CPS 109 - Lab 5

Agenda

1. Midterm Prep / Q&A
2. A1 FAQ / Q&A
3. Recursion
4. Mutability (review)
5. Quiz (shorter/easier than usual)

Midterm Preliminary Advice

- A quick reminder: your midterm is worth **25% of your final mark.** This is more than both of your assignments put together
- Your midterm will be held in your lab section during your lab time. **Ensure you attend the lab you're officially signed up for!**
- We'll go through a tutorial to familiarize you with the Ubuntu OS, as well as some FAQ
- Good preparation might include:
  - Doing the practice midterm (mandatory)
  - Reading over the lecture slides
  - Trying out the CodingBat problems
  - Doing the other sections' lab quizzes (extra practice)
  - https://github.com/ChrisKolios/CPS109_Fall2022/tree/master/Review/AlexMidtermQuestionBank (even more practice)

Midterm Prep 1 (Logging In)

Firstly, ensure that you're logged in to the Ubuntu partition on the lab computers.

- If your computer is currently in Windows mode, you can press the power button on the bottom right of the monitor to restart/shut your computer off. Press the power button again after a few seconds, and as your computer boots back up and you see a blue screen, press the Up arrow key to select the Linux/Ubuntu partition then hit Enter. It may take a few minutes to boot. Enter your CS lab credentials.
- **CRITICAL:** If you cannot log in via Linux let us know!!

Midterm Prep 2 (Opening IDE)

- Once you're in, the next thing you'll need to do is open your IDE of choice. Let's use Spyder for the sake of this example (though VSCode is good too). To open Spyder, you'll hit the button in the top left corner of the screen, then navigate to the "Programming" directory, then click "Spyder". It should launch.

Midterm Prep 3 (Saving to Folder)

- Now that you've got your IDE of choice open (e.g. Spyder), create a simple script. Type the following code:

```
print("Hello World!")
my_input = input("What should I print?: ")
print(my_input)
```

- To save your file to a specific folder, you can click the "Save As" button (or "Ctrl+Shift+S" on your keyboard)
- Navigate to the root folder. This folder will have your name on it. Changes you make to this folder will persist across multiple login sessions. You can create a new subdirectory, or save your file directly to your root. Do so.

Midterm Prep 4 (Running Code)

- Once you have saved your file, you can run it via the IDE. Give it a run to ensure that everything is working correctly.
- To run a file in Spyder, you can click the big Play button above the code, or use the keyboard shortcut “F5”
- You should see “Hello World” in the console to the right, and after you type in some input within the console your program should print it back to you.

Midterm FAQ 1

1. DO THE PRACTICE MIDTERM
   a. Note that this counts as your lab5 submission for this week, and it is fantastic practice for the midterm. It will show you what kinds of questions to expect, and the format they'll be in.
2. What will the format of the midterm exam be like?
   a. See 1. There will be multiple choice questions and code implementation questions. From your CMF:

Evaluation Guidelines	Notes: Midterm tests and final exam will consist of questions presented on D2L (multiple-choice questions, short answer questions and questions requiring code development). **The midterm tests are tentatively planned in the lab time during the week of October 24-28, 2022.** Part of the evaluation may be a brief oral defense of your programs.

3. Will I have access to the internet during the quiz?
   a. No, with the exception of D2L. I believe this is either monitored/proctored or software-enforced.
4. Will I have to submit my code?
   b. Yes, you will submit both the quiz and your code to D2L.

Midterm FAQ 2

1. What are the contents of the midterm?
   a. There will be an announcement on this soon. The first four chapters of the textbook. No recursion.
2. Am I allowed to use Python's help() function?
   a. Yes! We asked one of the Professors this and they said there would not be an issue with it. Note that this can be really helpful if you're in a bind and want to know how something works.
3. Are there any limitations to functions I'm allowed to use?
   a. No! (With the exception of a few select questions that might specify not to use certain methods (as they would trivialize the question))
4. What format will I submit my code as?
   a. You will submit a .py. You'll get more detailed instructions regarding your submission on your midterm.

