(Task 1 of 9) In this tutorial, we will review previously learned content.

(Task 2 of 9) What is the result of running this program?

Lispy [Run]

```
(defvar a 5)
(deffun (k b)
  (set! b 3)
  a)
(k a)
```

JavaScript

```
let a = 5;
function k(b) {
  b = 3;
  return a;
}
console.log(k(a));
```

```
5
```

You predicted the output correctly 🎉🎉🎉

The function call binds `b` to `5`. The `set!` mutates the value of `b` to `3`, but `a` is still bound to `5`.

Click here to run this program in the Stacker.

(Task 3 of 9) What is the result of running this program?

Lispy [Run]

```
(defvar a 6)
(defvar b a)
(set! a 5)
a
b
```

Python

```
a = 6
b = a
a = 5
print(a)
print(b)
```

```
5 6
```

You predicted the output correctly 🎉🎉🎉

The first definition binds `a` to `6`. The second definition binds `b` to the value of `a`, which is `6`. The `set!` mutates the binding of `a`, so `a` is now bound to `5`. But `b` is still bound to `6`.

Click here to run this program in the Stacker.

(Task 4 of 9) What is the result of running this program?

Lispy [Run]

```
(defvar zz (mvec 88 88))
(deffun (f aa)
  (vec-set! aa 0 97))
(f zz)
zz
```

Pseudo

```
let zz = vec[88, 88]
fun f(aa):
  aa[0] = 97
  return
end
print(f(zz))
print(zz)
```

```
print(zz)
```

```
#(97 88)
```

8

You predicted the output correctly 🎉🎉🎉

9

The program first creates a two-element vector. The elements are both `88`. After that, the program defines a function `f`. The function call `(f zz)` replaces the first vector element with `97`. After that, the vector is printed. The vector now refers `97` and `88`, so the result is `#(97 88)`.

Click here to run this program in the Stacker.

Lispy |

(Task 5 of 9) What is the result of running this program?

Lispy [Run ▶]

```
(defvar m (mvec 66))
(defvar z (mvec m 66 66))
(set! m (mvec 43))
z
```

Python

```
m = [ 66 ]
z = [ m, 66, 66 ]
m = [ 43 ]
print(z)
```

```
#(#(66) 66 66)
```

11

You predicted the output correctly 🎉🎉🎉

12

`m` is first bound to a one-element vector. `z` is bound to a three-element vector, of which the first element is the one-element vector. `(set! m (mvec 43))` binds `m` to a new vector. This doesn't impact `z` because the 0-th element of `z` is still the one-element vector.

Click here to run this program in the Stacker.

Lispy |

(Task 6 of 9) What is the result of running this program?

Lispy [Run ▶]

```
(defvar ns (mvec 74 85))
(vec-set! ns 0 ns)
(vec-ref ns 1)
```

Pseudo

```
let ns = vec[74, 85]
ns[0] = ns
print(ns[1])
```

```
85
```

14

You predicted the output correctly 🎉🎉🎉

15

`ns` is bound to a vector. `(vec-set! ns 0 ns)` replaces the 0-th element of the vector with the vector itself. This is fine because a vector element can be any value, including itself. Besides, the vector is not copied, so the mutation finishes immediately.

Click here to run this program in the Stacker.

Lispy | 16

(Task 7 of 9) What is the result of running this program?

Lispy [Run ▶]

```
(defvar n 7)
(deffun (foo)
  n)
(deffun (bar)
  (set! n 3)
  (foo))
(bar)
(set! n 5)
(foo)
```

Pseudo

```
let n = 7
fun foo():
  return n
end
fun bar():
  n = 3
  return foo()
end
print(bar())
n = 5
print(foo())
```

17

```
3 5
```

18

You predicted the output correctly 🎉🎉🎉

There is exactly one variable `n`. The `n` in `(set! n 3)` refers to that variable. Calling `bar` evaluates `(set! n 3)`, which mutates `n`. When the value of `n` is eventually printed, we see the new value.

Click here to run this program in the Stacker.

Lispy | 19

(Task 8 of 9) What is the result of running this program?

Lispy [Run ▶]

```
(defvar nn (mvec 66))
(defvar mm (mvec 77 nn))
(vec-set! nn 0 77)
mm
```

Pseudo

```
let nn = vec[66]
let mm = vec[77, nn]
nn[0] = 77
print(mm)
```

20

```
#(77 #(77))
```

21

You predicted the output correctly 🎉🎉🎉

`nn` is bound to a one-element vector. The only element of this vector is `66`. `mm` is bound to a two-element vector. The elements of this vector are `77` and the one-element vector. The subsequent `(vec-set! nn 0 77)` mutates the one-element vector. Finally, the value of `mm` is printed.

Click here to run this program in the Stacker.

Lispy | 22

(Task 9 of 9) What is the result of running this program?

Lispy [Run ▶]

```
(defvar p (mvec 72))
(defvar q p)
(vec-set! p 0 94)
```

JavaScript

```
let p = [ 72 ];
let q = p;
p[0] = 94;
```

```
q                                    console.log(q);
```

```
#(94)
```

23

You predicted the output correctly 🎉🎉🎉

24

The program creates a one-element vector (the only element being 72) and binds it to both `p` and `q`. The `0`-th element of the vector is then replaced with `94`. So, when the vector is printed as `#(94)`.

Click here to run this program in the Stacker.

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: 1738368143961