Syntax: Lispy JS PY Scala 3

In this tutorial, we will learn about **lambda expressions**, which are expressions that create functions.

The following program illustrates how to create a function.

```
print(lambda n: n + 1(2))
```

Run ▶

This program program produces 3. The top-level block contains one expression, a function call. The only (actual) parameter of the function call is 2. The function of the function call is created by

```
print(lambda n: n + 1)
```

Run ▶

This function takes only one parameter n, and returns (the value of) n + 1. So, the result of the whole program is the value of 2 + 1, which is 3.

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

What is the result of running this program?

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

Run ▶

Syntax: Lispy JS PY Scala 3

3

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

This program is essentially the same as

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

Run ▶

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

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

Syntax: Lispy JS PY Scala 3

```
1
```

Syntax: Lispy JS PY Scala 3

The answer is 2. You are right that x is bound to 1 when the lambda function is created. You might think the function remembers the value 1. However, the function does not remember the value of x. Rather, it remembers the environment and hence always refers to the latest value of x. 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 ▶

```
a = 2
def make():
    return lambda b: a + b
fun = make()
a = 1
print(fun(1))
```

Syntax: Lispy JS PY Scala 3

```
2
```

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 ▶

```
def bar(y):
    return lambda x: x + y
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 a lambda function defined in an environment where `y` is bound to `2`. The value of `bar(4)` is *another* lambda function defined in an environment where `y` is bound to `4`. The two lambda functions are *different* values. So, the value of `f(2)` is `12`, while the value of `g(2)` is `52`.

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 foobar():
    n = 0
    return lambda: ("WARNING: the translation might be inaccurate",
f = foobar()
g = foobar()
print(f())
print(f())
print(g())
```

Syntax: Lispy JS PY Scala 3

```
1 2 1
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Every time `foobar` is called, it creates a *new* environment that binds `n`. So, `f` and `g` have different bindings for the `n` variable. When `f` is called the first time, it mutates its binding for the `n` variable. So, the second call to `f` produces `2` rather than `1`. `g` has its own binding for the `n` variable, which still binds `n` to `0`. So, `g()` produces `1` rather than `3`.

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

```
def f(x, y, z):
    return body
```
is a shorthand for `f = lambda x,y,z: body`.

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 rewrite this function definition with as a variable definition that binds lambda function.

Run ▶

```
def f(x):
    return x + 1
```

Syntax: Lispy JS PY Scala 3

f = lambda x: x + 1

Syntax: Lispy JS PY Scala 3

The correct answer is `f = lambda x: x + 1`.

The following are common wrong answers:

- `(deffun f (lambda (x) (+ x 1)))`, which didn't replace the definition keyword
- `f = lambda f,x: x + 1`, which makes `f` a function that takes two parameters rather than one

Syntax: Lispy JS PY Scala 3

Here is a program that confused many students

Run ▶

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

Please

1. Run this program in the stacker by clicking the green run button above;
2. The stacker would show how this program produces its result(s);
3. Keep clicking ⏭ Next until you reach a configuration that you find particularly helpful;
4. Click 🔗 Share This Configuration to get a link to your configuration;
5. Submit your link below;

Syntax: Lispy JS PY Scala 3

https://smol-tutor.xyz/stacker/?syntax=PY&randomSeed=smol-tutor&nNext=0&program=%28deffun+%28foo+n%29%0A++%28deffun+%28bar%29%0A++++%28set%21+n+%28%2B+n+1%29%29%0A++++n%29%0A++bar%29%0A%28defvar+f+%28foo+0%29%29%0A%28defvar+g+%28foo+0%29%29%0A%0A%28f%29%0A%28f%29%0A%28g%29%0A&readOnlyMode=

Syntax: Lispy JS PY Scala 3

Please write a couple of sentences to explain how your configuration explains the result(s) of the program.

Syntax: Lispy JS PY Scala 3

Every time foo is called, it creates a new environment frame that binds n. So, f and g have different bindings for the n variable. When f is called the first time, it mutates its binding for the n variable. So, the second call to f produces 2 rather than 1. g has its own binding for the n variable, which still binds n to 0. So, g() produces 1 rather than 3

Syntax: Lispy JS PY Scala 3

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

In this tutorial, we will learn about **lambda expressions**, which are expressions that create functions.

The following program illustrates how to create a function.

```
print(lambda n: n + 1(2))
```

Run ▶

This program program produces `3`. The top-level block contains one expression, a function call. The only (actual) parameter of the function call is `2`. The function of the function call is created by

```
print(lambda n: n + 1)
```

Run ▶

This function takes only one parameter `n`, and returns (the value of) `n + 1`. So, the result of the whole program is the value of `2 + 1`, which is `3`.

```
def f(x, y, z):
    return body
```

is a shorthand for `f = lambda x,y,z: body`.

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: 1711101482136