΑΧΝΟΥΝ ΤΟ ΠΟΣΟ;

# 93 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `coins`, `amount`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([1, 2, 5], 5)` | `4` |
| `solve([], 0)` | `1` |
| `solve([], 5)` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/93-coin-combinations/exercise.rb.

Tests: exercises/93-coin-combinations/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/93-coin-combinations/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Ο εξωτερικός βρόχος είναι πάνω στις διαφορετικές αξίες και ο εσωτερικός πάνω στα ποσά από μικρά προς μεγάλα.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 458.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 94 Η φθίνουσα σειρά εμποδίζει την επαναχρησιμοποίηση

Υπάρχει ένα μόνο αντικείμενο αξίας ένα και θετικού δοσμένου βάρους. Θα υπολογίσουμε τη βέλτιστη αξία για κάθε χωρητικότητα.

RUBY

```ruby
def example(weight, capacity)
  best = Array.new(capacity + 1, 0)
  capacity.downto(weight) do |space|
    best[space] = [best[space], best[space - weight] + 1].max
  end
  best
end
```

## Διάβασε τις γραμμές

01 Κάθε μικρότερη θέση που διαβάζεται παραμένει στην κατάσταση πριν από το αντικείμενο.

02 Καμία απάντηση δεν πρέπει να ξεπεράσει το ένα, αφού υπάρχει μόνο ένα αντικείμενο.

03 Όταν το βάρος ξεπερνά όλη τη χωρητικότητα, ο βρόχος δεν εκτελείται.

## Ακολούθησε μια κλήση

Η `example(2, 5)` επιστρέφει `[0, 0, 1, 1, 1, 1]`.

Η `example(1, 0)` επιστρέφει `[0]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(1, 3)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 1, 1, 1]`. Το επιπλέον διαθέσιμο βάρος δεν δημιουργεί δεύτερο αντίγραφο του αντικειμένου.

Συνέχισε στην εκφώνηση της άσκησης 94, στη σελίδα 332.

Η ΑΣΚΗΣΗ

# 94 Διάλεξε αντικείμενα με περιορισμένο βάρος

## Τι θα φτιάξεις

Οι πίνακες `weights` και `values` έχουν ίδιο μήκος και περιγράφουν διαφορετικά αντικείμενα. Διάλεξε υποσύνολο με συνολικό βάρος το πολύ `capacity` και μέγιστη συνολική αξία. Κάθε θέση επιλέγεται το πολύ μία φορά. Επιτρέπεται η κενή επιλογή.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `weights` έχει από 0 έως 20 στοιχεία. Οι ακέραιες τιμές του είναι από 1 έως 20. Το `values` έχει από 0 έως 20 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 1000. Το `capacity` είναι από 0 έως 100.

Η κλήση

```
solve([2, 3, 4], [4, 5, 7], 5)
```

↓ Η τιμή που επιστρέφεται

```
9
```

## Η ιδέα

Για κάθε αντικείμενο σύγκρινε την παράλειψή του με την επιλογή του πάνω σε μια προηγούμενη λύση μικρότερης χωρητικότητας.

**Στόχος απόδοσης:** `O(n c)` χρόνος και `O(c)` πρόσθετος χώρος, όπου `c` η χωρητικότητα.

ΔΙΑΛΕΞΕ ΑΝΤΙΚΕΙΜΕΝΑ ΜΕ ΠΕΡΙΟΡΙΣΜΕΝΟ ΒΑΡΟΣ

# 94 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `weights`, `values`, `capacity`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([2, 3, 4], [4, 5, 7], 5)` | `9` |
| `solve([], [], 5)` | `0` |
| `solve([2], [3], 4)` | `3` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/94-zero-one-knapsack/exercise.rb.

Tests: exercises/94-zero-one-knapsack/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/94-zero-one-knapsack/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Με έναν μόνο πίνακα, διάβασε τις χωρητικότητες από μεγάλες προς μικρές.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 459.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 95 Ένα κελί εξαρτάται από πάνω και αριστερά

Σε κενό ορθογώνιο πλέγμα επιτρέπονται μόνο κινήσεις δεξιά και κάτω. Θα μετρήσουμε τις διαδρομές από πάνω αριστερά μέχρι κάτω δεξιά.

RUBY

```ruby
def example(rows, columns)
  return 0 if rows == 0 || columns == 0
  ways = Array.new(columns, 1)
  (1...rows).each do
    (1...columns).each do |column|
      ways[column] += ways[column - 1]
    end
  end
  ways[-1]
end
```

## Διάβασε τις γραμμές

01 Η πρώτη γραμμή και η πρώτη στήλη έχουν μία μόνο διαδρομή προς κάθε κελί.

02 Πριν από την ενημέρωση, το `ways[column]` είναι η τιμή από πάνω. Η αριστερή θέση έχει ήδη ενημερωθεί για τη σημερινή γραμμή.

03 Ένας πίνακας μπορεί να αντικαταστήσει ολόκληρο πλέγμα καταστάσεων όταν η εξάρτηση περιορίζεται στην προηγούμενη γραμμή.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 95 Από τον κώδικα στην τιμή

**Η κλήση**

```
example(2, 3)
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Ακολούθησε μια κλήση

Η `example(2, 3)` επιστρέφει `3`.

Η `example(0, 3)` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(3, 3)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `6`. Οι διαδρομές από πάνω και αριστερά σχηματίζουν ξένες κατηγορίες.

Συνέχισε στην εκφώνηση της άσκησης 95, στη σελίδα 336.

Η ΑΣΚΗΣΗ

# 95 Η μεγαλύτερη κοινή υποακολουθία

## Τι θα φτιάξεις

Βρες το μήκος της μεγαλύτερης κοινής υποακολουθίας των δύο κειμένων. Μπορείς να παραλείπεις χαρακτήρες αλλά δεν μπορείς να αλλάξεις τη σειρά τους. Η υποακολουθία δεν χρειάζεται να είναι συνεχόμενη. Η σύγκριση είναι ευαίσθητη σε πεζά και κεφαλαία και χρησιμοποιεί χαρακτήρες Ruby.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `left` έχει από 0 έως 100 χαρακτήρες. Το `right` έχει από 0 έως 100 χαρακτήρες.

Η κλήση

```
solve("abcde", "ace")
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Η ιδέα

Η κατάσταση περιγράφει δύο προθέματα. Ίσοι τελευταίοι χαρακτήρες μπορούν να επεκτείνουν την κοινή λύση των μικρότερων προθεμάτων. Διαφορετικοί απαιτούν παράλειψη από μία πλευρά.

**Στόχος απόδοσης:** `O((n + 1)(m + 1))` χρόνος και `O(m)` πρόσθετος χώρος για δεξί κείμενο μήκους `m`.

Η ΜΕΓΑΛΥΤΕΡΗ ΚΟΙΝΗ ΥΠΟΑΚΟΛΟΥΘΙΑ

# 95 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `left`, `right`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("abcde", "ace")` | `3` |
| `solve("", "abc")` | `0` |
| `solve("abc", "")` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/95-longest-common-subsequence/exercise.rb.

Tests: exercises/95-longest-common-subsequence/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/95-longest-common-subsequence/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Χρειάζεσαι την προηγούμενη γραμμή, την τρέχουσα αριστερή τιμή και την προηγούμενη διαγώνια τιμή.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 460.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 96 Οι αρχικές γραμμές είναι ήδη λυμένα προβλήματα

Θα δημιουργήσουμε τα όρια ενός πίνακα αποστάσεων. Τα εσωτερικά κελιά μένουν -1 για να φαίνεται ότι δεν έχουν ακόμη λυθεί.

RUBY

```ruby
def example(rows, columns)
  table = Array.new(rows + 1) { Array.new(columns + 1, -1) }
  (0..rows).each { |row| table[row][0] = row }
  (0..columns).each { |column| table[0][column] = column }
  table
end
```

## Διάβασε τις γραμμές

01 Η επιπλέον γραμμή και στήλη περιγράφουν κενά προθέματα.

02 Το πέρασμα από ή προς κενό κείμενο απαιτεί μία ενέργεια για κάθε χαρακτήρα.

03 Οι σωστές βασικές περιπτώσεις επιτρέπουν στον ίδιο κανόνα να χειριστεί και τα πρώτα εσωτερικά κελιά.

## Ακολούθησε μια κλήση

Η `example(1, 2)` επιστρέφει `[[0, 1, 2], [1, -1, -1]]`.

Η `example(0, 2)` επιστρέφει `[[0, 1, 2]]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example(2, 0)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[[0], [1], [2]]`. Με μηδενικές στήλες απομένουν μόνο κόστη διαγραφής.

Συνέχισε στην εκφώνηση της άσκησης 96, στη σελίδα 339.

Η ΑΣΚΗΣΗ

# 96 Πόσες αλλαγές χωρίζουν δύο κείμενα;

## Τι θα φτιάξεις

Βρες την ελάχιστη απόσταση μετατροπής του `left` στο `right`. Κάθε εισαγωγή, διαγραφή ή αντικατάσταση ενός χαρακτήρα κοστίζει ένα. Ίδιοι χαρακτήρες δεν χρειάζονται αλλαγή. Δεν επιτρέπεται ανταλλαγή δύο χαρακτήρων ως μία κίνηση. Χρησιμοποίησε χαρακτήρες Ruby.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `left` έχει από 0 έως 100 χαρακτήρες. Το `right` έχει από 0 έως 100 χαρακτήρες.

Η κλήση

```
solve("kitten", "sitting")
```

↓ Η τιμή που επιστρέφεται

```
3
```

## Η ιδέα

Για δύο προθέματα, η τελευταία ενέργεια είναι διαγραφή, εισαγωγή ή αντιστοίχιση των τελευταίων χαρακτήρων. Δοκίμασε και τις τρεις προηγούμενες καταστάσεις.

**Στόχος απόδοσης:** `O((n + 1)(m + 1))` χρόνος και `O(m)` πρόσθετος χώρος.

ΠΟΣΕΣ ΑΛΛΑΓΕΣ ΧΩΡΙΖΟΥΝ ΔΥΟ ΚΕΙΜΕΝΑ;

# 96 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `left`, `right`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve("kitten", "sitting")` | `3` |
| `solve("", "")` | `0` |
| `solve("", "abc")` | `3` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/96-edit-distance/exercise.rb.

Tests: exercises/96-edit-distance/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/96-edit-distance/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η απόσταση από κενό κείμενο προς πρόθεμα μήκους `j` είναι `j`, όχι μηδέν.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|

Η λύση βρίσκεται στη σελίδα 462.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 97 Βελτίωσε μία θέση χωρίς να χαλάσεις τη σειρά

Σε ταξινομημένο πίνακα υποψήφιων τελών θα αντικαταστήσουμε το πρώτο στοιχείο που δεν είναι μικρότερο από μια νέα τιμή. Αν λείπει, θα την προσθέσουμε στο τέλος.

RUBY

```ruby
def example(tails, candidate)
  result = tails.dup
  position = result.bsearch_index { |value| value >= candidate }
  result[position || result.length] = candidate
  result
end
```

## Διάβασε τις γραμμές

01 Η αναζήτηση δίνει όριο που διατηρεί τη διάταξη του πίνακα μετά την αντικατάσταση.

02 Μια ίση τιμή αντικαθιστά την προηγούμενη χωρίς να αλλάζει το μήκος.

03 Η άσκηση επαναλαμβάνει αυτό το βήμα και χρειάζεται να αιτιολογήσεις τι σημαίνει κάθε θέση του πίνακα.

## Ακολούθησε μια κλήση

Η `example([2, 5, 9], 4)` επιστρέφει `[2, 4, 9]`.

Η `example([], 3)` επιστρέφει `[3]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 3, 6], 3)`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[1, 3, 6]`. Το ίσο τριάρι δεν δημιουργεί μεγαλύτερο μήκος.

Συνέχισε στην εκφώνηση της άσκησης 97, στη σελίδα 342.

Η ΑΣΚΗΣΗ

# 97 Η μεγαλύτερη αύξουσα υποακολουθία

## Τι θα φτιάξεις

Βρες το μήκος της μεγαλύτερης αυστηρά αύξουσας υποακολουθίας του `numbers`. Επιτρέπεται να παραλείπεις θέσεις, αλλά η σειρά των επιλεγμένων στοιχείων διατηρείται. Ίσες τιμές δεν επεκτείνουν την υποακολουθία. Στόχος είναι λύση `O(n log n)`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `numbers` έχει από 0 έως 500 στοιχεία. Οι ακέραιες τιμές του είναι από -1000 έως 1000.

Η κλήση

```
solve([10, 9, 2, 5, 3, 7, 101, 18])
```

↓ Η τιμή που επιστρέφεται

```
4
```

## Η ιδέα

Για κάθε εφικτό μήκος κράτα τη μικρότερη δυνατή τελευταία τιμή. Μικρότερο τέλος αφήνει τουλάχιστον τις ίδιες δυνατότητες μελλοντικής επέκτασης.

**Στόχος απόδοσης:** `O(n log n)` χρόνος και `O(n)` πρόσθετος χώρος.

Η ΜΕΓΑΛΥΤΕΡΗ ΑΥΞΟΥΣΑ ΥΠΟΑΚΟΛΟΥΘΙΑ

# 97 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `numbers`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([10, 9, 2, 5, 3, 7, 101, 18])` | `4` |
| `solve([])` | `0` |
| `solve([4, 4, 4])` | `1` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/97-longest-increasing-subsequence/exercise.rb.

Tests: exercises/97-longest-increasing-subsequence/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/97-longest-increasing-subsequence/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Η πρώτη αποθηκευμένη τιμή που είναι τουλάχιστον ίση με τον νέο αριθμό είναι η θέση που μπορεί να βελτιωθεί.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 463.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 98 Το πλήθος συμβατών προηγουμένων είναι δείκτης

Σε ζεύγη αρχής και τέλους θα μετρήσουμε πόσες εργασίες τελειώνουν μέχρι την αρχή καθεμιάς, αφού ταξινομηθούν κατά τέλος. Το παράδειγμα χρησιμοποιεί απλή καταμέτρηση για να φανεί η έννοια.

RUBY

```ruby
def example(jobs)
  ordered = jobs.sort_by { |job| job[1] }
  finishes = ordered.map { |job| job[1] }
  ordered.map do |start, _finish|
    finishes.count { |finish| finish <= start }
  end
end
```

## Διάβασε τις γραμμές

01 Η `sort_by` ταξινομεί με βάση το αποτέλεσμα του block.

02 Με θετική διάρκεια, καμία εργασία δεν είναι συμβατή με τον εαυτό της.

03 Στην άσκηση, η γραμμική καταμέτρηση αντικαθίσταται από δυαδική αναζήτηση πάνω στα ίδια ταξινομημένα τέλη.

## Ακολούθησε μια κλήση

Η `example([[0, 2], [1, 3], [2, 4]])` επιστρέφει `[0, 0, 1]`.

Η `example([])` επιστρέφει `[]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([[0, 1], [1, 2], [2, 3]])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 1, 2]`. Κάθε νέα εργασία αρχίζει αφού τελειώσουν όλες οι προηγούμενες.

Συνέχισε στην εκφώνηση της άσκησης 98, στη σελίδα 345.

Η ΑΣΚΗΣΗ

# 98 Το πιο κερδοφόρο πρόγραμμα εργασιών

## Τι θα φτιάξεις

Κάθε γραμμή είναι `[start, finish, profit]`, με `start < finish` και μη αρνητικό κέρδος. Διάλεξε εργασίες χωρίς χρονική επικάλυψη, με μέγιστο συνολικό κέρδος. Εργασία που τελειώνει τη στιγμή που αρχίζει άλλη είναι συμβατή. Κάθε καταγεγραμμένη εργασία επιλέγεται το πολύ μία φορά.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `jobs` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 3 έως 3 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 1000.

Η κλήση

```
solve([[1, 3, 5], [2, 5, 6], [3, 6, 5]])
```

↓ Η τιμή που επιστρέφεται

```
10
```

## Η ιδέα

Ταξινόμησε κατά τέλος. Μια βέλτιστη λύση είτε παραλείπει την τρέχουσα εργασία είτε τη συνδυάζει με τη βέλτιστη λύση όλων όσων τελειώνουν εγκαίρως.

**Στόχος απόδοσης:** `O(n log n)` χρόνος και `O(n)` πρόσθετος χώρος.

ΤΟ ΠΙΟ ΚΕΡΔΟΦΟΡΟ ΠΡΟΓΡΑΜΜΑ ΕΡΓΑΣΙΩΝ

# 98 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `jobs`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[1, 3, 5], [2, 5, 6], [3, 6, 5]])` | `10` |
| `solve([])` | `0` |
| `solve([[1, 2, 7]])` | `7` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/98-weighted-interval-scheduling/exercise.rb.

Tests: exercises/98-weighted-interval-scheduling/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/98-weighted-interval-scheduling/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Στον πίνακα τελών, βρες την πρώτη τιμή αυστηρά μεγαλύτερη από τη νέα αρχή. Ο δείκτης είναι το πλήθος συμβατών προηγούμενων εργασιών.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 464.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 99 Μια νέα προσφορά αλλάζει μόνο αν είναι καλύτερη

Θα εφαρμόσουμε υποψήφιες αποστάσεις σε έναν πίνακα γνωστών τιμών. Η τιμή -1 δηλώνει ακόμη άγνωστη απόσταση.

RUBY

```ruby
def example(known, offers)
  result = known.dup
  offers.each do |vertex, candidate|
    if result[vertex] == -1 || candidate < result[vertex]
      result[vertex] = candidate
    end
  end
  result
end
```

## Διάβασε τις γραμμές

01 Η χαλάρωση κρατά τη μικρότερη απόσταση που έχει προταθεί μέχρι τώρα.

02 Η πρώτη εφικτή πρόταση αντικαθιστά την άγνωστη τιμή, ακόμη κι αν είναι μεγάλη.

03 Η τελευταία πρόταση δεν είναι απαραίτητα η καλύτερη. Δεν κάνουμε τυφλή αντικατάσταση.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

# 99 Από τον κώδικα στην τιμή

**Η κλήση**

```
example([0, -1, 10], [[1, 3], [2, 5], [2, 7]])
```

↓ Η τιμή που επιστρέφεται

```
[0, 3, 5]
```

## Ακολούθησε μια κλήση

Η `example([0, -1, 10], [[1, 3], [2, 5], [2, 7]])` επιστρέφει `[0, 3, 5]`.

Η `example([0], [])` επιστρέφει `[0]`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([0, -1], [[1, 0], [1, 4]])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `[0, 0]`. Η μηδενική διαδρομή παραμένει καλύτερη από τη μεταγενέστερη πρόταση τέσσερα.

Συνέχισε στην εκφώνηση της άσκησης 99, στη σελίδα 349.

