POETRY OF PROGRAMMING

CODE READING EXERCISES IN CLOJURE

Contents

Sections correspond to chapters of the book `https://egri-nagy.github.io/popbook/`.

1. Instructions

The best strategy to solve these *What is the output?*, *What does the expression evaluate to?* type of problems is to think first, write down the answer, then check it in a REPL. It is kind of a reading exercise. If you try read the expression aloud, the answer should become obvious. Most questions are typical usage examples, but some of them are twisted (if not wicked). These exercises will not teach how to write code. One has to solve actual problems for that. However, reading skills are important as well: understanding someone else's code, not to mention a debugging situation.

– Write the output after the arrow below each box.
– *Each box is a separate question*, definitions are only valid in the box they are defined in.
– When the expression leads to an error message, *it is enough to state the error*, no need to specify which exception is thrown.
– *Indicate the type of the output clearly.* For example, sequence operations return lists, so when the output is `(1 2 3)`, writing `1 2 3` or `[1 2 3]` are not acceptable answers. Similarly, when the output is `"hello"`, then writing `hello` is incorrect.

2. Function composition first

```
10
```

$\longrightarrow$

```
(identity 10)
```

$\longrightarrow$

```
(inc 2)
```

$\longrightarrow$

```
(dec 9)
```

$\longrightarrow$

```
(inc (dec 0))
```

$\longrightarrow$

```
((comp inc dec) 0)
```

$\longrightarrow$

```
(dec (dec 0))
```

$\longrightarrow$

```
((comp dec dec dec) 0)
```

$\longrightarrow$

```
((comp (comp dec dec) (comp dec dec)) 5)
```

$\longrightarrow$

```
((comp identity identity) 1)
```

$\longrightarrow$

```
((comp identity inc) 1)
```

$\longrightarrow$

```
(inc 11.11)
```

$\longrightarrow$

```
(dec 0.01)
```

$\longrightarrow$

```
((comp inc) 1)
```

$\longrightarrow$

```
((comp) 42)
```

$\longrightarrow$

```
(((comp comp comp) dec dec) 1)
```

$\longrightarrow$

3. Arithmetic done with functions

```
(+ (- 6 4) (/ 6 3))
```

$\longrightarrow$

```
(+ 1 (* 2 (- 1 (/ 12 4))))
```

$\longrightarrow$

```
(+)
```

$\longrightarrow$

```
(-)
```

$\longrightarrow$

```
(*)
```

$\longrightarrow$

```
(/)
```

$\longrightarrow$

```
(+ 3)
```

$\longrightarrow$

```
(* 5)
```

$\longrightarrow$

```
(/ 3)
```

$\longrightarrow$

```
(- 5)
```

$\longrightarrow$

```
(+ (* 5 2) (- 4 3) (*))
```

$\longrightarrow$

```
((comp - *) 2 3 4)
```

$\longrightarrow$

```
((comp / *) 2 3)
```

$\longrightarrow$

```
(* (inc 1) (inc (inc 1)))
```

$\longrightarrow$

```
(/ 10 4)
```

$\longrightarrow$

```
(/ 10.0 4)
```

$\longrightarrow$

```
(/ 10 4.0)
```

$\longrightarrow$

4. Asking yes-or-no questions: predicates

```
(zero? 0)
```

$\longrightarrow$

```
(zero? 1)
```

$\longrightarrow$

```
(pos? 1)
```

$\longrightarrow$

```
(pos? 1111)
```

$\longrightarrow$

```
(pos? 0)
```

$\longrightarrow$

```
(neg? 0)
```

$\longrightarrow$

```
(neg? -2)
```

$\longrightarrow$

```
(= (+ 1 2 3) (* 1 2 3))
```

$\longrightarrow$

```
(<= 2 2 2)
```

$\longrightarrow$

```
(<= 2 1 2)
```

$\longrightarrow$

```
(< 2 1 2)
```

$\longrightarrow$

```
(< 2 2 2)
```

$\longrightarrow$

```
(< 2 3 4)
```

$\longrightarrow$

```
(> 2 3)
```

$\longrightarrow$

```
(> 2 2)
```

$\longrightarrow$

```
(>= 2 2)
```

$\longrightarrow$

```
(fn? +)
```

$\longrightarrow$

```
(fn? -)
```

$\longrightarrow$

```
(fn? identity)
```

$\longrightarrow$

```
(fn? (+))
```

$\longrightarrow$

```
(number? (+))
```

$\longrightarrow$

```
(rational? (/ 7 3))
```

$\longrightarrow$

```
(rational? 2)
```

$\longrightarrow$

```
(float? 2)
```

$\longrightarrow$

```
(float? 2.0)
```

$\longrightarrow$

```
(integer? (+ 1 2 3 4 5))
```

$\longrightarrow$

```
(integer? (+ 1 2 3.0 4 5))
```

$\longrightarrow$

```
(float? (+ 1 2 3.0 4 5))
```

$\longrightarrow$

```
(number? 12.1)
```

$\longrightarrow$

```
(number? 0)
```

$\longrightarrow$

```
(number? (/ 1 19))
```

$\longrightarrow$

5. Strings

```
(char? \x)
```

$\longrightarrow$

```
(char? \space)
```

$\longrightarrow$

```
(char? \8)
```

$\longrightarrow$

```
(= 9 \9)
```

$\longrightarrow$

```
(string? "Granny Weatherwax")
```

$\longrightarrow$

```
(string? "      ")
```

$\longrightarrow$

```
(string? "")
```

$\longrightarrow$

```
(string? "12")
```

$\longrightarrow$

```
(number? "12")
```

$\longrightarrow$

```
(= \space " ")
```

$\longrightarrow$

```
(= (str \space) " ")
```

$\longrightarrow$

```
(str \1 2 "3" (- 5 1))
```

$\longrightarrow$

```
(str "The answer:" 42)
```

$\longrightarrow$

```
(str "The answer: " 42)
```

$\longrightarrow$

```
(str "The answer: " 42 ".")
```

$\longrightarrow$

```
(str "The answer: " (* 6 7) ".")
```

⟶

WARNING! The following questions assume that the string library is loaded by `(require ’[clojure.string :as string])`.

```
(string/upper-case "helloooo!")
```

