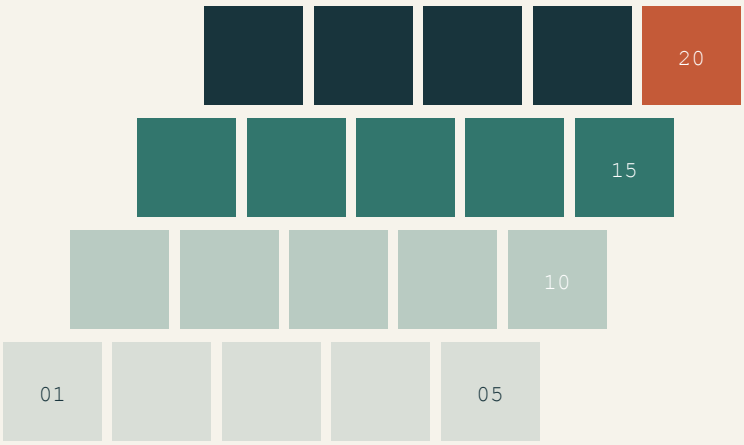
ΤΕΤΡΑΔΙΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Προγραμματισμός με μικρά βήματα

20 πολύ εύκολες ασκήσεις

Από μια σταθερή απάντηση στο FizzBuzz



Elixir

ΠΡΩΤΟ ΜΕΡΟΣ

ΕΛΛΗΝΙΚΗ ΕΚΔΟΣΗ

ΔΙΑΒΑΣΕ ΚΑΙ ΔΟΚΙΜΑΣΕ

Η σειρά σου

Πριν ξεκινήσεις · Η γλώσσα · Testing · Η διαδρομή

Οι τίτλοι και οι αριθμοί σελίδων είναι σύνδεσμοι.

ΑΠΟ ΤΟ ΜΗΔΕΝ

Πριν ξεκινήσεις

Ξεκινάς από το μηδέν και λύνεις είκοσι μικρές ασκήσεις σε Elixir. 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 `elixir` 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.ex. Το exercise_test.exs περιέχει τις δοκιμές της.

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 Elixir με μια ματιά

Η Elixir οργανώνει τον κώδικα σε modules και συναρτήσεις. Εδώ γράφεις συναρτήσεις σε αρχεία `.ex` και tests σε αρχεία `.exs`. Το Mix μεταγλωττίζει τον κώδικα και εκτελεί τα tests με `mix test`.

Εγκατάσταση στο macOS

Αν έχεις Homebrew, η επίσημη οδηγία της Elixir δίνει την εντολή.

```sh
brew install elixir
elixir --version
```

Η εγκατάσταση περιλαμβάνει και την απαιτούμενη Erlang/OTP, πάνω στην οποία τρέχει η Elixir. Η εντολή έκδοσης εμφανίζει πληροφορίες και για τις δύο. Αν δεν έχεις Homebrew, η ίδια επίσημη σελίδα δίνει εγκατάσταση με script. Δεν χρειάζεσαι πρόσθετα πακέτα Hex για το βιβλίο.

Διάβασε μια μικρή συνάρτηση

```elixir
defmodule Example do
  def value() do
    7
  end
end
```

Το `defmodule` ορίζει μια ομάδα συναρτήσεων με όνομα `Example`. Το `def value()` ορίζει μια συνάρτηση χωρίς παραμέτρους. Το `do` ξεκινά το σώμα και το `end` το κλείνει. Η τελευταία έκφραση, εδώ το `7`, είναι η τιμή που επιστρέφει η συνάρτηση. Δεν υπάρχει ανάγκη να γράψεις `return`.

Η κλήση `Example.value()` δίνει 7. Στις ασκήσεις τα modules λέγονται `Exercise01` έως `Exercise20` και η συνάρτηση `solve`. Για παράδειγμα, η πρώτη καλείται με `Exercise01.solve()`.

Στο `def solve(n) do`, το `n` είναι το όνομα της παραμέτρου. Οι επικεφαλίδες μας δεν δηλώνουν τύπους. Οι εκφωνήσεις ορίζουν αν μια είσοδος είναι ακέραιος και τι αποτέλεσμα ζητείται. Τα κείμενα γράφονται με διπλά εισαγωγικά και οι λογικές τιμές ως `true` και `false`.

Στην άσκηση 9 θα δεις `rem(a, b)` για το υπόλοιπο διαίρεσης. Στις αποφάσεις, το `if` ή το `cond` δίνει μια τιμή που μπορεί να είναι κατευθείαν το αποτέλεσμα της συνάρτησης. Οι σχετικές ασκήσεις εξηγούν αυτά τα εργαλεία όταν τα χρειαστείς.

ΤΟ ΕΡΓΑΛΕΙΟ TESTING

ExUnit / mix test

Το ExUnit περιλαμβάνεται στην Elixir και παρέχει ονομασμένα tests, assertions, setup και υποστήριξη ταυτόχρονης εκτέλεσης. Το Mix αναλαμβάνει τη μεταγλώττιση και την ανακάλυψη των tests σε ένα project. Το ExUnit μπορεί επίσης να τρέξει σε μεμονωμένο αρχείο με την εντολή `elixir`.

Υπάρχει και το ESpec, ένα χωριστό framework εμπνευσμένο από το RSpec, με `describe`, `context`, `it` και matchers. Η ύπαρξη μιας σύνταξης κοντά στη Ruby δεν είναι από μόνη της λόγος να την επιλέξουμε για την εκμάθηση της Elixir.

Στο βιβλίο χρησιμοποιούμε **ExUnit με mix test**, ώστε οι εντολές να συνδέονται με τον τρόπο που οργανώνεται ένα κανονικό project Elixir. Στη συνέχεια εξηγούμε τα `test`, `assert`, το `test_helper.exs` και η εκτέλεση ενός συγκεκριμένου test.

