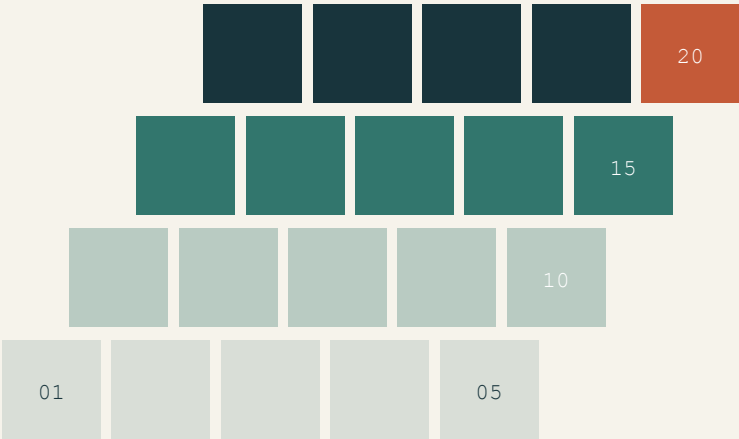


Προγραμματισμός με μικρά βήματα

20 πολύ εύκολες ασκήσεις
Από μια σταθερή απάντηση στο FizzBuzz



Ένας αριθμός. Μια μικρή ιδέα. Λίγες γραμμές.

Δεν χρειάζεται να τα μάθεις όλα μαζί.
Διάβασε, γράψε, δοκίμασε και εξήγησε
με δικά σου λόγια τι έκανε ο κώδικάς σου.



Μια συνάρτηση μπορεί, για παράδειγμα, να τριπλασιάζει.

Προγραμματισμός με μικρά βήματα · Elixir

Πρώτο μέρος · 20 ασκήσεις · Σεπτέμβριος 2026

Το βιβλίο συνοδεύεται από τον φάκελο prog.

Οι διαδρομές και οι εντολές ξεκινούν από αυτόν τον φάκελο.

Οι λύσεις βρίσκονται στο τελευταίο μέρος του βιβλίου.

ΠΕΡΙΕΧΟΜΕΝΑ

Ο χάρτης του βιβλίου

Πριν από την πρώτη άσκηση / 4

Ετοιμάζουμε το Mac / 8

Η Elixir με μια ματιά / 10

Η σειρά των 20 ασκήσεων / 13

Βήμα	Μάθημα και άσκηση	Αρχή	Λύση
01	Μια σταθερή απάντηση	15	84
02	Ο ίδιος αριθμός	19	85
03	Ο επόμενος αριθμός	22	86
04	Δύο αριθμοί μαζί	25	87
05	Το διπλάσιο	28	88
06	Από λεπτά σε δευτερόλεπτα	31	89
07	Η διαφορά δύο αριθμών	35	90
08	Γύρω από το ορθογώνιο	38	91
09	Τι περισσεύει	41	92
10	Είναι μηδέν;	44	93
11	Είναι θετικός;	47	94
12	Έφτασε το όριο;	50	95
13	Ζυγός αριθμός	53	96
14	Απόσταση από το μηδέν	56	97
15	Ο μεγαλύτερος από τους δύο	60	98
16	Τρεις περιπτώσεις	64	99
17	Μέσα στα όρια	68	100
18	Χωρίς υπόλοιπο	72	101
19	Η πρώτη λέξη του FizzBuzz	75	102
20	Οι κανόνες του FizzBuzz	79	103

Πάτησε έναν τίτλο ή έναν αριθμό σελίδας για να μεταφερθείς εκεί.

ΔΙΑΒΑΣΕ ΜΕ ΤΟΝ ΔΙΚΟ ΣΟΥ ΡΥΘΜΟ

Πριν από την πρώτη άσκηση

Ο στόχος είναι να γράφεις λίγες γραμμές και να καταλάβεις γιατί δουλεύουν. Ξεκινάμε από μια σταθερή απάντηση. Προσθέτουμε παραμέτρους, απλές πράξεις, συγκρίσεις και μικρές αποφάσεις. Σε ορισμένες ασκήσεις επαναλαμβάνουμε κάτι που ήδη ξέρεις. Αυτό είναι μέρος της σειράς.

Δεν χρειάζεται να έχεις ξαναγράψει κώδικα. Χρειάζεται να μπορείς να ανοίξεις έναν φάκελο, να αλλάξεις ένα αρχείο κειμένου και να αποθηκεύσεις την αλλαγή. Στο επόμενο κεφάλαιο θα ετοιμάσεις τα εργαλεία.

Η δική σου έκδοση

Αυτή η έκδοση χρησιμοποιεί Elixir. Ακολούθησε την εισαγωγή και τις ασκήσεις με τη σειρά. Οι άλλες γλώσσες έχουν χωριστά βιβλία.

Τι είναι μια συνάρτηση

Σκέψου μια συνάρτηση σαν έναν μικρό κανόνα με όνομα. Της δίνεις μια τιμή, εφαρμόζει τον κανόνα και σου δίνει ένα αποτέλεσμα. Για παράδειγμα, μια συνάρτηση που τριπλασιάζει έναν αριθμό δέχεται το 4 και επιστρέφει το 12.

Στο βιβλίο η συνάρτηση λέγεται `solve`. Το `solve(4)` σημαίνει «κάλεσε τη συνάρτηση δίνοντάς της το 4». Το `solve()` σημαίνει ότι δεν της δίνεις καμία τιμή. Η πρώτη άσκηση είναι τέτοια περίπτωση.

Το **όρισμα** είναι η συγκεκριμένη τιμή που δίνεις σε μια κλήση. Η **παράμετρος** είναι το όνομα με το οποίο τη χρησιμοποιεί η συνάρτηση. Στον ορισμό της συνάρτησης ονομάζουμε την παράμετρο `n`. Στην κλήση `solve(4)` το 4 είναι το όρισμα που παίρνει τη θέση της. Αυτή η διάκριση θα γίνει πιο καθαρή στη δεύτερη άσκηση.

Επιστρέφω και εμφανίζω

Το να επιστρέφεις μια τιμή σημαίνει ότι τη δίνεις στον κώδικα που σε κάλεσε. Το να την εμφανίζεις σημαίνει ότι γράφεις κάτι στην οθόνη. Οι ασκήσεις εδώ ζητούν **επιστροφή τιμής**. Τα tests παραλαμβάνουν την τιμή και την ελέγχουν. Δεν χρειάζεται να διαβάσεις από το πληκτρολόγιο ή να γράφεις `print`, `console.log` ή `printf` στη λύση.

Η Elixir επιστρέφει την τιμή της τελευταίας έκφρασης. Η εισαγωγή της γλώσσας εξηγεί τη σύνταξη.

Τρεις μορφές αποτελέσματος

Στην αρχή συναντάμε ακέραιους, όπως 0, 8 και -3, κείμενο μέσα σε εισαγωγικά, όπως "Hello!" και αργότερα λογικές τιμές: αληθές ή ψευδές. Το "8" είναι κείμενο. Το 8 είναι αριθμός. Δεν είναι ίδια τιμή για τα tests. Το κενό κείμενο γράφεται "" και έχει μηδέν χαρακτήρες. Το " " έχει ένα κενό.

Πώς δουλεύεις κάθε άσκηση

- 01 Διάβασε το μικρό μάθημα πριν από την εκφώνηση. Εξήγησε κάθε γραμμή του παραδείγματος και απάντησε στην ερώτηση κατανόησης. Μετά διάβασε την εκφώνηση.
- 02 Άνοιξε τον υποφάκελο της γλώσσας σου και το αρχείο `exercise`.
- 03 Κράτησε το όνομα της συνάρτησης και τις παραμέτρους της. Αντικατάστησε το προσωρινό σώμα κάτω από το `TODO` με τη λύση σου.
- 04 Αποθήκευσε το αρχείο. Με το `run.sh` δες την απάντησή σου στην οθόνη και με το `test.sh` τρέξε τους έτοιμους ελέγχους.
- 05 Αν αποτύχουν, σύγκρινε το αποτέλεσμα που περίμεναν με αυτό που επέστρεψες.
- 06 Διάβασε την εξήγηση της λύσης όταν έχεις δοκιμάσει μόνος σου.