⟶

```
(string/capitalize (string/lower-case "HELLO!"))
```

⟶

```
(string/ends-with? "mango" "go")
```

⟶

```
(string/ends-with? "mango" "GO")
```

⟶

```
(string/starts-with? "How to solve it?" "?")
```

⟶

```
(string/replace "banana" "a" "e")
```

⟶

```
(string/replace (string/replace "banana" "a" "-") "-n" "x")
```

⟶

6. Making memories

```
(def a 2)
(* 3 a)
```

⟶

```
(def a 5)
(def b 7)
(* a b)
```

⟶

```
(= 'x "x")
```

⟶

```
(def x "x")
(= "x" x)
```

⟶

```
(def x "grape")
(def x "apple")
x
```

⟶

7. List, the most fundamental collection

```
'(1 2 3)
```

$\longrightarrow$

```
(1 2 3)
```

$\longrightarrow$

```
(list 4 5)
```

$\longrightarrow$

```
(list 4 5 '(6 7))
```

$\longrightarrow$

```
(list 4 5 '(6 7 (8 9)))
```

$\longrightarrow$

```
(cons 11 '(19 13))
```

$\longrightarrow$

```
(cons 7 (cons 5 (list 3 4)))
```

$\longrightarrow$

```
(first '(8 3 5))
```

$\longrightarrow$

```
(first '())
```

$\longrightarrow$

```
(rest (5 3 8))
```

$\longrightarrow$

```
(rest '(5 3 8))
```

$\longrightarrow$

```
(rest '())
```

$\longrightarrow$

```
((comp rest rest) '(2 3 4))
```

$\longrightarrow$

```
((comp first rest) '(2 3 4))
```

$\longrightarrow$

```
((comp first first) '(2 3 4))
```

$\longrightarrow$

```
(last '(9 3 2))
```

$\longrightarrow$

```
(last '())
```

$\longrightarrow$

```
(reverse '(1 2 3))
```

$\longrightarrow$

```
((comp first reverse) '(1 2 3))
```

$\longrightarrow$

```
(empty? ())
```

$\longrightarrow$

```
(empty? '(1))
```

$\longrightarrow$

```
(count ())
```

$\longrightarrow$

```
(count (list \a \b))
```

$\longrightarrow$

```
(concat '(1) '(3) '(2))
```

$\longrightarrow$

```
(concat)
```

$\longrightarrow$

8. Lazy lists of numbers

```
(range 2)
```

$\longrightarrow$

```
(range 1 2)
```

$\longrightarrow$

```
(range 1 2 3)
```

$\longrightarrow$

```
(range 2 1 3)
```

$\longrightarrow$

```
(range 0 1 0.4)
```

$\longrightarrow$

```
(take 2 (drop 2 (range)))
```

$\longrightarrow$

```
(take 3 '(3 3 3 3))
```

$\longrightarrow$

```
(take-while neg? (range -2 2))
```

$\longrightarrow$

```
(drop-while neg? (range -2 2))
```

$\longrightarrow$

```
(take-while zero? '(-1 0 1))
```

$\longrightarrow$

```
(drop-while zero? '(-1 0 1))
```

$\longrightarrow$

```
(drop-while zero? '(-1 0 1 -1))
```

$\longrightarrow$

```
(drop-while neg? '(-1 0 1 -1))
```

$\longrightarrow$

```
(count (take 5 (range)))
```

$\longrightarrow$

9. Vectors

```
(vector \a \b \c)
```

$\longrightarrow$

```
(vector [1 2])
```

$\longrightarrow$

```
(vec [1 2])
```

$\longrightarrow$

```
(vec '(\a \b))
```

$\longrightarrow$

```
(vec [])
```

$\longrightarrow$

```
(vector [])
```

$\longrightarrow$

```
(count [1 ”two” 3])
```

$\longrightarrow$

```
(count [])
```

$\longrightarrow$

```
(count [[]])
```

$\longrightarrow$

```
(count [[[]]])
```

$\longrightarrow$

```
(count [[[[[[[]]]]]]])
```

$\longrightarrow$

```
(count [[][][]])
```

$\longrightarrow$

```
(nth [\a \b \c] 0)
```

$\longrightarrow$

```
(nth [\a \b \c] 2)
```

$\longrightarrow$

```
(nth [\a \b \c] 3)
```

$\longrightarrow$

```
([\a \b \c] 0)
```

$\longrightarrow$

```
([\a \b \c] 1)
```

$\longrightarrow$

```
([\a \b \c] 3)
```

$\longrightarrow$

```
(nth [] 0)
```

$\longrightarrow$

```
(conj [] 1)
```

$\longrightarrow$

```
(conj [] 1 2)
```

$\longrightarrow$

```
(conj [] 1 2 3)
```

$\longrightarrow$

```
(conj [\a \b]  3)
```

$\longrightarrow$

```
(conj [] [])
```

$\longrightarrow$

```
(conj [] [] [])
```

$\longrightarrow$

```
(conj [1 2] [3])
```

$\longrightarrow$

```
(vec (range 3))
```

$\longrightarrow$

```
(vector (range 3))
```

$\longrightarrow$

```
((comp ["hello" "world" "!"] [2 1 0]) 2)
```

$\longrightarrow$