Πηγές: ExUnit, Mix, ESpec.

EXUNIT / MIX TEST

Ετοίμασε το project

Από τον φάκελο `elixir`:

```sh
elixir --version
mix --version
```

Το ExUnit περιλαμβάνεται στην Elixir. Το `mix.exs` δηλώνει το project και τις θέσεις του κώδικα και των tests. Το `exercises/test_helper.exs` περιέχει το `ExUnit.start()`, που ξεκινά το framework πριν φορτωθούν οι δοκιμές.

Κάθε άσκηση έχει δικό της module, από `Exercise01` μέχρι `Exercise20`. Έτσι μπορούν να μεταγλωττίζονται μαζί χωρίς να αντικαθιστά η μία τη συνάρτηση της άλλης. Το αρχείο που αλλάζεις στην πρώτη άσκηση είναι το `exercises/01-hello/exercise.ex`.

EXUNIT / MIX TEST

Διάβασε το πρώτο test

Το διπλανό `exercise_test.exs` περιέχει:

ELIXIR

```elixir
defmodule Exercise01Test do
  use ExUnit.Case
  alias Exercise01, as: Exercise

  test "exact greeting" do
    assert Exercise.solve() === "Hello!"
  end
end
```

Το `use ExUnit.Case` δίνει στο module τα εργαλεία του ExUnit. Το `alias Exercise01, as: Exercise` επιτρέπει να αναφερόμαστε στη συγκεκριμένη άσκηση με το σύντομο όνομα `Exercise`. Το `test` δίνει περιγραφή στη δοκιμή. Το `assert` απαιτεί να είναι αληθής η σύγκριση που ακολουθεί. Το `===` ελέγχει αυστηρή ισότητα.

EXUNIT / MIX TEST

Τρέξε την πρώτη άσκηση

Από τον φάκελο `elixir`:

```sh
mix test exercises/01-hello/exercise_test.exs --trace
```

Το Mix μεταγλωττίζει τα `.ex`, φορτώνει το test helper και εκτελεί το συγκεκριμένο αρχείο δοκιμών. Το `--trace` εμφανίζει τα ονόματά τους. Το `_build` που δημιουργείται περιέχει αποτελέσματα της μεταγλώττισης.

Αρχικά το `raise` με `TODO` θα προκαλέσει αποτυχία. Με σωστή υλοποίηση η αναφορά δείχνει ότι πέρασαν όλα τα tests: `Result: 1 passed` στην έκδοση που ελέγχθηκε ή `0 failures` σε παλαιότερες εκδόσεις. Αν αποτύχει μια σύγκριση, το ExUnit εμφανίζει την έκφραση, τις τιμές και τη γραμμή του test.

EXUNIT / MIX TEST

Πρόσθεσε δικό σου test

Στο module `Exercise02Test`, πριν από το τελευταίο `end` του `exercises/02-echo/exercise_test.exs`, πρόσθεσε:

ELIXIR

```elixir
test "my number" do
  assert Exercise.solve(23) === 23
end
```

Τρέξε `mix test exercises/02-echo/exercise_test.exs --trace`. Θα εμφανιστεί και το `my number`.

Το `mix test` χωρίς διαδρομή εκτελεί όλα τα tests του project. Οι άλυτες ασκήσεις θα αποτύχουν. Αργότερα μπορείς να γνωρίσεις το `setup` για προετοιμασία δεδομένων και το `async: true` για ανεξάρτητα tests που επιτρέπεται να τρέξουν ταυτόχρονα.

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.

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.

ELIXIR

```
1  defmodule Example do
2    def example() do
3      "Ready!"
4    end
5  end
```

Διάβασε τις γραμμές

01 Το `defmodule Example do` αρχίζει ένα module, ένα όνομα που ομαδοποιεί συναρτήσεις. Το τελευταίο `end` κλείνει αυτό το module.

02 Το `def example() do` ορίζει τη συνάρτηση `example`. Οι κενές `()` σημαίνουν ότι δεν δέχεται παραμέτρους. Το εσωτερικό `end` την κλείνει.

03 Το `do` ανοίγει ένα σώμα και το αντίστοιχο `end` το κλείνει. Η εσοχή βοηθά να δεις σε ποιο σώμα ανήκει κάθε γραμμή.

04 Το `"Ready!"` είναι κείμενο και η τελευταία έκφραση της συνάρτησης. Η Elixir επιστρέφει την τιμή του χωρίς εντολή `return`. Τα διπλά εισαγωγικά δεν είναι χαρακτήρες του αποτελέσματος.

05 Η `Example.example()` καλεί τη συνάρτηση. Η τελεία χωρίζει το module από το όνομα της συνάρτησης. Η κλήση επιστρέφει τιμή, δεν την εμφανίζει μόνη της.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

01 Από τον κώδικα στην τιμή

Ακολούθησε μια κλήση

Η `Example.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

Στην άσκηση το module λέγεται `Exercise01` και η συνάρτηση `solve`. Κράτησε αυτά τα ονόματα και τα `do` και `end`. Αντικατάστησε το σχόλιο `TODO` και το `raise` με το δικό σου κείμενο.

Συνέχισε στην εκφώνηση της άσκησης 01, στη σελίδα 16.

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 **elixir**.