Η ΑΣΚΗΣΗ

# 99 Συντομότερες διαδρομές με βάρη

## Τι θα φτιάξεις

Οι κορυφές είναι `0...size`. Κάθε γραμμή `[from, to, weight]` είναι κατευθυνόμενη ακμή με έγκυρους δείκτες και μη αρνητικό ακέραιο βάρος. Επέστρεψε τις ελάχιστες αποστάσεις από την κορυφή μηδέν, με `-1` για μη προσβάσιμες κορυφές. Με μηδενικές κορυφές οι ακμές είναι κενές και η έξοδος `[]`. Διπλές ακμές και μηδενικά βάρη επιτρέπονται.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `size` είναι από 0 έως 30. Το `edges` έχει από 0 έως 100 γραμμές. Κάθε γραμμή έχει από 3 έως 3 στοιχεία. Οι ακέραιες τιμές του είναι από 0 έως 1000.

**Η κλήση**

```
solve(4, [[0, 1, 5], [0, 2, 1], [2, 1, 1], [1, 3, 2]])
```

↓ Η τιμή που επιστρέφεται

```
[0, 2, 1, 4]
```

## Η ιδέα

Η ουρά BFS δεν αρκεί όταν οι κινήσεις έχουν διαφορετικό κόστος. Οριστικοποίησε κάθε φορά την ακόμη εκκρεμή κορυφή με τη μικρότερη γνωστή απόσταση.

**Στόχος απόδοσης:** `O(v² + e)` χρόνος και `O(v + e)` χώρος. Η σάρωση των υποψηφίων αποφεύγει την ανάγκη heap στα δοσμένα όρια.

ΣΥΝΤΟΜΟΤΕΡΕΣ ΔΙΑΔΡΟΜΕΣ ΜΕ ΒΑΡΗ

# 99 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `size`, `edges`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve(4, [[0, 1, 5], [0, 2, 1], [2, 1, 1], [1, 3, 2]])` | `[0, 2, 1, 4]` |
| `solve(0, [])` | `[]` |
| `solve(1, [])` | `[0]` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/99-dijkstra-distances/exercise.rb.

Tests: exercises/99-dijkstra-distances/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/99-dijkstra-distances/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Μια μη γνωστή απόσταση μπορεί να είναι `nil`. Μετά την επιλογή κορυφής, δοκίμασε αν κάθε εξερχόμενη ακμή βελτιώνει τον προορισμό της.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 465.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

# 100 Ένα σύνολο χωρά στα bits ενός ακεραίου

Θα μετατρέψουμε μια μικρή λίστα μη αρνητικών δεικτών σε bitmask. Το bit κάθε δείκτη δηλώνει αν η τιμή υπάρχει στο σύνολο.

RUBY

```ruby
def example(vertices)
  mask = 0
  vertices.each do |vertex|
    mask |= 1 << vertex
  end
  mask
end
```

## Διάβασε τις γραμμές

01 Το `1 << vertex` μετακινεί το μοναδικό ενεργό bit στη θέση του δείκτη.

02 Το bitwise `|` ενώνει ενεργά bits. Η συντομογραφία `|=` αποθηκεύει την ένωση στο ίδιο όνομα.

03 Ο έλεγχος `(mask & bit) != 0` βρίσκει παρουσία. Η επανεισαγωγή του ίδιου bit δεν αλλάζει το σύνολο.

## Ακολούθησε μια κλήση

Η `example([0, 2])` επιστρέφει `5`.

Η `example([])` επιστρέφει `0`.

## Στάσου λίγο

Τι θα επιστρέψει η `example([1, 1, 3])`;

**Έλεγξε τη σκέψη σου.** Η απάντηση είναι `10`. Τα bits ένα και τρία αντιστοιχούν στις τιμές δύο και οκτώ.

Συνέχισε στην εκφώνηση της άσκησης 100, στη σελίδα 352.

Η ΑΣΚΗΣΗ

# 100 Η φθηνότερη διαδρομή από όλες τις πόλεις

## Τι θα φτιάξεις

Το `costs` είναι τετραγωνικός πίνακας `n` επί `n`, με μηδενική διαγώνιο. Κάθε άλλη τιμή είναι μη αρνητικό κόστος απευθείας μετάβασης, που μπορεί να διαφέρει στην αντίστροφη κατεύθυνση. Ξεκίνα από την πόλη 0, επισκέψου κάθε άλλη πόλη ακριβώς μία φορά και επέστρεψε στην 0. Βρες το ελάχιστο συνολικό κόστος. Για μηδέν ή μία πόλη επέστρεψε `0`.

Μην αλλάξεις τα ορίσματα εισόδου.

**Όρια:** Το `costs` έχει από 0 έως 10 γραμμές. Κάθε γραμμή έχει από 0 έως 10 στοιχεία. Οι γραμμές έχουν ίδιο μήκος. Οι ακέραιες τιμές του είναι από 0 έως 1000.

**Η κλήση**

```
solve([[0, 2, 9], [1, 0, 6], [15, 7, 0]])
```

↓ Η τιμή που επιστρέφεται

```
17
```

## Η ιδέα

Δύο μερικές διαδρομές με το ίδιο σύνολο επισκέψεων και την ίδια τελευταία πόλη έχουν ακριβώς τις ίδιες επιλογές συνέχειας. Αρκεί να κρατήσεις το μικρότερο κόστος ανά τέτοια κατάσταση.

**Στόχος απόδοσης:** $O(n^2 \cdot 2^n)$ χρόνος και $O(n \cdot 2^n)$ χώρος. Το όριο δέκα πόλεων κρατά τον εκθετικό χώρο διαχειρίσιμο.

Η ΦΘΗΝΟΤΕΡΗ ΔΙΑΔΡΟΜΗ ΑΠΟ ΟΛΕΣ ΤΙΣ ΠΟΛΕΙΣ

# 100 Γράψε και έλεγξε

Τα ορίσματα γράφονται με τη σειρά `costs`.

| Κλήση | Αναμενόμενο αποτέλεσμα |
|---|---|
| `solve([[0, 2, 9], [1, 0, 6], [15, 7, 0]])` | `17` |
| `solve([])` | `0` |
| `solve([[0]])` | `0` |

Αποθήκευσε την προσπάθεια. Ξεκίνα από τον φάκελο **ruby**.

Προσπάθεια: exercises/100-minimum-round-trip/exercise.rb.

Tests: exercises/100-minimum-round-trip/exercise_spec.rb.

```sh
bundle exec rspec \
  exercises/100-minimum-round-trip/exercise_spec.rb
```

Το αρχικό TODO προκαλεί αποτυχία. Μετά την υλοποίηση, διάβασε ποιο test απέτυχε και ποια τιμή περίμενε. Οι επιλογές της εντολής εξηγούνται στο κεφάλαιο testing.

## Μικρή βοήθεια

Κωδικοποίησε το σύνολο επισκέψεων με bitmask. Η κατάσταση `dp[mask][last]` τελειώνει στην πόλη `last` και έχει επισκεφτεί ακριβώς το σύνολο `mask`.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
|---|---|
| | |

Η λύση βρίσκεται στη σελίδα 467.

ΜΕΡΟΣ Β

# Οι λύσεις

Σύγκρινε τη σκέψη σου με μια πιθανή λύση.

Μετά κλείσε το βιβλίο και ξαναδοκίμασε μόνος σου.

01 02 03 04 05 06 07 08 09 10

11 12 13 14 15 16 17 18 19 20

Η ΛΥΣΗ

# 01 Μια σταθερή απάντηση

## Η σκέψη

Δεν υπάρχει υπολογισμός. Η συνάρτηση επιστρέφει το συγκεκριμένο κείμενο κάθε φορά που καλείται.

RUBY

```ruby
def solve()
  "Hello!"
end
```

## Διάβασε τον κώδικα

Το `"Hello!"` είναι η τελευταία έκφραση της μεθόδου και επιστρέφεται. Το `end` κλείνει τον ορισμό. Δεν χρειάζεται `puts` ή `return`.

## Πρόσεξε αυτό

Αν εμφανίσεις το μήνυμα στην οθόνη χωρίς να το επιστρέψεις, το test δεν θα πάρει την τιμή που περιμένει.

Επιστροφή στην άσκηση 01 · σελίδα 26

Η ΛΥΣΗ

# 02 Ο ίδιος αριθμός

## Η σκέψη

Επιστρέφουμε την παράμετρο χωρίς να την αλλάξουμε. Η ίδια γραμμή δουλεύει για θετικούς αριθμούς, αρνητικούς και μηδέν.

RUBY

```ruby
def solve(n)
  n
end
```

## Διάβασε τον κώδικα

Το `n` στο σώμα διαβάζει την τιμή της παραμέτρου. Ως τελευταία έκφραση επιστρέφει τον ίδιο αριθμό που δώσαμε στην κλήση.

## Πρόσεξε αυτό

Το `"n"` είναι το γράμμα n ως κείμενο. Το `n` είναι η τιμή της παραμέτρου.

Επιστροφή στην άσκηση 02 · σελίδα 29

Η ΛΥΣΗ

# 03 Ο επόμενος αριθμός

## Η σκέψη

Υπολογίζουμε `n + 1` και επιστρέφουμε το αποτέλεσμα. Για -1, η πρόσθεση δίνει 0.

RUBY

```ruby
def solve(n)
  n + 1
end
```

## Διάβασε τον κώδικα

Το `n + 1` υπολογίζεται πριν επιστραφεί. Δεν αλλάζει την παράμετρο και δεν χρειάζεται ενδιάμεση μεταβλητή.

## Πρόσεξε αυτό

Δεν χρειάζεται να αλλάξεις την παράμετρο ή να γράψεις ξεχωριστό κανόνα για τους αρνητικούς.

Επιστροφή στην άσκηση 03 · σελίδα 32

Η ΛΥΣΗ

# 04 Δύο αριθμοί μαζί

## Η σκέψη

Προσθέτουμε τις δύο παραμέτρους. Δεν χρησιμοποιούμε σταθερούς αριθμούς από τα παραδείγματα.

RUBY

```ruby
def solve(a, b)
  a + b
end
```

## Διάβασε τον κώδικα

Το κόμμα χωρίζει τις παραμέτρους `a` και `b`. Η τελευταία έκφραση επιστρέφει το άθροισμα των τιμών τους.

## Πρόσεξε αυτό

Η λύση πρέπει να δουλεύει με οποιοδήποτε ζευγάρι μέσα στα όρια, όχι μόνο με το πρώτο παράδειγμα.

Επιστροφή στην άσκηση 04 · σελίδα 35

Η ΛΥΣΗ

# 05 Το διπλάσιο

## Η σκέψη

Πολλαπλασιάζουμε το `n` με το 2. Το μηδέν παραμένει μηδέν και οι αρνητικοί αριθμοί παραμένουν αρνητικοί.

RUBY

```ruby
def solve(n)
  n * 2
end
```

## Διάβασε τον κώδικα

Το `*` πολλαπλασιάζει το `n` με το `2`. Η μέθοδος επιστρέφει την τιμή της έκφρασης `n * 2`.

## Πρόσεξε αυτό

Το `n + 2` προσθέτει δύο μονάδες. Δεν δίνει γενικά το διπλάσιο.

Επιστροφή στην άσκηση 05 · σελίδα 38

Η ΛΥΣΗ

# 06 Από λεπτά σε δευτερόλεπτα

## Η σκέψη

Δίνουμε όνομα στην τιμή 60, τα δευτερόλεπτα ενός λεπτού. Πολλαπλασιάζουμε το `minutes` με αυτό το όνομα, αποθηκεύουμε το αποτέλεσμα στο `seconds` και το επιστρέφουμε. Ο υπολογιστής πολλαπλασιάζει τις τιμές των δύο ονομάτων. Ο τελεστής `*` είναι ο ίδιος με εκείνον της προηγούμενης άσκησης.

RUBY

```ruby
def solve(minutes)
  seconds_per_minute = 60
  seconds = minutes * seconds_per_minute
  seconds
end
```

## Διάβασε τον κώδικα

Το `seconds_per_minute = 60` δίνει όνομα στη σταθερή ποσότητα. Το `seconds = ...` αποθηκεύει το γινόμενο. Το τελικό `seconds` επιστρέφει την αποθηκευμένη τιμή.

## Πρόσεξε αυτό

Τα tests ελέγχουν το αριθμητικό αποτέλεσμα. Δεν μπορούν να βεβαιώσουν ότι έδωσες όνομα στην τιμή 60 ή ότι χρησιμοποίησες το `seconds`. Έλεγξέ τα και διαβάζοντας τον κώδικά σου.

Επιστροφή στην άσκηση 06 · σελίδα 41

Η ΛΥΣΗ

# 07 Η διαφορά δύο αριθμών

## Η σκέψη

Επιστρέφουμε `a - b`. Η πρώτη παράμετρος είναι η αρχική τιμή και η δεύτερη αυτή που αφαιρούμε. Αν αφαιρέσουμε έναν αρνητικό αριθμό, το αποτέλεσμα αυξάνεται. Για -4 και -6 έχουμε -4 - (-6) = 2.

RUBY

```ruby
def solve(a, b)
  a - b
end
```

## Διάβασε τον κώδικα

Το `a - b` αφαιρεί τη δεύτερη παράμετρο από την πρώτη. Η μέθοδος επιστρέφει τη διαφορά, ακόμη και όταν είναι αρνητική.

## Πρόσεξε αυτό

Το `b - a` αντιστρέφει τη σειρά και συνήθως αλλάζει το αποτέλεσμα. Δοκίμασε δύο διαφορετικούς αριθμούς για να ελέγξεις τη σειρά.

Επιστροφή στην άσκηση 07 · σελίδα 44

Η ΛΥΣΗ

# 08 Γύρω από το ορθογώνιο

## Η σκέψη

Προσθέτουμε πλάτος και ύψος και πολλαπλασιάζουμε ολόκληρο το άθροισμα με το 2. Ισοδύναμα θα μπορούσαμε να προσθέσουμε και τις τέσσερις πλευρές.

RUBY

```ruby
def solve(width, height)
  2 * (width + height)
end
```

## Διάβασε τον κώδικα

Οι παρενθέσεις κάνουν πρώτα την πρόσθεση `width + height`. Το `2 *` διπλασιάζει ολόκληρο το άθροισμα και επιστρέφεται η περίμετρος.

## Πρόσεξε αυτό

Το `2 * width + height` διπλασιάζει μόνο το πλάτος.

Επιστροφή στην άσκηση 08 · σελίδα 47

Η ΛΥΣΗ

# 09 Τι περισσεύει

## Η σκέψη

Παίρνουμε το υπόλοιπο της διαίρεσης `candies` με `children`. Αν οι καραμέλες μοιράζονται ακριβώς, το αποτέλεσμα είναι 0.

RUBY

```ruby
def solve(candies, children)
  candies % children
end
```

## Διάβασε τον κώδικα

Το `%` δίνει το υπόλοιπο της διαίρεσης `candies` με `children`. Οι είσοδοι έχουν μη αρνητικές καραμέλες και τουλάχιστον ένα παιδί.

## Πρόσεξε αυτό

Η σειρά έχει σημασία. Το υπόλοιπο 8 διά 3 δεν είναι το ίδιο με το υπόλοιπο 3 διά 8. Δεν διαιρούμε ποτέ με μηδέν εδώ.

Επιστροφή στην άσκηση 09 · σελίδα 50

Η ΛΥΣΗ

# 10 Είναι μηδέν;

## Η σκέψη

Συγκρίνουμε το `n` με το 0. Δεν χρειάζεται ακόμη να γράψουμε `if`, γιατί η σύγκριση δίνει ήδη τη σωστή λογική τιμή.

RUBY

```ruby
def solve(n)
  n == 0
end
```

## Διάβασε τον κώδικα

Το `==` συγκρίνει το `n` με το μηδέν. Η μέθοδος επιστρέφει `true` ή `false`, χωρίς εισαγωγικά και χωρίς να χρειάζεται `if`.

## Πρόσεξε αυτό

Η απάντηση είναι λογική τιμή. Τα κείμενα `"true"` και `"false"` είναι διαφορετικός τύπος. Το `=` δεν είναι ο τελεστής ισότητας.

Επιστροφή στην άσκηση 10 · σελίδα 53

Η ΛΥΣΗ

# 11 Είναι θετικός;

## Η σκέψη

Επιστρέφουμε την τιμή της σύγκρισης `n > 0`.

RUBY

```ruby
def solve(n)
  n > 0
end
```

## Διάβασε τον κώδικα

Το `>` δεν περιλαμβάνει την ισότητα. Το `n > 0` επιστρέφει `false` τόσο για το μηδέν όσο και για αρνητικούς αριθμούς.

## Πρόσεξε αυτό

Αν γράψεις `>=`, το μηδέν θα δώσει λάθος απάντηση.

Επιστροφή στην άσκηση 11 · σελίδα 56

Η ΛΥΣΗ

# 12 Έφτασε το όριο;

## Η σκέψη

Συγκρίνουμε τη βαθμολογία με το όριο χρησιμοποιώντας `>=`. Το όριο δίνεται ως παράμετρος και μπορεί να αλλάζει σε κάθε κλήση.

RUBY

```ruby
def solve(score, threshold)
  score >= threshold
end
```

## Διάβασε τον κώδικα

Το `>=` περιλαμβάνει και την ισότητα. Το αποτέλεσμα της σύγκρισης `score >= threshold` επιστρέφεται ως λογική τιμή.

## Πρόσεξε αυτό

Μην γράψεις ένα σταθερό όριο από κάποιο παράδειγμα. Χρησιμοποίησε το `threshold`.

Επιστροφή στην άσκηση 12 · σελίδα 59

Η ΛΥΣΗ

# 13 Ζυγός αριθμός

## Η σκέψη

Υπολογίζουμε το υπόλοιπο με το 2 και το συγκρίνουμε με το 0. Η σύγκριση δίνει τη λογική τιμή που ζητά η εκφώνηση.

RUBY

```ruby
def solve(n)
  n % 2 == 0
end
```

## Διάβασε τον κώδικα

Πρώτα υπολογίζεται το `n % 2`. Το `== 0` ελέγχει αν το υπόλοιπο είναι μηδέν. Επιστρέφεται η σύγκριση, όχι το υπόλοιπο.

## Πρόσεξε αυτό

Μην επιστρέψεις το ίδιο το υπόλοιπο. Το αποτέλεσμα πρέπει να είναι λογική τιμή.

Επιστροφή στην άσκηση 13 · σελίδα 62

Η ΛΥΣΗ

# 14 Απόσταση από το μηδέν

## Η σκέψη

Αν `n < 0`, επιστρέφουμε `-n`. Αλλιώς επιστρέφουμε `n`. Το 0 ανήκει στη δεύτερη περίπτωση και μένει 0.

RUBY

```ruby
def solve(n)
  if n < 0
    -n
  else
    n
  end
end
```

## Διάβασε τον κώδικα

Το `if` δίνει `-n` όταν το `n` είναι αρνητικό και `n` στην άλλη περίπτωση. Το πρώτο `end` κλείνει το `if` και το δεύτερο τη μέθοδο.

## Πρόσεξε αυτό

Αν αλλάξεις το πρόσημο σε κάθε είσοδο, οι θετικοί αριθμοί θα γίνουν αρνητικοί. Τα tests ελέγχουν τη συμπεριφορά, ενώ εσύ ελέγχεις ότι χρησιμοποίησες τη ζητούμενη συνθήκη.

