CS 120: Intro to Computer Programming II

In-Class Activity - 02 References - Day 1

Activity 1 - Turn in this one

Discuss each of these questions with your group. Make sure that everybody is involved; don't just give the answers. It's important that everybody agree, and understand the answers!

- What is a "reference?" If a variable `x` is a reference to an array, does `x` actually contain the array?
- In the video and the slides, we said that the line of code

  ```
  x = [1,2,3,4,5]
  ```

  was executed in two steps. What were these steps?
- Arrays have a very special **structural** property (that is, a property of how they are built in memory) which was good for performance. What was that property, and why was that useful?

Solution: Question 1
A reference is a "pointer" to another location in memory. (Down in the guts, it's an integer, which holds the **address** where an object is stored.

If `x` is a reference to an array, it does **NOT** contain the array itself; instead, it stores the address which will allow us to find the array elsewhere in memory.

Question 2
In an assignment statement, we will first build the objecte (in this case, an array) on the right-hand side; and then, as a second step, make the variable on the left point at that object.

Question 3
All of the slots in an array are the same size. This makes it possible to jump **directly** to the destination slot, in one operation, since it's easy to calculate the address where that slot will reside.

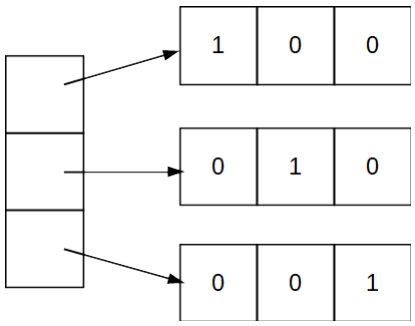
Activity 2 - Turn in this one

Draw the data structure diagram (that is, the reference diagram) for the following:

```
[ [1,0,0], [0,1,0], [0,0,1] ]
```

As always, **make sure to discuss this with your group.** Make sure that everybody can see the diagram. Pay attention to see if you agree how many arrays there are, what their contents are, and where they are stored.

Solution:



Activity 3 - Turn in this one

Draw the data structure diagram for the code snippet below; at each point where you see `DRAW HERE`, draw a picture of how the variables look at exactly that point.

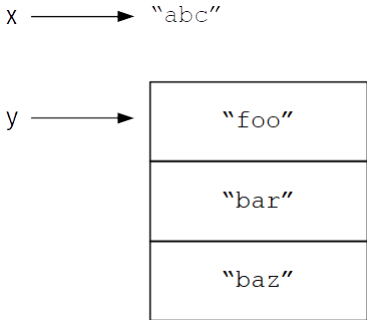
Yes, that means that you will have to re-draw objects, multiple times - sorry about that! But I want you to show how the variables **changes** over time.

```
x = "abc"
y = ["foo", "bar", "baz"]
# DRAW HERE

y[0] = "hello"
y[1] = "world"
y[2] = [ "here", "is", "data" ]
# DRAW HERE

y = "nothing here, anymore"
# DRAW HERE
```

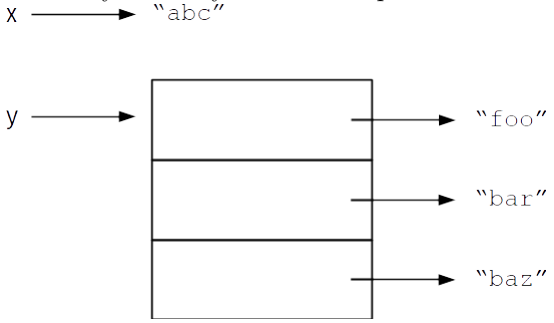
Solution: Important Note:
It's true that strings are, down in the guts, actually arrays. And I've drawn them like that in the Python Review from time to time. But when you draw a reference diagram, **do not draw them as arrays!** If you draw them that way, then I will think that you're drawing an actual array, with lots of single-character strings:

```
i_will_think_you_mean = ['f', 'o', 'o', 'b', 'a', 'r']
```