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

Tests: exercises/01-hello/exercise_test.exs.

```sh
mix test exercises/01-hello/exercise_test.exs --trace
```

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 83.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    value
  end
end
```

Διάβασε τις γραμμές

01 Το `value` μέσα στις παρενθέσεις είναι η παράμετρος. Εδώ δεν γράφουμε τύπο δίπλα της. Η κλήση της δίνει έναν ακέραιο. Το `value` στο σώμα επιστρέφει την τιμή αυτή ως τελευταία έκφραση.

02 Στον ορισμό, το `value` είναι παράμετρος. Στην κλήση με `6`, το `6` είναι όρισμα. Δεν γράφουμε ούτε τύπο ούτε όνομα παραμέτρου στην κλήση.

03 Το `value` στο σώμα διαβάζει την τιμή της παραμέτρου. Το `"value"` με εισαγωγικά θα ήταν κείμενο, όχι η τιμή της.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 19.

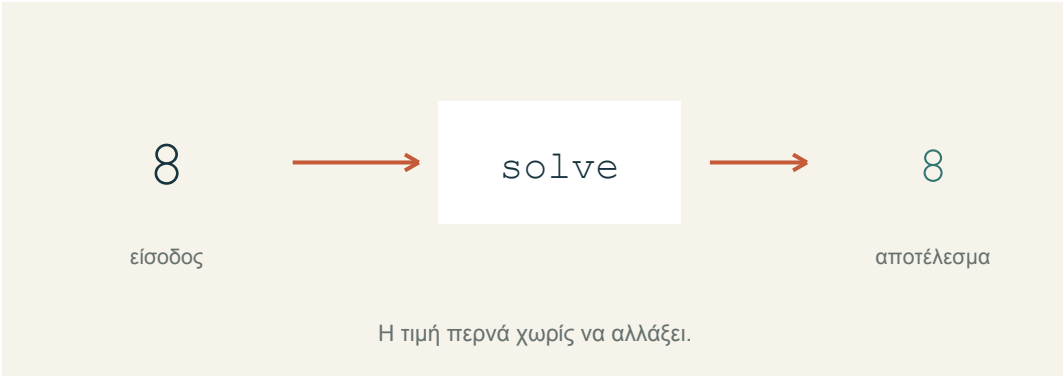
Η ΑΣΚΗΣΗ

02 Ο ίδιος αριθμός

Τι θα φτιάξεις

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

Όρια: -1000 ≤ n ≤ 1000.



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 **elixir**.

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

Tests: exercises/02-echo/exercise_test.exs.

```sh
mix test exercises/02-echo/exercise_test.exs --trace
```

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 84.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    value + 3
  end
end
```

Διάβασε τις γραμμές

01 Το `+` ανάμεσα σε δύο ακεραίους υπολογίζει το άθροισμα. Πρώτα διαβάζεται η τιμή του `value` και μετά προστίθεται το `3`.

02 Το αποτέλεσμα της έκφρασης επιστρέφεται. Δεν αλλάζουμε την παράμετρο `value` και δεν χρειάζεται νέο τοπικό όνομα.

03 Το `3` χωρίς εισαγωγικά είναι αριθμός. Στο `"3"` τα εισαγωγικά θα το έκαναν κείμενο, ακατάλληλο για αυτή την ακέραια πρόσθεση.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 22.

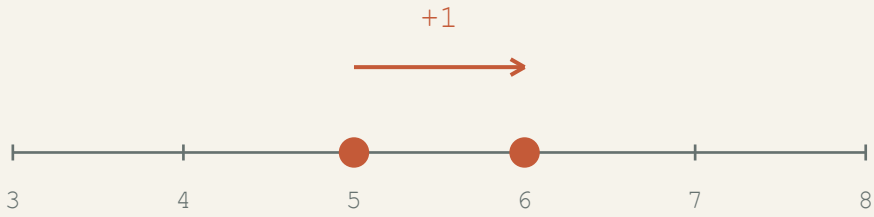
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/03-next-number/exercise_test.exs.

```sh
mix test exercises/03-next-number/exercise_test.exs --trace
```

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 85.

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.

ELIXIR

```elixir
defmodule Example do
  def example(apples, pears) do
    apples + pears
  end
end
```

Διάβασε τις γραμμές

01 Στο `apples, pears`, το κόμμα χωρίζει τις δύο παραμέτρους. Κάθε παράμετρος 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 και τις δύο τιμές και επιστρέφει το άθροισμα. Τα ονόματα βοηθούν να καταλάβουμε τι μετράμε.

Ακολούθησε μια κλήση

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

Η `Example.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 25.

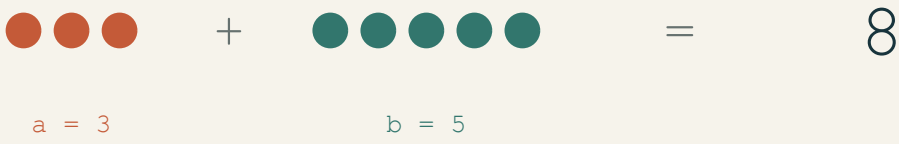
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/04-add/exercise_test.exs.

```sh
mix test exercises/04-add/exercise_test.exs --trace
```

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 86.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    value * 3
  end
end
```

Διάβασε τις γραμμές

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

02 Στο `value * 3` το `value` διαβάζεται από την κλήση, ενώ το `3` είναι σταθερό. Το αποτέλεσμα παραμένει ακέραιος.

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

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 28.

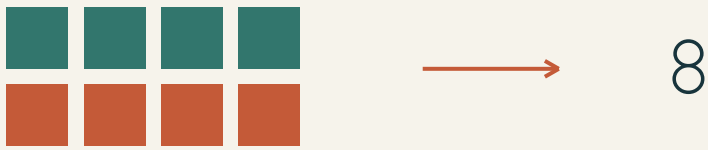
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/05-double/exercise_test.exs.