A1 FAQ

1. Am I allowed to use external libraries?
   a. NO. You are allowed to import any default libraries/packages that come bundled with Python (e.g. math), but no external libraries (e.g. numpy)
2. When is it due again and how much is it worth?
   a. Check your CMF and D2L… But it's due Sunday October 23rd at 11:59 pm and it is worth 10% of your final grade.
3. If my program has multiple outputs (e.g. for different user inputs), what should my output.txt contain?
   a. You should compose your output.txt with multiple runs through your code if you have different use cases. Make sure it's all in one file and separate them with "-----" or something similar.
4. What does 2. To write statements using the boolean primitive data types. mean?
   a. You should have an explicit True/False in a statement somewhere in your code. E.g. if (x == True). Do not make it implicit (e.g. if (x == 3))!
5. Who should I contact if I require accommodation?
   a. Your professor. You can find their email via the CMF. We're not authorized to provide extensions / etc. Feel free to email us with questions though.

Content: Recursive Functions

Last lab was functions. This lab, let's talk about *recursive functions*.

I know some of you have encountered "classes" with testing. We will go over that next week.

Recursive Functions

What's a recursive function? Well, first let's talk about what it means to be recursive.

For something to be recursive, it has to reference itself in order to work.

Recursive Functions

In all seriousness, for something to be recursive, it has to call on itself in order to work.

Recursive Definitions

Let's look at an example from an assignment I was given last year:

(60 marks) Let w be a string of brackets { and }. Then w is called balanced if it satisfies the following recursive definition:

- w is the empty string, or
- w is of the form { x } where x is a balanced string, or
- w is of the form xy where x and y are balanced strings.

Recursive Functions

So what does this mean to you? Well, we can use this concept to write functions!

Recursive Functions

Why would this ever be useful? Well, there are some instances where recursion is the only way to solve a problem.

Can any of you think of a problem that may require recursive functionality?

Base Cases

First, however, we need to talk about base cases. These are super important for designing recursive functions, since they keep your program from recursing indefinity.

Base Cases

What is a base case?

Simply put: it’s the default behaviour you need to account for when writing a recursive function. Base cases are reserved for when the function’s input is very simple, like 1.

Recursive Functions

One application that you guys have already gone over is the Fibonacci sequence. Another one we can approach with recursion is factorials.

Recursive Functions

```
def factorial(num):
    if(num < 1):
        return 1
    else:
        new_num = num * factorial(num - 1)
        return new_num
```

A Word On Mutability

I’m sure that by now, you’ve heard one of us mention that lists are “mutable” and strings are “immutable”. What do these mean?

A Word On Mutability

Something is **mutable** if its contents can be changed (like a list).

Something is **immutable** if its contents cannot be changed (like a string or a tuple).

But what about str.replace()?

You'll notice that for string manipulation methods (.replace(), .lower(), etc.) the method returns a new string.

You need to overwrite your old string with the new string returned by the method.

Mutability Example 2

```python
# Additional Mutability Example

if __name__ == "__main__":

    teenage_mutable_ninja_turtles = ["Raphael", "Donatello", "Leonardo", "Michelangelo"] # Remember, lists are mutable - strings are not
    teenage_mutable_ninja_turtles.append("Splinter") # We can add new elements
    teenage_mutable_ninja_turtles.pop() # We can also remove elements (Shoo Splinter, you aren't a turtle!)

    for immutable_turtle in teenage_mutable_ninja_turtles:
        immutable_turtle = "B" + immutable_turtle[1:]
    print(teenage_mutable_ninja_turtles) # Notice how it doesn't change. This is due to how for loops work (we're modifying the
                                         # immutable_turtle variable, which is local to the for loop).

    for i in range(len(teenage_mutable_ninja_turtles)):
        teenage_mutable_ninja_turtles[i] = "B" + teenage_mutable_ninja_turtles[i][1:]
    print(teenage_mutable_ninja_turtles) # Notice how it does change, but we need an equals sign to do so (we are making a new string)

    #teenage_mutable_ninja_turtles[0][0] = "R" # We can't change strings like this!
    teenage_mutable_ninja_turtles[0] = "Baphael!" # But we can change list items to whole new strings

    teenage_mutable_ninja_turtles[0].replace("B", "R", 1)
    print(teenage_mutable_ninja_turtles[0]) # Notice how it doesn't change
    teenage_mutable_ninja_turtles[0] = teenage_mutable_ninja_turtles[0].replace("B", "R", 1)
    print(teenage_mutable_ninja_turtles[0]) # Notice how it does change, but we are setting it to the new string returned by the .replace() method
```

```
['Raphael', 'Donatello', 'Leonardo', 'Michelangelo']
['Baphael', 'Bonatello', 'Beonardo', 'Bichelangelo']
Baphael!
Raphael!
```