Επιστροφή στην άσκηση 14 · σελίδα 65

Η ΛΥΣΗ

# 15 Ο μεγαλύτερος από τους δύο

## Η σκέψη

Αν `a > b`, επιστρέφουμε `a`. Αλλιώς επιστρέφουμε `b`. Στην ισότητα η δεύτερη επιλογή είναι σωστή, επειδή οι δύο τιμές είναι ίδιες.

RUBY

```ruby
def solve(a, b)
  if a > b
    a
  else
    b
  end
end
```

## Διάβασε τον κώδικα

Ο πρώτος κλάδος δίνει το `a` όταν είναι μεγαλύτερο. Το `else` δίνει το `b`. Η τιμή του επιλεγμένου κλάδου επιστρέφεται από τη μέθοδο.

## Πρόσεξε αυτό

Με αρνητικούς αριθμούς, το -2 είναι μεγαλύτερο από το -7. Δεν συγκρίνουμε τις αποστάσεις τους από το μηδέν.

Επιστροφή στην άσκηση 15 · σελίδα 69

Η ΛΥΣΗ

# 16 Τρεις περιπτώσεις

## Η σκέψη

Πρώτα ελέγχουμε `n < 0`, μετά `n == 0` και τέλος επιστρέφουμε 1. Αν οι δύο πρώτοι έλεγχοι απέτυχαν, ο αριθμός είναι θετικός.

RUBY

```ruby
def solve(n)
  if n < 0
    -1
  elsif n == 0
    0
  else
    1
  end
end
```

## Διάβασε τον κώδικα

Το `elsif n == 0` ελέγχεται μόνο όταν το `n` δεν είναι αρνητικό. Το τελικό `else` καλύπτει τους θετικούς αριθμούς. Επιστρέφεται μία από τις τιμές `-1`, `0`, `1`.

## Πρόσεξε αυτό

Η απάντηση δεν είναι ο ίδιος ο αριθμός. Για είσοδο 42, ζητάμε 1.

Επιστροφή στην άσκηση 16 · σελίδα 73

Η ΛΥΣΗ

# 17 Μέσα στα όρια

## Η σκέψη

Συνδυάζουμε `n >= 0` με `n <= high`. Το αποτέλεσμα είναι αληθές μόνο όταν και οι δύο συγκρίσεις είναι αληθείς.

RUBY

```ruby
def solve(n, high)
  (n >= 0) && (n <= high)
end
```

## Διάβασε τον κώδικα

Το `&&` απαιτεί να πετύχουν και οι δύο συγκρίσεις. Τα `>=` και `<=` περιλαμβάνουν τα άκρα. Η τελική τιμή είναι `true` ή `false`.

## Πρόσεξε αυτό

Το λογικό «ή» θα δεχόταν αριθμούς έξω από τα όρια. Η ισότητα πρέπει να γίνεται δεκτή και στα δύο άκρα.

Επιστροφή στην άσκηση 17 · σελίδα 76

Η ΛΥΣΗ

# 18 Χωρίς υπόλοιπο

## Η σκέψη

Υπολογίζουμε το υπόλοιπο του `n` με το `divisor` και ελέγχουμε αν είναι 0. Ο διαιρέτης δεν είναι ποτέ μηδέν, σύμφωνα με την εκφώνηση.

RUBY

```ruby
def solve(n, divisor)
  n % divisor == 0
end
```

## Διάβασε τον κώδικα

Το `n % divisor` δίνει το υπόλοιπο. Το `== 0` αναγνωρίζει το πολλαπλάσιο. Ο θετικός διαιρέτης εξασφαλίζεται από τα όρια της εκφώνησης.

## Πρόσεξε αυτό

Το 2 δεν είναι πολλαπλάσιο του 5, αλλά το 0 είναι. Η συνάρτηση δεν πρέπει να έχει σταθερό διαιρέτη.

Επιστροφή στην άσκηση 18 · σελίδα 79

Η ΛΥΣΗ

# 19 Η πρώτη λέξη του FizzBuzz

## Η σκέψη

Αν το υπόλοιπο με το 3 είναι 0, επιστρέφουμε `"Fizz"`. Αλλιώς επιστρέφουμε `""`. Ακόμη και το 15 δίνει μόνο Fizz σε αυτή την άσκηση, επειδή υπάρχει μόνο ο κανόνας του 3.

RUBY

```ruby
def solve(n)
  if n % 3 == 0
    "Fizz"
  else
    ""
  end
end
```

## Διάβασε τον κώδικα

Το `if` επιλέγει `"Fizz"` όταν το υπόλοιπο είναι μηδέν. Το `else` δίνει `""`. Η μέθοδος επιστρέφει κείμενο και στις δύο περιπτώσεις.

## Πρόσεξε αυτό

Το `""` δεν έχει κενό χαρακτήρα. Το `" "` έχει έναν και θα αποτύχει στον έλεγχο.

Επιστροφή στην άσκηση 19 · σελίδα 83

Η ΛΥΣΗ

# 20 Οι κανόνες του FizzBuzz

## Η σκέψη

Ελέγχουμε πρώτα τα δύο υπόλοιπα μαζί και επιστρέφουμε FizzBuzz. Έπειτα ελέγχουμε μόνο το 3, μετά μόνο το 5 και τέλος δίνουμε το κενό κείμενο. Το 30 ακολουθεί τον πρώτο κλάδο, το 9 τον δεύτερο, το 10 τον τρίτο και το 7 τον τελευταίο.

RUBY

```ruby
def solve(n)
  if (n % 3 == 0) && (n % 5 == 0)
    "FizzBuzz"
  elsif n % 3 == 0
    "Fizz"
  elsif n % 5 == 0
    "Buzz"
  else
    ""
  end
end
```

## Διάβασε τον κώδικα

Ο πρώτος κλάδος καλύπτει τον κοινό κανόνα με `&&`. Τα `elsif` δοκιμάζουν τους απλούς κανόνες και το `else` δίνει κενό κείμενο. Επιστρέφεται η πρώτη περίπτωση που ταιριάζει.

## Πρόσεξε αυτό

Αν ελέγξεις πρώτα μόνο το 3 και επιστρέψεις αμέσως Fizz, το 15 δεν θα φτάσει ποτέ στον κοινό έλεγχο. Το πλήρες FizzBuzz από το 1 έως το 100 θα προστεθεί αφού μάθουμε επανάληψη.

Επιστροφή στην άσκηση 20 · σελίδα 87

Η ΛΥΣΗ

# 21 Δύο αυξήσεις στην ίδια τιμή

## Η σκέψη

Μια μεταβλητή μπορεί να πάρει νέα τιμή. Στη δεξιά πλευρά διαβάζουμε την προηγούμενη τιμή και μετά αντικαθιστούμε εκείνη που κρατούσε το όνομα.

RUBY

```ruby
def solve(n)
  value = n
  value = value + 1
  value = value + 1
  value
end
```

## Διάβασε τον κώδικα

Το πρώτο `value = value + 1` προσθέτει μία μονάδα. Η επόμενη γραμμή διαβάζει τη νέα τιμή και προσθέτει άλλη μία. Το τελικό `value` επιστρέφεται.

## Πρόσεξε αυτό

Τα tests ελέγχουν την τελική τιμή. Διάβασε και τον κώδικά σου για να βεβαιωθείς ότι χρησιμοποίησες δύο αναθέσεις.

Επιστροφή στην άσκηση 21 · σελίδα 91

Η ΛΥΣΗ

# 22 Ανέβα με βήματα των δύο

## Η σκέψη

Το `while` ελέγχει μια συνθήκη πριν από κάθε επανάληψη. Αν είναι ήδη ψευδής, το σώμα δεν εκτελείται ούτε μία φορά.

RUBY

```ruby
def solve(start, limit)
  value = start
  while value < limit
    value = value + 2
  end
  value
end
```

## Διάβασε τον κώδικα

Η συνθήκη συγκρίνει την τρέχουσα τιμή με το όριο. Κάθε πέρασμα προσθέτει `2`, οπότε η συνθήκη τελικά γίνεται ψευδής. Επιστρέφεται η τιμή μετά το τέλος του βρόχου.

## Πρόσεξε αυτό

Αν ξεχάσεις την ενημέρωση μέσα στο σώμα, η συνθήκη μπορεί να μένει πάντα αληθής. Το `Control-C` σταματά μια εκτέλεση που δεν τελειώνει.

Επιστροφή στην άσκηση 22 · σελίδα 94

Η ΛΥΣΗ

# 23 Άθροισμα από το 1 έως το n

## Η σκέψη

Η μία μεταβλητή δείχνει ποιον αριθμό εξετάζουμε. Η άλλη κρατά το αποτέλεσμα από όλα τα περάσματα που έχουν ήδη γίνει.

RUBY

```ruby
def solve(n)
  total = 0
  number = 1
  while number <= n
    total = total + number
    number = number + 1
  end
  total
end
```

## Διάβασε τον κώδικα

Το `total` ξεκινά έξω από το `while`, ώστε να κρατά όσα προσθέσαμε. Το `number` αυξάνεται σε κάθε πέρασμα. Το `<=` περιλαμβάνει και το `n`.

## Πρόσεξε αυτό

Αν βάλεις το `total = 0` μέσα στον βρόχο, θα χάνεις το προηγούμενο άθροισμα σε κάθε επανάληψη.

Επιστροφή στην άσκηση 23 · σελίδα 98

Η ΛΥΣΗ

# 24 Πόσοι άρτιοι μέχρι το όριο;

## Η σκέψη

Η επανάληψη επισκέπτεται κάθε αριθμό. Η συνθήκη αποφασίζει αν ο συγκεκριμένος αριθμός θα αυξήσει τον μετρητή.

RUBY

```ruby
def solve(n)
  count = 0
  number = 1
  while number <= n
    if number % 2 == 0
      count = count + 1
    end
    number = number + 1
  end
  count
end
```

## Διάβασε τον κώδικα

Το `if` βρίσκεται μέσα στο `while`. Μόνο το `count` ενημερώνεται υπό συνθήκη. Το `number` ενημερώνεται μετά το `if`, για να προχωρά πάντα ο βρόχος.

## Πρόσεξε αυτό

Η ακολουθία αρχίζει στο `1`. Το μηδέν είναι άρτιο, αλλά δεν ανήκει στο διάστημα αυτής της άσκησης.

Επιστροφή στην άσκηση 24 · σελίδα 102

Η ΛΥΣΗ

# 25 Η ίδια κατάθεση πολλές φορές

## Η σκέψη

Όταν ξέρουμε το πλήθος των επαναλήψεων, η Ruby μπορεί να το μετρήσει για εμάς. Το σώμα περιγράφει μόνο τι γίνεται κάθε φορά.

RUBY

```ruby
def solve(count, amount)
  total = 0
  count.times do
    total = total + amount
  end
  total
end
```

## Διάβασε τον κώδικα

Το `count.times` εκτελεί το block ακριβώς `count` φορές. Το τοπικό `total`, που ορίστηκε πριν από το block, ενημερώνεται μέσα του και επιστρέφεται μετά.

## Πρόσεξε αυτό

Η τελευταία γραμμή πρέπει να επιστρέφει το `total`. Η ίδια η `times` επιστρέφει τον αριθμό των επαναλήψεων.

Επιστροφή στην άσκηση 25 · σελίδα 105

Η ΛΥΣΗ

# 26 Δύο τιμές σε έναν πίνακα

## Η σκέψη

Ένας πίνακας είναι μία τιμή που κρατά πολλά στοιχεία με σειρά. Μπορούμε να τον επιστρέψουμε όπως επιστρέφαμε έναν αριθμό.

RUBY

```ruby
def solve(a, b)
  [a, b]
end
```

## Διάβασε τον κώδικα

Οι αγκύλες φτιάχνουν έναν νέο πίνακα. Το πρώτο στοιχείο είναι το `a` και το δεύτερο το `b`. Η σειρά και οι επαναλαμβανόμενες τιμές διατηρούνται.

## Πρόσεξε αυτό

Το `[a, b]` είναι πίνακας. Το `"[a, b]"` θα ήταν ένα κείμενο και θα αποτύγχανε στα tests.

Επιστροφή στην άσκηση 26 · σελίδα 108

Η ΛΥΣΗ

# 27 Το πρώτο στοιχείο

## Η σκέψη

Ο δείκτης είναι ο αριθμός μιας θέσης. Η πρώτη θέση έχει δείκτη `0`, η δεύτερη `1` και η τρίτη `2`.

RUBY

```ruby
def solve(numbers)
  numbers[0]
end
```

## Διάβασε τον κώδικα

Το `numbers[0]` διαβάζει την πρώτη θέση. Δεν αφαιρεί ούτε μετακινεί κανένα στοιχείο.

## Πρόσεξε αυτό

Η είσοδος έχει τουλάχιστον ένα στοιχείο. Η θέση `1` είναι η δεύτερη, όχι η πρώτη.

Επιστροφή στην άσκηση 27 · σελίδα 111

Η ΛΥΣΗ

# 28 Πόσα στοιχεία έχει;

## Η σκέψη

Το πλήθος των στοιχείων είναι διαφορετικό από τις τιμές τους. Ο πίνακας `[100]` έχει ένα στοιχείο.

RUBY

```ruby
def solve(numbers)
  numbers.length
end
```

## Διάβασε τον κώδικα

Το `numbers.length` δίνει έναν ακέραιο. Κάθε θέση μετρά μία φορά, ακόμη και αν πολλές θέσεις έχουν την ίδια τιμή.

## Πρόσεξε αυτό

Το πλήθος ενός μη κενού πίνακα είναι κατά ένα μεγαλύτερο από τον τελευταίο δείκτη του.

Επιστροφή στην άσκηση 28 · σελίδα 114

Η ΛΥΣΗ

# 29 Φτιάξε τους αριθμούς από το 1

## Η σκέψη

Ξεκινάμε με κενό πίνακα. Σε κάθε πέρασμα προσθέτουμε στο τέλος την επόμενη τιμή.

RUBY

```ruby
def solve(n)
  numbers = []
  number = 1
  while number <= n
    numbers << number
    number = number + 1
  end
  numbers
end
```

## Διάβασε τον κώδικα

Το `numbers << number` μεγαλώνει τον πίνακα κατά ένα στοιχείο. Μετά ενημερώνουμε τον μετρητή. Επιστρέφουμε τον πίνακα όταν τελειώσει το `while`.

## Πρόσεξε αυτό

Αν δημιουργείς τον κενό πίνακα μέσα στο `while`, θα χάνεις όσα στοιχεία είχες ήδη προσθέσει.

Επιστροφή στην άσκηση 29 · σελίδα 118

Η ΛΥΣΗ

# 30 Το άθροισμα ενός πίνακα

## Η σκέψη

Δεν χρειάζεται να κρατάμε δείκτη όταν θέλουμε να χρησιμοποιήσουμε κάθε στοιχείο. Η `each` δίνει τις τιμές με τη σειρά.

RUBY

```ruby
def solve(numbers)
  total = 0
  numbers.each do |number|
    total = total + number
  end
  total
end
```

## Διάβασε τον κώδικα

Η `each` δίνει κάθε στοιχείο στο `number`. Το `total` ενημερώνεται και παραμένει διαθέσιμο μετά το block. Η τελευταία γραμμή επιστρέφει το άθροισμα.

## Πρόσεξε αυτό

Η ίδια η `each` επιστρέφει τον αρχικό πίνακα. Για αυτό χρειάζεται η τελική γραμμή `total`.

Επιστροφή στην άσκηση 30 · σελίδα 121

Η ΛΥΣΗ

# 31 Πόσα στοιχεία είναι θετικά;

## Η σκέψη

Η `each` επισκέπτεται όλες τις θέσεις. Το `if` αποφασίζει αν η συγκεκριμένη θέση θα μετρήσει στο αποτέλεσμα.

RUBY

```ruby
def solve(numbers)
  count = 0
  numbers.each do |number|
    if number > 0
      count = count + 1
    end
  end
  count
end
```

## Διάβασε τον κώδικα

Το `number > 0` αποκλείει και το μηδέν. Μετράμε θέσεις, άρα δύο ίδια θετικά στοιχεία αυξάνουν τον μετρητή δύο φορές.

## Πρόσεξε αυτό

Το άθροισμα των θετικών στοιχείων δεν είναι το πλήθος τους. Για `[4, 7]`, ζητάμε `2`.

Επιστροφή στην άσκηση 31 · σελίδα 125

Η ΛΥΣΗ

# 32 Διπλασίασε κάθε στοιχείο

## Η σκέψη

Η `map` φτιάχνει ένα αποτέλεσμα για κάθε αρχικό στοιχείο. Το πλήθος και η σειρά των θέσεων παραμένουν ίδια.

RUBY

```ruby
def solve(numbers)
  numbers.map do |number|
    number * 2
  end
end
```

## Διάβασε τον κώδικα

Το block δίνει το `number * 2`. Η `map` συγκεντρώνει αυτές τις τιμές σε νέο πίνακα, ο οποίος επιστρέφεται από τη μέθοδο.

## Πρόσεξε αυτό

Μη χρησιμοποιήσεις `map!`. Η εκδοχή με θαυμαστικό αλλάζει τον αρχικό πίνακα.

Επιστροφή στην άσκηση 32 · σελίδα 128

Η ΛΥΣΗ

# 33 Κράτα μόνο τους άρτιους

## Η σκέψη

Η `select` αποφασίζει αν θα κρατήσει κάθε στοιχείο. Κρατά την αρχική τιμή, όχι το αποτέλεσμα της σύγκρισης.

RUBY

```ruby
def solve(numbers)
  numbers.select do |number|
    number % 2 == 0
  end
end
```

## Διάβασε τον κώδικα

Το `number % 2 == 0` είναι αληθές για τους άρτιους, συμπεριλαμβανομένου του μηδενός. Η `select` κρατά αυτές τις τιμές με την ίδια σειρά.

## Πρόσεξε αυτό

Με `map` θα έπαιρνες πίνακα λογικών τιμών. Η `select` επιστρέφει τα στοιχεία που πέρασαν τον έλεγχο.

Επιστροφή στην άσκηση 33 · σελίδα 131

Η ΛΥΣΗ

# 34 Ένας αριθμός γίνεται κείμενο

## Η σκέψη

Ο ίδιος αριθμός μπορεί να αναπαρασταθεί ως κείμενο για να συμμετέχει σε ένα μήνυμα. Η αριθμητική τιμή και το κείμενο είναι διαφορετικοί τύποι.

RUBY

```ruby
def solve(n)
  n.to_s
end
```

## Διάβασε τον κώδικα

Το `n.to_s` δημιουργεί κείμενο με τα ψηφία και, όταν χρειάζεται, το πρόσημο. Η παράμετρος παραμένει αριθμός.

## Πρόσεξε αυτό

Το `"n"` είναι το γράμμα n. Δεν μετατρέπει την τιμή της παραμέτρου.

Επιστροφή στην άσκηση 34 · σελίδα 134

Η ΛΥΣΗ

# 35 Ένα μήνυμα με αριθμό

## Η σκέψη

Το `+` μπορεί να ενώνει δύο κείμενα. Πρώτα μετατρέπουμε τον αριθμό σε κείμενο, ώστε οι δύο πλευρές να έχουν κατάλληλο τύπο.

