

Syntax: Lispy JS PY Scala 3

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

The following program illustrates the new concepts.

```
x = 2
print(x)
x = x + 1
print(x)
x = x * 2
print(x)
```

Run ▶

This program produces 2 3 6. It first defines `x` and binds `x` to 2. The first variable assignment `x = x + 1` **mutates** (the binding of) `x`. After that, `x` is bound to the value of `x + 1`; this uses the new value of `x`, which is 2, resulting in `2 + 1`, which is 3. The next variable assignment again mutates `x` and binds `x` to the value of `3 * 2`, which is 6.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

```
rent = 10
rent = 10 * 2
print(rent)
```

Run ▶

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Syntax: Lispy JS PY Scala 3

`rent` is bound to 10 initially. The `set !` mutates `rent` to 20. So, when we print the value of `rent` after the `set !`, we see 20.

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

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

```
x = 12
y = x
x = 0
print(x)
print(y)
```

Run ▶

Syntax: Lispy JS PY Scala 3

Please briefly explain why you think the answer is 0 0.

Syntax: Lispy JS PY Scala 3

y is a reference to x

Syntax: Lispy JS PY Scala 3

The answer is 0 12. You might think `y = x` binds `y` to `x`, so changing `x` will change `y`. However, we learned in previous tutorials that variables are bound to values, so `y` is bound to the 12, i.e., *the value of* `x`. In SMoL, variable assignments change *only* the mutated variables.

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

What is the result of running this program?

Syntax: Lispy JS PY Scala 3

```
a = 1
b = a
a = 2
print(a)
print(b)
```

Run ▶

2 1

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Syntax: Lispy JS PY Scala 3

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

Run ▶

0

Syntax: Lispy JS PY Scala 3

The answer is 12. You might think the function call `f(x)` binds `y` to `x`, so changing `x` will change `y`. However, we learned in previous tutorials that variables are bound to values, so `y` is bound to 12, i.e., *the value of* `x`. In SMoL, variable assignments change *only* the mutated variables.


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

What is the result of running this program?

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

```
a = 1
def foobar(b):
    global a
    a = 2
    return b
print(foobar(a))
print(a)
```

Run 

1 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?

Syntax: Lispy JS PY Scala 3

```
m = 40
n = m
n = 22
print(m)
print(n)
```

Run 

40 22

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Syntax: Lispy JS PY Scala 3

`m` and `n` are bound to `40`. The variable assignment mutates the binding of `n`. So, `n` is eventually bound to `22`.

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

What is the result of running this program?

Syntax: Lispy JS PY Scala 3

```
x = 12
def f(y):
    y = 0
    return x
print(f(x))
```

Run 

12

Syntax: Lispy JS PY Scala 3

Pleaes briefly explain why you think the answer is 12.

Syntax: Lispy JS PY Scala 3

Y is a copy of the x sent into the function

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Syntax: Lispy JS PY Scala 3

The function call binds y to 12. The variable assignment mutates the value of y to 0, but x is still bound to 12.

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

What is the result of running this program?

Syntax: Lispy JS PY Scala 3

```
x = [55]
v = [x, 55, 55]
x = [66]
print(v)
```

Run ▶

`[[55], 55, 55]`

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Syntax: Lispy JS PY Scala 3

x is first bound to a one-element list. v is bound to a three-element list, of which the first element is the one-element list. x = [66] binds x to a new list. This doesn't impact v because the 0-th element of v is still the one-element list.

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

What is the result of running this program?

Syntax: Lispy JS PY Scala 3

```
x = 59
v = [68, 57, x]
x = 74
print(v)
```

Run ▶

`[68, 57, 59]`

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Syntax: Lispy JS PY Scala 3

v is bound to the list created by [68, 57, x]. [68, 57, x] is a function call. x is evaluated at the point of evaluating [68, 57, x]. That is why the list's content is 68, 57, and 59. The subsequent variable assignment mutates x. But this doesn't impact the list because the list refers the value 59 rather than x.

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

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

What did you learn about variable assignment from these programs?

Syntax: Lispy JS PY Scala 3

That variables are always assigned values, and that they are not references in python...

Syntax: Lispy JS PY Scala 3

Variable assignments change *only* the mutated variables. That is, variables are not aliased.

Syntax: Lispy JS PY Scala 3

(Note: some programming languages (e.g., C++ and Rust) allow variables to be aliased. However, even in those languages, variables are not aliased by default.)

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 above point.

Syntax: Lispy JS PY Scala 3

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

*(You selected 1 programs)*

Syntax: Lispy JS PY Scala 3

Okay. How does this program ([29](#)) support the point?

Syntax: Lispy JS PY Scala 3

The subsequent variable assignment mutates x. But this doesn't impact the list because the list refers the value 59 rather than x.

Syntax: Lispy JS PY Scala 3

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

Syntax: Lispy JS PY Scala 3

Variable assignments change *only* the mutated variables. That is, variables are not aliased.

(Note: some programming languages (e.g., C++ and Rust) allow variables to be aliased. However, even in those languages, variables are not aliased by default.)

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: 1711098855546