Το άλυτο αρχείο είναι `exercise.exs`. Το αρχείο της έτοιμης λύσης μένει χωριστά από την προσπάθειά σου.

Δες τη συνάρτησή σου να δουλεύει

Μόλις λύσεις την πρώτη άσκηση, από τον φάκελο `prog` τρέξε.

```
SH
bash scripts/run.sh 01 elixir
```

Θα δεις `Hello!`. Στη δεύτερη άσκηση η εντολή δέχεται και τον αριθμό.

```
SH
bash scripts/run.sh 02 elixir 8
```

Το script καλεί την προσπάθειά σου και εμφανίζει το αποτέλεσμα. Η συνάρτηση συνεχίζει να επιστρέφει μια τιμή. Την εμφάνιση αναλαμβάνει το έτοιμο πρόγραμμα γύρω της. Η εισαγωγή της γλώσσας σου δίνει τη δική της εντολή.

Τι κάνουν τα tests

Ένα test καλεί τη συνάρτηση με μια γνωστή είσοδο και συγκρίνει το αποτέλεσμα με μια αναμενόμενη τιμή. Για έναν τριπλασιασμό, θα μπορούσε να ελέγχει ότι το 4 δίνει 12 και ότι το 0 δίνει 0.

Το `TODO` στα άλυστα αρχεία προκαλεί επίτηδες αποτυχία. Έτσι βλέπεις ότι εκτελείται το σωστό αρχείο. Δεν χρειάζεται να μάθεις τη σύνταξη των tests πριν λύσεις την πρώτη άσκηση. Οι έλεγχοι έχουν ήδη γραφτεί.

Τα tests καλύπτουν παραδείγματα και επιλεγμένα όρια. Δεν είναι απόδειξη για κάθε δυνατή είσοδο. Η εκφώνηση ορίζει τι πρέπει να ισχύει συνολικά. Στις πρώτες 20 ασκήσεις θεωρούμε τις εισόδους έγκυρες, μέσα στα όρια που γράφει κάθε εκφώνηση. Ο έλεγχος λανθασμένων εισόδων θα έρθει αργότερα.

ΤΡΕΙΣ ΜΙΚΡΕΣ ΕΡΩΤΗΣΕΙΣ

Πότε προχωράς

Προχώρα όταν μπορείς να πεις με δικά σου λόγια τι δέχεται η συνάρτηση, τι επιστρέφει και γιατί περνούν τα παραδείγματα. Αν η προηγούμενη άσκηση σε δυσκόλεψε, ξανακάνε την ίδια με άλλους μικρούς αριθμούς. Δεν υπάρχει χρόνος επίλυσης ή στόχος για ασκήσεις ανά ημέρα.

Στην 20ή άσκηση θα αναγνωρίζεις τη λέξη του FizzBuzz για έναν αριθμό. Η πλήρης ακολουθία από το 1 έως το 100 χρειάζεται επανάληψη και παρουσίαση πολλών αποτελεσμάτων. Αυτά μένουν για το επόμενο μέρος.

1

Τι δέχεται;

Μπορώ να περιγράψω την είσοδο.

2

Τι επιστρέφει;

Μπορώ να προβλέψω μια απάντηση.

3

Γιατί δουλεύει;

Μπορώ να εξηγήσω γιατί περνούν τα παραδείγματα.

Ο ΧΩΡΟΣ ΟΠΟΥ ΘΑ ΔΟΥΛΕΥΕΙΣ

Ετοιμάζουμε το Mac

Οι οδηγίες ελέγχθηκαν στις επίσημες πηγές στις 7 Σεπτεμβρίου 2026. Η πραγματική εκτέλεση του υλικού καταγράφεται χωριστά στο `VERIFICATION.md`. Εγκατάστησε μόνο τα εργαλεία της γλώσσας που επέλεξες.

Terminal και φάκελοι

Άνοιξε το Terminal από τις Εφαρμογές → Βοηθήματα ή από το Spotlight. Το Terminal περιμένει εντολές κειμένου. Με το Enter εκτελείς μία εντολή. Στα παραδείγματα αντιγράφεις μόνο το περιεχόμενο του πλαισίου.

Η εντολή `pwd` δείχνει σε ποιον φάκελο βρίσκεσαι. Η `ls` δείχνει τα αρχεία του. Η `cd` αλλάζει φάκελο. Γράψε `cd` με ένα κενό, σύρε τον φάκελο `prog` από το Finder στο Terminal και πάτησε Enter. Αυτό δουλεύει και όταν η διαδρομή έχει κενά ή ο φάκελος βρίσκεται αλλού στον δικό σου υπολογιστή.

```
SH
pwd
ls
```

Θα πρέπει να βλέπεις τα `README.md`, `book`, `exercises` και `scripts`. Στα παραδείγματα που λένε «από τον `prog`», ξεκινάς από αυτόν τον φάκελο. Η εντολή `cd ..` σε πηγαίνει έναν φάκελο προς τα πάνω.

Επεξεργασία αρχείων

Χρησιμοποίησε τον editor κώδικα που έχεις ήδη. Άνοιξε σε αυτόν ολόκληρο τον φάκελο `prog`. Αποθήκευε με `Command-S` πριν τρέξεις tests. Τα αρχεία κώδικα πρέπει να μένουν απλό κείμενο, με το ίδιο όνομα και κατάληξη. Ένα αρχείο `exercise.py.txt` δεν είναι το `exercise.py` που περιμένουν τα tests.

Εγκατάσταση ανά γλώσσα

Η εισαγωγή στη Elixir έχει συγκεκριμένες οδηγίες και μια πρώτη εκτέλεση.

- Elixir

Για εντολές που ξεκινούν με `brew`, χρειάζεται το Homebrew. Αν δεν το έχεις, η επίσημη σελίδα του Homebrew δίνει τον installer και τα βήματα μετά την

εγκατάσταση. Ακολούθησε και τα βήματα για το `PATH` που εμφανίζει για το δικό σου Mac. Το `PATH` είναι η λίστα φακέλων όπου το Terminal ψάχνει για εντολές. Επιβεβαίωσε με `brew --version`. Οι εγκαταστάσεις Python, Node.js και Go μπορούν επίσης να γίνουν από τους επίσημους installers τους, χωρίς Homebrew.

Στις λήψεις, `arm64` ή `aarch64` σημαίνει Apple Silicon. `x64` ή `amd64` σημαίνει Intel. Δες το Apple → Πληροφορίες για αυτό το Mac για τον επεξεργαστή σου.

Αν κάτι δεν τρέχει

Το `command not found` συνήθως σημαίνει ότι λείπει η εγκατάσταση ή ότι το Terminal δεν τη βρίσκει στο `PATH`. Κλείσε και άνοιξε το Terminal μετά την εγκατάσταση. Δοκίμασε την εντολή έκδοσης της γλώσσας σου.

Το `No such file or directory` συχνά σημαίνει ότι βρίσκεσαι σε άλλον φάκελο. Έλεγξε με `pwd` και `ls`. Το `TODO` σημαίνει ότι τρέχει το άλτο αρχείο. Ένα συντακτικό σφάλμα δείχνει συνήθως αρχείο και γραμμή: ξεκίνα από εκεί, έλεγξε παρενθέσεις, εισαγωγικά και εσοχές και ξανατρέξε την ίδια εντολή.

