

(Task 1 of 11) 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, vectors, 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*.

(Task 2 of 11) In this tutorial, we will learn about **definitions**.

The following program illustrates **variable definitions**.

```
Lispy [Run] Scala 3
(defvar x 1)   val x = 1
(defvar y 2)   val y = 2
x             println(x)
y             println(y)
(+ x y)       println(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
```

Lispy | 

(Task 3 of 11) What is the result of running this program?

<small>Lispy</small> [Run 	<small>Pseudo</small>
<code>(defvar x 1)</code>	<code>let x = 1</code>
<code>(defvar y (+ x 2))</code>	<code>let y = x + 2</code>
<code>x</code>	<code>print(x)</code>
<code>y</code>	<code>print(y)</code>

1 3


3

You predicted the output correctly 🎉🎉🎉

4

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.Lispy | (Task 4 of 11) The following program illustrates **function definitions**.

<small>Lispy</small> [Run 	<small>Scala 3</small>
<code>(deffun (f x y)</code>	<code>def f(x : Int, y : Int) =</code>
<code> (/ (+ x y) 2))</code>	<code> (x + y) / 2</code>
	<code> println(f(2 * 3, 4))</code>
<code>(f (* 2 3) 4)</code>	

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`.

Lispy | 

(Task 5 of 11) What is the result of running this program?

<small>Lispy</small> [Run 	<small>Python</small>
<code>(deffun (f x y z)</code>	<code>def f(x, y, z):</code>
<code> (+ x (+ y z)))</code>	<code> return x + (y + z)</code>
<code>(f 2 1 3)</code>	<code>print(f(2, 1, 3))</code>

6

7

You predicted the output correctly 🎉🎉🎉

8

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

(Task 6 of 11) Function definitions can contain definitions, for example

<p>Lispy [Run ▶]</p> <pre>(deffun (f x y) (defvar n (+ x y)) (/ n 2)) (f (* 2 3) 4)</pre>	<p>Pseudo</p> <pre>fun f(x, y): let n = x + y return n / 2 end print(f(2 * 3, 4))</pre>
--	--

(Task 7 of 11) What is the result of running this program?

<p>Lispy [Run ▶]</p> <pre>(deffun (f x y z) (defvar p (* y z)) (+ x p)) (f 2 1 3)</pre>	<p>Scala 3</p> <pre>def f(x : Int, y : Int, z : Int) = val p = y * z x + p println(f(2, 1, 3))</pre>
--	---

5

11

You predicted the output correctly 🎉🎉🎉🎉

12

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

(Task 8 of 11) 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:

<p>Lispy [Run ▶]</p> <pre>(defvar n 42) (deffun (f x) (defvar y 1) (+ x y)) (deffun (g) (deffun (h m) (* 2 m)) (f (h 3))) (g)</pre>	<p>Pseudo</p> <pre>let n = 42 fun f(x): let y = 1 return x + y end fun g(): fun h(m): return 2 * m end return f(h(3)) end print(g())</pre>
--	---

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`

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

14

(You skipped the question.)

15

(Task 9 of 11) 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**.

Lispy |

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

```

Lispy [Run] Scala 3
(defvar x 23) val x = 23
(/ x 0)      println(x / 0)

```

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

```

Lispy [Run] Scala 3
#t      println(true)
#f      println(false)
(+ #t #f) println(true + false)

```

(Task 10 of 11) What is the result of running this program?

Lispy |

```

Lispy [Run] Python
(defvar xyz 42) xyz = 42
abc           print(abc)

```

`error`

18

You predicted the output correctly 🎉🎉🎉

19

The variable `abc` is not defined.

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

(Task 11 of 11) It is an error to evaluate an undefined variable.

20


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

21

(You skipped the question.)

22

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

Lispy | 

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:

```

Lispy [Run] Scala 3
(defvar n 42)      val n = 42
                   def f(x : Int) =
(deffun (f x)      val y = 1
                   (defvar y 1)    x + y
                   (+ x y))        def g =
                                   def h(m : Int) =
(deffun (g)        2 * m
                   (deffun (h m)    f(h(3))
                       (* 2 m))      println(g)
                   (f (h 3)))

```

(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 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: 1737138466678