Syntax: Lispy JS PY Scala 3

In this tutorial, we will learn more about **variable assignments** and **mutable variables**.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
x = 12
def f():
    return x
def g():
    global x
    x = 0
    return f()
print(g())
x = 1
print(f())
```

Syntax: Lispy JS PY Scala 3

```
0 1
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

There is exactly one variable `x`. The `x` in `x = 0` refers to that variable. Calling `f` evaluates `x = 0`, which mutates `x`. When the value of `x` is eventually printed, we see the new value.

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(n):
    return x + n
x = 2
print(f(30))
```

Syntax: Lispy JS PY Scala 3

```
32
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

The program binds `x` to `1` and then defines a function `f`. `x` is then bound to `2`. So, `f(30)` is `x + 30`, which is `32`.s

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

```
foobar = 2
print(foobar)
```

Run ▶

Syntax: Lispy JS PY Scala 3

`error`

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Mutating an undefined variable (in this case, `foobar`) errors rather than defining the variable. So, this program errors.

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

What did you learn about variable assignment from these programs?

Syntax: Lispy JS PY Scala 3

Why is foobar not a variable name?

Syntax: Lispy JS PY Scala 3

- Variable assignments mutate existing bindings and do not create new bindings.
- Functions refer to the latest values of variables defined outside their definitions. That is, functions do not remember the values of those variables from when the functions were defined.

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 following point:

> Variable assignments mutate existing bindings and do not create new bindings.

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

Syntax: Lispy JS PY Scala 3

(*You selected 1 programs*)

Syntax: Lispy JS PY Scala 3

Okay. How does this program (7) support the point?

Syntax: Lispy JS PY Scala 3

foobar is not defined, according to stack

Syntax: Lispy JS PY Scala 3

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

> Functions refer to the latest values of variables defined outside their definitions. That is, functions do not remember the values of those variables from when the functions were 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 (1,4) support the point?

Syntax: Lispy JS PY Scala 3

That's how it works

Syntax: Lispy JS PY Scala 3

Although we keep saying "this variable is mutated", the variables themselves are *not* mutated. What is actually mutated is *the binding between the variables and their values*.

Blocks have been a good way to describe these bindings. However, they can't explain variable mutations: a block is a piece of source code, and we can't change the source code by mutating a variable. Besides, blocks can't explain how parameters might be bound differently in different function calls. Consider the following program:

Run ▶

```
def f(n):
    return n + 1
print(f(2))
print(f(3))
```

In this program, `n` is bound to `2` in this first function call, and `3` in the second. Blocks can't explain how `n` is bound differently because the two calls share the same block: the body of `f`.

What is a better way to describe the binding between variables and their values?

Syntax: Lispy JS PY Scala 3

Each function call creates a new local scope where n is temporarily bound to the argument passed in that call.

Syntax: Lispy JS PY Scala 3

Environments (rather than blocks) bind variables to values.

Similar to lists, environments are created as programs run.

Environments are created from blocks. They form a tree-like structure, respecting the tree-like structure of their corresponding blocks. So, we have a primordial environment, a top-level environment, and environments created from function environment, a top-level environment, and environments created from function bodies.

Every function call creates a new environment. This is very different from the block perspective: every function corresponds to exactly one block, its 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

Which language construct(s) create new bindings?

Syntax: Lispy JS PY Scala 3

✓ Definitions
☐ Variable mutations (e.g., `x = 3`)
☐ Variable references
✓ Function calls

Syntax: Lispy JS PY Scala 3

Which language construct(s) mutate bindings?

Syntax: Lispy JS PY Scala 3

☐ Definitions
✓ Variable mutations (e.g., `x = 3`)
✓ Variable references
☐ Function calls

Syntax: Lispy JS PY Scala 3

You should NOT have chosen “Variable references”. Only variable assignments mutate bindings. Definitions and function calls create new bindings but don't mutate existing bindings.

Syntax: Lispy JS PY Scala 3

Here is a program that confused many students

```
x = 5
def f(x, y):
    return (x := y)
print(f(x, 6))
print(x)
```

Run ▶

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=Python&randomSeed=smol-tutor&nNext=4&program=%28defvar+x+5%29%0A%28deffun+%28f+x+y%29%0A++%28set%21+x+y%29%29%0A%28f+x+6%29%0Ax%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

The ':=' operator is new and explains how a variable is assigned within an expression.

Syntax: Lispy JS PY Scala 3

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

- Variable assignments mutate existing bindings and do not create new bindings.
- Functions refer to the latest values of variables defined outside their definitions. That is, functions do not remember the values of those variables from when the functions were defined.

Environments (rather than blocks) bind variables to values.

Similar to lists, environments are created as programs run.

Environments are created from blocks. They form a tree-like structure, respecting the tree-like structure of their corresponding blocks. So, we have a primordial environment, a top-level environment, and environments created from function bodies.

Every function call creates a new environment. This is very different from the block perspective: every function corresponds to exactly one block, its 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: 1711099375880