Το κοινό script εκτελεί το άλτο αρχείο όταν παραλείπεις την τελευταία λέξη.

```
SH
```

```
bash scripts/test.sh 01 elixir
bash scripts/test.sh 01 elixir solution
```

Η πρώτη εντολή ελέγχει τη δική σου προσπάθεια. Η δεύτερη την έτοιμη λύση. Δεν αντιγράφει τη λύση πάνω στο αρχείο σου. Για τις γλώσσες με compiler, τα προσωρινά εκτελέσιμα δημιουργούνται σε ξεχωριστό προσωρινό φάκελο.

Η ΓΛΩΣΣΑ ΣΟΥ

Η Elixir με μια ματιά

Η Elixir οργανώνει τον κώδικα σε modules και συναρτήσεις. Εδώ θα δουλεύεις με αρχεία `.exs`, τα οποία εκτελούνται ως scripts με την εντολή `elixir`. Οι ίδιες μικρές πράξεις γράφονται με λίγο διαφορετική σύνταξη από τις υπόλοιπες γλώσσες του βιβλίου.

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 χρειάζεται πρόσθετα πακέτα Hex για το βιβλίο.

Διάβασε μια μικρή συνάρτηση

```
ELIXIR
1  defmodule Example do
2    def value() do
3      7
4    end
5  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 το module λέγεται `Exercise` και η συνάρτηση `solve`, οπότε τα tests καλούν `Exercise.solve()` ή δίνουν τα ορίσματα που ζητά η άσκηση.

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.

Η πρώτη προσπάθεια

Άνοιξε το `exercises/01-hello/elixir/exercise.exs`. Αντικατάστησε το προσωρινό σώμα μετά το `TODO` με τη λύση σου. Περιλαμβάνει τις γραμμές `_ = n` όπου υπάρχουν και τη γραμμή `raise ....` Το `_ = n` αποφεύγει την προειδοποίηση για αχρησιμοποίητη παράμετρο. Το προσωρινό `raise` προκαλεί την αρχική αποτυχία. Κράτησε τα `defmodule`, `def` και τα δύο `end`.

Από τον `prog`.

```
SH
bash scripts/test.sh 01 elixir
```

Δες τη δική σου απάντηση

Αφού ολοκληρώσεις την πρώτη άσκηση, από τον `prog`.

```
SH
bash scripts/run.sh 01 elixir
```

Θα εμφανιστεί το `Hello!` που επιστρέφει η συνάρτησή σου. Στη δεύτερη άσκηση μπορείς να δώσεις και έναν αριθμό.

```
SH
bash scripts/run.sh 02 elixir 8
```

Για να δεις την έτοιμη λύση πριν συμπληρώσεις το αρχείο σου.

```
SH
bash scripts/run.sh 02 elixir --solution 8
```

Το κοινό script χρειάζεται μόνο Bash και τα εργαλεία της γλώσσας σου. Οι λογικές τιμές εμφανίζονται ως `true` ή `false`. Το κενό κείμενο δίνει μια κενή γραμμή. Τα ορίσματα δίνονται με τη σειρά που γράφει η εκφώνηση.

Απευθείας tests

Το `ExUnit` περιλαμβάνεται στην Elixir. Το αρχείο `tests.exs` το ξεκινά και φορτώνει την επιλεγμένη υλοποίηση. Από τον `prog`.

```
SH
cd exercises/01-hello/elixir
elixir tests.exs
```

Για την έτοιμη λύση, από τον ίδιο φάκελο.

```
SH
PROG_VARIANT=solution elixir tests.exs
```

Θα δεις αριθμό tests και failures. Η μεταβλητή `PROG_VARIANT` ισχύει μόνο για την εντολή στην οποία γράφεται. Κάθε εκτέλεση φορτώνει ένα από τα δύο αρχεία, επειδή και τα δύο ορίζουν το ίδιο module `Exercise`.

Για να δεις την τιμή της έτοιμης πρώτης λύσης, από τον ίδιο φάκελο.

```
SH
elixir -r solution.exs -e 'IO.inspect(Exercise.solve())'
```

Η `IO.inspect` εμφανίζει την τιμή έξω από τη συνάρτηση.

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.

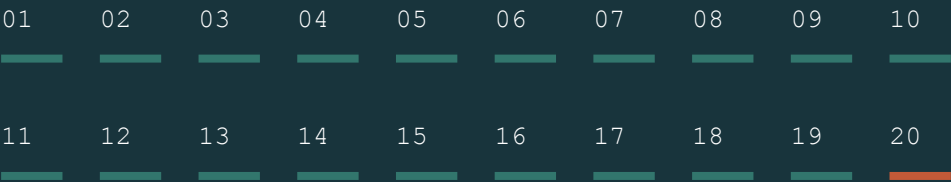
Βήμα	Η νέα ιδέα
01	Επιστροφή μιας σταθερής τιμής
02	Μία παράμετρος
03	Πρόσθεση με +
04	Δύο παράμετροι
05	Πολλαπλασιασμός με *
06	Δίνουμε ονόματα σε τιμές και αποτελέσματα
07	Αφαίρεση με -
08	Παρενθέσεις σε μια έκφραση
09	Υπόλοιπο διαίρεσης
10	Σύγκριση ισότητας και λογική τιμή
11	Σύγκριση με >
12	Σύγκριση 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.

Οι ασκήσεις

Πρώτα ένα μικρό μάθημα σύνταξης.

Μετά η δική σου προσπάθεια, ένα βήμα κάθε φορά.



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

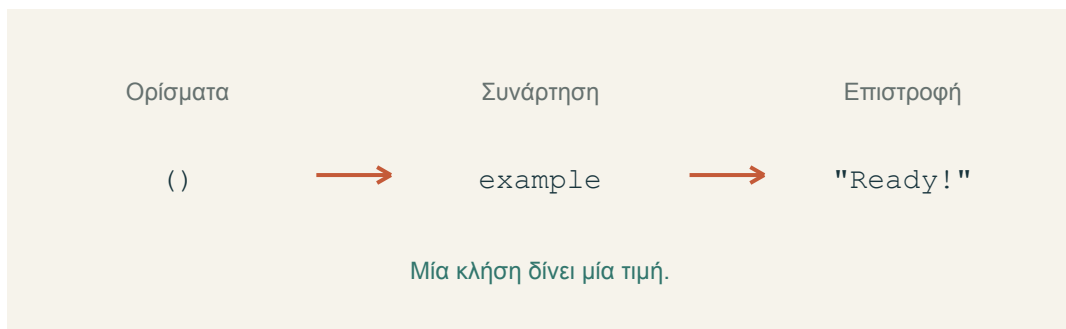
```
1  defmodule Example do
2    def example() do
3      "Ready!"
4    end
5  end
```

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



Ακολουθήσε 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;

Έλεγε 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 `run.sh` καλεί τη συνάρτηση της άσκησης και εμφανίζει την τιμή της.

Το αρχείο της προσπάθειάς σου

Στην άσκηση το module λέγεται `Exercise` 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 17.

Η ΑΣΚΗΣΗ

01 Μια σταθερή απάντηση

Επιστροφή μιας σταθερής τιμής.

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.



```
solve() —————→ "Hello!"
```

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 Δοκίμασέ το

Η συνάρτηση καλείται χωρίς ορίσματα.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve()</code>	<code>"Hello!"</code>

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/01-hello/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 01 elixir
```

```
bash scripts/run.sh 01 elixir
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 84.

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

```
1 defmodule Example do
2   def example(value) do
3     value
4   end
5 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.

Έλεγε 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 20.

Η ΑΣΚΗΣΗ

02 Ο ίδιος αριθμός

Μία παράμετρος.

Τι θα φτιάξεις

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

Όρια: $-1000 \leq n \leq 1000$.



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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά `n`.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(8)</code>	8
<code>solve(-4)</code>	-4
<code>solve(0)</code>	0
<code>solve(-1000)</code>	-1000
<code>solve(1000)</code>	1000

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/02-echo/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 02 elixir
bash scripts/run.sh 02 elixir 8
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

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

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 85.