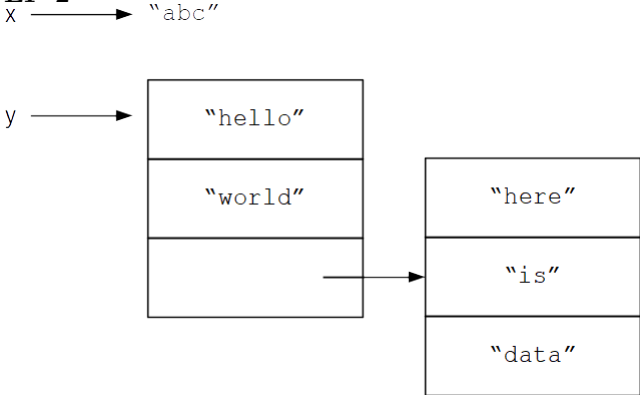
STEP 1
Here are a couple of perfectly acceptible drawings for the **first step.** The first one is how I would recommend you draw things, for simplicity:



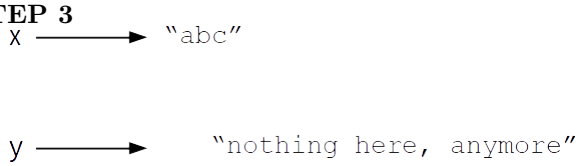
This picture is more **literally** true, because an array doesn't really contain the strings, but instead contains references to strings. While it's more true than the previous one, I find it annoying to draw, which is why I allow you to do the previous version:



STEP 2



STEP 3



(activity continues on the next page)

Activity 4 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

In the previous activity, the last line of code was interesting: it **replaced** an array entirely; it changed the variable `y`, which used to have an array, to point to a string instead.

Discuss with your group: What do you think happened to the old array? Where is it now? (If you know the answer, don't shout it out; let your group discuss the possible options.)

In this activity, I want you to turn in **all** of the various ideas that people in your group reported, even if you think they are wrong. Even if somebody knows the right answer (or at least, thinks they do!), report the other answers as well.

Solution: Here's the actual answer (I hope your group came up with many interesting guesses!)

In Python, when the language detects that the last reference to a certain object has gone away, Python will **automatically, silently** clean up the memory, and make it available for other purposes. And as it turns out, most objects are quite easy to "garbage collect."

But sometimes, an object becomes garbage, and it's not immediately obvious. For those, a more expensive scan through memory is required, and we don't run that often. But, if Python starts to run short of memory, it automatically kicks off the "garbage collector," which finds all of those hard-to-detect objects, and cleans them up as well.

All of this happens automatically, without you doing anything.

Activity 5 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

What does the `id()` function return in Python? If two variables have different id's, what does that tell you about what's going on, inside memory? If two variables have the `same` id, what does that tell you?

Solution: The `id()` function returns an integer, which functions as a unique identifier (no two different objects will have the same ID, guaranteed).

I have speculated that **probably** the ID is simply the address of the object in memory, but I'm not certain that this is the case.

Challenge Activity - Do not turn in this one

Draw the data structure diagram for the following code snippet. (Execute it in Python, and print out all of the id's, to check your work.)

```
x = [1,2,3]
y = [4,5,6]
z = x
```

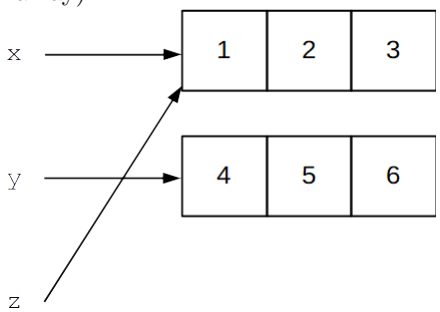
Next, draw the data structure diagram for the following code snippet. Execute this in Python as well, to double-check your id's.

```
w    = [0,0,0]
w[0] = [1,2,3]
```

```
w[1] = [4,5,6]
w[2] = w[0]
```

Finally, print out `w`. What do you see?

Solution: This is the first drawing. Notice that the variables `x,z` point to the same object in memory (an array).



This is the second drawing. Notice that it looks **exactly the same** as the previous picture - except that the 3 references are elements in the array, not named variables.

In fact, elements in an array **are,** more or less, simply more variables, and all of the rules for assignment work for them, just like anything other variable!