

Variables v. Objects Exercises - due tuesday 09/07

Exercise 1

Using the space below, write out, one line at a time, the execution of the following block of code which first creates a dictionary with the names of all the animals in my very small (and weird) zoo.

Now, given this code, what would the value of `animals_in_my_zoo['Meerkat']` be?

- Both `x` and `animals_in_my_zoo` are variables (data frames) that point to the same list (object) of animals in the zoo. Hence when we add a Meerkat called Peeps to the list that object `x` points to, if we ask Python to pull information from the shelf that `animals_in_my_zoo` points to, we end up with the same information because it points to the same shelf as `x`. I.e. we can find the Meerkat called Peeps. Hence, the value of `animals_in_my_zoo['Meerkat']` would give us the name "Peeps". **See the last three rows on page 2 on the code template pdf for a sketch of the code.**

Once you've gotten an answer, open up Python and run the code to see if you were correct. If not, can you tell why?

```
In [25]: animals_in_my_zoo = {'Zebras': ['Stripey', 'Black and White'],
                             'Polar Bears': 'Fluffy',
                             'Dinosaurs': 'Bitey'}
x = animals_in_my_zoo
x['Meerkat'] = 'Peeps'
```

Checking whether our assumption was correct:

```
In [26]: print(animals_in_my_zoo['Meerkat'])
```

Peeps

Yay, we were correct!

Exercise 2

Now let's do a few list manipulations. For these manipulations, you may need to open Python to check the documentation for different functions to figure out whether they mutate the list in-place, or whether they create a new object. Sketch out this code, and make a prediction for the value of y. Then run the code in Python to check your answer.

```
In [27]: x = [1, 2, 3, 4]
         y = x
         x.pop(1)
```

```
Out[27]: 2
```

- As y is set to point to the same objects that x is pointing to, whatever manipulation done to x should be found with y too. Our manipulation mutates the list in-place. I.e. when we remove the first position (2) from the list that x is pointing to when we call y we will also no longer find the 2. y should return the following value [1, 3, 4]. **Check row 1-3 in the code-drawing template please for the sketch of the code.**

```
In [28]: y
```

```
Out[28]: [1, 3, 4]
```

Correct!

Exercise 3

Now sketch this code. Again, use Python to check the documentation for sorted if you aren't sure if it mutates or not.

```
In [29]: x = [1, 4, 3, 2]
         y = x
         sorted(x)
```

```
Out[29]: [1, 2, 3, 4]
```

Sketch out this code, and make a prediction for the value of y. Then run the code in Python to check your answer.

- **Check the associated rows on the template please.** As we can see it's an in-place manipulation and y and x both point to one list. When we sort the shelf x points to we get the same results when we pull the shelf from y. I.e. prediction for y: [1,2,3,4]

In [30]:

```
y
```

Out[30]: [1, 4, 3, 2]

In [31]:

```
x
```

Out[31]: [1, 4, 3, 2]

Incorrect. Running the code showed that the prediction was wrong. We understand this is because the command was `sorted(x)`, displaying x sorted only for that line. When running y without that property, y returns the values of the original list. When we check and run x without the property of `sorted` then it also returns the values unsorted. With `sorted` the shelf is not permanently changed. If however we had used `x.sort()` instead, the command would have changed the list and shelf permanently and this change would have been reflected in y too. Let's check:

In [32]:

```
x.sort()  
y
```

Out[32]: [1, 2, 3, 4]

And this assumption was correct!

Exercise 4

Now sketch out this code, and make a prediction for the value of x and the value of y. Then run Python to check your answer.

If the result surprises you, you can find some discussion of what likely happened here.

In [33]:

```
x = [1, 4, 3, 2]  
y = x  
x = x.sort()
```

- We would assume, as outlined in Exercise 3 that y would now take the value of [1,2,3,4] because the `x.sort()` command applies to the shelf that x and y both point at. x would also take that value of [1,2,3,4]. **See code template.**

In [34]:

`y`Out[34]: `[1, 2, 3, 4]`**Correct!**

In [35]:

`print(x)`

None

Incorrect. What happened here? We reassigned the the result of the method to x. Why is x *none* then?

Exercise 5

One of the thing that's very useful about lists, dictionaries, and sets in Python is that they are recursive, meaning a list can contain a list. While powerful, though, this can also be tricky! Let's draw out the following code:

In [36]:

```
x = [3.14, 42]
z = {'boring numbers': [1, 2, 3],
     'interesting numbers': x}
y = x
y.append(1.618)
```

Now, what is the value of `z['interesting numbers']`?

- the value should be [3.14, 42, 1.618]

In [37]:

`z['interesting numbers']`Out[37]: `[3.14, 42, 1.618]`**Correct!**

Exercise 6

As noted in our tutorial, in general, almost all collections of things in Python are mutable with the exception of tuples. Tuples are, quite simply, immutable lists. Let's play a little with tuples.

```
In [38]: x = [42, -1]
         y = (1, 2, x)
         x.append(-2)
```

Write down what you think will be the values of x and y? Now check your answers in Python.

If you got this wrong, you can find discussion of this result [here](#)

- We would assume x to append to [42, -1, -2]. As y is *recursive*, i.e. includes x which is a reference of the shelf, we expect y to be (1,2,x), i.e. (1,2, [41,-1,-2]). **See code template**

```
In [40]: x
```

```
Out[40]: [42, -1, -2]
```

```
In [39]: y
```

```
Out[39]: (1, 2, [42, -1, -2])
```

Correct!