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

```
1  defmodule Example do
2    def example(value) do
3      value + 3
4    end
5  end
```

Διάβασε τις γραμμές

- 01 Το `+` ανάμεσα σε δύο ακέραιους 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 23.

Η ΑΣΚΗΣΗ

03 Ο επόμενος αριθμός

Πρόσθεση με +.

Τι θα φτιάξεις

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

Όρια: $-1000 \leq n \leq 1000$.



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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(5)</code>	6
<code>solve(0)</code>	1
<code>solve(-1)</code>	0
<code>solve(-1000)</code>	-999
<code>solve(1000)</code>	1001

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/03-next-number/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 03 elixir
bash scripts/run.sh 03 elixir 5
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 86.

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

```
1 defmodule Example do
2   def example(apples, pears) do
3     apples + pears
4   end
5 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 26.

Η ΑΣΚΗΣΗ

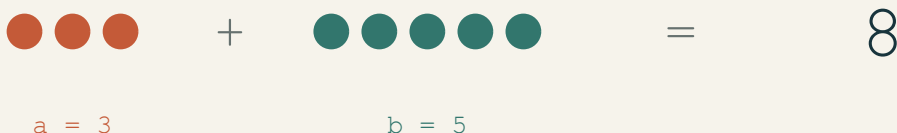
04 Δύο αριθμοί μαζί

Δύο παράμετροι.

Τι θα φτιάξεις

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

Όρια: $-1000 \leq a, b \leq 1000$.


$$\begin{array}{ccccccc} \bullet & \bullet & \bullet & + & \bullet & \bullet & \bullet & \bullet & \bullet & = & 8 \\ a = 3 & & & & b = 5 & & & & & & \end{array}$$

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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3, 5)</code>	8
<code>solve(0, 0)</code>	0
<code>solve(-3, 5)</code>	2
<code>solve(-4, -6)</code>	-10
<code>solve(1000, 1000)</code>	2000

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/04-add/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 04 elixir
bash scripts/run.sh 04 elixir 3 5
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

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

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 87.

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

```
1 defmodule Example do
2   def example(value) do
3     value * 3
4   end
5 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 $\times$ 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 29.

Η ΑΣΚΗΣΗ

05 Το διπλάσιο

Πολλαπλασιασμός με $*$.

Τι θα φτιάξεις

Δέξου έναν ακέραιο n και επέστρεψε το διπλάσιό του.

Όρια: $-1000 \leq n \leq 1000$.



8

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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(4)</code>	8
<code>solve(1)</code>	2
<code>solve(0)</code>	0
<code>solve(-3)</code>	-6
<code>solve(1000)</code>	2000

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/05-double/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 05 elixir
bash scripts/run.sh 05 elixir 4
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 88.

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

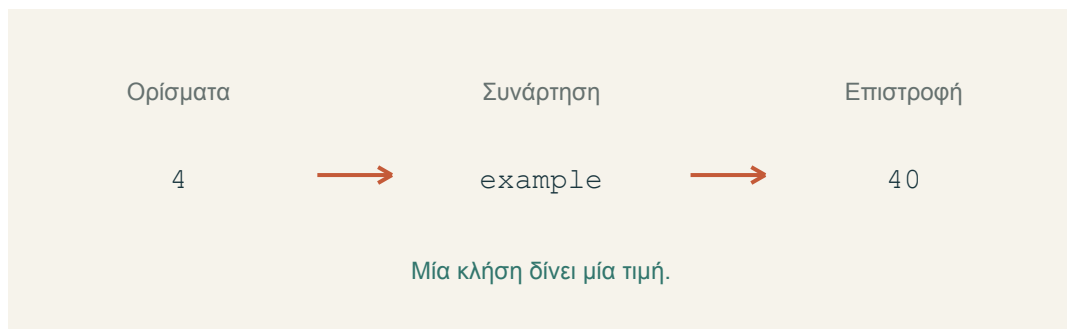
```
1  defmodule Example do
2    def example(boxes) do
3      items_per_box = 10
4      total = boxes * items_per_box
5      total
6    end
7  end
```

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



Ακολουθήσε 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 33.

Η ΑΣΚΗΣΗ

06 Από λεπτά σε δευτερόλεπτα

Δίνουμε ονόματα σε τιμές και αποτελέσματα.

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: $0 \leq \text{minutes} \leq 1000$. Δουλεύουμε μόνο με ολόκληρα λεπτά.



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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά `minutes`.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3)</code>	180
<code>solve(1)</code>	60
<code>solve(0)</code>	0
<code>solve(60)</code>	3600
<code>solve(1000)</code>	60000

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/06-minutes-to-seconds/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 06 elixir
bash scripts/run.sh 06 elixir 3
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

Για 3 λεπτά έχεις τρεις ομάδες των 60 δευτερολέπτων.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 89.

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

```
1 defmodule Example do
2   def example(price, coupon) do
3     price - coupon
4   end
5 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 36.

Η ΑΣΚΗΣΗ

07 Η διαφορά δύο αριθμών

Αφαίρεση με '-'.

Τι θα φτιάξεις

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

Όρια: $-1000 \leq a, b \leq 1000$.



$$8 - 3 = 5$$

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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά a, b .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(8, 3)</code>	5
<code>solve(3, 8)</code>	-5
<code>solve(0, 0)</code>	0
<code>solve(-4, -6)</code>	2
<code>solve(1000, -1000)</code>	2000

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/07-difference/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 07 elixir
bash scripts/run.sh 07 elixir 8 3
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 90.

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

```
1 defmodule Example do
2   def example(first, second) do
3     (first + second) * 3
4   end
5 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 39.

Η ΑΣΚΗΣΗ

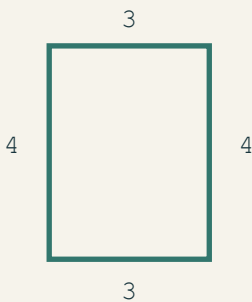
08 Γύρω από το ορθογώνιο

Παρενθέσεις σε μια έκφραση.

Τι θα φτιάξεις

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

Όρια: $0 \leq \text{width}, \text{height} \leq 1000$. Εφαρμόζουμε τον ίδιο τύπο και για μηδενικές διαστάσεις.



$$2 \times (3 + 4) = 14$$

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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3, 4)</code>	14
<code>solve(1, 1)</code>	4
<code>solve(0, 5)</code>	10
<code>solve(5, 0)</code>	10
<code>solve(0, 0)</code>	0

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/08-rectangle-perimeter/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 08 elixir
bash scripts/run.sh 08 elixir 3 4
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 91.

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

```
1 defmodule Example do
2   def example(tokens, boxes) do
3     rem(tokens, boxes)
4   end
5 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 42.

Η ΑΣΚΗΣΗ

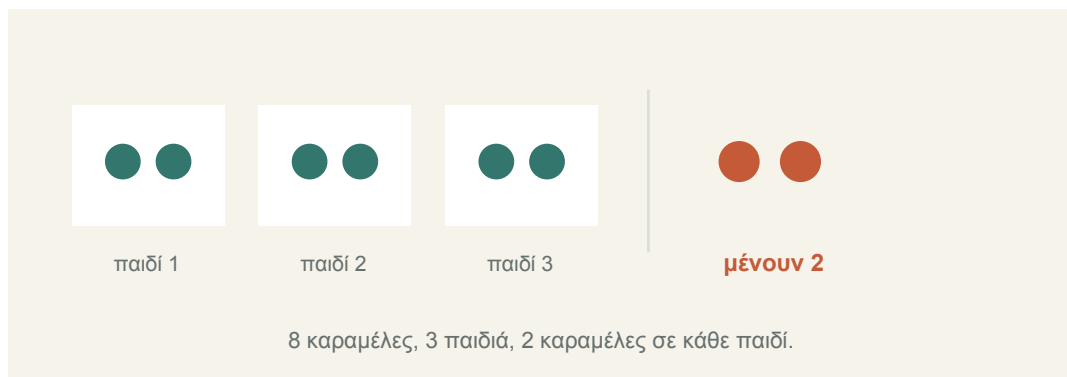
09 Τι περισσεύει

Υπόλοιπο διαίρεσης.

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: $0 \leq \text{candies} \leq 1000$ και $1 \leq \text{children} \leq 1000$. Υπάρχει πάντα τουλάχιστον ένα παιδί.



Η ιδέα

Ας ξεκινήσουμε από τη μοιρασιά. Με 8 καραμέλες και 4 παιδιά, η διαίρεση $8 \div 4$ δίνει 2 καραμέλες σε κάθε παιδί. Με 3 παιδιά δίνεις πάλι 2 στο καθένα, άρα μοιράζεις $3 \times 2 = 6$ 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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(8, 3)</code>	2
<code>solve(12, 4)</code>	0
<code>solve(2, 5)</code>	2
<code>solve(0, 3)</code>	0
<code>solve(7, 1)</code>	0

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/09-leftover/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 09 elixir
bash scripts/run.sh 09 elixir 8 3
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 92.

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

```
1 defmodule Example do
2   def example(value) do
3     value == 5
4   end
5 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 45.