```sh
mix test exercises/05-double/exercise_test.exs --trace
```

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 87.

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.

ELIXIR

```elixir
defmodule Example do
  def example(boxes) do
    items_per_box = 10
    total = boxes * items_per_box
    total
  end
end
```

Διάβασε τις γραμμές

01 Το `items_per_box = 10` συνδέει το νέο όνομα με την τιμή `10`. Εδώ χρησιμοποιούμε το `=` σε αυτή την απλή μορφή. Η γενικότερη χρήση του για αντιστοίχιση προτύπων θα έρθει αργότερα.

02 Το `total = boxes * items_per_box` δίνει όνομα στο γινόμενο. Το τελικό `total` επιστρέφει την τιμή του ως τελευταία έκφραση.

03 Και οι δύο πλευρές του `*` είναι τώρα ονόματα μεταβλητών. Χρησιμοποιούνται οι τιμές τους. Τα τοπικά ονόματα ανήκουν στη συνάρτηση και ορίζονται ξανά σε κάθε κλήση.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

06 Από τον κώδικα στην τιμή

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 32.

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 **elixir**.

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

Tests: exercises/06-minutes-to-seconds/exercise_test.exs.

```sh
mix test exercises/06-minutes-to-seconds/exercise_test.exs --trace
```

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 88.

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.

ELIXIR

```elixir
defmodule Example do
  def example(price, coupon) do
    price - coupon
  end
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. Αυτό δεν φαινόταν στην πρόσθεση, αλλά εδώ είναι ουσιαστικό.

03 Στο `5 - 20` το - κάνει αφαίρεση δύο τιμών. Στο αποτέλεσμα `-15` δηλώνει ότι ο αριθμός είναι αρνητικός.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 35.

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 **elixir**.

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

Tests: exercises/07-difference/exercise_test.exs.

```sh
mix test exercises/07-difference/exercise_test.exs --trace
```

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 89.

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.

ELIXIR

```elixir
defmodule Example do
  def example(first, second) do
    (first + second) * 3
  end
end
```

Διάβασε τις γραμμές

01 Οι εσωτερικές παρενθέσεις ομαδοποιούν το άθροισμα. Πρώτα υπολογίζεται το `first + second` και μετά το γινόμενο.

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 εδώ ομαδοποιούν πράξεις. Οι παρενθέσεις δίπλα στο όνομα της συνάρτησης έχουν άλλο ρόλο, τις παραμέτρους ή τα ορίσματα.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 38.

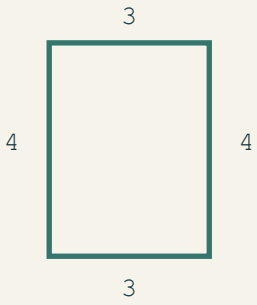
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/08-rectangle-perimeter/exercise_test.exs.

```sh
mix test exercises/08-rectangle-perimeter/exercise_test.exs --trace
```

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 90.

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.

ELIXIR

```elixir
defmodule Example do
  def example(tokens, boxes) do
    rem(tokens, boxes)
  end
end
```

Διάβασε τις γραμμές

01 Η έτοιμη συνάρτηση `rem(tokens, boxes)` δίνει το υπόλοιπο. Το `rem` καλείται με δύο ορίσματα. Το `div(tokens, boxes)` θα έδινε το ακέραιο πηλίκο, που είναι διαφορετικό αποτέλεσμα.

02 Οι παρενθέσεις και το κόμμα της `rem` ακολουθούν τον ίδιο κανόνα κλήσης που ήδη ξέρεις.

03 Στα παραδείγματα οι μάρκες είναι μη αρνητικές και τα κουτιά περισσότερα από μηδέν. Δεν διαιρούμε ποτέ με το μηδέν.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 41.

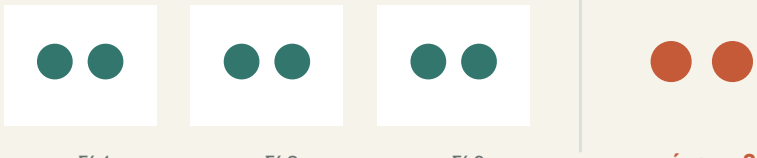
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/09-leftover/exercise_test.exs.

```sh
mix test exercises/09-leftover/exercise_test.exs --trace
```

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 91.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    value == 5
  end
end
```

Διάβασε τις γραμμές

01 Το `==` συγκρίνει την αριστερή τιμή με τη δεξιά. Δεν αλλάζει το `value`. Το μονό `=` δεν είναι ο έλεγχος ισότητας.

02 Η σύγκριση επιστρέφει `true` ή `false`. Είναι λογικές τιμές. Δεν γράφουμε εισαγωγικά γύρω τους.

03 Δεν χρειάζεται να γράψουμε ξεχωριστή απόφαση. Η ίδια η σύγκριση παράγει την απάντηση που θα επιστρέψουμε.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 44.

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 **elixir**.

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

Tests: exercises/10-is-zero/exercise_test.exs.

```sh
mix test exercises/10-is-zero/exercise_test.exs --trace
```

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 92.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

11 Μεγαλύτερο σημαίνει αυστηρά μεγαλύτερο

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

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    value > 10
  end
end
```

Διάβασε τις γραμμές

01 Το `value > 10` είναι μια σύγκριση, όπως η ισότητα. Δίνει λογική τιμή, όχι τη διαφορά των δύο αριθμών.

02 Όταν οι δύο τιμές είναι ίσες, το `>` δίνει ψευδές. Δοκίμαζε πάντα και τον ίδιο τον αριθμό του ορίου.

03 Το αντίστροφο σύμβολο `<` σημαίνει «μικρότερο». Για παράδειγμα, το `9 < 10` είναι αληθές. Η ανοιχτή πλευρά του συμβόλου κοιτά προς τη μεγαλύτερη τιμή.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 47.

