Lab 5 Manual

Lab 5 Outline

Part 1: Experimenting with Python [35 mins]

For some of the exercises, it may be useful to copy the code into pythontutor.com to visualize the execution.

- *Stack frame tracing*
- *Updating list in place*
- *Global vs local variables*

Part 2: Quiz [15 mins]

Now that you have completed the lab, you will take a short mandatory quiz on the lab material. To take the quiz, **your lab tutors will provide you with the passcode to access the Lab Quiz on your Ed account**. This will be a **timed (15 mins)**, multiple choice quiz on the lab material. If you completed the lab, you will be able to answer all of the questions on the quiz.

You are required to take the quiz during your lab session. Make sure to leave enough time to take the quiz before the end of the lab session.

Stack frame tracing

Please first read through the following five function definitions:

```
1  def days_to_hours(days):
2      hours = days * 24
3      return hours
4
5  def hours_to_minutes(hours):
6      minutes = hours * 60
7      return minutes
8
9  def minutes_to_seconds(minutes):
10     seconds = minutes * 60
11     return seconds
12
13 def hours_to_seconds(hours):
14     minutes = hours_to_minutes(hours)
15     seconds = minutes_to_seconds(minutes)
16     return seconds
17
18 def days_to_seconds(days):
19     hours = days_to_hours(days)
20     seconds = hours_to_seconds(hours)
21     return seconds
22
23 hours_to_seconds(20)
24 days_to_seconds(6)
```

This code snippet defines five functions that convert different time units into one another. Now think about the following questions:

- How many stack frames are created when you execute `hours_to_seconds(20)`?
- How many stack frames are created when you execute `days_to_seconds(6)`?
- How many frames (including global frame) are created in total for the above code snippet?

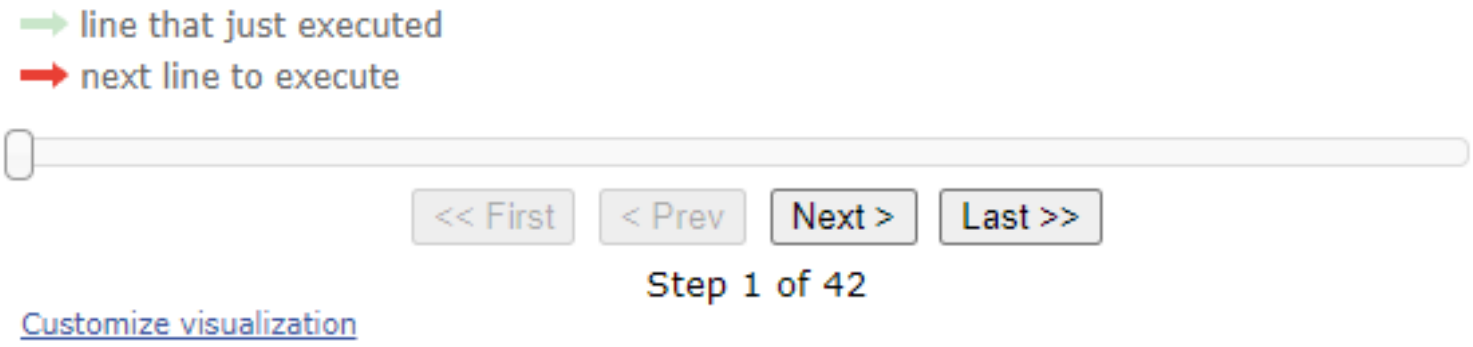
Discuss these questions with your partner.

Note: A stack frame in Python is created whenever you execute a function call. In addition, there is always a "global frame" that manages the global execution of a Python module (i.e. a Python program).

Stack Frame Visualization

Please go to pythontutor.com, and click on "Start visualizing your code now". **Type** the above code snippet (including the two function calls at the end) into the editor.

Then click "Visualize Execution" to start, and there should be a total of 42 steps:



You can click "Next" to execute the next line of code. **Observe when a stack frame is created, and count how many frames are created along the whole execution. Does the answer match your expectation?**

If you are stuck, feel free to discuss with your lab partner or ask questions to your lab tutors!

Updating list in place

Consider the following code, and try to trace it manually first. What do you think will be printed? Now, create a new file called `update_list.py`. Copy and paste the code into that file and run the file.

```
# update_list function squares every element in a list in place
def update_list(nums):
    for i in range(len(nums)):
        nums[i] = nums[i]**2

list1 = [0,1,2,3]
list2 = [0,1,2,3]
list3 = list1

print("before update")
print("list1:", list1)
print("list2:", list2)
print("list3:", list3)

# only call update_list on list1
update_list(list1)

print("after update")
print("list1:", list1)
print("list2:", list2)
print("list3:", list3)
```

Did `list2` and `list3` change after calling the `update_list` function on `list1`? Why or why not?

What does this tell you about references?

What does this tell you about object references in function calls?

Discuss the above questions with your partner.

Global vs local variables

Consider the following code. Trace it manually and think about the questions below.

```
num = 10

def multiply_local(number,multiple):
    num = number * multiple
    print("num inside function: " + str(num))

def multiply_global(number,multiple):
    # "global" keyword declares a variable as global
    global num
    num = number * multiple
    print("num inside function: " + str(num))

multiply_local(num,4)
print("num after multiple_local: " + str(num))
multiply_global(num,4)
print("num after multiple_glocal: " + str(num))
```

- How many global variables are in the program? What are their names? What are their values?
- How many local variables are in the program? What are their names? What are their values?
- Which names in the program are used as parameter names?
- What do you expect `num` to be after each function call?

Now copy and paste this code into `multiply.py` and run it. Did it match your prediction?

If you are unsure about any of these questions, feel free to discuss amongst yourselves or ask your lab tutors.