Η ΑΣΚΗΣΗ

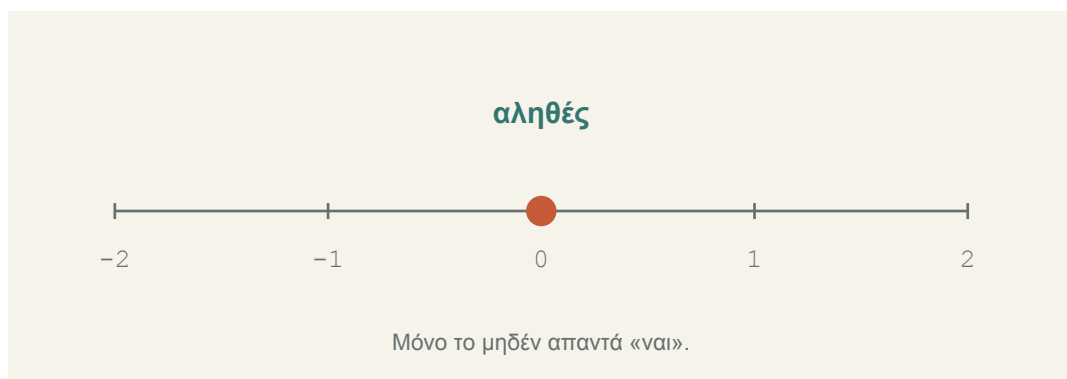
10 Είναι μηδέν;

Σύγκριση ισότητας και λογική τιμή.

Τι θα φτιάξεις

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

Όρια: $-1000 \leq n \leq 1000$.



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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(0)</code>	<code>true</code>
<code>solve(1)</code>	<code>false</code>
<code>solve(-1)</code>	<code>false</code>
<code>solve(1000)</code>	<code>false</code>
<code>solve(-1000)</code>	<code>false</code>

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/10-is-zero/elixir/`
 Αρχείο `exercise.exs` · Tests `tests.exs`

```
SH
bash scripts/test.sh 10 elixir
bash scripts/run.sh 10 elixir 0
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος	αποτέλεσμα
---------	------------

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 93.

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

```
1 defmodule Example do
2   def example(value) do
3     value > 10
4   end
5 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.

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

Συνέχισε στην εκφώνηση της άσκησης 11, στη σελίδα 48.

Η ΑΣΚΗΣΗ

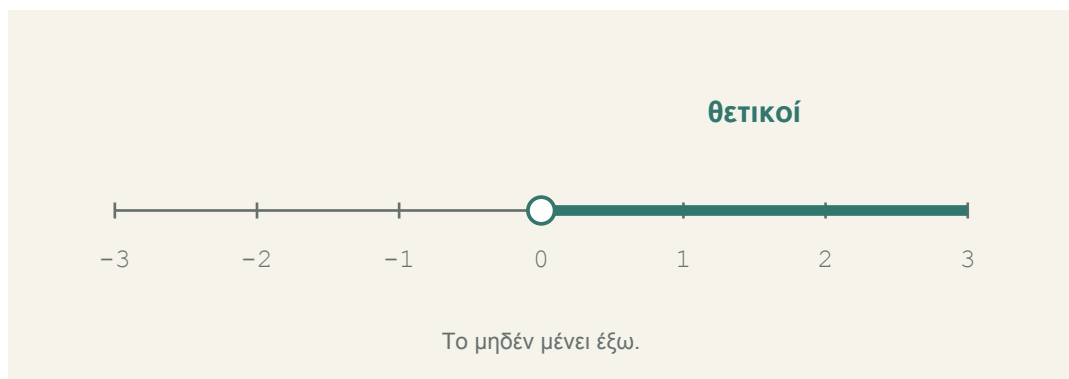
11 Είναι θετικός;

Σύγκριση με $>$.

Τι θα φτιάξεις

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

Όρια: $-1000 \leq n \leq 1000$.



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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3)</code>	<code>true</code>
<code>solve(1)</code>	<code>true</code>
<code>solve(0)</code>	<code>false</code>
<code>solve(-1)</code>	<code>false</code>
<code>solve(1000)</code>	<code>true</code>

Στον υπολογιστή σου

Από τον φάκελο `prolog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/11-is-positive/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 11 elixir
bash scripts/run.sh 11 elixir 3
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 94.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

12 Όταν το όριο συμπεριλαμβάνεται

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

ELIXIR

```
1 defmodule Example do
2   def example(available, needed) do
3     available >= needed
4   end
5 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 51.

Η ΑΣΚΗΣΗ

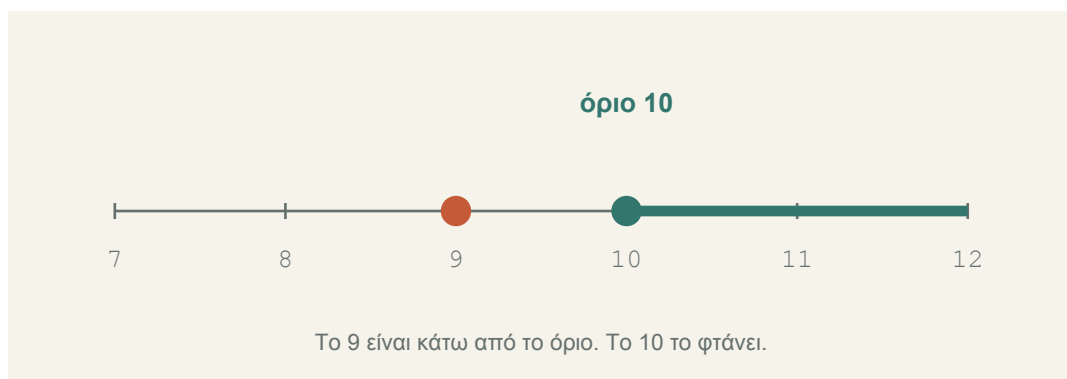
12 Έφτασε το όριο;

Σύγκριση που περιλαμβάνει την ισότητα.

Τι θα φτιάξεις

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

Όρια: $0 \leq \text{score}, \text{threshold} \leq 1000$.



Η ιδέα

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

ΕΦΤΑΣΕ ΤΟ ΟΡΙΟ;

12 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(9, 10)</code>	<code>false</code>
<code>solve(10, 10)</code>	<code>true</code>
<code>solve(11, 10)</code>	<code>true</code>
<code>solve(0, 0)</code>	<code>true</code>
<code>solve(0, 1)</code>	<code>false</code>

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/12-at-least/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

```
SH
```

```
bash scripts/test.sh 12 elixir
```

```
bash scripts/run.sh 12 elixir 9 10
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 95.

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 περισεύει μία μονάδα μετά τον σχηματισμό ζευγαριών.