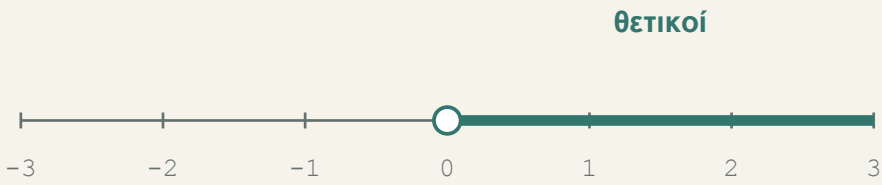
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/11-is-positive/exercise_test.exs.

SH
```
mix test exercises/11-is-positive/exercise_test.exs --trace
```

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 93.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

12 Όταν το όριο συμπεριλαμβάνεται

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

ELIXIR

```elixir
defmodule Example do
  def example(available, needed) do
    available >= needed
  end
end
```

Διάβασε τις γραμμές

01 Το `>=` σημαίνει «μεγαλύτερο ή ίσο». Οι δύο χαρακτήρες γράφονται ενωμένοι και με αυτή τη σειρά. Δεν γράφουμε `=>`.

02 Η σύγκριση χρησιμοποιεί δύο παραμέτρους. Το όριο είναι η τιμή του `needed` σε αυτή την κλήση.

03 Το αντίστοιχο `<=` σημαίνει «μικρότερο ή ίσο». Και οι δύο συγκρίσεις περιλαμβάνουν την ισότητα.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 50.

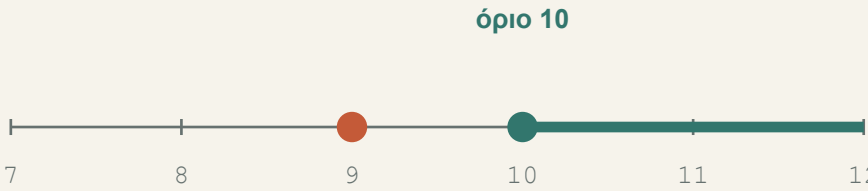
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/12-at-least/exercise_test.exs.

```sh
mix test exercises/12-at-least/exercise_test.exs --trace
```

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 94.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    rem(value, 2) == 1
  end
end
```

Διάβασε τις γραμμές

01 Πρώτα υπολογίζεται το `rem(value, 2)`. Έπειτα το `== 1` συγκρίνει το υπόλοιπο με το `1`.

02 Η έκφραση έχει αριθμητικό ενδιάμεσο αποτέλεσμα, αλλά η τελική τιμή της είναι λογική. Ο τύπος αποτελέσματος το περιγράφει.

03 Εδώ χρησιμοποιούμε μη αρνητικούς ακεραίους. Το υπόλοιπο στη διαίρεση με το `2` είναι τότε είτε `0` είτε `1`.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 53.

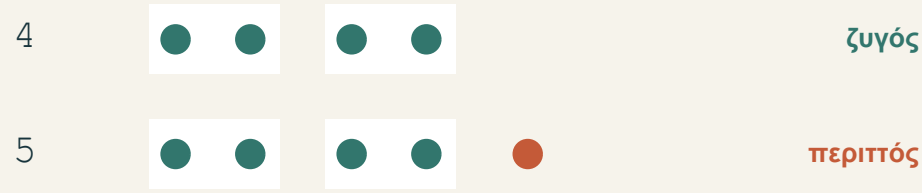
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/13-is-even/exercise_test.exs.

```sh
mix test exercises/13-is-even/exercise_test.exs --trace
```

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 95.

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`.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    if value > 10 do
      1
    else
      0
    end
  end
end
```

Διάβασε τις γραμμές

01 Το `if value > 10 do` ανοίγει τον πρώτο κλάδο. Το `else` αρχίζει τον δεύτερο και το εσωτερικό `end` κλείνει την επιλογή. Το `if` δίνει την τελευταία τιμή του κλάδου που εκτελέστηκε. Είναι και η τελευταία έκφραση της συνάρτησης, άρα αυτή επιστρέφεται.

02 Κλάδος είναι ο κώδικας που αντιστοιχεί σε μία επιλογή. Σε αυτή την κλήση εκτελείται μόνο ένας από τους δύο κλάδους.

03 Η συνθήκη είναι λογική, αλλά η απάντηση της συνάρτησης είναι ακέραια. Άλλο ο έλεγχος και άλλο η τιμή που επιλέγει.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

14 Από τον κώδικα στην τιμή

Μία κλήση δίνει μία τιμή.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 57.

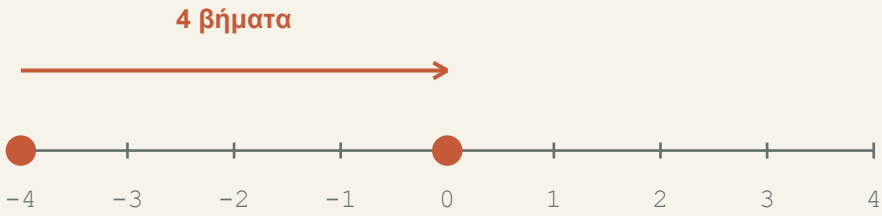
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/14-absolute/exercise_test.exs.

```sh
mix test exercises/14-absolute/exercise_test.exs --trace
```

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 96.

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.

ELIXIR

```elixir
defmodule Example do
  def example(a, b) do
    if a > b do
      b
    else
      a
    end
  end
end
```

Διάβασε τις γραμμές

01 Το `if` ελέγχει αν το `a` είναι μεγαλύτερο από το `b`. Αν ναι, μικρότερο είναι το `b`. Διαφορετικά κρατάμε το `a`.

