

(Task 1 of 8) In this tutorial, we will learn about using functions as values.

0

Note: some programming languages do NOT consider functions or methods a kind of value. But many programming languages, from Python to Rust, do.

(Task 2 of 8) What is the result of running this program?

Lispy | 

```

Lispy [Run ▶]
(deffun (twice f x) (f (f x)))
(deffun (double x) (+ x x))
(twice double 1)

JavaScript
function twice(f, x) {
  return f(f(x));
}
function double(x) {
  return x + x;
}
console.log(twice(double, 1));

```

4

2

Please briefly explain why you think the answer is 4.

3

We call the function passed into twice twice

4

You predicted the output correctly 🎉🎉🎉

5

The value of `(twice double 1)` is the value of `(f (f x))`, where `f` is bound to `double` and `x` is bound to `1`. So, the result is the value of `(double (double 1))`, which is 4.

Click [here](#) to run this program in the Stacker.

(Task 3 of 8) What is the result of running this program?

Lispy | 

```

Lispy [Run ▶]
(deffun (inc x) (+ x 1))
(deffun (g) inc)
(defvar f (g))
(f 10)

Scala 3
def inc(x : Int) =
  x + 1
def g =
  inc
val f = g
println(f(10))

```

error

7

The answer is 11

8

THE ANSWER IS 11.


▼ Textual explanation

You might think that `(g)` errors because it returns a function, and that it would not error if it returns a value of other kinds (e.g., numbers and vectors). However, it does not error. In SMoL, functions are (also) *first-class* citizens of the value world.

Click [here](#) to run this program in the Stacker.

Lispy | 

What is the result of running this program?

Lispy 	JavaScript
<pre>(deffun (fun1) (deffun (average x y) (/ (+ x y) 2)) average) (defvar x (fun1)) (x 20 40)</pre>	<pre>function fun1() { function average(x, y) { return (x + y) / 2; } return average; } let x = fun1(); console.log(x(20, 40));</pre>

30


10

You predicted the output correctly 🎉🎉🎉🎉

11

Lispy | 

(Task 4 of 8) What is the result of running this program?

Lispy 	JavaScript
<pre>(deffun (inc n) (+ n 1)) (defvar v (mvec inc inc)) ((vec-ref v 0) 2)</pre>	<pre>function inc(n) { return n + 1; } let v = [inc, inc]; console.log(v[0](2));</pre>

3

13

You predicted the output correctly 🎉🎉🎉🎉

14

`v` is bound to a vector that refers to the function `inc`. The value of `(vec-ref v 0)` is the function `inc`. So, the value of `((vec-ref v 0) 2)` is the value of `(inc 2)`, which is 3.

Click [here](#) to run this program in the Stacker.

Lispy | 

(Task 5 of 8) What is the result of running this program?

(Task 5 of 8) What is the result of running this program?

Lispy [Run ▶]	Python
(defun (f) 42)	def f(): return 42
(defvar g f)	g = f
(defvar h g)	h = g
(h)	print(h())

42

16

You predicted the output correctly 🎉🎉🎉

17

This program binds `f` to a function that returns `42`, and then binds `g` and `h` to that function. Finally, calling that function produces `42`.

Click [here](#) to run this program in the Stacker.

(Task 6 of 8) What did you learn about functions from these programs?

18

Functions exist in the same way variables do. They can be passed around and all normal variable rules apply to functions

19

(Task 7 of 8) Functions are (also) *first-class* citizens of the value world. Specifically,

20

- Variables (notably parameters) can be bound to functions,
- Functions can return functions, and
- Vectors can refer to functions.

Any feedback regarding these statements? Feel free to skip this question.

21

(You skipped the question.)

22

(Task 8 of 8) Please scroll back and select 1-3 programs that make the point above.

23

You don't need to select *all* such programs.

(You selected 3 programs)

24

Okay. How do these programs (9,12,15) support the point?

25

That is all of what these programs do

26

Let's review what we have learned in this tutorial.

27

Functions are (also) *first-class* citizens of the value world. Specifically,

- Variables (notably parameters) can be bound to functions,
- Functions can return functions, and
- Vectors can refer to functions.

You have finished this tutorial 🎉🎉🎉

Please the finished tutorial to a PDF file so you can review the content in the future. **Your instructor (if any) might require you to submit the PDF.**

Start time: 1737764497818