RUBY

```ruby
def solve(n)
  "Number: " + n.to_s
end
```

## Διάβασε τον κώδικα

Το αριστερό κείμενο δίνει την ετικέτα και το κενό. Το `n.to_s` δίνει τα ψηφία. Το `+` επιστρέφει την ένωσή τους.

## Πρόσεξε αυτό

Το `"Number: " + n` προσπαθεί να ενώσει κείμενο με ακέραιο και προκαλεί σφάλμα τύπου.

Επιστροφή στην άσκηση 35 · σελίδα 137

Η ΛΥΣΗ

# 36 Μία γραμμή για κάθε κείμενο

## Η σκέψη

Η `join` βάζει το διαχωριστικό ανάμεσα στα στοιχεία. Για μηδέν στοιχεία δίνει κενό κείμενο και για ένα στοιχείο δεν χρειάζεται διαχωριστικό.

RUBY

```ruby
def solve(words)
  words.join("\n")
end
```

## Διάβασε τον κώδικα

Η `words.join("\n")` ενώνει τις τιμές με αλλαγή γραμμής μόνο ανάμεσα σε διαδοχικά στοιχεία. Τα άδεια κείμενα εξακολουθούν να καταλαμβάνουν θέση.

## Πρόσεξε αυτό

Αν το τελευταίο στοιχείο είναι κενό, το αποτέλεσμα μπορεί να τελειώνει σε αλλαγή γραμμής λόγω του διαχωριστικού. Μην προσθέτεις άλλη χειροκίνητα.

Επιστροφή στην άσκηση 36 · σελίδα 140

Η ΛΥΣΗ

# 37 Ένα συνεχόμενο διάστημα

## Η σκέψη

Το διάστημα περιγράφει την αρχή και το τέλος. Η `to_a` συγκεντρώνει τις ακέραιες τιμές του σε πίνακα.

```ruby
def solve(n)
  (1..n).to_a
end
```

## Διάβασε τον κώδικα

Το `(1..n)` περιγράφει το διάστημα. Η `.to_a` το μετατρέπει σε πίνακα. Με `n = 0`, η αρχή είναι μεγαλύτερη από το τέλος και ο πίνακας είναι κενός.

## Πρόσεξε αυτό

Το `..` έχει δύο τελείες. Η σύνταξη με τρεις τελείες έχει διαφορετικό κανόνα για το τελευταίο στοιχείο.

Επιστροφή στην άσκηση 37 · σελίδα 143

Η ΛΥΣΗ

# 38 Βάλε Fizz στις σωστές θέσεις

## Η σκέψη

Το block της `map` μπορεί να επιλέγει τιμή με `if`. Όποιος κλάδος εκτελεστεί δίνει το κείμενο της συγκεκριμένης θέσης.

RUBY

```ruby
def solve(numbers)
  numbers.map do |number|
    if number % 3 == 0
      "Fizz"
    else
      number.to_s
    end
  end
end
```

## Διάβασε τον κώδικα

Κάθε πέρασμα επιστρέφει είτε `"Fizz"` είτε το `number.to_s`. Η `map` συγκεντρώνει πάντα κείμενα, κρατώντας τη σειρά.

## Πρόσεξε αυτό

Η άσκηση 19 επέστρεφε κενό κείμενο για τους υπόλοιπους αριθμούς. Εδώ θέλουμε να φαίνεται ο αριθμός.

Επιστροφή στην άσκηση 38 · σελίδα 147

Η ΛΥΣΗ

# 39 FizzBuzz για κάθε στοιχείο

## Η σκέψη

Οι κανόνες της άσκησης 20 εφαρμόζονται τώρα σε κάθε στοιχείο. Η πρώτη επιτυχημένη συνθήκη καθορίζει την τιμή της συγκεκριμένης θέσης.

RUBY

```ruby
def solve(numbers)
  numbers.map do |number|
    if (number % 3 == 0) && (number % 5 == 0)
      "FizzBuzz"
    elsif number % 3 == 0
      "Fizz"
    elsif number % 5 == 0
      "Buzz"
    else
      number.to_s
    end
  end
end
```

## Διάβασε τον κώδικα

Η `map` εκτελεί την αλυσίδα για κάθε `number`. Το `&&` αναγνωρίζει την κοινή περίπτωση πριν από τους απλούς κανόνες. Όλα τα αποτελέσματα είναι κείμενα.

## Πρόσεξε αυτό

Μην αλλάζεις τη σειρά του πίνακα και μην παραλείπεις στοιχεία. Κάθε αρχική θέση πρέπει να έχει ακριβώς μία απάντηση.

Επιστροφή στην άσκηση 39 · σελίδα 151

Η ΛΥΣΗ

# 40 Το πλήρες FizzBuzz

## Η σκέψη

Χτίζουμε την ακολουθία των αριθμών, δίνουμε μία ετικέτα σε κάθε θέση και ενώνουμε τα κείμενα. Στους αριθμούς χωρίς λέξη κρατάμε τα ψηφία τους.

RUBY

```ruby
def solve(n)
  labels = (1..n).map do |number|
    if (number % 3 == 0) && (number % 5 == 0)
      "FizzBuzz"
    elsif number % 3 == 0
      "Fizz"
    elsif number % 5 == 0
      "Buzz"
    else
      number.to_s
    end
  end
  labels.join("\n")
end
```

## Διάβασε τον κώδικα

Η `(1..n).map` εφαρμόζει τους γνωστούς κανόνες σε όλους τους αριθμούς του διαστήματος. Το `labels` κρατά πίνακα κειμένων και η `join("\n")` δίνει το τελικό κείμενο.

## Πρόσεξε αυτό

Η μέθοδος επιστρέφει το κείμενο. Το RSpec ελέγχει την τιμή που επιστρέφει. Μην προσθέσεις `puts` μέσα στη λύση σου.

Επιστροφή στην άσκηση 40 · σελίδα 155

Η ΛΥΣΗ

# 41 Όλα τα ενδιάμεσα αθροίσματα

## Η σκέψη

Κράτα το άθροισμα όσων στοιχείων έχουν ήδη περάσει. Κάθε νέο στοιχείο το ενημερώνει μία φορά και δημιουργεί μία νέα απάντηση.

RUBY

```ruby
def solve(numbers)
  total = 0
  numbers.map do |number|
    total += number
  end
end
```

## Διάβασε τον κώδικα

Μετά τη θέση `i`, το `total` ισούται με το άθροισμα του αντίστοιχου προθέματος. Η ανάθεση επιστρέφει τη νέα τιμή, οπότε η `map` τη γράφει στον νέο πίνακα. Αυτή η ιδιότητα ισχύει αρχικά για το κενό πρόθεμα και διατηρείται σε κάθε βήμα.

**Κόστος:** `O(n)` χρόνος και `O(n)` χώρος για την έξοδο, με `O(1)` πρόσθετο χώρο.

## Πρόσεξε αυτό

Οι αρνητικοί αριθμοί μπορούν να μειώσουν το τρέχον άθροισμα. Δεν ψάχνουμε μόνο τα μεγαλύτερα αθροίσματα.

Επιστροφή στην άσκηση 41 · σελίδα 162

Η ΛΥΣΗ

# 42 Πολλά αθροίσματα διαστημάτων

## Η σκέψη

Αποθήκευσε μία φορά τα αθροίσματα προθεμάτων. Το ζητούμενο διάστημα είναι η διαφορά ανάμεσα σε δύο τέτοια αθροίσματα.

RUBY

```ruby
def solve(numbers, queries)
  prefix = [0]
  numbers.each { |number| prefix << prefix[-1] + number }
  queries.map do |left, right|
    prefix[right + 1] - prefix[left]
  end
end
```

## Διάβασε τον κώδικα

Το `prefix[i]` μετρά τα πρώτα `i` στοιχεία. Αφαιρώντας τα πρώτα `left` από τα πρώτα `right + 1`, μένουν ακριβώς όσα ζητά η ερώτηση. Τα `|left, right|` μοιράζουν τα δύο στοιχεία κάθε μικρού πίνακα σε δύο ονόματα.

**Κόστος:** `O(n + q)` χρόνος και `O(n + q)` χώρος μαζί με την έξοδο, όπου `q` οι ερωτήσεις.

## Πρόσεξε αυτό

Το δεξί άκρο περιλαμβάνεται. Γι' αυτό διαβάζουμε το πρόθεμα στη θέση `right + 1`.

Επιστροφή στην άσκηση 42 · σελίδα 165

Η ΛΥΣΗ

# 43 Το γινόμενο όλων των άλλων

## Η σκέψη

Κάθε απάντηση χωρίζεται σε ένα γινόμενο αριστερά και ένα δεξιά. Δύο περάσματα αρκούν για να τα συνδυάσουν.

RUBY

```ruby
def solve(numbers)
  result = Array.new(numbers.length, 1)
  left = 1
  numbers.each_with_index do |number, i|
    result[i] = left
    left *= number
  end
  right = 1
  (numbers.length - 1).downto(0) do |i|
    result[i] *= right
    right *= numbers[i]
  end
  result
end
```

## Διάβασε τον κώδικα

Πριν ενημερωθεί το `left`, περιέχει μόνο τις θέσεις πριν από την `i`. Αντίστοιχα, το `right` περιέχει μόνο τις θέσεις μετά την `i`. Η ίδια η τιμή δεν συμμετέχει στην απάντησή της. Η έξοδος χρησιμοποιείται και ως προσωρινή αποθήκευση.

**Κόστος:** `O(n)` χρόνος, `O(n)` χώρος εξόδου και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Τα μηδενικά καταρρίπτουν τη λύση «συνολικό γινόμενο διά το στοιχείο». Με δύο μηδενικά κάθε απάντηση είναι μηδέν.

Επιστροφή στην άσκηση 43 · σελίδα 169

Η ΛΥΣΗ

# 44 Περιστροφή χωρίς χαμένες τιμές

## Η σκέψη

Μια πλήρης περιστροφή δεν αλλάζει τίποτα. Αφού μικρύνει το πλήθος βημάτων, κάθε αρχική θέση έχει μία ακριβή τελική θέση.

RUBY

```ruby
def solve(numbers, steps)
  return [] if numbers.empty?
  size = numbers.length
  shift = steps % size
  result = Array.new(size)
  numbers.each_with_index do |number, i|
    result[(i + shift) % size] = number
  end
  result
end
```

## Διάβασε τον κώδικα

Η μετατόπιση με modulo αντιστοιχίζει κάθε θέση σε διαφορετική θέση προορισμού. Έτσι γράφονται όλες οι θέσεις ακριβώς μία φορά. Δεν χρειάζονται διαδοχικές μετακινήσεις ολόκληρου του πίνακα.

**Κόστος:** `O(n)` χρόνος και `O(n)` χώρος για την έξοδο, ανεξάρτητα από το μέγεθος του `steps`.

## Πρόσεξε αυτό

Έλεγξε το κενό input πριν από το modulo. Η διαίρεση με μηδενικό μήκος δεν ορίζεται.

Επιστροφή στην άσκηση 44 · σελίδα 172

Η ΛΥΣΗ

# 45 Συγχώνευση δύο ταξινομημένων πινάκων

## Η σκέψη

Το μικρότερο διαθέσιμο στοιχείο βρίσκεται πάντα σε ένα από τα δύο μέτωπα. Αφού επιλεγεί, προχωρά μόνο ο δικός του δείκτης.

RUBY

```ruby
def solve(left, right)
  result = []
  i = 0
  j = 0
  while i < left.length || j < right.length
    if j == right.length ||
       (i < left.length && left[i] <= right[j])
      result << left[i]
      i += 1
    else
      result << right[j]
      j += 1
    end
  end
  result
end
```

## Διάβασε τον κώδικα

Πριν από κάθε επανάληψη, η έξοδος είναι ταξινομημένη και περιέχει ακριβώς όσα έχουμε καταναλώσει. Το μικρότερο από τα δύο διαθέσιμα στοιχεία διατηρεί αυτή την ιδιότητα. Το `||` αποφεύγει τη σύγκριση όταν ο δεξιός πίνακας έχει τελειώσει.

**Κόστος:** `O(n + m)` χρόνος και `O(n + m)` χώρος εξόδου.

## Πρόσεξε αυτό

Οι ίσες τιμές δεν αφαιρούνται. Η συγχώνευση διατηρεί το συνολικό πλήθος στοιχείων.

Επιστροφή στην άσκηση 45 · σελίδα 175

Η ΛΥΣΗ

# 46 Τα μηδενικά στο τέλος

## Η σκέψη

Διαβάζουμε όλες τις θέσεις, αλλά γράφουμε μπροστά μόνο τις μη μηδενικές τιμές. Οι υπόλοιπες θέσεις της νέας εξόδου μένουν μηδενικές.

RUBY

```ruby
def solve(numbers)
  result = Array.new(numbers.length, 0)
  write = 0
  numbers.each do |number|
    next if number == 0
    result[write] = number
    write += 1
  end
  result
end
```

## Διάβασε τον κώδικα

Ο δείκτης `write` μετρά πόσες μη μηδενικές τιμές είδαμε. Καθώς αυτές αντιγράφονται με τη σειρά ανάγνωσης, η σχετική τους σειρά διατηρείται. Το `next` παραλείπει μόνο το τρέχον πέρασμα του block.

**Κόστος:** `O(n)` χρόνος και `O(n)` χώρος εξόδου, με `O(1)` πρόσθετο χώρο.

## Πρόσεξε αυτό

Το `false` και το `nil` είναι ψευδή στη Ruby, αλλά το `0` είναι αληθές. Έλεγξε ρητά `number == 0`.

Επιστροφή στην άσκηση 46 · σελίδα 178

Η ΛΥΣΗ

# 47 Η μεγαλύτερη ανοδική διαδρομή

## Η σκέψη

Κράτα χωριστά το μήκος της διαδρομής που τελειώνει τώρα και το καλύτερο μήκος που έχει εμφανιστεί οπουδήποτε.

RUBY

```ruby
def solve(numbers)
  return 0 if numbers.empty?
  current = 1
  best = 1
  (1...numbers.length).each do |i|
    current = numbers[i] > numbers[i - 1] ? current + 1 : 1
    best = [best, current].max
  end
  best
end
```

## Διάβασε τον κώδικα

Η `current` περιγράφει μόνο το τμήμα που τελειώνει στην τρέχουσα θέση. Αυτό είτε επεκτείνει το προηγούμενο είτε ξεκινά από την αρχή. Η `best` καταγράφει το μέγιστο όλων αυτών των υποψήφιων τμημάτων.

**Κόστος:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Αυτό είναι συνεχόμενο τμήμα. Η μη συνεχόμενη αύξουσα υποακολουθία θα εμφανιστεί στην άσκηση 97.

Επιστροφή στην άσκηση 47 · σελίδα 181

Η ΛΥΣΗ

# 48 Μία αγορά και μία πώληση

## Η σκέψη

Για κάθε πιθανή ημέρα πώλησης αρκεί να ξέρουμε τη μικρότερη τιμή που προηγήθηκε. Δεν χρειάζεται να δοκιμάσουμε κάθε ζεύγος ημερών.

RUBY

```ruby
def solve(prices)
  return 0 if prices.length < 2
  cheapest = prices[0]
  best = 0
  (1...prices.length).each do |i|
    price = prices[i]
    best = [best, price - cheapest].max
    cheapest = [cheapest, price].min
  end
  best
end
```

## Διάβασε τον κώδικα

Στην αρχή κάθε περάσματος, το `cheapest` είναι η χαμηλότερη προηγούμενη τιμή. Η καλύτερη συναλλαγή που τελειώνει σήμερα έχει κέρδος `price - cheapest`. Η `best` αρχίζει από μηδέν ώστε η αποχή να παραμένει διαθέσιμη επιλογή.

**Κόστος:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η μεγαλύτερη διαφορά δύο τιμών δεν αρκεί. Η ημέρα αγοράς πρέπει να έρχεται πρώτη.

Επιστροφή στην άσκηση 48 · σελίδα 184

Η ΛΥΣΗ

# 49 Το πιο κερδοφόρο συνεχόμενο τμήμα

## Η σκέψη

Το καλύτερο τμήμα που τελειώνει σε μία θέση είτε αρχίζει εκεί είτε συνεχίζει το καλύτερο τμήμα που τελείωνε αμέσως πριν.

RUBY

```ruby
def solve(numbers)
  ending = numbers[0]
  best = ending
  (1...numbers.length).each do |i|
    ending = [numbers[i], ending + numbers[i]].max
    best = [best, ending].max
  end
  best
end
```

## Διάβασε τον κώδικα

Η κατάσταση `ending` περιγράφει όλα τα τμήματα που τελειώνουν στην τρέχουσα θέση μέσω της καλύτερης τιμής τους. Ένα αρνητικό προηγούμενο άθροισμα δεν βοηθά την επέκταση, οπότε κερδίζει το νέο ξεκίνημα. Αυτό είναι το βασικό βήμα του αλγορίθμου Kadane.

**Κόστος:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Μην αρχικοποιήσεις τη συνολική απάντηση από μηδέν. Σε πίνακα μόνο αρνητικών αριθμών αυτό θα επέτρεπε ένα απαγορευμένο κενό τμήμα.

Επιστροφή στην άσκηση 49 · σελίδα 187

Η ΛΥΣΗ

# 50 Μια θέση με ίσα βάρη

## Η σκέψη

Αφού γνωρίζουμε το συνολικό άθροισμα και την αριστερή πλευρά, η δεξιά προκύπτει με δύο αφαιρέσεις.

RUBY

```ruby
def solve(numbers)
  total = numbers.sum
  left = 0
  numbers.each_with_index do |number, i|
    right = total - left - number
    return i if left == right
    left += number
  end
  -1
end
```

## Διάβασε τον κώδικα

Πριν από τον έλεγχο, το `left` περιέχει μόνο προηγούμενες θέσεις. Η διαφορά `total - left - number` περιέχει μόνο επόμενες. Επειδή οι δείκτες εξετάζονται από αριστερά, η πρώτη επιστροφή είναι και ο μικρότερος έγκυρος δείκτης.

**Κόστος:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Μπορεί να είναι σωστή η πρώτη ή η τελευταία θέση. Αρνητικές τιμές επιτρέπουν επίσης πολλές σωστές θέσεις.

Επιστροφή στην άσκηση 50 · σελίδα 190

Η ΛΥΣΗ

# 51 Παλίνδρομο μέσα στον θόρυβο

## Η σκέψη

Πρώτα απομάκρυνε τις διαφορές που η εκφώνηση θεωρεί αδιάφορες. Μετά σύγκρινε συμμετρικές θέσεις από τα άκρα προς το κέντρο.

RUBY

```ruby
def solve(text)
  clean = text.downcase.gsub(/[^a-z0-9]/, "")
  left = 0
  right = clean.length - 1
  while left < right
    return false if clean[left] != clean[right]
    left += 1
    right -= 1
  end
  true
end
```

## Διάβασε τον κώδικα

Κάθε επιτυχής σύγκριση αφαιρεί ένα ζεύγος άκρων από το υπόλοιπο πρόβλημα. Όταν οι δείκτες συναντηθούν ή περάσουν ο ένας τον άλλον, δεν απομένει ζεύγος που θα μπορούσε να διαφωνεί. Η κανονικοποίηση δημιουργεί νέο κείμενο.

