Syntax: Lispy JS PY Scala 3

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

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

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
def inc(x):
    return x + 1
def g():
    return inc
f = g()
print(f(10))
```

Syntax: Lispy JS PY Scala 3

```
error
```

Syntax: Lispy JS PY Scala 3

The answer is `11`. 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 lists). 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.

Syntax: Lispy JS PY Scala 3

What is your thought? (Feel free to skip this question.)

Syntax: Lispy JS PY Scala 3

in g(), a value is not passed into the inc() fuction, i thought it would give an error

Syntax: Lispy JS PY Scala 3

Thank you!

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
def fun1():
    def average(x, y):
        return (x + y) / 2
    return average
x = fun1()
print(x(20, 40))
```

Syntax: Lispy JS PY Scala 3

```
30
```

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 twice(f, x):
    return f(f(x))
def double(x):
    return x + x
print(twice(double, 1))
```

Syntax: Lispy JS PY Scala 3

4

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

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.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
def f():
    return 42
g = f
h = g
print(h())
```

Syntax: Lispy JS PY Scala 3

42

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

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.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
def inc(n):
    return n + 1
v = [inc, inc]
print(v[0](2))
```

Syntax: Lispy JS PY Scala 3

`3`

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

`v` is bound to a list that refers to the function `inc`. The value of `v[0]` is the function `inc`. So, the value of `v[0](2)` is the value of `inc(2)`, which is `3`.

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

What did you learn about functions from these programs?

Syntax: Lispy JS PY Scala 3

You can have an array with functions :S

Syntax: Lispy JS PY Scala 3

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

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

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 above.

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

Syntax: Lispy JS PY Scala 3

(*You selected 3 programs*)

Syntax: Lispy JS PY Scala 3

Okay. How do these programs (10,13,16) support the point?

Syntax: Lispy JS PY Scala 3

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

Syntax: Lispy JS PY Scala 3

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

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

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

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: 1711100816761