Lispy |

(Task 1 of 11) In this tutorial, we will learn about **variable assignments** and **mutable variables**.

The following program illustrates the new concepts.

Lispy [Run] Python

```
(defvar x 2)        x = 2
x                   print(x)
(set! x (+ x 1))    x = x + 1
x                   print(x)
(set! x (* x 2))    x = x * 2
x                   print(x)
```

This program produces `2 3 6`. It first defines `x` and binds `x` to `2`. The first variable assignment `(set! 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`.

Lispy |

(Task 2 of 11) What is the result of running this program?

Lispy [Run] Pseudo

```
(defvar rent 10)        let rent = 10
(set! rent (* 10 2))    rent = 10 * 2
rent                    print(rent)
```

`20`

2

You predicted the output correctly 🎉🎉🎉

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.

Lispy |

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

Lispy [Run] Pseudo

```
(defvar x (mvec 55))          let x = vec[55]
(defvar v (mvec x 55 55))     let v = vec[x, 55, 55]
(set! x (mvec 66))            x = vec[66]
v                             print(v)
```

`#(#(66) 55 55)`

5

The answer is `#(#(55) 55 55)`.

▼ Textual explanation

You might think the `0`-th and first element of `v` refers to `x`. However, vectors refer to values, so it actually refers to the one-element vector, i.e., *the value of* `x`. In SMoL, variable assignments change *only* the mutated variables.

Click here to run this program in the Stacker.

6

Lispy |

What is the result of running this program?

Lispy [Run]

```
(defvar x (mvec 0))
(defvar v (mvec 2 x 3))
(set! x (mvec 1))
v
```

JavaScript

```
let x = [ 0 ];
let v = [ 2, x, 3 ];
x = [ 1 ];
console.log(v);
```

`#(2 #(0) #3)`

8

You predicted the output correctly 🎉🎉🎉

9

Lispy |

(Task 4 of 11) What is the result of running this program?

Lispy [Run]

```
(defvar x 12)
(deffun (f y)
  (set! y 0)
  x)
(f x)
```

Pseudo

```
let x = 12
fun f(y):
  y = 0
  return x
end
print(f(x))
```

`0`

11

The answer is `12`.

▼ Textual explanation

You might think the function call `(f x)` binds `y` to `x`, so changing `y` will change `x`. 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.

12

Lispy |

What is the result of running this program?

Lispy [Run]

```
(defvar s 1)
(deffun (foobar t)
  (set! t 2)
  s)
(foobar s)
```

Scala 3

```
var s = 1
def foobar(t : Int) =
  t = 2
  s
println(foobar(s))
```

14

```
1
```

15

You predicted the output correctly 🎉🎉🎉

Lispy |

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

Lispy [Run]

```
(defvar m 40)
(defvar n m)
(set! n 22)
m
n
```

Scala 3

```
var m = 40
var n = m
n = 22
println(m)
println(n)
```

17

```
40 22
```

18

You predicted the output correctly 🎉🎉🎉

`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 Stacker.

Lispy |

(Task 6 of 11) What is the result of running this program?

Lispy [Run]

```
(defvar x 12)
(deffun (f y)
  (set! x 0)
  y)
(f x)
```

Pseudo

```
let x = 12
fun f(y):
  x = 0
  return y
end
print(f(x))
```

20

```
12
```

21

You predicted the output correctly 🎉🎉🎉

You predicted the output correctly 🎉🎉🎉

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

Click here to run this program in the Stacker.

Lispy |

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

Lispy [Run]

```
(defvar x 59)
(defvar v (mvec 68 57 x))
(set! x 74)
v
```

JavaScript

```
let x = 59;
let v = [ 68, 57, x ];
x = 74;
console.log(v);
```

23

`#(68 57 59)`

24

You predicted the output correctly 🎉🎉🎉

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

Click here to run this program in the Stacker.

Lispy |

(Task 8 of 11) What is the result of running this program?

Lispy [Run]

```
(defvar x 12)
(defvar y x)
(set! x 0)
x
y
```

Python

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

26

`0 12`

27

You predicted the output correctly 🎉🎉🎉

The first definition binds `x` to `12`. The second definition binds `y` to the value of `x`, which is `12`. The variable assignment mutates the binding of `x`, so `x` is now bound to `0`. But `y` is still bound to `12`.

Click here to run this program in the Stacker.

(Task 9 of 11) What did you learn about variable assignment from these programs? 28

They mutate the variable bound in the environment, by assigning an new heap allocated value to the variable. Previously retained references to the old variable e.g. vectors within vectors retain references to the old vector 29

(Task 10 of 11) Variable assignments change *only* the mutated variables. That is, variables are not aliased. 30

(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. 31

(You skipped the question.) 32

(Task 11 of 11) Please scroll back and select 1-3 programs that make the above point. 33

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

(*You selected 2 programs*) 34

Okay. How do these programs (4,7) support the point? 35

Previously retained references to the old variable e.g. vectors within vectors retain references to the old vector 36

Let's review what we have learned in this tutorial. 37

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 🎉🎉🎉

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: 1737761241944