ELIXIR

```
1 defmodule Example do
2   def example(value) do
3     rem(value, 2) == 1
4   end
5 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 ενδιαμέσο 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 54.

Η ΑΣΚΗΣΗ

13 Ζυγός αριθμός

Εμπέδωση: υπόλοιπο και ισότητα.

Τι θα φτιάξεις

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

Όρια: $0 \leq n \leq 1000$. Το 0 θεωρείται ζυγός αριθμός.

The diagram illustrates the concept of even and odd numbers using pairs of dots. For the number 4, there are two pairs of dots, each pair enclosed in a white box, with the label 'ζυγός' (even) to the right. For the number 5, there are two pairs of dots in white boxes and one single red dot, with the label 'περιττός' (odd) to the right. Below the diagram, the text reads: 'Χωρίζουμε σε ζευγάρια και κοιτάμε αν περισσεύει ένα.' (We divide into pairs and see if one is left over.)

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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(4)</code>	<code>true</code>
<code>solve(3)</code>	<code>false</code>
<code>solve(0)</code>	<code>true</code>
<code>solve(1)</code>	<code>false</code>
<code>solve(2)</code>	<code>true</code>

Στον υπολογιστή σου

Από τον φάκελο `prolog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/13-is-even/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 13 elixir
```

```
bash scripts/run.sh 13 elixir 4
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 96.

ΜΙΚΡΟ ΜΑΘΗΜΑ ΣΥΝΤΑΞΗΣ

14 Δύο δρόμοι με if και else

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

ELIXIR

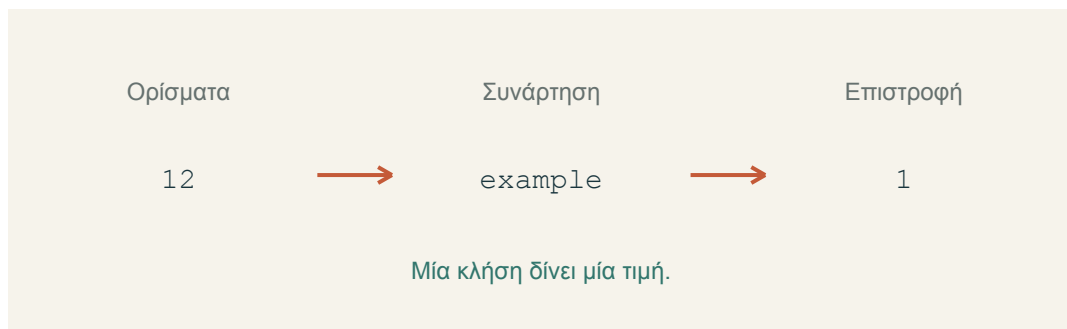
```
1  defmodule Example do
2    def example(value) do
3      if value > 10 do
4        1
5      else
6        0
7      end
8    end
9  end
```

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



Ακολουθήσε 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 58.

Η ΑΣΚΗΣΗ

14 Απόσταση από το μηδέν

Πρώτη επιλογή με `if / else`.

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: $-1000 \leq n \leq 1000$.



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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά `n`.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(-4)</code>	4
<code>solve(4)</code>	4
<code>solve(0)</code>	0
<code>solve(-1)</code>	1
<code>solve(1)</code>	1

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/14-absolute/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 14 elixir
bash scripts/run.sh 14 elixir -4
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 97.

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

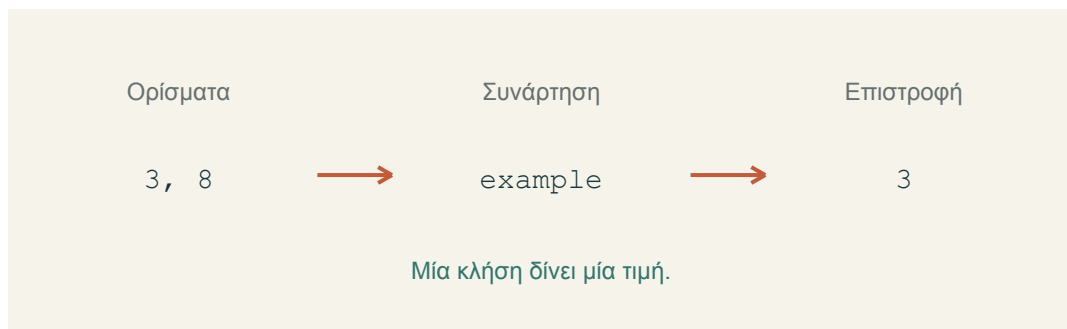
```
1  defmodule Example do
2    def example(a, b) do
3      if a > b do
4        b
5      else
6        a
7      end
8    end
9  end
```

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



Ακολουθήσε 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 62.

Η ΑΣΚΗΣΗ

15 Ο μεγαλύτερος από τους δύο

Εμπέδωση: επιλέγουμε ανάμεσα σε δύο παραμέτρους.

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: $-1000 \leq a, b \leq 1000$.



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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(7, 2)</code>	7
<code>solve(2, 7)</code>	7
<code>solve(5, 5)</code>	5
<code>solve(-7, -2)</code>	-2
<code>solve(0, -1)</code>	0

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/15-larger/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 15 elixir
bash scripts/run.sh 15 elixir 7 2
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

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

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 98.

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

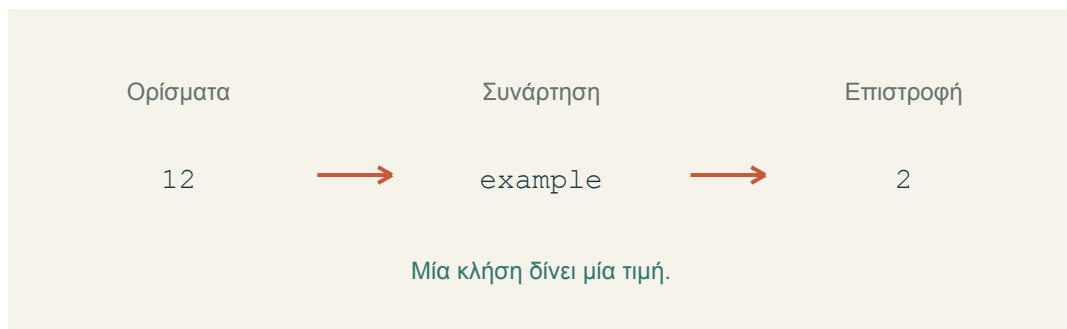
```
1  defmodule Example do
2    def example(value) do
3      cond do
4        value > 10 -> 2
5        value == 10 -> 1
6        true -> 0
7      end
8    end
9  end
```

Διάβασε τις γραμμές

- 01 Το `cond do` ελέγχει τις γραμμές με τη σειρά. Σε κάθε γραμμή το `->` χωρίζει τη συνθήκη από το αποτέλεσμα 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



Ακολουθήσε 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 66.

Η ΑΣΚΗΣΗ

16 Τρεις περιπτώσεις

Περισσότερα από δύο ενδεχόμενα.

Τι θα φτιάξεις

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

Όρια: $-1000 \leq n \leq 1000$.

αρνητικός

-1

μηδέν

0

θετικός

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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά `n`.

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(-7)</code>	-1
<code>solve(0)</code>	0
<code>solve(42)</code>	1
<code>solve(-1)</code>	-1
<code>solve(1)</code>	1

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/16-sign/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 16 elixir
bash scripts/run.sh 16 elixir -7
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

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

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 99.

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

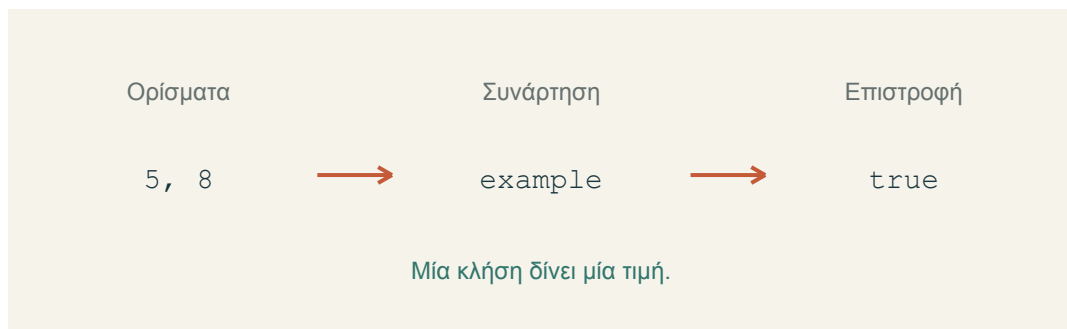
```
1  defmodule Example do
2    def example(value, limit) do
3      value >= 2 and value <= limit
4    end
5  end
```

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



Ακολουθήσε 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.

Έλεγε 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 70.

Η ΑΣΚΗΣΗ

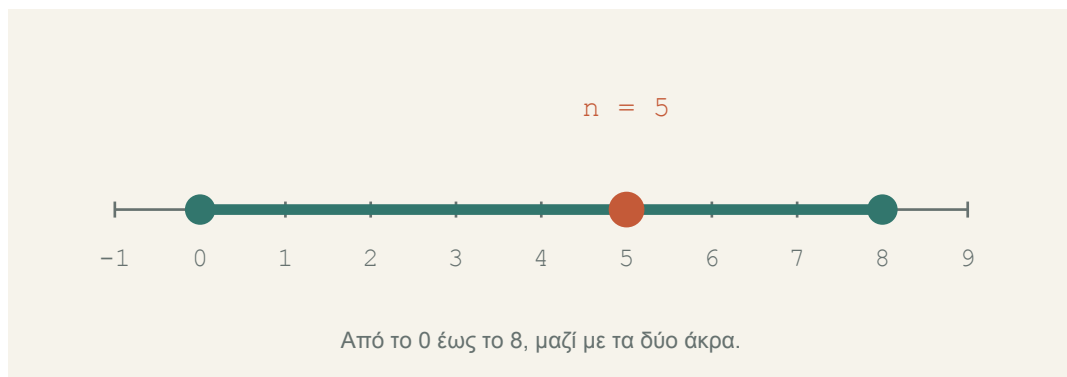
17 Μέσα στα όρια

Συνδυάζουμε δύο ελέγχους με και.

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: $-1000 \leq n \leq 1000$ και $0 \leq \text{high} \leq 1000$.



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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(5, 8)</code>	<code>true</code>
<code>solve(0, 8)</code>	<code>true</code>
<code>solve(8, 8)</code>	<code>true</code>
<code>solve(-1, 8)</code>	<code>false</code>
<code>solve(9, 8)</code>	<code>false</code>

Στον υπολογιστή σου

Από τον φάκελο `prolog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/17-in-range/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 17 elixir
```

```
bash scripts/run.sh 17 elixir 5 8
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

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

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 100.

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 προσθ εται νέα εντολή.

ELIXIR

