In this tutorial, we will learn *more* about using functions as values.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
def foo():
    n = 0
    def bar():
        nonlocal n
        n = n + 1
        return n
    return bar
f = foo()
g = foo()
print(f())
print(f())
print(g())
```

Syntax: Lispy JS PY Scala 3

`1 1 1`

Syntax: Lispy JS PY Scala 3

Pleaes briefly explain why you think the answer is `1 1 1`.

Syntax: Lispy JS PY Scala 3

Because the value of n is reset to 0 everytime the function is called? :S

Syntax: Lispy JS PY Scala 3

The answer is `1 2 1`. You are right that `n` is bound to `0` when `bar` is bound to a function. You might think the function remembers the value `0`. However, `bar` does not remember the value of `n`. Rather, it remembers the environment and hence always refers to the latest value of `n`. `foo` is called twice, so two environments are created. `f()` mutates the first, while `g()` mutates the second. In SMoL, functions refer to the latest values of variables defined outside their definitions.

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
def f():
    n = 1
    def dbl():
        nonlocal n
        n = n * 2
        return n
    return dbl
dbl1 = f()
dbl2 = f()
print(dbl1())
```

```
print(dbl2())
print(dbl1())
```

Syntax: Lispy JS PY Scala 3

```
2 2 4
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶️

```
x = 1
def f():
    def addx(y):
        return x + y
    return addx
g = f()
x = 2
print(g(0))
```

Syntax: Lispy JS PY Scala 3

```
2
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

`x` is bound to `1`. `g` is bound to the function `addx` `x = 2` binds `x` to `2`. So, the value of `g(0)` is the value of `addx(0)`, which is the value of `2 + 0`, which is `2`.

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶️

```
def bar(y):
    def addy(x):
        return x + y
    return addy
f = bar(2)
g = bar(4)
print(f(2))
print(g(2))
```

Syntax: Lispy JS PY Scala 3

```
4 6
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

The value of `bar(2)` is the function `addy` defined in an environment where `y` is bound

The value of `bar(2)` is the function `addy` defined in an environment where `y` is bound to `2`. The value of `bar(4)` is *another* `addy` defined in an environment where `y` is bound to `4`. The two `addy` functions are *different* values. So, the value of `f(2)` is `4`, while the value of `g(2)` is `6`.

Click here to run this program in the Stacker.

What did you learn about functions from these programs?

Syntax: Lispy JS PY Scala 3

The answer is 1 2 1. You are right that n is bound to 0 when bar is bound to a function. You might think the function remembers the value 0. However, bar does not remember the value of n. Rather, it remembers the environment and hence always refers to the latest value of n. foo is called twice, so two environments are created. f() mutates the first, while g() mutates the second. In SMoL, functions refer to the latest values of variables defined outside their definitions.

Functions remember the environment in which they are defined. That is, function bodies are "enclosed" by the environments in which the function values are created. So, function values are called *closures*.

Syntax: Lispy JS PY Scala 3

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

Syntax: Lispy JS PY Scala 3

(You skipped the question.)

Syntax: Lispy JS PY Scala 3

Please scroll back and select 1-3 programs that make the point that

> Functions remember the environment in which they are defined.

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

Syntax: Lispy JS PY Scala 3

(*You selected 2 programs*)

Syntax: Lispy JS PY Scala 3

Okay. How do these programs (1,6) support the point?

Syntax: Lispy JS PY Scala 3

it remembers the environment and hence always refers to the latest value of n.

Syntax: Lispy JS PY Scala 3

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

Functions remember the environment in which they are defined. That is, function bodies are "enclosed" by the environments in which the function values are created.

So, function values are called *closures*.

You have finished this tutorial 🎉🎉🎉

Please print 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: 1711101113318