**Κόστος:** `O(n)` χρόνος και `O(n)` πρόσθετος χώρος για το καθαρό κείμενο.

## Πρόσεξε αυτό

Το αρχικό κείμενο πρέπει να μείνει ίδιο. Η `gsub!` ή η `downcase!` μπορεί να το μεταβάλει.

Επιστροφή στην άσκηση 51 · σελίδα 194

Η ΛΥΣΗ

# 52 Συμπίεση διαδοχικών χαρακτήρων

## Η σκέψη

Για κάθε αρχή ομάδας βρες την πρώτη διαφορετική θέση. Η διαφορά των δύο δεικτών δίνει το πλήθος χωρίς ξεχωριστό μετρητή.

RUBY

```ruby
def solve(text)
  pieces = []
  left = 0
  while left < text.length
    right = left + 1
    right += 1 while right < text.length && text[right] == text[left]
    pieces << text[left] + (right - left).to_s
    left = right
  end
  pieces.join
end
```

## Διάβασε τον κώδικα

Οι ομάδες χωρίζουν το κείμενο σε ξένα μεταξύ τους συνεχόμενα τμήματα. Ο εσωτερικός βρόχος δεν ξαναδιαβάζει προηγούμενη ομάδα. Η έξοδος συγκεντρώνεται ως κομμάτια και ενώνεται στο τέλος.

**Κόστος:** `O(n)` χρόνος και `O(n)` χώρος μαζί με την έξοδο.

## Πρόσεξε αυτό

Το ίδιο γράμμα σε δύο χωριστές ομάδες καταγράφεται δύο φορές. Δεν ζητείται συνολικό histogram.

Επιστροφή στην άσκηση 52 · σελίδα 197

Η ΛΥΣΗ

# 53 Ξαναφτιάξε το συμπιεσμένο κείμενο

## Η σκέψη

Χώρισε την είσοδο σε ζεύγη γράμματος και πλήθους. Κάθε ζεύγος παράγει ανεξάρτητα ένα κομμάτι του αποτελέσματος.

RUBY

```ruby
def solve(encoded)
  encoded.scan(/([a-z])([1-9][0-9]*)/).map do |letter, count|
    letter * count.to_i
  end.join
end
```

## Διάβασε τον κώδικα

Η `scan` επιστρέφει όλες τις ομάδες στην αρχική σειρά. Το `count` είναι ακόμη κείμενο, οπότε μετατρέπεται με `to_i`. Ο πολλαπλασιασμός ενός κειμένου με ακέραιο επαναλαμβάνει το κείμενο. Η λύση βασίζεται στη ρητή προϋπόθεση έγκυρης κωδικοποίησης.

**Κόστος:** `O(n + r)` χρόνος και χώρος, όπου `r` το μήκος του αναπτυγμένου κειμένου.

## Πρόσεξε αυτό

Το πλήθος μπορεί να έχει πάνω από ένα ψηφίο. Το `a12` σημαίνει δώδεκα γράμματα, όχι δύο χωριστούς αριθμούς.

Επιστροφή στην άσκηση 53 · σελίδα 200

Η ΛΥΣΗ

# 54 Η συχνότερη λέξη

## Η σκέψη

Αποθήκευσε έναν μετρητή ανά διαφορετική λέξη. Η τελική επιλογή έχει δύο κριτήρια: μεγαλύτερη συχνότητα και μικρότερη λέξη.

RUBY

```ruby
def solve(words)
  counts = Hash.new(0)
  words.each { |word| counts[word] += 1 }
  counts.keys.min_by do |word|
    [-counts[word], word]
  end || ""
end
```

## Διάβασε τον κώδικα

Το αρνητικό πλήθος κάνει τη μεγαλύτερη συχνότητα μικρότερο πρώτο κριτήριο. Αν δύο πρώτα κριτήρια είναι ίσα, συγκρίνονται οι ίδιες οι λέξεις. Η `min_by` σε κενή συλλογή δίνει `nil`, που αντικαθίσταται με κενό κείμενο από το `||`.

**Κόστος:** `O(n)` αναμενόμενος χρόνος και `O(u)` χώρος για `u` διαφορετικές λέξεις, με φραγμένο μήκος λέξης.

## Πρόσεξε αυτό

Η σειρά εμφάνισης δεν λύνει την ισοβαθμία. Η λέξη που εμφανίστηκε πρώτη μπορεί να μην είναι η ζητούμενη.

Επιστροφή στην άσκηση 54 · σελίδα 203

Η ΛΥΣΗ

# 55 Ο πρώτος μοναδικός χαρακτήρας

## Η σκέψη

Το πρώτο πέρασμα μαθαίνει όλες τις συχνότητες. Το δεύτερο διατηρεί την αρχική σειρά και βρίσκει την πρώτη μοναδική θέση.

RUBY

```ruby
def solve(text)
  chars = text.chars
  counts = Hash.new(0)
  chars.each { |char| counts[char] += 1 }
  chars.each_with_index do |char, i|
    return i if counts[char] == 1
  end
  -1
end
```

## Διάβασε τον κώδικα

Οι συχνότητες είναι οριστικές πριν αρχίσει η αναζήτηση. Το `each_with_index` πάνω στον πίνακα χαρακτήρων δίνει τη σωστή μονάδα μέτρησης ακόμη και για ελληνικά γράμματα. Η έγκαιρη επιστροφή επιλέγει την πρώτη έγκυρη θέση.

**Κόστος:** `O(n)` αναμενόμενος χρόνος και `O(n)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η σειρά των κλειδιών ενός Hash δεν αντικαθιστά τον δείκτη χαρακτήρα. Χρειαζόμαστε θέση στην αρχική ακολουθία.

Επιστροφή στην άσκηση 55 · σελίδα 206

Η ΛΥΣΗ

# 56 Ίδιοι χαρακτήρες, άλλη σειρά

## Η σκέψη

Δες το ένα κείμενο ως προσθήκες σε μετρητές και το άλλο ως αφαιρέσεις. Η ισότητα σημαίνει ότι δεν περισσεύει τίποτα.

RUBY

```ruby
def solve(left, right)
  counts = Hash.new(0)
  left.each_char { |char| counts[char] += 1 }
  right.each_char { |char| counts[char] -= 1 }
  counts.values.all? { |count| count == 0 }
end
```

## Διάβασε τον κώδικα

Κάθε τελικός μετρητής είναι η διαφορά των εμφανίσεων του χαρακτήρα στα δύο κείμενα. Όλες οι διαφορές είναι μηδέν ακριβώς όταν τα πολυσύνολα είναι ίσα. Η `all?` στο κενό Hash επιστρέφει `true`, όπως χρειάζεται για δύο κενά κείμενα.

**Κόστος:** `O(n + m)` αναμενόμενος χρόνος και `O(u)` χώρος για τους διαφορετικούς χαρακτήρες.

## Πρόσεξε αυτό

Δεν αρκεί να είναι ίδια τα σύνολα χαρακτήρων. Το `aab` και το `abb` έχουν διαφορετικές συχνότητες.

Επιστροφή στην άσκηση 56 · σελίδα 209

Η ΛΥΣΗ

# 57 Κοινά στοιχεία χωρίς διπλότυπα

## Η σκέψη

Χρειάζεται έλεγχος παρουσίας, όχι συχνότητας. Κράτα ένα σύνολο τιμών από τον πρώτο πίνακα και ένα δεύτερο για τις κοινές τιμές.

RUBY

```ruby
def solve(left, right)
  present = {}
  left.each { |value| present[value] = true }
  common = {}
  right.each do |value|
    common[value] = true if present.key?(value)
  end
  common.keys.sort
end
```

## Διάβασε τον κώδικα

Η `key?` απαντά αν υπάρχει κλειδί, ανεξάρτητα από την τιμή του. Η επανεγγραφή του ίδιου κλειδιού στο `common` δεν προσθέτει δεύτερο στοιχείο. Η ταξινόμηση γίνεται μόνο πάνω στις διαφορετικές κοινές τιμές.

**Κόστος:** `O(n + m + u log u)` αναμενόμενος χρόνος και `O(n + u)` χώρος, όπου `u` οι κοινές διαφορετικές τιμές.

## Πρόσεξε αυτό

Η σειρά της εξόδου είναι μέρος της εκφώνησης. Μην επιστρέψεις απλώς τα κλειδιά με τη σειρά που βρέθηκαν.

Επιστροφή στην άσκηση 57 · σελίδα 212

Η ΛΥΣΗ

# 58 Η μεγαλύτερη ακολουθία διαδοχικών τιμών

## Η σκέψη

Μια τιμή είναι αρχή ακολουθίας μόνο αν λείπει η αμέσως προηγούμενη. Ξεκινώντας μόνο από τέτοιες αρχές, αποφεύγεις να ξαναμετράς το ίδιο τμήμα.

RUBY

```ruby
def solve(numbers)
  present = {}
  numbers.each { |number| present[number] = true }
  best = 0
  present.each_key do |number|
    next if present.key?(number - 1)
    finish = number
    finish += 1 while present.key?(finish)
    best = [best, finish - number].max
  end
  best
end
```

## Διάβασε τον κώδικα

Κάθε διαφορετική τιμή συμμετέχει στην επέκταση μόνο της μοναδικής αρχής της ακολουθίας της. Οι υπόλοιποι εξωτερικοί έλεγχοι σταματούν αμέσως. Έτσι ο εσωτερικός βρόχος συνολικά παραμένει γραμμικός.

**Κόστος:** `O(n)` αναμενόμενος χρόνος και `O(n)` χώρος.

## Πρόσεξε αυτό

Αν επεκτείνεις από κάθε τιμή, ένας ήδη διαδοχικός πίνακας προκαλεί τετραγωνικό αριθμό ελέγχων.

Επιστροφή στην άσκηση 58 · σελίδα 215

Η ΛΥΣΗ

# 59 Δύο θέσεις με δοσμένο άθροισμα

## Η σκέψη

Όταν εξετάζεις μια νέα τιμή, ξέρεις ακριβώς ποια προηγούμενη τιμή θα συμπλήρωνε τον στόχο. Αποθήκευσε την πρώτη θέση κάθε τιμής.

RUBY

```ruby
def solve(numbers, target)
  first = {}
  numbers.each_with_index do |number, j|
    wanted = target - number
    return [first[wanted], j] if first.key?(wanted)
    first[number] = j unless first.key?(number)
  end
  []
end
```

## Διάβασε τον κώδικα

Το Hash περιέχει μόνο θέσεις μικρότερες από το τρέχον `j`, άρα αποκλείεται η επαναχρησιμοποίηση θέσης. Η σειρά του βρόχου επιλέγει το μικρότερο δεξί άκρο και η διατήρηση της πρώτης εμφάνισης επιλέγει το μικρότερο αριστερό.

**Κόστος:** `O(n)` αναμενόμενος χρόνος και `O(n)` χώρος.

## Πρόσεξε αυτό

Μην αντικαθιστάς την πρώτη θέση ενός διπλότυπου. Είναι αυτή που κερδίζει την ισοβαθμία.

Επιστροφή στην άσκηση 59 · σελίδα 218

Η ΛΥΣΗ

# 60 Πόσα τμήματα έχουν το σωστό άθροισμα;

## Η σκέψη

Ένα τμήμα έχει το ζητούμενο άθροισμα όταν δύο προθέματα διαφέρουν κατά `target`. Για κάθε νέο πρόθεμα χρειάζεσαι πόσα κατάλληλα παλιότερα προθέματα υπάρχουν.

RUBY

```ruby
def solve(numbers, target)
  seen = Hash.new(0)
  seen[0] = 1
  prefix = 0
  answer = 0
  numbers.each do |number|
    prefix += number
    answer += seen[prefix - target]
    seen[prefix] += 1
  end
  answer
end
```

## Διάβασε τον κώδικα

Πριν από την αύξηση του `seen[prefix]`, το Hash περιέχει μόνο παλιότερα προθέματα. Κάθε εμφάνιση του `prefix - target` δίνει διαφορετικό αριστερό άκρο. Αυτή η σειρά αποκλείει το κενό τμήμα όταν ο στόχος είναι μηδέν.

**Κόστος:** `O(n)` αναμενόμενος χρόνος και `O(n)` χώρος.

## Πρόσεξε αυτό

Αποθήκευσε πλήθος εμφανίσεων, όχι μόνο παρουσία. Με πολλά μηδενικά υπάρχουν πολλά διαφορετικά τμήματα με ίδια αθροίσματα.

Επιστροφή στην άσκηση 60 · σελίδα 221

Η ΛΥΣΗ

# 61 Η πρώτη εμφάνιση με δυαδική αναζήτηση

## Η σκέψη

Κράτα ένα διάστημα πιθανών θέσεων. Όταν το μέσο είναι αρκετά μεγάλο, η πρώτη εμφάνιση δεν μπορεί να βρίσκεται δεξιότερα από αυτό.

RUBY

```ruby
def solve(numbers, target)
  left = 0
  right = numbers.length
  while left < right
    middle = (left + right) / 2
    if numbers[middle] < target
      left = middle + 1
    else
      right = middle
    end
  end
  left < numbers.length && numbers[left] == target ? left : -1
end
```

## Διάβασε τον κώδικα

Οι θέσεις πριν από το `left` είναι μικρότερες από τον στόχο. Οι θέσεις από το `right` και μετά είναι τουλάχιστον ίσες με αυτόν. Στη σύγκλιση βρίσκουμε το πρώτο στοιχείο που δεν είναι μικρότερο και ελέγχουμε αν είναι πραγματικά ίσο.

**Κόστος:** `O(log n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η εύρεση ίσης τιμής δεν ολοκληρώνει την αναζήτηση. Μπορεί να υπάρχουν ίδιες τιμές πιο αριστερά.

Επιστροφή στην άσκηση 61 · σελίδα 226

Η ΛΥΣΗ

# 62 Πού θα μπει η νέα τιμή;

## Η σκέψη

Εδώ μας ενδιαφέρει το όριο ανάμεσα στις μικρότερες και τις όχι μικρότερες τιμές. Το όριο παραμένει χρήσιμο ακόμη και όταν ο στόχος απουσιάζει.

RUBY

```ruby
def solve(numbers, target)
  left = 0
  right = numbers.length
  while left < right
    middle = (left + right) / 2
    if numbers[middle] >= target
      right = middle
    else
      left = middle + 1
    end
  end
  left
end
```

## Διάβασε τον κώδικα

Στο τελικό όριο, όλες οι προηγούμενες τιμές είναι αυστηρά μικρότερες και όλες οι επόμενες τουλάχιστον ίσες. Άρα η εισαγωγή εκεί διατηρεί την ταξινόμηση και προηγείται των διπλότυπων. Ο κενός πίνακας δίνει φυσικά θέση μηδέν.

**Κόστος:** `O(log n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Το μήκος είναι έγκυρη θέση εισαγωγής. Δεν είναι όμως έγκυρη θέση ανάγνωσης στοιχείου.

Επιστροφή στην άσκηση 62 · σελίδα 229

Η ΛΥΣΗ

# 63 Πού αρχίζουν και πού τελειώνουν τα ίσα;

## Η σκέψη

Το πρώτο όριο βρίσκει τιμές τουλάχιστον ίσες με τον στόχο. Το δεύτερο βρίσκει τιμές αυστηρά μεγαλύτερες. Ανάμεσά τους βρίσκονται όλες οι ίσες.

RUBY

```ruby
def solve(numbers, target)
  first = numbers.bsearch_index { |value| value >= target }
  return [] if first.nil? || numbers[first] != target
  after = numbers.bsearch_index { |value| value > target }
  [first, (after || numbers.length) - 1]
end
```

## Διάβασε τον κώδικα

Η διαφορά `>=` και `>` καθορίζει αν οι ίσες τιμές ανήκουν στη δεξιά περιοχή. Το δεύτερο όριο δείχνει μετά την τελευταία εμφάνιση, γι' αυτό αφαιρούμε ένα. Αν δεν υπάρχει μεγαλύτερη τιμή, το όριο είναι το μήκος.

**Κόστος:** `O(log n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Μην περπατήσεις γραμμικά από την πρώτη εμφάνιση μέχρι την τελευταία. Σε πίνακα ίσων τιμών χάνεται ο στόχος απόδοσης.

Επιστροφή στην άσκηση 63 · σελίδα 232

Η ΛΥΣΗ

# 64 Τετραγωνική ρίζα χωρίς δεκαδικά

## Η σκέψη

Όταν ένα τετράγωνο είναι ήδη πολύ μεγάλο, όλα τα μεγαλύτερα είναι επίσης πολύ μεγάλα. Αναζητούμε την πρώτη αποτυχία αυτής της μονοτονικής συνθήκης.

RUBY

```ruby
def solve(number)
  left = 0
  right = number + 1
  while left < right
    middle = (left + right) / 2
    if middle * middle <= number
      left = middle + 1
    else
      right = middle
    end
  end
  left - 1
end
```

## Διάβασε τον κώδικα

Το διάστημα αναζήτησης περιέχει το πρώτο `x` με πολύ μεγάλο τετράγωνο. Η ισότητα ανήκει ακόμη στις επιτυχείς τιμές και μετακινεί το αριστερό όριο δεξιά. Η Ruby κρατά ακέραιη αριθμητική σε αυτούς τους πολλαπλασιασμούς.

**Κόστος:** `O(log(number + 1))` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Έλεγξε χωριστά στο μυαλό σου το μηδέν, το ένα και τους αριθμούς ακριβώς πριν από ένα τέλειο τετράγωνο.

Επιστροφή στην άσκηση 64 · σελίδα 235

Η ΛΥΣΗ

# 65 Η μικρότερη χωρητικότητα

## Η σκέψη

Για δεδομένη χωρητικότητα μπορείς να μετρήσεις τις απαιτούμενες ημέρες με ένα πέρασμα. Αν μια χωρητικότητα αρκεί, κάθε μεγαλύτερη αρκεί επίσης.

RUBY

```ruby
def solve(weights, days)
  low = weights.max
  high = weights.sum
  while low < high
    capacity = (low + high) / 2
    used = 1
    load = 0
    weights.each do |weight|
      if load + weight > capacity
        used += 1
        load = 0
      end
      load += weight
    end
    if used <= days
      high = capacity
    else
      low = capacity + 1
    end
  end
  low
end
```

Επιστροφή στην άσκηση 65 · σελίδα 239

Η ΜΙΚΡΟΤΕΡΗ ΧΩΡΗΤΙΚΟΤΗΤΑ

# 65 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Για μια σταθερή χωρητικότητα, το γέμισμα κάθε ημέρας με όσο το δυνατόν περισσότερα επόμενα δέματα δεν μπορεί να απαιτήσει περισσότερες ημέρες από μια νωρίτερη διακοπή. Αυτό δίνει σωστό έλεγχο εφικτότητας. Η δυαδική αναζήτηση εντοπίζει την πρώτη εφικτή χωρητικότητα.

**Κόστος:** `O(n log S)` χρόνος και `O(1)` πρόσθετος χώρος, όπου `S` το συνολικό βάρος.

## Πρόσεξε αυτό

Δεν επιτρέπεται αναδιάταξη βαρών. Η λύση δεν είναι απλώς το συνολικό βάρος διαιρεμένο με τις ημέρες.