02 Μια παράμετρος μπορεί να μπει στη θέση μιας σταθερής απάντησης. Ο κλάδος επιστρέφει την τρέχουσα τιμή της.

03 Η ισότητα οδηγεί στον δεύτερο κλάδο, επειδή χρησιμοποιούμε `>`. Εδώ αυτό είναι σωστό, αφού οι τιμές είναι ίδιες.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

15 Από τον κώδικα στην τιμή

Μία κλήση δίνει μία τιμή.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 61.

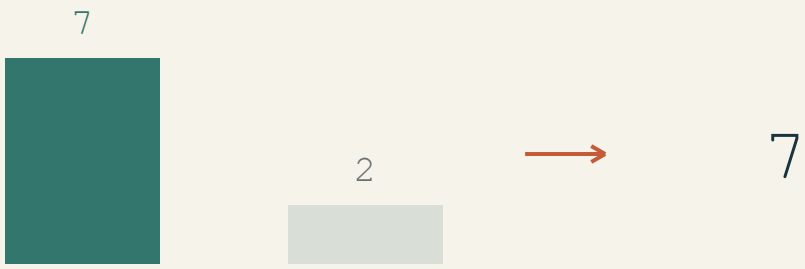
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/15-larger/exercise_test.exs.

```sh
mix test exercises/15-larger/exercise_test.exs --trace
```

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 97.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    cond do
      value > 10 -> 2
      value == 10 -> 1
      true -> 0
    end
  end
end
```

Διάβασε τις γραμμές

01 Το `cond do` ελέγχει τις γραμμές με τη σειρά. Σε κάθε γραμμή το `->` χωρίζει τη συνθήκη από το αποτέλεσμά της. Το `true -> 0` είναι η τελευταία, πάντα αληθής περίπτωση, ώστε να υπάρχει απάντηση όταν αποτύχουν οι προηγούμενες.

02 Διαβάζουμε από πάνω προς τα κάτω. Η πρώτη συνθήκη που πετυχαίνει καθορίζει το αποτέλεσμα. Δεν αθροίζουμε τις απαντήσεις.

03 Και οι τρεις κλάδοι δίνουν ακέραιο. Χρειάζεται απάντηση και για τιμή μικρότερη, ίση και μεγαλύτερη από το όριο.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

16 Από τον κώδικα στην τιμή

Μία κλήση δίνει μία τιμή.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 65.

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 **elixir**.

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

Tests: exercises/16-sign/exercise_test.exs.

```sh
mix test exercises/16-sign/exercise_test.exs --trace
```

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 98.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value, limit) do
    value >= 2 and value <= limit
  end
end
```

Διάβασε τις γραμμές

01 Το `and` ενώνει δύο λογικές συγκρίσεις. Δίνει αληθές μόνο όταν και οι δύο δίνουν αληθές.

02 Η πρώτη σύγκριση περιλαμβάνει το κάτω όριο `2`. Η δεύτερη περιλαμβάνει το πάνω όριο `limit`. Αν η πρώτη είναι ψευδής, η δεύτερη δεν χρειάζεται να υπολογιστεί.

03 Κάθε σύγκριση έχει και το δικό της `value`. Γράφουμε τους δύο ελέγχους ολόκληρους για να φαίνονται καθαρά.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

17 Από τον κώδικα στην τιμή

Μία κλήση δίνει μία τιμή.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 69.

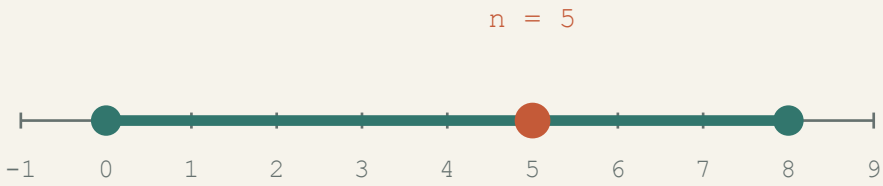
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/17-in-range/exercise_test.exs.

```sh
mix test exercises/17-in-range/exercise_test.exs --trace
```

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 99.

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.

ELIXIR

```elixir
defmodule Example do
  def example(total, size) do
    rem(total, size) == 0
  end
end
```

Διάβασε τις γραμμές

01 Στο `rem(total, size)`, το `size` παίρνει τον ρόλο του διαιρέτη. Χρησιμοποιούμε την τιμή της παραμέτρου, όχι σταθερό αριθμό.

02 Το `== 0` ελέγχει αν δεν περίσσεψε τίποτα. Δεν επιστρέφουμε το ίδιο το υπόλοιπο αλλά το αποτέλεσμα της σύγκρισης.

03 Το `size` πρέπει να είναι θετικό. Μηδενικό `total` με θετικό `size` αφήνει μηδενικό υπόλοιπο και περνά τον έλεγχο.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 72.

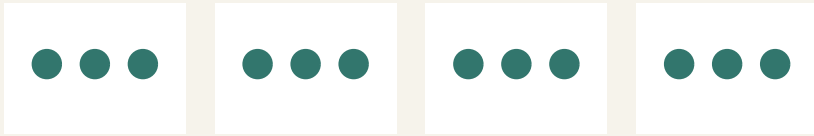
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/18-is-multiple/exercise_test.exs.

