


Hello World

src/hello_world.clj

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

Instructions Tests Results

ALL TESTS PASSED

**Sweet. Looks like you've solved the exercise!**

Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring.

Submit

1 test passed >

Reverse String (easy)


[← Back to Exercise](#) Clojure / Reverse String

src/reverse_string.clj

```
1 (ns reverse-string)
2
3 (defn reverse-string [s] ;; <- arglist goes here
4   println (apply str (reverse s)))
5 )
6
```

Instructions Tests Results

ALL TESTS PASSED

**Sweet. Looks like you've solved the exercise!**

Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring.

Submit

6 tests passed >

Armstrong Numbers (easy)


[← Back to Exercise](#) Clojure / Armstrong Numbers

src/armstrong_numbers.clj

```
1 (ns armstrong-numbers)
2
3 (defn armstrong? [num] ;; <- arglist goes here
4   (let
5     [digits (map #(Character/getNumericValue %) (str num))
6      count (count digits)
7      pow-digits (map #(apply * (repeat count %)) digits)]
8     (= num (apply + pow-digits))))
```

Instructions Tests Results

ALL TESTS PASSED

**Sweet. Looks like you've solved the exercise!**

Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring.

Submit

9 tests passed >

Allergies (medium)

src/allergies.cloj

```
1 (ns allergies)
2
3 (def ^:const als {1 :eggs 2 :peanuts 4 :shellfish 8 :strawberries
4                  16 :tomatoes 32 :chocolate 64 :pollen 128 :cats})
5
6 (defn allergies [n] ;; <- arglist goes here
7   (map val (filter (fn [[k _]] (= k (bit-and n k))) als))
8 )
9
10 (defn allergic-to? [i k] ;; <- arglist goes here
11   (contains? (set (allergies i)) k)
12 )
13
```

Instructions Tests Results

ALL TESTS PASSED



Sweet. Looks like you've solved the exercise!
Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring.

Submit

14 tests passed >


Clock (Medium)

src/clock.cloj

```
1 (ns clock)
2
3 (defn clock->string [clock] ;; <- arglist goes here
4   (let
5     [{hours :hours
6      minutes :minutes} clock]
7     (format "%02d:%02d" hours minutes)
8   )
9 )
10
11 (defn clock [hours minutes] ;; <- arglist goes here
12   (let
13     [m (mod minutes 60)
14      h-from-m (+ (if (neg? minutes) -1 0) (quot minutes 60))
15      h (mod (+ hours h-from-m) 24)]
16     {:hours h
17      :minutes m})
18 )
19
20 (defn add-time [clock time] ;; <- arglist goes here
21   (let
22     [{hours :hours
23      minutes :minutes} clock]
24     (clock/clock hours (+ minutes time)))
25 )
26
```

Instructions Tests Results

ALL TESTS PASSED



Sweet. Looks like you've solved the exercise!
Good job! You can continue to improve your code or, if you're done, submit an iteration to get automated feedback and optionally request mentoring.

Submit

3 tests passed >