Syntax: Lispy JS PY Scala 3

This series of tutorials revolve around a central idea, **SMoL**, the Standard Model of Languages. This is the embodiment of the computational core of many of our widely-used programming languages, from C# and Java to JavaScript, and Python to Scala and Racket. All these languages (and many others), to a large extent, have a common computational core:

- lexical scope
- nested scope
- eager evaluation
- sequential evaluation (per “thread”)
- mutable first-order variables
- mutable first-class structures (objects, lists, etc.)
- higher-order functions that close over bindings
- automated memory management (e.g., garbage collection)

What goes in SMoL is, of course, a judgment call: when a feature isn't present across a large number of diverse languages (like static types), or shows too much variation between languages that have it (like objects), we do not include that in the *standard* model. But it is not a value judgment: the standard model is about what languages *are*, rather than what languages *should be*.

Syntax: Lispy JS PY Scala 3

In this tutorial, we will learn about **definitions**.

The following program illustrates **variable definitions**.

Run ▶

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

In this program, the first variable definition **binds** `x` to `1`, and the second variable definition binds `y` to `2`.

This program produces three values: the value of `x`, the value of `y`, and the value of `x + y`. These values are `1`, `2`, and `3`, respectively. In this Tutor, we will write the result of running this program on a single line as

```
1 2 3
```

rather than

```
1
2
3
```

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

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

Run ▶

Syntax: Lispy JS PY Scala 3

```
1 3
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

The first definition binds `x` to the value `1`. The second definition binds `y` to the value of `x + 2`, which is `3`. So, the result of this program is `1 3`.

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

The following program illustrates **function definitions**.

```
def f(x, y):
    return (x + y) / 2
print(f(2 * 3, 4))
```

Run ▶

This program produces `5`. It defines a function named `f`, which has two **formal parameters**, `x` and `y`, and **calls** the function with the **actual parameters** `6` and `4`.

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

```
def f(x, y, z):
    return x + (y + z)
print(f(2, 1, 3))
```

Run ▶

Syntax: Lispy JS PY Scala 3

```
6
```

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

Function definitions can contain definitions, for example

```
def f(x, y):
    n = x + y
    return n / 2
print(f(2 * 3, 4))
```

Run ▶

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

Run ▶

```
def f(x, y, z):
    p = y * z
    return x + p
print(f(2, 1, 3))
```

Syntax: Lispy JS PY Scala 3

`5`

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

We have two kinds of places where a definition might happen: the top-level **block** and function bodies (which are also **blocks**). A block is a sequence of definitions and expressions.

Blocks form a tree-like structure in a program. For example, we have four blocks in the following program:

Run ▶

```
n = 42
def f(x):
    y = 1
    return x + y
def g():
    def h(m):
        return 2 * m
    return f(h(3))
print(g())
```

The blocks are:

- the top-level block, where the definitions of `n`, `f`, and `g` appear
- the body of `f`, where the definition of `y` appears, which is a sub-block of the top-level block
- the body of `g`, where the definition of `h` appears, which is also a sub-block of the top-level block, and
- the body of `h`, where no local definition appears, which is a sub-block of the body of `g`

Syntax: Lispy JS PY Scala 3

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

(You skipped the question.)

Syntax: Lispy JS PY Scala 3

We use the term **values** to refer to the typical result computations. These include numbers, strings, booleans, etc. However, running a program can also produce an **error**.

For example, the result of the following program is an `error` because you can't divide a number by 0.

```
x = 23
print(x / 0)
```

Run ▶

The result of the following program is `True False error` because you can't add two boolean values.

```
print(True)
print(False)
print(True + False)
```

Run ▶

Syntax: Lispy JS PY Scala 3

What is the result of running this program?

```
xyz = 42
print(abc)
```

Run ▶

Syntax: Lispy JS PY Scala 3

`error`

Syntax: Lispy JS PY Scala 3

You got it right! 🎉🎉🎉

The variable `abc` is not defined.

Click here to run this program in the Stacker.

Syntax: Lispy JS PY Scala 3

It is an error to evaluate an undefined variable.

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

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

We have two kinds of places where a definition might happen: the top-level **block** and

function bodies (which are also **blocks**). A block is a sequence of definitions and expressions.

Blocks form a tree-like structure in a program. For example, we have four blocks in the following program:

Run ▶️

```
n = 42
def f(x):
    y = 1
    return x + y
def g():
    def h(m):
        return 2 * m
    return f(h(3))
print(g())
```

The blocks are:

- the top-level block, where the definitions of `n`, `f`, and `g` appear
- the body of `f`, where the definition of `y` appears, which is a sub-block of the top-level block
- the body of `g`, where the definition of `h` appears, which is also a sub-block of the top-level block, and
- the body of `h`, where no local definition appears, which is a sub-block of the body of `g`

It is an error to evaluate an undefined variable.

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: 1711095274422