(Task 1 of 7) In this tutorial, we will learn even more about **mutable values**, illustrated with **vectors**. 0

Lispy | 🎲 1

(Task 2 of 7) Which choice best describes the heap at the end of the following program?

(**Note**: we use `@ddd` (e.g., `@123`, `@200`, and `@100`) to represent heap addresses. Heap addresses are *random*. The numbers don't mean anything.)

```
              Lispy [Run ▶]              JavaScript
(defvar x (mvec 2))    let x = [ 2 ];
(vec-set! x 0 33)      x[0] = 33;
x                      console.log(x);
```

2

A. `@100 = #(2); @200 = #(33)`
B. `@200 = #(2)`
C. `@200 = #(33)`
D. `@200 = 33`

`@200 = #(33)` 3

You gave the correct answer 🎉🎉🎉 4

Exactly one vector was created. So, there must be at most one vector on the heap. This rules out **A**.

For now, the heap maps addresses only to vectors. This rules out **D**.

The heap looks like **B** after evaluating `(mvec 2)`. However, the subsequent mutation changes the vector. So, the correct answer is **C**.

Lispy | 🎲 5

(Task 3 of 7) Which choice best describes the heap at the end of the following program?

```
                 Lispy [Run ▶]                  Python
(defvar v (mvec 1 2 3))         v = [ 1, 2, 3 ]
(defvar vv (mvec v v))          vv = [ v, v ]
(vec-set! (vec-ref vv 1) 0 6)   vv[1][0] = 6
(vec-ref vv 0)                  print(vv[0])
```

6

A. `@100 = #(1 2 3); @200 = #(@100 @100)`
B. `@100 = #(1 2 3); @200 = #(@100 @300); @300 = #(6 2 3)`

B. `@100 = #(1 2 3); @200 = #(@100 @300); @300 = #(6 2 3)`
C. `@100 = #(1 2 3); @200 = #(#(1 2 3) #(6 2 3))`
D. `@100 = #(1 2 3); @200 = #(1 2 3); @300 = #(6 2 3); @400 = #(@200 @300)`
E. `@100 = #(1 2 3); @200 = #(6 @100)`
F. `@100 = #(6 2 3); @200 = #(@100 @100)`
G. `@100 = #(6 2 3); @200 = #(#(1 2 3) #(6 2 3))`

`@100 = #(6 2 3); @200 = #(@100 @100)` 7

You gave the correct answer 🎉🎉🎉 8

Vectors refer values (e.g., `1` and `@200`). This rules out **C** and **G**.

Two vectors are created. So, there must be two vectors on the heap. This rules out **B** and **D**.

`vv` is bound to a 2-element vector, and the 1-th element of the 2-element vector must be the 3-element vector. The mutation replaces the 0-th element in the 3-element vector with `6`. So, **F** is the correct answer, while **A** does not reflect the effect of the mutation, and **E** mutates the wrong vector.

Lispy | 🎲 9

(Task 4 of 7) Which choice best describes the heap at the end of the following program?

Lispy [Run ▶] **Python**

```
(defvar x (mvec 3))         x = [ 3 ]
(defvar v (mvec 1 2 x))     v = [ 1, 2, x ]
(vec-set! x 0 4)            x[0] = 4
v                           print(v)
```

10

A. `@100 = #(3); @200 = #(1 2 @100)`
B. `@100 = #(3); @200 = #(1 2 @300); @300 = #(4)`
C. `@100 = #(4); @200 = #(1 2 @100)`
D. `@100 = #(4); @200 = #(1 2 #(3))`
E. `@100 = #(4); @200 = #(1 2 #(4))`
F. `@100 = #(4); @200 = #(1 2 3)`
G. `@100 = #(4); @200 = #(1 2 4)`

`@100 = #(4); @200 = #(1 2 @100)` 11

You gave the correct answer 🎉🎉🎉 12

D and **E** are wrong because vectors refer values. `#(3)` and `#(4)` are not values, although they can be the printed representation of a vector.

F and **G** are wrong because the 2-th element of the 3-element vector must be a vector. The 3-element vector is created by `(mvec 1 2 x)`. The value of `x` is a vector at that moment. This 3-element vector is never mutated.

B is wrong because only two vectors are created. There must be at most two vectors on the heap.

A can be the heap after the two vectors are created. However, the subsequent mutation changes the shorter vector. So, **C** is the correct answer.

Lispy | 🎲

13

(Task 5 of 7) Which choice best describes the heap at the end of the following program?

Lispy [Run ▶] Pseudo

```
(defvar m (mvec 1 2))        let m = vec[1, 2]
(vec-set! m 1 (mvec 3 4))    m[1] = vec[3, 4]
```

14

A. `@100 = #(@200 1); @200 = #(3 4)`
B. `@100 = #(1 @200); @200 = #(3 4)`
C. `@100 = #(1 #(3 4))`
D. `@100 = #(1 #(3 4)); @200 = #(3 4)`
E. `@100 = #(1 2)`
F. `@100 = #(1 2); @200 = #(3 4)`

15

`@100 = #(1 @200); @200 = #(3 4)`

16

You gave the correct answer 🎉🎉🎉

Two vectors are created. So, there must be at least two things on the heap. This rules out **C** and **E**.

D is wrong because vectors refer values (e.g., `1`, `2`, and `@200`). `#(3 4)` is not itself a value; it's the *printing* of the value that resides at `@200`. **F** can be the heap after the second vector is created. However, the subsequent mutation changes `@100`. So, **F** is wrong.

The mutation replaces the 1-th (i.e., second) element rather than the 0-th element, so **B** is correct, while **A** is wrong.

Lispy | 🎲

(Task 6 of 7) Which choice best describes the heap at the end of the following program?

(Task 6 of 7) Which choice best describes the heap at the end of the following program?

```
              Lispy [Run ▶]              Python
 (defvar x (mvec 1 0 2))   x = [ 1, 0, 2 ]
 (vec-set! x 1 x)          x[1] = x
 (vlen x)                  print(len(x))
```

18

A. `@100 = #(1 @100 2)`
B. `@100 = #(1 @200 2); @200 = #(1 0 2)`
C. `@100 = #(1 #(1 0 2) 2)`
D. `@100 = #(1 0 2)`

19

`@100 = #(1 @200 2); @200 = #(1 0 2)`

20

The answer is `@100 = #(1 @100 2)`.

C is wrong because vectors refer values. `#(1 0 2)` is not a value, although it can be some vector values printed.

B is wrong because only one vector is created. There must not be two vectors on the heap.

D can be the heap after the vector is created. But the subsequent mutation replaces the 1-th element of `@100` with `@100`. So, **A** is the correct answer.

Lispy | 🎲

21

(Task 7 of 7) Which choice best describes the heap at the end of the following program?

```
              Lispy [Run ▶]              Pseudo
 (defvar mv (mvec 4 5))   let mv = vec[4, 5]
 (vec-ref mv 0)           print(mv[0])
```

22

A. `@100 = #(@200 @300); @200 = 4; @300 = 5`
B. `@100 = #(4 5)`
C. `@100 = #(4 5); @200 = 4`

23

`@100 = #(4 5)`

24

You gave the correct answer 🎉🎉🎉

For now, the heap maps addresses only to vectors. This rules out **A** and **C**.

So, **B** is correct

So, **D** is correct.

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: 1737146456883