Η ΛΥΣΗ

# 66 Το καλύτερο παράθυρο σταθερού μήκους

## Η σκέψη

Δύο γειτονικά παράθυρα διαφέρουν σε δύο μόνο θέσεις. Η υπόλοιπη δουλειά του προηγούμενου αθροίσματος μπορεί να διατηρηθεί.

RUBY

```ruby
def solve(numbers, width)
  current = (0...width).sum { |i| numbers[i] }
  best = current
  (width...numbers.length).each do |i|
    current += numbers[i] - numbers[i - width]
    best = [best, current].max
  end
  best
end
```

## Διάβασε τον κώδικα

Η αρχική κατάσταση αντιστοιχεί στο πρώτο επιτρεπτό παράθυρο. Κάθε ενημέρωση διατηρεί ακριβώς το νέο παράθυρο ίδιου μήκους. Η αρχικοποίηση της `best` από πραγματικό παράθυρο καλύπτει και τις εισόδους μόνο με αρνητικές τιμές.

**Κόστος:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Το μήκος είναι ακριβές. Δεν επιτρέπεται να διαλέξεις μικρότερο τμήμα επειδή περιέχει λιγότερες αρνητικές τιμές.

Επιστροφή στην άσκηση 66 · σελίδα 242

Η ΛΥΣΗ

# 67 Το συντομότερο αρκετά μεγάλο τμήμα

## Η σκέψη

Πρόσθεσε στοιχεία από δεξιά ώσπου το παράθυρο να αρκεί. Τότε αφαίρεσε από αριστερά όσο συνεχίζει να αρκεί, καταγράφοντας τα μικρότερα μήκη.

RUBY

```ruby
def solve(numbers, target)
  left = 0
  total = 0
  best = numbers.length + 1
  numbers.each_with_index do |number, right|
    total += number
    while total >= target
      best = [best, right - left + 1].min
      total -= numbers[left]
      left += 1
    end
  end
  best <= numbers.length ? best : 0
end
```

## Διάβασε τον κώδικα

Για κάθε δεξί άκρο εξετάζουμε όλες τις αφαιρέσεις που διατηρούν αρκετό άθροισμα. Μόλις το άθροισμα γίνει μικρό, άλλες αφαιρέσεις δεν μπορούν να βοηθήσουν. Κάθε στοιχείο μπαίνει και βγαίνει από το παράθυρο το πολύ μία φορά.

**Κόστος:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Αυτή η λογική δεν εφαρμόζεται γενικά σε αρνητικές τιμές. Η θετικότητα είναι ουσιαστική προϋπόθεση της άσκησης.

Επιστροφή στην άσκηση 67 · σελίδα 245

Η ΛΥΣΗ

# 68 Κείμενο χωρίς επαναλαμβανόμενο χαρακτήρα

## Η σκέψη

Η τελευταία θέση ενός χαρακτήρα δείχνει πόσο πρέπει να μετακινηθεί το αριστερό άκρο όταν τον ξανασυναντάς.

RUBY

```ruby
def solve(text)
  last = {}
  left = 0
  best = 0
  text.chars.each_with_index do |char, right|
    if last.key?(char)
      left = [left, last[char] + 1].max
    end
    last[char] = right
    best = [best, right - left + 1].max
  end
  best
end
```

## Διάβασε τον κώδικα

Πριν από κάθε ενημέρωση της απάντησης, το παράθυρο δεν έχει διπλότυπα. Η νέα επανάληψη διορθώνεται περνώντας ακριβώς μετά την προηγούμενη θέση του ίδιου χαρακτήρα, εφόσον αυτή βρίσκεται μέσα στο παράθυρο.

**Κόστος:** `O(n)` αναμενόμενος χρόνος και `O(n)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η απάντηση αφορά συνεχόμενο τμήμα, όχι όλες τις διαφορετικές τιμές του κειμένου.

Επιστροφή στην άσκηση 68 · σελίδα 248

Η ΛΥΣΗ

# 69 Παράθυρο με λίγες διαφορετικές τιμές

## Η σκέψη

Το Hash πρέπει να περιγράφει μόνο το τρέχον παράθυρο. Όταν προστεθεί παραπάνω διαφορετική τιμή, αφαίρεσε από αριστερά ώσπου να αποκατασταθεί το όριο.

RUBY

```ruby
def solve(numbers, limit)
  counts = Hash.new(0)
  left = 0
  best = 0
  numbers.each_with_index do |number, right|
    counts[number] += 1
    while counts.length > limit
      old = numbers[left]
      counts[old] -= 1
      counts.delete(old) if counts[old] == 0
      left += 1
    end
    best = [best, right - left + 1].max
  end
  best
end
```

## Διάβασε τον κώδικα

Μετά τον εσωτερικό βρόχο, οι μετρητές είναι θετικοί και αντιστοιχούν ακριβώς στις τιμές του έγκυρου παραθύρου. Το αριστερό άκρο έχει μετακινηθεί όσο ήταν απαραίτητο, οπότε για το συγκεκριμένο δεξί άκρο παίρνουμε το μεγαλύτερο επιτρεπτό μήκος.

**Κόστος:** `O(n)` αναμενόμενος χρόνος και `O(min(n, limit + 1))` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η αφαίρεση μίας εμφάνισης δεν αφαιρεί απαραίτητα ολόκληρη την τιμή από το παράθυρο.

Επιστροφή στην άσκηση 69 · σελίδα 251

Η ΛΥΣΗ

# 70 Πόσα αναγράμματα κρύβονται στο κείμενο;

## Η σκέψη

Κάθε υποψήφιο παράθυρο έχει μήκος ίσο με το pattern. Ενημέρωσε τις 26 συχνότητές του όταν μπαίνει και βγαίνει ένας χαρακτήρας.

RUBY

```ruby
def solve(text, pattern)
  wanted = Array.new(26, 0)
  pattern.each_byte { |byte| wanted[byte - 97] += 1 }
  window = Array.new(26, 0)
  width = pattern.length
  answer = 0
  text.each_byte.with_index do |byte, i|
    window[byte - 97] += 1
    if i >= width
      window[text.getbyte(i - width) - 97] -= 1
    end
    answer += 1 if i >= width - 1 && window == wanted
  end
  answer
end
```

## Διάβασε τον κώδικα

Οι δύο πίνακες περιγράφουν πολυσύνολα χαρακτήρων. Η ισότητά τους ελέγχει και παρουσία και πλήθος. Η σύγκριση 26 θέσεων έχει σταθερό κόστος ως προς το μήκος εισόδου. Τα bytes χρησιμοποιούνται επειδή εδώ το αλφάβητο ορίζεται ρητά ως ASCII.

**Κόστος:** `O(n + m)` χρόνος για σταθερό αλφάβητο 26 γραμμάτων και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Έλεγχε την ισότητα μόνο αφού σχηματιστεί πλήρες παράθυρο. Το μεγαλύτερο pattern δεν έχει καμία εμφάνιση.

Επιστροφή στην άσκηση 70 · σελίδα 254

Η ΛΥΣΗ

# 71 Σωστά ζευγάρια παρενθέσεων

## Η σκέψη

Το επόμενο κλείσιμο πρέπει να ταιριάζει με το πιο πρόσφατο άνοιγμα που δεν έχει ακόμη κλείσει. Αυτή είναι ακριβώς η σειρά μιας στοίβας.

RUBY

```ruby
def solve(text)
  pairs = {")" => "(", "]" => "[", "}" => "{"}
  stack = []
  text.each_char do |char|
    if pairs.key?(char)
      return false if stack.pop != pairs[char]
    else
      stack << char
    end
  end
  stack.empty?
end
```

## Διάβασε τον κώδικα

Η στοίβα περιέχει τα ανοίγματα που περιμένουν κλείσιμο, με το πιο πρόσφατο στην κορυφή. Το `pop` από κενή στοίβα δίνει `nil`, που δεν μπορεί να ταιριάξει με κανένα άνοιγμα. Στο τέλος πρέπει να μην απομένει κανένα ανοιχτό ζεύγος.

**Κόστος:** `O(n)` χρόνος και `O(n)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Ίσα πλήθη ανοιγμάτων και κλεισιμάτων δεν αρκούν. Το `([)]` έχει λάθος εμφώλευση.

Επιστροφή στην άσκηση 71 · σελίδα 258

Η ΛΥΣΗ

# 72 Πόσες παρενθέσεις λείπουν;

## Η σκέψη

Ένα κλείσιμο χωρίς διαθέσιμο άνοιγμα απαιτεί ένα νέο άνοιγμα πριν από αυτό. Όσα ανοίγματα περισσέψουν στο τέλος απαιτούν ισάριθμα κλεισίματα.

RUBY

```ruby
def solve(text)
  open = 0
  missing = 0
  text.each_char do |char|
    if char == "("
      open += 1
    elsif open > 0
      open -= 1
    else
      missing += 1
    end
  end
  missing + open
end
```

## Διάβασε τον κώδικα

Κάθε κλείσιμο χωρίς προηγούμενο άνοιγμα επιβάλλει τουλάχιστον μία προσθήκη πριν από τη θέση του. Κάθε άνοιγμα που περισσεύει επιβάλλει τουλάχιστον ένα κλείσιμο μετά από αυτό. Ο αλγόριθμος κάνει ακριβώς αυτές τις αναπόφευκτες προσθήκες.

**Κόστος:** `O(n)` χρόνος και `O(1)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Το `)(` χρειάζεται δύο προσθήκες, παρότι έχει ίσο πλήθος ανοιγμάτων και κλεισιμάτων.

Επιστροφή στην άσκηση 72 · σελίδα 261

Η ΛΥΣΗ

# 73 Υπολόγισε μια έκφραση με στοίβα

## Η σκέψη

Κάθε αριθμός περιμένει στη στοίβα μέχρι να εμφανιστεί ο τελεστής που τον χρησιμοποιεί. Το αποτέλεσμα της πράξης γίνεται νέα διαθέσιμη τιμή.

RUBY

```ruby
def solve(tokens)
  stack = []
  tokens.each do |token|
    if ["+", "-", "*", "/"].include?(token)
      right = stack.pop
      left = stack.pop
      value = case token
              when "+" then left + right
              when "-" then left - right
              when "*" then left * right
              when "/" then left.div(right)
              end
      stack << value
    else
      stack << token.to_i
    end
  end
  stack[0]
end
```

Επιστροφή στην άσκηση 73 · σελίδα 265

ΥΠΟΛΟΓΙΣΕ ΜΙΑ ΕΚΦΡΑΣΗ ΜΕ ΣΤΟΙΒΑ

# 73 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Η στοίβα κρατά αποτελέσματα υποεκφράσεων που έχουν ήδη ολοκληρωθεί. Ένας τελεστής αφαιρεί δύο τέτοιες τιμές και τις αντικαθιστά με μία. Η έγκυρη είσοδος εξασφαλίζει ότι στο τέλος απομένει ακριβώς ένα αποτέλεσμα.

**Κόστος:** `O(n)` πράξεις στοίβας και `O(n)` χώρος. Το κόστος αριθμητικής μεγάλων ακεραίων εξαρτάται από τα ψηφία τους.

## Πρόσεξε αυτό

Μην εκτελέσεις την είσοδο ως κώδικα με `eval`. Οι τέσσερις επιτρεπτές πράξεις μπορούν να επιλεγούν ρητά.

Η ΛΥΣΗ

# 74 Η επόμενη αυστηρά μεγαλύτερη τιμή

## Η σκέψη

Από δεξιά προς τα αριστερά, ορισμένες τιμές παύουν να είναι χρήσιμες υποψήφιες όταν συναντήσουν μια πιο κοντινή, τουλάχιστον ίση τιμή.

RUBY

```ruby
def solve(numbers)
  result = Array.new(numbers.length, -1)
  stack = []
  (numbers.length - 1).downto(0) do |i|
    stack.pop while !stack.empty? && stack[-1] <= numbers[i]
    result[i] = stack[-1] unless stack.empty?
    stack << numbers[i]
  end
  result
end
```

## Διάβασε τον κώδικα

Μια αφαιρεμένη τιμή δεν μπορεί να είναι η πρώτη καλύτερη επιλογή για καμία θέση πιο αριστερά: η τρέχουσα τιμή είναι τουλάχιστον ίση και πιο κοντά. Η κορυφή που απομένει είναι η πρώτη μεγαλύτερη. Κάθε τιμή εισάγεται και αφαιρείται το πολύ μία φορά.

**Κόστος:** `O(n)` χρόνος και `O(n)` χώρος.

## Πρόσεξε αυτό

Η ισότητα δεν αρκεί. Αν κρατήσεις ίσες τιμές στην κορυφή, θα δώσεις λάθος απάντηση σε διπλότυπα.

Επιστροφή στην άσκηση 74 · σελίδα 268

Η ΛΥΣΗ

# 75 Πόσες ημέρες μέχρι να ζεστάνει;

## Η σκέψη

Κράτα τους δείκτες ημερών που περιμένουν ακόμη απάντηση. Μια νέα υψηλότερη θερμοκρασία μπορεί να λύσει πολλές εκκρεμότητες μαζί.

RUBY

```ruby
def solve(temperatures)
  result = Array.new(temperatures.length, 0)
  waiting = []
  temperatures.each_with_index do |temperature, i|
    while !waiting.empty? && temperature > temperatures[waiting[-1]]
      previous = waiting.pop
      result[previous] = i - previous
    end
    waiting << i
  end
  result
end
```

## Διάβασε τον κώδικα

Οι δείκτες που περιμένουν έχουν μη αυξανόμενες θερμοκρασίες. Μόλις η σημερινή ξεπεράσει την κορυφή, είναι η πρώτη που το καταφέρνει μετά από εκείνη την ημέρα. Όσα μείνουν στη στοίβα δεν βρήκαν ποτέ απάντηση και κρατούν το αρχικό μηδέν.

**Κόστος:** `O(n)` χρόνος και `O(n)` χώρος.

## Πρόσεξε αυτό

Οι ίσες θερμοκρασίες δεν λύνουν εκκρεμότητες. Η απόσταση μετριέται σε θέσεις, όχι σε βαθμούς.

Επιστροφή στην άσκηση 75 · σελίδα 271

Η ΛΥΣΗ

# 76 Το μεγαλύτερο ορθογώνιο στο ιστόγραμμα

## Η σκέψη

Όταν εμφανιστεί χαμηλότερη στήλη, ορισμένα προηγούμενα ύψη δεν μπορούν να επεκταθούν άλλο δεξιά. Η στοίβα διατηρεί τα αριστερά όρια που χρειαζόμαστε.

RUBY

```ruby
def solve(heights)
  stack = []
  best = 0
  (0..heights.length).each do |i|
    height = i == heights.length ? 0 : heights[i]
    while !stack.empty? && heights[stack[-1]] > height
      top = stack.pop
      left = stack.empty? ? -1 : stack[-1]
      area = heights[top] * (i - left - 1)
      best = [best, area].max
    end
    stack << i
  end
  best
end
```

## Διάβασε τον κώδικα

Οι δείκτες στη στοίβα έχουν μη φθίνοντα ύψη. Όταν ένα ύψος αφαιρεθεί, η τρέχουσα θέση είναι το πρώτο μικρότερο ύψος στα δεξιά. Η νέα κορυφή περιορίζει το πλάτος αριστερά. Ίσα ύψη μπορούν να περιμένουν και τελικά ένα από αυτά θα εξεταστεί με όλο το διαθέσιμο πλάτος.

**Κόστος:** `O(n)` χρόνος και `O(n)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Μετά την αφαίρεση ενός δείκτη, το αριστερό όριο είναι μία θέση μετά τη νέα κορυφή, όχι μετά τον δείκτη που μόλις έφυγε.

Επιστροφή στην άσκηση 76 · σελίδα 274

Η ΛΥΣΗ

# 77 Ένωση επικαλυπτόμενων διαστημάτων

## Η σκέψη

Μετά την ταξινόμηση κατά αρχή, το επόμενο διάστημα μπορεί να αλληλεπιδράσει μόνο με το τελευταίο ήδη κατασκευασμένο διάστημα.

RUBY

```ruby
def solve(intervals)
  result = []
  intervals.sort.each do |start, finish|
    if result.empty? || start > result[-1][1]
      result << [start, finish]
    else
      result[-1][1] = [result[-1][1], finish].max
    end
  end
  result
end
```

## Διάβασε τον κώδικα

Η ταξινόμηση ζευγών χρησιμοποιεί πρώτα την αρχή και μετά το τέλος. Τα ήδη ολοκληρωμένα διαστήματα πριν από το τελευταίο δεν μπορούν να επηρεαστούν από μεγαλύτερη νέα αρχή. Τα νέα ζεύγη αποκλείουν μεταβολή των γραμμών εισόδου.

**Κόστος:** `O(n log n)` χρόνος και `O(n)` χώρος μαζί με την έξοδο.

## Πρόσεξε αυτό

Οι εσωτερικοί πίνακες της εισόδου δεν πρέπει να αλλάξουν. Κατασκεύασε καινούργια ζεύγη στην έξοδο.

Επιστροφή στην άσκηση 77 · σελίδα 277

Η ΛΥΣΗ

# 78 Πόσες αίθουσες χρειάζονται;

## Η σκέψη

Μετέτρεψε τις αρχές και τα τέλη σε γεγονότα αύξησης και μείωσης του πλήθους ενεργών συναντήσεων. Το μέγιστο πλήθος είναι η ζητούμενη χωρητικότητα.

```ruby
def solve(meetings)
  events = meetings.flat_map do |start, finish|
    [[start, 1], [finish, -1]]
  end
  active = 0
  best = 0
  events.sort.each do |_time, change|
    active += change
    best = [best, active].max
  end
  best
end
```

## Διάβασε τον κώδικα

Η ταξινόμηση ζευγών βάζει το `-1` πριν από το `1` όταν οι χρόνοι είναι ίσοι. Κάθε στιγμή με `k` ενεργές συναντήσεις απαιτεί τουλάχιστον `k` αίθουσες. Η επαναχρησιμοποίηση όσων ελευθερώνονται αρκεί για να πετύχουμε αυτό το μέγιστο.

**Κόστος:** `O(n log n)` χρόνος και `O(n)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Εδώ τα άκρα συμπεριφέρονται διαφορετικά από τα κλειστά διαστήματα της άσκησης 77. Το κοινό άκρο δεν απαιτεί επιπλέον αίθουσα.

Επιστροφή στην άσκηση 78 · σελίδα 280

Η ΛΥΣΗ

# 79 Οι γραμμές γίνονται στήλες

## Η σκέψη

Η έξοδος έχει τόσες γραμμές όσες ήταν οι στήλες της εισόδου. Κάθε νέα γραμμή συλλέγει μία συγκεκριμένη στήλη.

RUBY

```ruby
def solve(matrix)
  return [] if matrix.empty?
  rows = matrix.length
  columns = matrix[0].length
  Array.new(columns) do |column|
    Array.new(rows) do |row|
      matrix[row][column]
    end
  end
end
```

## Διάβασε τον κώδικα

Κάθε στοιχείο έχει έναν μοναδικό προορισμό με ανταλλαγμένους δείκτες. Τα δύο blocks δημιουργούν νέες γραμμές και αντιγράφουν ακέραιες τιμές. Όταν οι στήλες είναι μηδέν, η εξωτερική `Array.new(0)` δίνει απευθείας κενή έξοδο.