Exercise 7

```
In [41]: x = {'dog_names': ['fido', 'woofer'],
             'cat_names': ['fluffy', 'bite-y']}
         y = x.copy()
         y['dinosaur_names'] = ['big guy', 'super lizard']
         y['cat_names'].append('tiger')
```

What would you get if you ran `x['dinosaur_names']`?

- We would get no results because there is no reference of the `dinosaur_names` list stored in `x`. **See code template.**

What would you get if you ran `x['cat_names']`?

- We would get `['fluffy', 'bite-y', 'tiger']`. We would get that because we did not do a deep copy but only a regular copy where we only copied the arrow of `cat_names` pointing to the shelf, hence when we append the shelf using `y`, the pointer `cat_names` in `x` finds those values too! **See code template.**

Check your answers with Python.

In [42]:

```
x['dinosaur_names']
```

```
-----  
KeyError                                Traceback (most recent call last)  
/var/folders/h9/g02cmrsn6y571zblv96gs56c0000gn/T/ipykernel_39698/1188399946  
.py in <module>  
----> 1 x['dinosaur_names']
```

```
KeyError: 'dinosaur_names'
```

Correct!

In [43]:

```
x['cat_names']
```






Out[43]: `['fluffy', 'bite-y', 'tiger']`

Correct!


Code-Drawing Template

E
x
e
r
c
i
s
e
2


E
x
e
r
c
i
s
e
3


Code <code>x = [1, 2, 3, 4]</code>	Variables <code>x</code> 	Objects <code>[1, 2, 3, 4]</code>
Code <code>x = y</code>	Variables <code>x, y</code> 	Objects <code>[1, 2, 3, 4]</code>
Code <code>x.pop(1)</code>	Variables <code>x, y</code> 	Objects <code>[1, 3, 4]</code>
Code <code>x = [1, 4, 3, 2]</code>	Variables <code>x</code> 	Objects <code>[1, 4, 3, 2]</code>
Code <code>y = x</code>	Variables <code>x, y</code> 	Objects <code>[1, 4, 3, 2]</code>

Exercise 3

Code	Variables	Objects
<code>sorted(x)</code>	<code>x, y</code> 	<code>[1, 2, 3, 4]</code>


Exercise 4


Code	Variables	Objects
<code>x = [1, 4, 3, 2]</code>	<code>x,</code> 	<code>[1, 4,3,2]</code>


Code	Variables	Objects
<code>y = x</code>	<code>x, y</code> 	<code>[1, 4,3,2]</code>

Code	Variables	Objects
<code>x = x.sort()</code>	<code>x, y</code> 	<code>[1, 2, 3, 4]</code>

Exercise 1

Code	Variables	Objects
<code>animals_in_my_zoo = {'Zebras': ['Stripey', 'Black and White'], 'Polar Bears': 'Fluffy', 'Dinosaurs': 'Bitey'}</code>	<code>animals_in_my_zoo</code> 	<code>'Zebras': ['Stripey', 'Black and White'], 'Polar Bears': 'Fluffy', 'Dinosaurs': 'Bitey'</code>








Code	Variables	Objects
<code>x = animals_in_my_zoo</code>	<code>animals_in_my_zoo, x</code> 	<code>'Zebras': ['Stripey', 'Black and White'], 'Polar Bears': 'Fluffy', 'Dinosaurs': 'Bitey'</code>

Code	Variables	Objects
<code>x['Meerkat'] = 'Peeps'</code>	<code>animals_in_my_zoo, x</code> 	<code>'Zebras': ['Stripey', 'Black and White'], 'Polar Bears': 'Fluffy', 'Dinosaurs': 'Bitey', 'Meerkat': 'Peeps'</code>

[]:

Code-Drawing Template

E
x
e
r
c
i
s
e
6

Code	Variables	Objects
<code>x = [42, -1]</code>	x 	[42, -1]
Code	Variables	Objects
<code>y = (1, 2, x)</code>	x,  y 	[42, -1]  (1, 2, x)
Code	Variables	Objects
<code>x.append(-2)</code>	x,  y 	[42, -1, -2]  (1, 2, x)
Code	Variables	Objects
Code	Variables	Objects

Exercise 7

Code	Variables	Objects
<pre>x = {'dog_names': ['fido', 'woofer'], 'cat_names': ['fluffy', 'bite-y']}</pre>	x	<pre>{'dog_names': ['fido', 'woofer'], 'cat_names': ['fluffy', 'bite-y']}</pre>
Code	Variables	Objects
<pre>y = x.copy()</pre>	x, y	<pre>{'dog_names': , 'cat_names' }</pre> <pre>{'dog_names' , 'cat_names' }</pre>
Code	Variables	Objects
<pre>y['dinosaur_names'] = ['big guy', 'super lizard']</pre>	x, y	<pre>{'dog_names': , 'cat_names' }</pre> <pre>{'dog_names' , 'cat_names' , 'dinosaur_names': ['big guy', 'super lizard']}</pre>
Code	Variables	Objects
<pre>y['cat_names'].append('tiger')</pre>	x, y	<pre>{'dog_names': , 'cat_names' }</pre> <pre>{'dog_names' , 'cat_names' , 'dinosaur_names': ['big guy', 'super lizard']}</pre>
Code	Variables	Objects
Code	Variables	Objects
Code	Variables	Objects

[]:

