

Clojure

Iteration 1LatestSubmitted via Editor, a few seconds ago

src/hello_world.clj

```
1 (ns hello-world)
2
3 (defn hello []
4   "Hello, World!")
5
```

AnalysisTests

Bobbi Towers gave this feedback on a solution very similar to yours:

You've entered the most common solution, which is to simply return the string "Hello, World!". Remember, every function implicitly returns the result of the last expression in the function body.

on1LatestSubmitted via Editor, a few seconds ago

her.clj

```
bird-watcher)

Fine the vector for last week's counts
last-week [0 2 5 3 7 8 4])

function to return today's count of birds
today [birds]
st birds))

function to increment today's count
inc-bird [birds]
nj (vec (butlast birds)) (inc (last birds))))


edicate function to check if there was a day without birds
day-without-birds? [birds]
lean (some zero? birds)))

function to count the number of birds in the first n days
n-days-count [birds n]
duce + (take n birds)))

function to count the number of busy days (5 or more birds)
busy-days [birds]
unt (filter #(>= % 5) birds)))

function to determine if the week had an odd pattern (1 0 1 0 ...)
odd-week? [birds]
birds (take (count birds) (cycle [1 0])))
```

AnalysisTests


No auto suggestions? Try human mentoring.
Get real 1-to-1 human mentoring on the Bird
Watcher exercise and start writing better
Clojure.
[Get mentoring](#)

Iteration 1Latest

Submitted via Editor, a few seconds ago


src/lucians_luscious_lasagna.clj

```
1 ;; 🍷 Hi there! Welcome to the Clojure Track.
2 ;; The online test-runner is powered by babashka and the Small Clojure Interpreter (SCI).
3 ;; Almost all language features are supported, with the exception of low-level constructs
4 ;; like `deftype`, and certain Java classes. For more info, see:
5 ;; https://github.com/babashka/babashka#differences-with-clojure
6
7 (ns lucians-luscious-lasagna)
8
9 ;; Define the expected oven time for the lasagna
10 (def expected-time 40)
11
12 ;; Function to calculate remaining oven time for the lasagna
13 (defn remaining-time
14   "Takes the actual time in minutes the lasagna has been in the oven,
15   and returns how many minutes the lasagna still has to remain in the oven."
16   [actual-time]
17     (- expected-time actual-time))
18
19 ;; Function to calculate preparation time based on the number of layers
20 (defn prep-time
21   "Takes the number of layers added to the lasagna,
22   and returns how many minutes you spent preparing the lasagna."
23   [num-layers]
24     (* 2 num-layers))
25
26 ;; Function to calculate total cooking and preparation time
27 (defn total-time
28   "Takes the number of layers of lasagna and the actual time in minutes it has been in the oven.
29   Returns how many minutes in total you've worked on cooking the lasagna."
30   [num-layers actual-time]
31     (+ (prep-time num-layers) actual-time))
```

AnalysisTests

Bobbi Towers gave this feedback on a solution very similar to yours:
Excellent! Your solution follows the most common approach.

Tracks / Clojure / Exercises / Cars, Assemble!



Cars, Assemble!

In-progressLearning Exercise

OverviewYour iterations1Community SolutionsCode Review

Continue in editor


Iteration 1Latest

Submitted via Editor, a few seconds ago

src/cars_assemble.clj

```
1 (ns cars-assemble)
2 (def cars-per-hour 221)
3 (defn success-rate [speed]
4   (condp <= speed
5     10 0.77
6     9  0.8
7     5  0.9
8     1  1.0
9     0.0))
10 (defn production-rate
11   "Returns the assembly line's production rate per hour,
12   taking into account its success rate"
13   [speed]
14     (* speed cars-per-hour (success-rate speed)))
15 (defn working-items
16   "Calculates how many working cars are produced per minute"
17   [speed]
18     (int (/ (production-rate speed) 60)))
19
```

AnalysisTests



No auto suggestions? Try human mentoring.

Get real 1-to-1 human mentoring on the Cars, Assemble! exercise and start writing better Clojure.

Get mentoring

Overview

Your Iterations 1

Community Solutions

Code Review

Continue in editor

Iteration 1

Latest

Submitted via Editor, a few seconds ago

src/interest_is_interesting.clj

```


1 (ns interest-is-interesting)
2 (defn abs
3   [n]
4   (if (neg? n) (- n) n)
5 )
6 (defn interest-rate
7   [balance]
8   (cond
9     (neg? balance) -3.213
10    (< balance 1000.0M) 0.5
11    (< balance 5000.0M) 1.621
12    :else 2.475))
13 (defn annual-balance-update
14   [balance]
15   (+
16    (* (bigdec (interest-rate balance)) 0.01M (abs balance))
17     balance))
18 (defn amount-to-donate
19   [balance tax-free-percentage]
20   (if (pos? balance) (int (* tax-free-percentage 0.01 balance 2)) 0))
21

```

Analysis

Tests

...



No auto suggestions? Try human mentoring.

Get real 1-to-1 human mentoring on the Interest is Interesting exercise and start writing better Clojure.