SH

```
mix test exercises/18-is-multiple/exercise_test.exs --trace
```

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 100.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    if rem(value, 2) == 0 do
      "Pair"
    else
      ""
    end
  end
end
```

Διάβασε τις γραμμές

01 Κάθε κλάδος τελειώνει με κείμενο. Η τιμή του επιλεγμένου κλάδου γίνεται η τιμή του `if` και της συνάρτησης.

02 Το `""` έχει μηδέν χαρακτήρες. Το `" "` θα είχε ένα κενό και θα ήταν διαφορετική απάντηση.

03 Η σύγκριση παραμένει λογική. Χρησιμοποιείται για την επιλογή κειμένου και δεν είναι η τελική απάντηση.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

19 Από τον κώδικα στην τιμή

Μία κλήση δίνει μία τιμή.

Ακολούθησε μια κλήση

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 76.

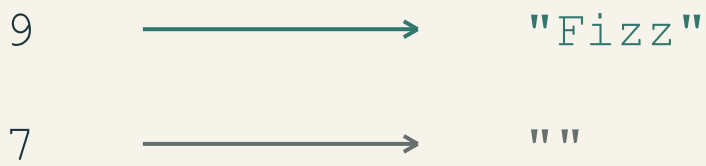
Η ΑΣΚΗΣΗ

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 **elixir**.

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

Tests: exercises/19-fizz/exercise_test.exs.

```sh
mix test exercises/19-fizz/exercise_test.exs --trace
```

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 101.

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.

ELIXIR

```elixir
defmodule Example do
  def example(value) do
    cond do
      rem(value, 2) == 0 and rem(value, 3) == 0 -> "PairTrio"
      rem(value, 2) == 0 -> "Pair"
      rem(value, 3) == 0 -> "Trio"
      true -> ""
    end
  end
end
```

Διάβασε τις γραμμές

01 Ο πρώτος έλεγχος ενώνει με `and` τους δύο κανόνες. Πρέπει να προηγείται, επειδή κάθε κοινό πολλαπλάσιο θα περνούσε και κάποιον από τους απλούς ελέγχους.

02 Το `cond` σταματά στον πρώτο επιτυχημένο έλεγχο. Η τελική γραμμή `true -> ""` καλύπτει τους υπόλοιπους αριθμούς.

03 Δεν εμφανίζουμε πολλά αποτελέσματα και δεν επαναλαμβάνουμε την εκτέλεση μέσα στη συνάρτηση. Μία κλήση δίνει μία ετικέτα.

ΤΟ ΙΔΙΟ ΠΑΡΑΔΕΙΓΜΑ, ΒΗΜΑ ΒΗΜΑ

20 Από τον κώδικα στην τιμή

Μία κλήση δίνει μία τιμή.

Ακολούθησε μια κλήση

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

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

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

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

Στάσου λίγο

Τι θα επιστρέψει η `Example.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 80.

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 **elixir**.

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

Tests: exercises/20-fizzbuzz-label/exercise_test.exs.

```sh
mix test exercises/20-fizzbuzz-label/exercise_test.exs --trace
```

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 102.

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.

ELIXIR

```elixir
defmodule Exercise01 do
  def solve() do
    "Hello!"
  end
end
```

Διάβασε τον κώδικα

Το κείμενο είναι η τελευταία έκφραση της `solve`, άρα είναι και η τιμή που επιστρέφει. Δεν γράφουμε `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 16

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.

ELIXIR

```elixir
defmodule Exercise02 do
  def solve(n) do
    n
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

Πρόσεξε αυτό

Το `"n"` είναι το γράμμα n ως κείμενο. Το `n` είναι η τιμή της παραμέτρου.

Επιστροφή στην άσκηση 02 · σελίδα 19

Η ΛΥΣΗ

03 Ο επόμενος αριθμός

Η σκέψη

Υπολογίζουμε `n + 1` και επιστρέφουμε το αποτέλεσμα. Για -1, η πρόσθεση δίνει 0.

ELIXIR

```elixir
defmodule Exercise03 do
  def solve(n) do
    n + 1
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

Πρόσεξε αυτό

Δεν χρειάζεται να αλλάξεις την παράμετρο ή να γράψεις ξεχωριστό κανόνα για τους αρνητικούς.

Επιστροφή στην άσκηση 03 · σελίδα 22

Η ΛΥΣΗ

04 Δύο αριθμοί μαζί

Η σκέψη

Προσθέτουμε τις δύο παραμέτρους. Δεν χρησιμοποιούμε σταθερούς αριθμούς από τα παραδείγματα.

ELIXIR

```elixir
defmodule Exercise04 do
  def solve(a, b) do
    a + b
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

Πρόσεξε αυτό

Η λύση πρέπει να δουλεύει με οποιοδήποτε ζευγάρι μέσα στα όρια, όχι μόνο με το πρώτο παράδειγμα.

Επιστροφή στην άσκηση 04 · σελίδα 25

Η ΛΥΣΗ

05 Το διπλάσιο

Η σκέψη

Πολλαπλασιάζουμε το `n` με το 2. Το μηδέν παραμένει μηδέν και οι αρνητικοί αριθμοί παραμένουν αρνητικοί.

ELIXIR

```elixir
defmodule Exercise05 do
  def solve(n) do
    n * 2
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

Πρόσεξε αυτό

Το `n + 2` προσθέτει δύο μονάδες. Δεν δίνει γενικά το διπλάσιο.

Επιστροφή στην άσκηση 05 · σελίδα 28

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.

ELIXIR

```elixir
defmodule Exercise06 do
  def solve(minutes) do
    seconds_per_minute = 60
    seconds = minutes * seconds_per_minute
    seconds
  end
end
```

Διάβασε τον κώδικα

Το `=` συνδέει το όνομα `seconds_per_minute` με την τιμή 60 και το `seconds` με την υπολογισμένη τιμή. Το τελευταίο `seconds` δίνει το αποτέλεσμα της συνάρτησης.

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 32

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.

ELIXIR

```elixir
defmodule Exercise07 do
  def solve(a, b) do
    a - b
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

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 35

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.