**Κόστος:** `O(r c)` χρόνος και χώρος εξόδου για `r` γραμμές και `c` στήλες.

## Πρόσεξε αυτό

Η `Array.new(rows, [])` μοιράζεται τον ίδιο εσωτερικό πίνακα σε όλες τις θέσεις. Οι γραμμές εδώ πρέπει να είναι ανεξάρτητες.

Επιστροφή στην άσκηση 79 · σελίδα 283

Η ΛΥΣΗ

# 80 Διάβασε τον πίνακα σπειροειδώς

## Η σκέψη

Περιέγραψε την αδιάβαστη περιοχή με πάνω, κάτω, αριστερό και δεξί όριο. Κάθε ολοκληρωμένη πλευρά αφαιρεί ένα από αυτά τα όρια.

RUBY

```ruby
def solve(matrix)
  return [] if matrix.empty? || matrix[0].empty?
  top = 0
  bottom = matrix.length - 1
  left = 0
  right = matrix[0].length - 1
  result = []
  while top <= bottom && left <= right
    (left..right).each { |c| result << matrix[top][c] }
    top += 1
    (top..bottom).each { |r| result << matrix[r][right] }
    right -= 1
    if top <= bottom
      right.downto(left) { |c| result << matrix[bottom][c] }
      bottom -= 1
    end
    if left <= right
      bottom.downto(top) { |r| result << matrix[r][left] }
      left += 1
    end
  end
  result
end
```

Επιστροφή στην άσκηση 80 · σελίδα 286

ΔΙΑΒΑΣΕ ΤΟΝ ΠΙΝΑΚΑ ΣΠΕΙΡΟΕΙΔΩΣ

# 80 Γιατί λειτουργεί

### Διάβασε τον κώδικα

Κάθε ανάγνωση αφορά την τρέχουσα εξωτερική πλευρά και ακολουθείται από μετακίνηση του αντίστοιχου ορίου. Τα guards αποτρέπουν επανάληψη της τελευταίας λεπτής περιοχής. Καμία τιμή δεν χρειάζεται να διαγραφεί ή να σημαδευτεί μέσα στην είσοδο.

**Κόστος:** `O(r c)` χρόνος, `O(r c)` χώρος εξόδου και `O(1)` πρόσθετος χώρος.

### Πρόσεξε αυτό

Μια τελευταία μονή γραμμή ή στήλη δεν πρέπει να διαβαστεί δύο φορές.

Η ΛΥΣΗ

# 81 Πόσα νησιά υπάρχουν στον χάρτη;

## Η σκέψη

Κάθε άγνωστο κελί γης αρχίζει ένα νέο νησί. Πριν συνεχίσεις τη γενική σάρωση, εξερεύνησε όλη τη συνδεδεμένη περιοχή του.

RUBY

```ruby
def solve(grid)
  return 0 if grid.empty?
  rows, cols = grid.length, grid[0].length
  seen = Array.new(rows) { Array.new(cols, false) }
  islands = 0
  grid.each_with_index do |row, r|
    row.each_index do |c|
      next if row[c] == 0 || seen[r][c]
      islands += 1
      stack = [[r, c]]
      seen[r][c] = true
      until stack.empty?
        x, y = stack.pop
        [[x-1,y], [x+1,y], [x,y-1], [x,y+1]].each do |nx, ny|
          next unless nx.between?(0, rows-1)
          next unless ny.between?(0, cols-1)
          next if grid[nx][ny] == 0 || seen[nx][ny]
          seen[nx][ny] = true
          stack << [nx, ny]
        end
      end
    end
  end
  islands
end
```

Επιστροφή στην άσκηση 81 · σελίδα 290

ΠΟΣΑ ΝΗΣΙΑ ΥΠΑΡΧΟΥΝ ΣΤΟΝ ΧΑΡΤΗ;

# 81 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Η στοίβα υλοποιεί αναζήτηση βάθους χωρίς αναδρομή. Όταν αδειάσει, όλα τα κελιά του νέου νησιού έχουν σημειωθεί. Η εξωτερική σάρωση δεν θα τα ξαναμετρήσει. Κάθε κελί γης μπαίνει στη στοίβα μία φορά και έχει το πολύ τέσσερις γείτονες.

**Κόστος:** `O(r c)` χρόνος και `O(r c)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Μην αντικαταστήσεις τη γη με νερό μέσα στην είσοδο. Χρησιμοποίησε χωριστό πίνακα επισκέψεων.

Η ΛΥΣΗ

# 82 Η συντομότερη διαδρομή στο πλέγμα

## Η σκέψη

Όταν κάθε κίνηση κοστίζει ένα, μια ουρά εξετάζει πρώτα όλες τις θέσεις απόστασης ένα, μετά δύο και ούτω καθεξής.

RUBY

```ruby
def solve(grid)
  return -1 if grid.empty? || grid[0][0] == 1
  rows, cols = grid.length, grid[0].length
  distance = Array.new(rows) { Array.new(cols, -1) }
  distance[0][0] = 0
  queue = [[0, 0]]
  head = 0
  while head < queue.length
    x, y = queue[head]
    head += 1
    return distance[x][y] if x == rows-1 && y == cols-1
    [[x-1,y], [x+1,y], [x,y-1], [x,y+1]].each do |nx, ny|
      next unless nx.between?(0, rows-1)
      next unless ny.between?(0, cols-1)
      next if grid[nx][ny] == 1 || distance[nx][ny] != -1
      distance[nx][ny] = distance[x][y] + 1
      queue << [nx, ny]
    end
  end
  -1
end
```

Επιστροφή στην άσκηση 82 · σελίδα 294

Η ΣΥΝΤΟΜΟΤΕΡΗ ΔΙΑΔΡΟΜΗ ΣΤΟ ΠΛΕΓΜΑ

# 82 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Η ουρά διατηρεί μη φθίνουσα σειρά αποστάσεων. Άρα η πρώτη ανακάλυψη κάθε κελιού είναι ήδη βέλτιστη. Ο δείκτης `head` διαβάζει την ουρά χωρίς μετακινήσεις των υπόλοιπων στοιχείων. Ένα ελεύθερο μοναδικό κελί έχει απόσταση μηδέν.

**Κόστος:** `O(r c)` χρόνος και `O(r c)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η αναζήτηση βάθους μπορεί να βρει διαδρομή, αλλά δεν εγγυάται ότι η πρώτη που βρίσκει είναι η συντομότερη.

Η ΛΥΣΗ

# 83 Αποστάσεις σε έναν γράφο

## Η σκέψη

Η αναζήτηση στο πλέγμα γενικεύεται όταν οι γείτονες δίνονται από λίστα αντί να υπολογίζονται από συντεταγμένες.

RUBY

```ruby
def solve(size, edges, start)
  graph = Array.new(size) { [] }
  edges.each do |a, b|
    graph[a] << b
    graph[b] << a
  end
  distance = Array.new(size, -1)
  distance[start] = 0
  queue = [start]
  head = 0
  while head < queue.length
    vertex = queue[head]
    head += 1
    graph[vertex].each do |neighbor|
      next if distance[neighbor] != -1
      distance[neighbor] = distance[vertex] + 1
      queue << neighbor
    end
  end
  distance
end
```

Επιστροφή στην άσκηση 83 · σελίδα 297

ΑΠΟΣΤΑΣΕΙΣ ΣΕ ΕΝΑΝ ΓΡΑΦΟ

# 83 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Η ανεξάρτητη λίστα κάθε κορυφής κατασκευάζεται με block. Το BFS επισκέπτεται κάθε προσβάσιμη κορυφή μία φορά. Οι διπλές ακμές και οι βρόχοι απορρίπτονται ως νέες επισκέψεις επειδή η απόσταση έχει ήδη οριστεί.

**Κόστος:** `O(v + e)` χρόνος και χώρος, με `v` κορυφές και `e` ακμές.

## Πρόσεξε αυτό

Οι απομονωμένες κορυφές υπάρχουν ακόμη κι αν δεν εμφανίζονται σε καμία ακμή. Το μήκος εξόδου πρέπει να είναι `size`.

Η ΛΥΣΗ

# 84 Πόσα χωριστά δίκτυα υπάρχουν;

## Η σκέψη

Κάθε νέα αναζήτηση από άγνωστη κορυφή ανακαλύπτει μία ολόκληρη συνιστώσα. Ήδη γνωστές κορυφές δεν αρχίζουν νέα αναζήτηση.

RUBY

```ruby
def solve(size, edges)
  graph = Array.new(size) { [] }
  edges.each do |a, b|
    graph[a] << b
    graph[b] << a
  end
  seen = Array.new(size, false)
  count = 0
  size.times do |root|
    next if seen[root]
    count += 1
    seen[root] = true
    stack = [root]
    until stack.empty?
      graph[stack.pop].each do |neighbor|
        next if seen[neighbor]
        seen[neighbor] = true
        stack << neighbor
      end
    end
  end
  count
end
```

Επιστροφή στην άσκηση 84 · σελίδα 300

ΠΟΣΑ ΧΩΡΙΣΤΑ ΔΙΚΤΥΑ ΥΠΑΡΧΟΥΝ;

# 84 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Μετά το άδειασμα της στοίβας, έχουν σημειωθεί ακριβώς οι κορυφές που συνδέονται με τη νέα ρίζα. Καμία από αυτές δεν μπορεί να αυξήσει ξανά τον μετρητή. Μια απομονωμένη ρίζα μετριέται και η αναζήτησή της ολοκληρώνεται αμέσως.

**Κόστος:** `O(v + e)` χρόνος και χώρος.

## Πρόσεξε αυτό

Ένας βρόχος από κορυφή στον εαυτό της δεν δημιουργεί δεύτερη συνιστώσα.

Η ΛΥΣΗ

# 85 Μια σειρά που σέβεται τις εξαρτήσεις

## Η σκέψη

Διαθέσιμη είναι μια εργασία χωρίς εκκρεμή προαπαιτούμενα. Εκτέλεσε τη μικρότερη διαθέσιμη και αφαίρεσε τη συμβολή της στις εξαρτήσεις των επόμενων.

RUBY

```ruby
def solve(size, edges)
  graph = Array.new(size) { [] }
  degree = Array.new(size, 0)
  edges.each do |a, b|
    graph[a] << b
    degree[b] += 1
  end
  used = Array.new(size, false)
  order = []
  size.times do
    vertex = (0...size).find { |v| !used[v] && degree[v] == 0 }
    return [] if vertex.nil?
    used[vertex] = true
    order << vertex
    graph[vertex].each { |neighbor| degree[neighbor] -= 1 }
  end
  order
end
```

Επιστροφή στην άσκηση 85 · σελίδα 303

ΜΙΑ ΣΕΙΡΑ ΠΟΥ ΣΕΒΕΤΑΙ ΤΙΣ ΕΞΑΡΤΗΣΕΙΣ

# 85 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Μόνο μια κορυφή με μηδενικό υπόλοιπο εισερχόμενων ακμών μπορεί να είναι επόμενη. Η επιλογή της μικρότερης τέτοιας κορυφής δίνει το μικρότερο δυνατό επόμενο στοιχείο κάθε έγκυρου προθέματος. Οι διπλές ακμές προστίθενται και αφαιρούνται με το ίδιο πλήθος.

**Κόστος:** $O(v^2 + e)$ χρόνος και $O(v + e)$ χώρος. Η γραμμική επιλογή είναι σκόπιμη για έως 30 κορυφές.

## Πρόσεξε αυτό

Αν απομένουν εργασίες αλλά καμία δεν είναι διαθέσιμη, υπάρχει κύκλος. Μια μερική σειρά δεν είναι αποδεκτή απάντηση.

Η ΛΥΣΗ

# 86 Χωρίζεται ο γράφος σε δύο ομάδες;

## Η σκέψη

Δώσε μια αυθαίρετη ομάδα στην πρώτη κορυφή κάθε συνιστώσας. Κάθε γείτονας πρέπει να πάρει την αντίθετη. Σύγκρουση σε ήδη χρωματισμένη κορυφή σημαίνει αποτυχία.

RUBY

```ruby
def solve(size, edges)
  graph = Array.new(size) { [] }
  edges.each do |a, b|
    graph[a] << b
    graph[b] << a
  end
  color = Array.new(size, -1)
  size.times do |root|
    next if color[root] != -1
    color[root] = 0
    queue = [root]
    head = 0
    while head < queue.length
      vertex = queue[head]
      head += 1
      graph[vertex].each do |neighbor|
        if color[neighbor] == -1
          color[neighbor] = 1 - color[vertex]
          queue << neighbor
        elsif color[neighbor] == color[vertex]
          return false
        end
      end
    end
  end
  true
end
```

Επιστροφή στην άσκηση 86 · σελίδα 306

ΧΩΡΙΖΕΤΑΙ Ο ΓΡΑΦΟΣ ΣΕ ΔΥΟ ΟΜΑΔΕΣ;

# 86 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Κάθε ακμή επιβάλλει έναν τοπικό περιορισμό αντίθετων χρωμάτων. Η εξερεύνηση μεταφέρει αυτούς τους περιορισμούς σε όλη τη συνιστώσα. Αν εμφανιστεί σύγκρουση, καμία αλλαγή της αρχικής ομάδας δεν τη διορθώνει, αφού θα αντέστρεφε όλα τα χρώματα μαζί.

**Κόστος:** `O(v + e)` χρόνος και χώρος.

## Πρόσεξε αυτό

Έλεγξε όλες τις συνιστώσες. Μια πρώτη σωστή συνιστώσα δεν εγγυάται ότι ο υπόλοιπος γράφος είναι σωστός.

Η ΛΥΣΗ

# 87 Όλες οι διαφορετικές μεταθέσεις

## Η σκέψη

Χτίσε την απάντηση μία θέση τη φορά. Δοκίμασε κάθε διαθέσιμη τιμή και αναίρεσε την επιλογή πριν δοκιμάσεις την επόμενη.

RUBY

```ruby
def solve(numbers)
  values = numbers.sort
  used = Array.new(values.length, false)
  path = []
  result = []
  visit = lambda do
    return result << path.dup if path.length == values.length
    values.each_index do |i|
      next if used[i]
      next if i > 0 && values[i] == values[i-1] && !used[i-1]
      used[i] = true
      path << values[i]
      visit.call
      path.pop
      used[i] = false
    end
  end
  visit.call
  result
end
```

Επιστροφή στην άσκηση 87 · σελίδα 309

ΟΛΕΣ ΟΙ ΔΙΑΦΟΡΕΤΙΚΕΣ ΜΕΤΑΘΕΣΕΙΣ

# 87 Γιατί λειτουργεί

### Διάβασε τον κώδικα

Η ταξινομημένη σειρά επιλογών παράγει λεξικογραφική έξοδο. Η παράλειψη μιας ίσης τιμής όταν η προηγούμενή της παραμένει αχρησιμοποίητη αποτρέπει ισοδύναμους κλάδους στο ίδιο επίπεδο. Κάθε αναδρομική κλήση αφήνει την κατάσταση όπως τη βρήκε πριν επιστρέψει.

**Κόστος:** `O(n · n!)` άνω φράγμα χρόνου και χώρου εξόδου· `O(n)` χώρος αναζήτησης πέρα από την έξοδο.

### Πρόσεξε αυτό

Αποθήκευσε `path.dup` όταν ολοκληρωθεί μια μετάθεση. Η αποθήκευση του ίδιου μεταβαλλόμενου πίνακα θα αλλοίωνε όλες τις απαντήσεις.

Η ΛΥΣΗ

# 88 Διάλεξε k τιμές χωρίς επαναλήψεις

## Η σκέψη

Αφού επιλέξεις μία τιμή, οι επόμενες πρέπει να είναι μεγαλύτερες. Έτσι κάθε σύνολο επιλογών παράγεται μία φορά και δεν χρειάζεται τελική αφαίρεση διπλότυπων.

RUBY

```ruby
def solve(size, count)
  result = []
  path = []
  visit = lambda do |start|
    return result << path.dup if path.length == count
    needed = count - path.length
    (start..(size - needed + 1)).each do |value|
      path << value
      visit.call(value + 1)
      path.pop
    end
  end
  visit.call(1)
  result
end
```

## Διάβασε τον κώδικα

Το `start` αποκλείει προηγούμενες τιμές και διαφορετικές σειρές του ίδιου συνδυασμού. Το άνω όριο αφήνει αρκετές μεγαλύτερες τιμές για να συμπληρωθεί η επιλογή. Η αντιγραφή της τελικής διαδρομής αποσυνδέει την έξοδο από το backtracking.

**Κόστος:** `O(k · C(n,k))` για τις έγκυρες εξόδους, με `O(k)` χώρο αναζήτησης πέρα από την αποθήκευσή τους.

## Πρόσεξε αυτό

Η επιλογή μηδενικών στοιχείων έχει μία έγκυρη λύση: την κενή επιλογή. Δεν είναι αδυναμία.

Επιστροφή στην άσκηση 88 · σελίδα 312

Η ΛΥΣΗ

# 89 Παρήγαγε μόνο σωστές παρενθέσεις

## Η σκέψη

Ένα σωστό τελικό κείμενο έχει σωστά προθέματα: ποτέ περισσότερα κλεισίματα από ανοίγματα. Κατασκεύασε μόνο κλάδους που μπορούν ακόμη να οδηγήσουν σε λύση.

RUBY

```ruby
def solve(pairs)
  result = []
  visit = lambda do |open, closed, text|
    return result << text if closed == pairs
    visit.call(open + 1, closed, text + "(") if open < pairs
    visit.call(open, closed + 1, text + ")") if closed < open
  end
  visit.call(0, 0, "")
  result
end
```

## Διάβασε τον κώδικα

Οι δύο περιορισμοί εξασφαλίζουν ότι κάθε παραγόμενο πρόθεμα είναι επεκτάσιμο. Κάθε έγκυρο τελικό κείμενο ακολουθεί μία μοναδική ακολουθία επιλογών. Η δοκιμή του ανοίγματος πριν από το κλείσιμο δίνει λεξικογραφική σειρά χωρίς τελική ταξινόμηση.

**Κόστος:** `O(n Cn)` χρόνος και χώρος εξόδου, όπου `Cn` το πλήθος Catalan· το βάθος αναδρομής είναι `O(n)`.

## Πρόσεξε αυτό

Η βασική περίπτωση πρέπει να αποθηκεύει και το κενό κείμενο όταν ζητούνται μηδενικά ζεύγη.

Επιστροφή στην άσκηση 89 · σελίδα 316

Η ΛΥΣΗ

# 90 Πόσες τοποθετήσεις βασιλισσών υπάρχουν;

## Η σκέψη

Τοποθέτησε μία βασίλισσα ανά γραμμή. Για κάθε υποψήφια στήλη αρκούν τρεις έλεγχοι: κατειλημμένη στήλη και οι δύο οικογένειες διαγωνίων.