Get mentoring


Haskell,

Tracks /

Haskell /

Exercises /

Hello World



Hello World

In-progress

Tutorial Exercise

Overview

Your Iterations 1

Continue in editor

Iteration 1

Latest

Submitted via Editor, a few seconds ago

src/HelloWorld.hs

package.yaml

...

```


1 module HelloWorld (hello) where
2
3 hello :: String
4 hello = "Hello, World!"
5

```

Analysis

Tests

...



Matthijs gave this feedback on a solution very similar to yours:

Welcome to Haskell!

It looks like you're all set to proceed. Good luck 🍀

pig latin

igpay atinlay

In-progress

Medium

Overview

Your Iterations 1

Community Solutions

Code Review

Continue in editor

▼

Iteration 1

Latest

Submitted via Editor, a few seconds ago

Passed

▼

src/PigLatin.hs


package.yaml

```
1 module PigLatin (translate) where
2 import Data.List
3 translate = unwords . map ((++"ay") . translateW) . words
4 translateW :: String -> String
5 translateW "" = ""
6 translateW xs@(x:xs')
7   | x `elem` vowels = xs
8   | take 2 xs `elem` ["xr", "yt"] = xs
9   | x == 'y' = xs' ++ "y"
10  | take 2 xs' == "qu" = drop 2 xs' ++ take 3 xs
11  | take 2 xs == "qu" = drop 2 xs ++ "qu"
12  | otherwise = dropWhile ('elem' consonants) xs ++ takeWhile ('elem' consonants) xs
13 vowels = "aeiou"
14 consonants = ['a'..'z'] \\ ('y':vowels)
15
```

Analysis

Tests

...



No auto suggestions? Try human mentoring.

Get real 1-to-1 human mentoring on the Pig Latin exercise and start writing better Haskell.

Get mentoring

House

In-progress

Medium

Overview

Your iterations 1

Community Solutions

Code Review

Continue in editor

▼

Iteration 1

Latest

Submitted via Editor, a few seconds ago

Passed

▼

src/House.hs


package.yaml

```
1 module House (rhyme) where
2
3 rhyme :: String
4 rhyme = unlines [
5   "This is the house that Jack built.",
6   "",
7   "This is the malt",
8   "that lay in the house that Jack built.",
9   "",
10  "This is the rat",
11  "that ate the malt",
12  "that lay in the house that Jack built.",
13  "",
14  "This is the cat",
15  "that killed the rat",
16  "that ate the malt",
17  "that lay in the house that Jack built.",
18  "",
19  "This is the dog",
20  "that worried the cat",
21  "that killed the rat",
22  "that ate the malt",
23  "that lay in the house that Jack built.",
24  "",
25  "This is the farmer who owned the house that Jack built."
26
```

Analysis

Tests

...



No auto suggestions? Try human mentoring.

Get real 1-to-1 human mentoring on the House exercise and start writing better Haskell.

Get mentoring

Overview

Your Iterations 1

Dig Deeper

Community Solutions

Code Review

Continue in editor

Iteration 1

Latest

Submitted via Editor, a few seconds ago

Passed


src/SpaceAge.hs

package.yaml

```
1 module SpaceAge (Planet(..), ageOn) where
2
3 data Planet = Mercury
4             | Venus
5             | Earth
6             | Mars
7             | Jupiter
8             | Saturn
9             | Uranus
10            | Neptune
11
12 ageOn :: Planet -> Float -> Float
13 ageOn planet seconds = ageInEarthYears / orbitalPeriod planet
14 where
15     ageInEarthYears = seconds / 31557600
16
17 orbitalPeriod :: Planet -> Float
18 orbitalPeriod Mercury = 0.2408467
19 orbitalPeriod Venus   = 0.61519726
20 orbitalPeriod Earth   = 1.0
21 orbitalPeriod Mars    = 1.8808158
22 orbitalPeriod Jupiter = 11.862615
23 orbitalPeriod Saturn  = 29.447498
24 orbitalPeriod Uranus  = 84.016846
25 orbitalPeriod Neptune = 164.79132
26
```

Analysis

Tests



No auto suggestions? Try human mentoring.

Get real 1-to-1 human mentoring on the Space Age exercise and start writing better Haskell.

Get mentoring

Overview

Your Iterations 1

Community Solutions

Code Review

Continue in editor

Iteration 1

Latest

Submitted via Editor, a few seconds ago

Passed 1


src/ReverseString.hs

package.yaml

```
1 module ReverseString (reverseString) where
2
3 reverseString :: String -> String
4 reverseString str = reverse str
5
```

Analysis

Tests



Matthijs gave this feedback on a solution very similar to yours:

Since the functions `f` and `\x -> f x` each produce exactly the same results when given the same arguments, they are deemed equivalent. That is: whenever you write `\x -> f x` you might as well write only `f` instead. This simplification is called **η -reduction**.

η -equivalence is relevant to this solution because

`f x = body`

is syntactic sugar for (and therefore equivalent to)

`f = \x -> body`