```
1 defmodule Example do
2   def example(total, size) do
3     rem(total, size) == 0
4   end
5 end
```

Διάβασε τις γραμμές

- 01 Στο `rem(total, size)`, το `size` παίρνει τον ρ λο του διαιρέτη. Χρησιμοποι με την τιμή της παραμέτρου,  χι 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 χ ρι πριν διαβάσεις παρακάτω.

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

Συνέχισε στην εκφ νηση της άσκησης 18, στη σελίδα 73.

Η ΑΣΚΗΣΗ

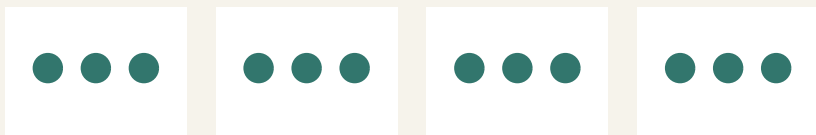
18 Χωρίς υπόλοιπο

Εμπέδωση: ο διαιρέτης γίνεται παράμετρος.

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: $0 \leq n \leq 1000$ και $1 \leq \text{divisor} \leq 1000$. 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 Δοκίμασέ το

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

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(12, 3)</code>	<code>true</code>
<code>solve(13, 3)</code>	<code>false</code>
<code>solve(0, 7)</code>	<code>true</code>
<code>solve(7, 1)</code>	<code>true</code>
<code>solve(2, 5)</code>	<code>false</code>

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/18-is-multiple/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 18 elixir
```

```
bash scripts/run.sh 18 elixir 12 3
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 101.

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

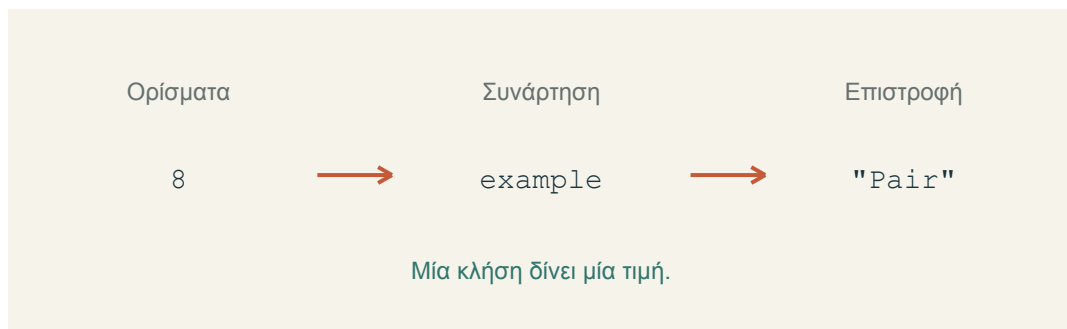
```
1  defmodule Example do
2    def example(value) do
3      if rem(value, 2) == 0 do
4        "Pair"
5      else
6        ""
7      end
8    end
9  end
```

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



Ακολουθήσε 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 77.

Η ΑΣΚΗΣΗ

19 Η πρώτη λέξη του FizzBuzz

Εμπέδωση: μια συνθήκη επιλέγει κείμενο.

Τι θα φτιάξεις

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

Όρια: $1 \leq n \leq 1000$. 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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(3)</code>	<code>"Fizz"</code>
<code>solve(1)</code>	<code>""</code>
<code>solve(5)</code>	<code>""</code>
<code>solve(6)</code>	<code>"Fizz"</code>
<code>solve(15)</code>	<code>"Fizz"</code>

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/19-fizz/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 19 elixir
bash scripts/run.sh 19 elixir 3
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

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

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 102.

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

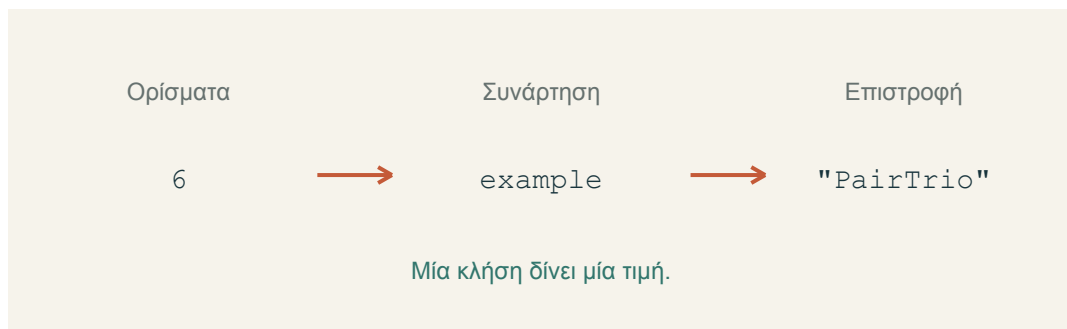
```
1  defmodule Example do
2    def example(value) do
3      cond do
4        rem(value, 2) == 0 and rem(value, 3) == 0 -> "PairTrio"
5        rem(value, 2) == 0 -> "Pair"
6        rem(value, 3) == 0 -> "Trio"
7        true -> ""
8      end
9    end
10  end
```

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



Ακολουθήσε 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.

Έλεγε 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 81.

Η ΑΣΚΗΣΗ

20 Οι κανόνες του FizzBuzz

Σειρά ελέγχων όταν οι περιπτώσεις συμπίπτουν.

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: $1 \leq n \leq 1000$. 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 Δοκίμασέ το

Τα ορίσματα γράφονται με τη σειρά n .

Κλήση	Αναμενόμενο αποτέλεσμα
<code>solve(15)</code>	<code>"FizzBuzz"</code>
<code>solve(9)</code>	<code>"Fizz"</code>
<code>solve(10)</code>	<code>"Buzz"</code>
<code>solve(7)</code>	<code>""</code>
<code>solve(1)</code>	<code>""</code>

Στον υπολογιστή σου

Από τον φάκελο `prog`, άνοιξε το αρχείο της προσπάθειάς σου.

`exercises/20-fizzbuzz-label/elixir/`

Αρχείο `exercise.exs` · Tests `tests.exs`

SH