RUBY

```ruby
def solve(size)
  columns = {}
  falling = {}
  rising = {}
  search = lambda do |row|
    return 1 if row == size
    total = 0
    size.times do |column|
      down = row - column
      up = row + column
      next if columns[column] || falling[down] || rising[up]
      columns[column] = true
      falling[down] = true
      rising[up] = true
      total += search.call(row + 1)
      columns.delete(column)
      falling.delete(down)
      rising.delete(up)
    end
    total
  end
  search.call(0)
end
```

Επιστροφή στην άσκηση 90 · σελίδα 319

ΠΟΣΕΣ ΤΟΠΟΘΕΤΗΣΕΙΣ ΒΑΣΙΛΙΣΣΩΝ ΥΠΑΡΧΟΥΝ;

# 90 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Η γραμμή είναι μοναδική λόγω του επιπέδου αναδρομής. Τα τρία Hash ελέγχουν τους υπόλοιπους περιορισμούς χωρίς να ξαναδιαβάζουν όλες τις προηγούμενες βασίλισσες. Κάθε πλήρης τοποθέτηση προσθέτει ένα, ενώ αδιέξοδος κλάδος προσθέτει μηδέν.

**Κόστος:** `O(n · n!)` άνω φράγμα χρόνου και `O(n)` χώρος αναζήτησης. Δεν αποθηκεύονται όλες οι τοποθετήσεις.

## Πρόσεξε αυτό

Αφαίρεσε και τους τρεις περιορισμούς όταν επιστρέψει η αναδρομή. Διαφορετικά ένας προηγούμενος κλάδος θα μπλοκάρει άσχετες επιλογές.

Η ΛΥΣΗ

# 91 Ανέβα τη σκάλα με κλειστά σκαλοπάτια

## Η σκέψη

Οι τρόποι άφιξης σε ένα ελεύθερο σκαλοπάτι προέρχονται από τα δύο αμέσως προηγούμενα. Μια απαγορευμένη θέση έχει μηδέν τρόπους, αλλά μπορεί να υπερπηδηθεί.

RUBY

```ruby
def solve(steps, blocked)
  forbidden = {}
  blocked.each { |step| forbidden[step] = true }
  ways = Array.new(steps + 1, 0)
  ways[0] = 1
  (1..steps).each do |step|
    next if forbidden.key?(step)
    ways[step] = ways[step - 1]
    ways[step] += ways[step - 2] if step >= 2
  end
  ways[steps]
end
```

## Διάβασε τον κώδικα

Οι δύο τελευταίες κινήσεις χωρίζουν τις διαδρομές σε ξένες κατηγορίες: όσες φτάνουν με άλμα ένα και όσες με άλμα δύο. Γι' αυτό τα πλήθη προστίθενται χωρίς διπλομέτρηση. Η σειρά από μικρά σε μεγάλα σκαλοπάτια εξασφαλίζει έτοιμες προηγούμενες απαντήσεις.

**Κόστος:** `O(n + b)` αναμενόμενος χρόνος και χώρος, για `n` σκαλοπάτια και `b` απαγορευμένες καταγραφές.

## Πρόσεξε αυτό

Το απαγορευμένο ενδιάμεσο σκαλοπάτι δεν αποκλείει ένα άλμα δύο θέσεων που περνά από πάνω του.

Επιστροφή στην άσκηση 91 · σελίδα 323

Η ΛΥΣΗ

# 92 Το ποσό με τα λιγότερα νομίσματα

## Η σκέψη

Αν το τελευταίο νόμισμα έχει αξία `coin`, πριν από αυτό πρέπει να έχει σχηματιστεί το υπόλοιπο ποσό. Δοκίμασε κάθε επιτρεπτή τελευταία αξία και κράτα την καλύτερη.

RUBY

```ruby
def solve(coins, amount)
  best = Array.new(amount + 1, amount + 1)
  best[0] = 0
  (1..amount).each do |value|
    coins.each do |coin|
      next if coin > value
      best[value] = [best[value], best[value - coin] + 1].min
    end
  end
  best[amount] <= amount ? best[amount] : -1
end
```

## Διάβασε τον κώδικα

Κάθε μικρότερο ποσό έχει ήδη τη βέλτιστη απάντησή του. Η εξέταση όλων των δυνατών τελευταίων νομισμάτων καλύπτει κάθε λύση. Ένα αδύνατο υπόλοιπο κρατά υποψήφια τιμή μεγαλύτερη από το επιτρεπτό πλήθος και δεν παράγει ψεύτικη εφικτή λύση.

**Κόστος:** `O(a m)` χρόνος και `O(a)` χώρος, για ποσό `a` και `m` καταγεγραμμένες αξίες.

## Πρόσεξε αυτό

Η άπληστη επιλογή του μεγαλύτερου νομίσματος μπορεί να αποτύχει. Για αξίες 1, 3 και 4, το ποσό 6 χρειάζεται δύο τριάρια.

Επιστροφή στην άσκηση 92 · σελίδα 326

Η ΛΥΣΗ

# 93 Πόσοι συνδυασμοί φτιάχνουν το ποσό;

## Η σκέψη

Επεξεργάσου μία αξία τη φορά. Ο πίνακας μετρά όσα ποσά φτιάχνονται χρησιμοποιώντας μόνο τις αξίες που έχουν ήδη εξεταστεί.

RUBY

```ruby
def solve(coins, amount)
  ways = Array.new(amount + 1, 0)
  ways[0] = 1
  coins.uniq.each do |coin|
    (coin..amount).each do |value|
      ways[value] += ways[value - coin]
    end
  end
  ways[amount]
end
```

## Διάβασε τον κώδικα

Στην επεξεργασία μιας αξίας, οι παλιές δυνατότητες μένουν και προστίθενται όσες χρησιμοποιούν ακόμη ένα τέτοιο νόμισμα. Η αύξουσα σειρά ποσών επιτρέπει επαναχρησιμοποίηση της ίδιας αξίας. Η σταθερή σειρά αξιών αποτρέπει διαφορετικές διατάξεις του ίδιου συνδυασμού.

**Κόστος:** `O(a u + m)` αναμενόμενος χρόνος και `O(a + u)` χώρος, όπου `u` οι διαφορετικές αξίες.

## Πρόσεξε αυτό

Αν βάλεις εξωτερικά το ποσό και εσωτερικά την τελευταία αξία, θα μετρήσεις διαφορετικές σειρές των ίδιων νομισμάτων.

Επιστροφή στην άσκηση 93 · σελίδα 329

Η ΛΥΣΗ

# 94 Διάλεξε αντικείμενα με περιορισμένο βάρος

## Η σκέψη

Για κάθε αντικείμενο σύγκρινε την παράλειψή του με την επιλογή του πάνω σε μια προηγούμενη λύση μικρότερης χωρητικότητας.

RUBY

```ruby
def solve(weights, values, capacity)
  best = Array.new(capacity + 1, 0)
  weights.each_with_index do |weight, i|
    capacity.downto(weight) do |space|
      candidate = best[space - weight] + values[i]
      best[space] = [best[space], candidate].max
    end
  end
  best[capacity]
end
```

## Διάβασε τον κώδικα

Πριν από ένα αντικείμενο, κάθε θέση αφορά μόνο προηγούμενα αντικείμενα. Η φθίνουσα ενημέρωση διαβάζει μικρότερες χωρητικότητες που δεν έχουν ακόμη αλλάξει στο τρέχον πέρασμα. Άρα η νέα αξία προστίθεται το πολύ μία φορά σε κάθε υποψήφια λύση.

**Κόστος:** `O(n c)` χρόνος και `O(c)` πρόσθετος χώρος, όπου `c` η χωρητικότητα.

## Πρόσεξε αυτό

Αύξουσα ενημέρωση θα μπορούσε να ξαναχρησιμοποιήσει το ίδιο αντικείμενο μέσα στο ίδιο πέρασμα.

Επιστροφή στην άσκηση 94 · σελίδα 332

Η ΛΥΣΗ

# 95 Η μεγαλύτερη κοινή υποακολουθία

## Η σκέψη

Η κατάσταση περιγράφει δύο προθέματα. Ίσοι τελευταίοι χαρακτήρες μπορούν να επεκτείνουν την κοινή λύση των μικρότερων προθεμάτων. Διαφορετικοί απαιτούν παράλειψη από μία πλευρά.

RUBY

```ruby
def solve(left, right)
  chars = right.chars
  previous = Array.new(chars.length + 1, 0)
  left.each_char do |a|
    current = [0]
    chars.each_with_index do |b, j|
      if a == b
        current << previous[j] + 1
      else
        current << [previous[j + 1], current[-1]].max
      end
    end
    previous = current
  end
  previous[-1]
end
```

Επιστροφή στην άσκηση 95 · σελίδα 336

Η ΜΕΓΑΛΥΤΕΡΗ ΚΟΙΝΗ ΥΠΟΑΚΟΛΟΥΘΙΑ

# 95 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Η θέση `j + 1` αντιστοιχεί στα πρώτα `j + 1` γράμματα του δεξιού κειμένου. Σε ισότητα επεκτείνουμε τη διαγώνια λύση. Διαφορετικά, μία τουλάχιστον τελευταία τιμή δεν μπορεί να χρησιμοποιηθεί μαζί με την άλλη, οπότε παίρνουμε το καλύτερο από τις δύο παραλείψεις.

**Κόστος:** `O((n + 1)(m + 1))` χρόνος και `O(m)` πρόσθετος χώρος για δεξί κείμενο μήκους `m`.

## Πρόσεξε αυτό

Η μεγαλύτερη κοινή συνεχόμενη συμβολοσειρά είναι άλλο πρόβλημα. Εδώ επιτρέπονται κενά μέσα στην επιλογή.

Η ΛΥΣΗ

# 96 Πόσες αλλαγές χωρίζουν δύο κείμενα;

## Η σκέψη

Για δύο προθέματα, η τελευταία ενέργεια είναι διαγραφή, εισαγωγή ή αντιστοίχιση των τελευταίων χαρακτήρων. Δοκίμασε και τις τρεις προηγούμενες καταστάσεις.

RUBY

```ruby
def solve(left, right)
  chars = right.chars
  previous = (0..chars.length).to_a
  left.each_char.with_index do |a, i|
    current = [i + 1]
    chars.each_with_index do |b, j|
      cost = a == b ? 0 : 1
      current << [previous[j + 1] + 1,
                  current[j] + 1,
                  previous[j] + cost].min
    end
    previous = current
  end
  previous[-1]
end
```

## Διάβασε τον κώδικα

Η πάνω τιμή συν μία αντιστοιχεί σε διαγραφή από το αριστερό κείμενο. Η αριστερή συν μία σε εισαγωγή. Η διαγώνια συν το κόστος χαρακτήρων σε αντιστοίχιση ή αντικατάσταση. Η μικρότερη από τις τρεις καλύπτει κάθε δυνατό τελευταίο βήμα μιας βέλτιστης μετατροπής.

**Κόστος:** `O((n + 1)(m + 1))` χρόνος και `O(m)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η αντιστοίχιση ίδιων χαρακτήρων έχει κόστος μηδέν. Η αντικατάσταση διαφορετικών έχει κόστος ένα.

Επιστροφή στην άσκηση 96 · σελίδα 339

Η ΛΥΣΗ

# 97 Η μεγαλύτερη αύξουσα υποακολουθία

## Η σκέψη

Για κάθε εφικτό μήκος κράτα τη μικρότερη δυνατή τελευταία τιμή. Μικρότερο τέλος αφήνει τουλάχιστον τις ίδιες δυνατότητες μελλοντικής επέκτασης.

RUBY

```ruby
def solve(numbers)
  tails = []
  numbers.each do |number|
    position = tails.bsearch_index { |tail| tail >= number }
    tails[position || tails.length] = number
  end
  tails.length
end
```

## Διάβασε τον κώδικα

Η θέση μηδέν κρατά το μικρότερο τέλος μήκους ένα, η θέση ένα μήκους δύο και ούτω καθεξής. Η αντικατάσταση δεν μειώνει κανένα εφικτό μήκος. Αν ο νέος αριθμός ξεπερνά όλα τα τέλη, δημιουργεί νέο μεγαλύτερο μήκος. Το `>=` εμποδίζει τα διπλότυπα να αυξήσουν την απάντηση.

**Κόστος:** `O(n log n)` χρόνος και `O(n)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Ο πίνακας των ελάχιστων τελών δεν είναι απαραίτητα μία πραγματική υποακολουθία. Το μήκος του είναι σωστό, αλλά οι τιμές του δεν αποτελούν ζητούμενη ανακατασκευή.

Επιστροφή στην άσκηση 97 · σελίδα 342

Η ΛΥΣΗ

# 98 Το πιο κερδοφόρο πρόγραμμα εργασιών

## Η σκέψη

Ταξινόμησε κατά τέλος. Μια βέλτιστη λύση είτε παραλείπει την τρέχουσα εργασία είτε τη συνδυάζει με τη βέλτιστη λύση όλων όσων τελειώνουν εγκαίρως.

RUBY

```ruby
def solve(jobs)
  ordered = jobs.sort_by { |job| job[1] }
  finishes = ordered.map { |job| job[1] }
  best = [0]
  ordered.each_with_index do |(start, _finish, profit), i|
    compatible = finishes.bsearch_index { |time| time > start }
    compatible ||= finishes.length
    best << [best[i], profit + best[compatible]].max
  end
  best[-1]
end
```

## Διάβασε τον κώδικα

Το `best[k]` είναι η βέλτιστη αξία των πρώτων `k` εργασιών κατά τέλος. Επειδή κάθε εργασία τελειώνει μετά την αρχή της, όλες οι συμβατές βρίσκονται πριν από την τρέχουσα και η αντίστοιχη απάντηση είναι ήδη έτοιμη. Η σύγκριση επιλέγει παράλειψη ή προσθήκη χωρίς επικάλυψη.

**Κόστος:** `O(n log n)` χρόνος και `O(n)` πρόσθετος χώρος.

## Πρόσεξε αυτό

Η πιο κερδοφόρα μεμονωμένη εργασία ή η πιο σύντομη δεν δίνει πάντα το καλύτερο συνολικό πρόγραμμα.

Επιστροφή στην άσκηση 98 · σελίδα 345

Η ΛΥΣΗ

# 99 Συντομότερες διαδρομές με βάρη

## Η σκέψη

Η ουρά BFS δεν αρκεί όταν οι κινήσεις έχουν διαφορετικό κόστος. Οριστικοποίησε κάθε φορά την ακόμη εκκρεμή κορυφή με τη μικρότερη γνωστή απόσταση.

RUBY

```ruby
def solve(size, edges)
  return [] if size == 0
  graph = Array.new(size) { [] }
  edges.each { |from, to, weight| graph[from] << [to, weight] }
  distance = Array.new(size)
  distance[0] = 0
  done = Array.new(size, false)
  size.times do
    candidates = (0...size).reject do |vertex|
      done[vertex] || distance[vertex].nil?
    end
    vertex = candidates.min_by { |v| distance[v] }
    break if vertex.nil?
    done[vertex] = true
    graph[vertex].each do |to, weight|
      candidate = distance[vertex] + weight
      if distance[to].nil? || candidate < distance[to]
        distance[to] = candidate
      end
    end
  end
  distance.map { |value| value || -1 }
end
```

Επιστροφή στην άσκηση 99 · σελίδα 349

ΣΥΝΤΟΜΟΤΕΡΕΣ ΔΙΑΔΡΟΜΕΣ ΜΕ ΒΑΡΗ

# 99 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Μια διαδρομή που περνά αργότερα από εκκρεμή κορυφή δεν μπορεί να βελτιώσει την τρέχουσα ελάχιστη απόσταση, επειδή όλα τα επιπλέον βάρη είναι μη αρνητικά. Έτσι η επιλεγμένη κορυφή οριστικοποιείται. Η χαλάρωση των ακμών διατηρεί την καλύτερη γνωστή προσέγγιση για όσες απομένουν.

**Κόστος:** `O(v² + e)` χρόνος και `O(v + e)` χώρος. Η σάρωση των υποψηφίων αποφεύγει την ανάγκη heap στα δοσμένα όρια.

## Πρόσεξε αυτό

Η εγγύηση του Dijkstra απαιτεί μη αρνητικά βάρη. Μια απλή πρώτη ανακάλυψη δεν είναι ακόμη τελική απάντηση.

Η ΛΥΣΗ

# 100 Η φθηνότερη διαδρομή από όλες τις πόλεις

## Η σκέψη

Δύο μερικές διαδρομές με το ίδιο σύνολο επισκέψεων και την ίδια τελευταία πόλη έχουν ακριβώς τις ίδιες επιλογές συνέχειας. Αρκεί να κρατήσεις το μικρότερο κόστος ανά τέτοια κατάσταση.

RUBY

```ruby
def solve(costs)
  size = costs.length
  return 0 if size < 2
  states = 1 << size
  dp = Array.new(states) { Array.new(size) }
  dp[1][0] = 0
  (1...states).each do |mask|
    size.times do |last|
      value = dp[mask][last]
      next if value.nil?
      (1...size).each do |city|
        bit = 1 << city
        next if (mask & bit) != 0
        next_mask = mask | bit
        candidate = value + costs[last][city]
        old = dp[next_mask][city]
        dp[next_mask][city] = candidate if old.nil? || candidate < old
      end
    end
  end
  (1...size).map { |last| dp[-1][last] + costs[last][0] }.min
end
```

Επιστροφή στην άσκηση 100 · σελίδα 352

Η ΦΘΗΝΟΤΕΡΗ ΔΙΑΔΡΟΜΗ ΑΠΟ ΟΛΕΣ ΤΙΣ ΠΟΛΕΙΣ

# 100 Γιατί λειτουργεί

## Διάβασε τον κώδικα

Η βάση έχει επισκεφτεί μόνο την πόλη μηδέν. Κάθε μετάβαση ενεργοποιεί ένα νέο bit, άρα πηγαίνει σε μεγαλύτερη μάσκα και σε κατάσταση που θα εξεταστεί αργότερα. Από κάθε πλήρη μάσκα προσθέτουμε την τελική ακμή επιστροφής. Η συγχώνευση ισοδύναμων μερικών διαδρομών είναι ο αλγόριθμος Held-Karp.

**Κόστος:** $O(n^2 \cdot 2^n)$ χρόνος και $O(n \cdot 2^n)$ χώρος. Το όριο δέκα πόλεων κρατά τον εκθετικό χώρο διαχειρίσιμο.

## Πρόσεξε αυτό

Η διαδρομή πρέπει να επιστρέψει στην αφετηρία. Η μικρότερη απάντηση που επισκέπτεται απλώς όλες τις πόλεις δεν αρκεί.

ΜΕΤΑ ΤΙΣ ΕΚΑΤΟ ΑΣΚΗΣΕΙΣ

# Δοκίμασε τη σκέψη σου

Διάλεξε μια άσκηση που σε δυσκόλεψε. Πρόσθεσε ένα test με διαφορετική είσοδο και εξήγησε πρώτα την αναμενόμενη απάντηση. Έπειτα τρέξε την εντολή του εργαλείου και διάβασε το αποτέλεσμα.

Αν το test αποτύχει, έλεγξε αν το λάθος βρίσκεται στην προσπάθειά σου ή στην αναμενόμενη τιμή που έγραψες. Όταν περάσει, εξήγησε ποια περίπτωση έλεγξες που έλειπε πριν.

**ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ**

| είσοδος | αποτέλεσμα |
| --- | --- |