ELIXIR

```elixir
defmodule Exercise08 do
  def solve(width, height) do
    2 * (width + height)
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

Πρόσεξε αυτό

Το `2 * width + height` διπλασιάζει μόνο το πλάτος.

Επιστροφή στην άσκηση 08 · σελίδα 38

Η ΛΥΣΗ

09 Τι περισσεύει

Η σκέψη

Παίρνουμε το υπόλοιπο της διαίρεσης `candies` με `children`. Αν οι καραμέλες μοιράζονται ακριβώς, το αποτέλεσμα είναι 0.

ELIXIR

```elixir
defmodule Exercise09 do
  def solve(candies, children) do
    rem(candies, children)
  end
end
```

Διάβασε τον κώδικα

Η `rem(a, b)` υπολογίζει το υπόλοιπο. Η Elixir δεν χρησιμοποιεί το `%` για αυτή την πράξη.

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 41

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.

ELIXIR

```elixir
defmodule Exercise10 do
  def solve(n) do
    n == 0
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

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 44

Η ΛΥΣΗ

11 Είναι θετικός;

Η σκέψη

Επιστρέφουμε την τιμή της σύγκρισης `n > 0`.

ELIXIR

```elixir
defmodule Exercise11 do
  def solve(n) do
    n > 0
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

Πρόσεξε αυτό

Αν γράψεις `>=`, το μηδέν θα δώσει λάθος απάντηση.

Επιστροφή στην άσκηση 11 · σελίδα 47

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.

ELIXIR

```elixir
defmodule Exercise12 do
  def solve(score, threshold) do
    score >= threshold
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

Πρόσεξε αυτό

Μην γράψεις ένα σταθερό όριο από κάποιο παράδειγμα. Χρησιμοποίησε το `threshold`.

Επιστροφή στην άσκηση 12 · σελίδα 50

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.

ELIXIR

```elixir
defmodule Exercise13 do
  def solve(n) do
    rem(n, 2) == 0
  end
end
```

Διάβασε τον κώδικα

Η `rem(a, b)` υπολογίζει το υπόλοιπο. Η Elixir δεν χρησιμοποιεί το `%` για αυτή την πράξη.

Πρόσεξε αυτό

Μην επιστρέψεις το ίδιο το υπόλοιπο. Το αποτέλεσμα πρέπει να είναι λογική τιμή.

Επιστροφή στην άσκηση 13 · σελίδα 53

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.

ELIXIR

```elixir
defmodule Exercise14 do
  def solve(n) do
    if n < 0 do
      -n
    else
      n
    end
  end
end
```

Διάβασε τον κώδικα

Το `if ... do ... else ... end` έχει ως τιμή το αποτέλεσμα του κλάδου που επιλέχθηκε.

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 57

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.

ELIXIR

```elixir
defmodule Exercise15 do
  def solve(a, b) do
    if a > b do
      a
    else
      b
    end
  end
end
```

Διάβασε τον κώδικα

Η τελευταία έκφραση στο σώμα της συνάρτησης είναι η τιμή που επιστρέφεται. Δεν χρειάζεται εντολή `return`.

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 61

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.

ELIXIR

```elixir
defmodule Exercise16 do
  def solve(n) do
    cond do
      n < 0 -> -1
      n == 0 -> 0
      true -> 1
    end
  end
end
```

Διάβασε τον κώδικα

Το `cond` δοκιμάζει συνθήκες με τη σειρά. Σε κάθε γραμμή, το `->` χωρίζει τη συνθήκη από την απάντηση. Το τελικό `true` ταιριάζει όταν έχουν αποτύχει οι προηγούμενες συνθήκες.

Πρόσεξε αυτό

Η απάντηση δεν είναι ο ίδιος ο αριθμός. Για είσοδο 42, ζητάμε 1.

Επιστροφή στην άσκηση 16 · σελίδα 65

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.

ELIXIR

```elixir
defmodule Exercise17 do
  def solve(n, high) do
    (n >= 0) and (n <= high)
  end
end
```

Διάβασε τον κώδικα

Το λογικό «και» γράφεται `and`. Κάθε πλευρά του είναι μια σύγκριση που δίνει λογική τιμή.

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 69

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.

ELIXIR

```elixir
defmodule Exercise18 do
  def solve(n, divisor) do
    rem(n, divisor) == 0
  end
end
```

Διάβασε τον κώδικα

Η `rem(a, b)` υπολογίζει το υπόλοιπο. Η Elixir δεν χρησιμοποιεί το `%` για αυτή την πράξη.

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 72

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.

ELIXIR

```elixir
defmodule Exercise19 do
  def solve(n) do
    if rem(n, 3) == 0 do
      "Fizz"
    else
      ""
    end
  end
end
```

Διάβασε τον κώδικα

Η `rem(a, b)` υπολογίζει το υπόλοιπο. Η Elixir δεν χρησιμοποιεί το `%` για αυτή την πράξη.

Πρόσεξε αυτό

Το `""` δεν έχει κενό χαρακτήρα. Το `" "` έχει έναν και θα αποτύχει στον έλεγχο.

Επιστροφή στην άσκηση 19 · σελίδα 76

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.

ELIXIR

```elixir
defmodule Exercise20 do
  def solve(n) do
    cond do
      (rem(n, 3) == 0) and (rem(n, 5) == 0) -> "FizzBuzz"
      rem(n, 3) == 0 -> "Fizz"
      rem(n, 5) == 0 -> "Buzz"
      true -> ""
    end
  end
end
```

Διάβασε τον κώδικα

Το `cond` που είδες στην άσκηση 16 ελέγχει πρώτα τη συνθήκη με `and`. Το `true -> ""` καλύπτει τους υπόλοιπους αριθμούς.

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 80

ΜΕΤΑ ΤΙΣ ΕΙΚΟΣΙ 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