```
bash scripts/test.sh 20 elixir
bash scripts/run.sh 20 elixir 15
```

Το πρώτο script ελέγχει τη συνάρτησή σου. Το δεύτερο εμφανίζει την τιμή που επιστρέφει. Για την έτοιμη λύση, πρόσθεσε `--solution` μετά τη γλώσσα στο `run.sh`.

Μικρή βοήθεια

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

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

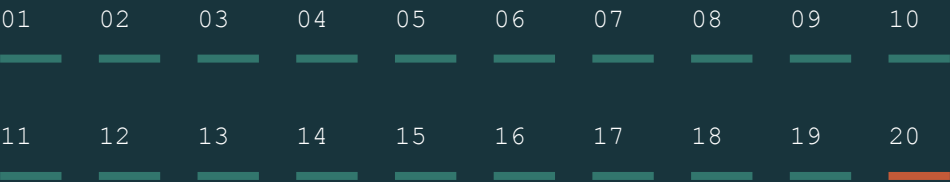
είσοδος

αποτέλεσμα

Όταν ολοκληρώσεις την προσπάθειά σου, δες τη λύση στη σελίδα 103.

Οι λύσεις

Σύγκρινε τη σκέψη σου με μια πιθανή λύση.
Βρες γιατί δουλεύει. Μετά ξαναδοκίμασε μόνος σου.



Η ΛΥΣΗ

01 Μια σταθερή απάντηση

Η σκέψη

Δεν υπάρχει υπολογισμός. Η συνάρτηση επιστρέφει το συγκεκριμένο κείμενο κάθε φορά που καλείται.

ELIXIR

```
1 defmodule Exercise do
2   def solve() do
3     "Hello!"
4   end
5 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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 01 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(n) do
3     n
4   end
5 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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 02 elixir solution
```

Η ΛΥΣΗ

03 Ο επόμενος αριθμός

Η σκέψη

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

ELIXIR

```
1 defmodule Exercise do
2   def solve(n) do
3     n + 1
4   end
5 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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 03 elixir solution
```

Η ΛΥΣΗ

04 Δύο αριθμοί μαζί

Η σκέψη

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

ELIXIR

```
1 defmodule Exercise do
2   def solve(a, b) do
3     a + b
4   end
5 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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 04 elixir solution
```

Η ΛΥΣΗ

05 Το διπλάσιο

Η σκέψη

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

ELIXIR

```
1 defmodule Exercise do
2   def solve(n) do
3     n * 2
4   end
5 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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 05 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(minutes) do
3     seconds_per_minute = 60
4     seconds = minutes * seconds_per_minute
5     seconds
6   end
7 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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 06 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(a, b) do
3     a - b
4   end
5 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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 07 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(width, height) do
3     2 * (width + height)
4   end
5 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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 08 elixir solution
```

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

```
1  defmodule Exercise do
2    def solve(candies, children) do
3      rem(candies, children)
4    end
5  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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 09 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(n) do
3     n == 0
4   end
5 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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 10 elixir solution
```

Η ΛΥΣΗ

11 Είναι θετικός;

Η σκέψη

Επιστρέφουμε την τιμή της σύγκρισης $n > 0$.

ELIXIR

```
1 defmodule Exercise do
2   def solve(n) do
3     n > 0
4   end
5 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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 11 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(score, threshold) do
3     score >= threshold
4   end
5 end
```

Διάβασε τον κώδικα

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

Πρόσεξε αυτό

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

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 12 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(n) do
3     rem(n, 2) == 0
4   end
5 end
```

Διάβασε τον κώδικα

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

Πρόσεξε αυτό

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

Έλεγε τη λύση

SH

```
bash scripts/test.sh 13 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(n) do
3     if n < 0 do
4       -n
5     else
6       n
7     end
8   end
9 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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 14 elixir solution
```

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

```
1  defmodule Exercise do
2    def solve(a, b) do
3      if a > b do
4        a
5      else
6        b
7      end
8    end
9  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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 15 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(n) do
3     cond do
4       n < 0 -> -1
5       n == 0 -> 0
6       true -> 1
7     end
8   end
9 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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 16 elixir solution
```

Η ΛΥΣΗ

17 Μέσα στα όρια

Η σκέψη

Συνδυάζουμε $n \geq 0$ με $n \leq \text{high}$. Το αποτέλεσμα είναι αληθές μόνο όταν και οι δύο συγκρίσεις είναι αληθείς.

ELIXIR

```
1 defmodule Exercise do
2   def solve(n, high) do
3     (n >= 0) and (n <= high)
4   end
5 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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 17 elixir solution
```

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

```
1 defmodule Exercise do
2   def solve(n, divisor) do
3     rem(n, divisor) == 0
4   end
5 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.

Έλεγχε τη λύση

SH

```
bash scripts/test.sh 18 elixir solution
```

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

```
1  defmodule Exercise do
2    def solve(n) do
3      if rem(n, 3) == 0 do
4        "Fizz"
5      else
6        ""
7      end
8    end
9  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.

Έλεγε τη λύση

SH

```
bash scripts/test.sh 19 elixir solution
```

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

```
1  defmodule Exercise do
2    def solve(n) do
3      cond do
4        (rem(n, 3) == 0) and (rem(n, 5) == 0) -> "FizzBuzz"
5        rem(n, 3) == 0 -> "Fizz"
6        rem(n, 5) == 0 -> "Buzz"
7        true -> ""
8      end
9    end
10  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.

Έλεγξε τη λύση

SH

```
bash scripts/test.sh 20 elixir solution
```

ΜΕΤΑ ΤΙΣ ΕΙΚΟΣΙ ΑΣΚΗΣΕΙΣ

Το επόμενο μικρό βήμα

Μπορείς πλέον να δεχτείς τιμές, να κάνεις απλές πράξεις, να συγκρίνεις και να επιλέξεις μια απάντηση. Στην τελευταία άσκηση συνδύασες αυτές τις ιδέες στους κανόνες του FizzBuzz.

Πριν προχωρήσεις, διάλεξε μια άσκηση που σε δυσκόλεψε. Γράψε ξανά τη λύση της χωρίς να τη βλέπεις. Δώσε μια καινούργια είσοδο και εξήγησε γιατί περιμένεις τη συγκεκριμένη απάντηση.

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.

Όταν νιώσεις έτοιμος

Το επόμενο μέρος μπορεί να ξεκινήσει από την επανάληψη μιας απλής εντολής, το μέτρημα από το 1 έως το n και την πλήρη ακολουθία FizzBuzz. Θα κρατήσουμε τον ίδιο ρυθμό, ένα μικρό βήμα κάθε φορά.

ΜΙΑ ΔΙΚΗ ΣΟΥ ΔΟΚΙΜΗ

είσοδος

αποτέλεσμα
