CPS 109 - Lab 2

Agenda

1 - Housekeeping (will likely bypass)
2 - Recap of Last Week
3 - Operators
4 - Lists
5 - Loops

Reminders:

Your default username: Your my.ryerson **Short ID**
Your default password: MMDDXXXX (birth month, birth day, last 4 digits of your student #)

Our Contact Information:
  Chris: ckolios@ryerson.ca
  Saghar: saghar.jiantavana@ryerson.ca
Course GitHub (this section):
  https://github.com/ChrisKolios/CPS109_Fall2022

Reminder 2

Every week, in addition to in-person labs (where you do your quizzes), there is also a lab assignment to submit

This week you'll have to submit a pdf of a screenshot from Codingbat Logic-1 and Logic-2

Housekeeping (optional)

We're going to briefly walk you through forwarding .cs emails to your regular Ryerson email (please follow along).

Step 1: Open cmd.exe (Windows) or Terminal (Mac/Linux)
Step 2: Type: ssh username@moon.scs.ryerson.ca (username is your Ryerson CS username)
Step 3: Enter your password (remember it won't show)
Step 4: Type: nano .forward
Step 5: Type: username@ryerson.ca
Step 6: Input: Ctrl + o (to save), then Enter (to confirm)
Step 7: Input: Ctrl + x to exit
Step 8: Type: chmod 600 .forward
Step 9: Exit via Ctrl + d or Type: exit

Recap of Last Week

Remember last week we went over a lot of new stuff! Not remembering all of it is completely fine. As you gain more experience / code more these things will sink in.

I would recommend rereading if/elif/else

Some New Syntax

Let's talk about this statement:

```
5   if __name__ == "__main__":
```

What is this line, exactly? In short, this tells the computer what code to execute when the main file in a program is run. Note: two underscores

General Program Structure

Whenever you are writing a Python program in the future, you'll want to have the following structure:

Function definitions / their logic

```
def f1():
   return 0 # or whatever
def f2(y):
   …

if __name__ == "__main__":
   Any variable declarations / non-function logic / calls to functions etc.
   x = 5
   print(f2(x))
   …
```

Operators

Operators are, in essence, the backbone of programming. They’re instructions that the computer recognizes and uses to manipulate or check data.

Anything from boolean comparisons, to mathematical operations, to comparisons.

Common Operator Examples

Boolean Comparisons:

and, or, not

Mathematical Operations:

+, -, *, /, //, **, %

Comparators:

Greater than / or equal to	> / >=
Less than / or equal to	< / <=
Equal to / not equal to	== / !=

Lists

A list is simply a collection of data/variables/objects denoted with a set of [] and separated by commas, like so:

```
this_list = ["this", "is", "our", "list"]
```

Lists Cont.

Let’s run through a quick example (which is posted on the Github as: ListCrashCourse.py) to hopefully answer some questions you may have regarding lists.

Introduction to Loops

Do you want to repeat a bunch of code until you achieve something specific? Maybe until you reach a certain number of iterations or you find what you're looking for in a list?

You need a loop for that.

Introduction to Loops

What is a loop?

A loop is a certain set of instructions (syntax) that the computer knows to repeat until a condition is met.

Kinds of loops: while, for, nested

Loops: For

A for loop will repeat a specific set of instructions (written by you) *for* the amount of time/number of iterations specified.

Simply put: the computer will do something x amount of times.

Loops: For

```
1    for i in range(10):
2        print(f'loop number {i}')
3
4
```

Notice the range function starts iteration at 0 and ends at 9 (not inclusive on stopping point)

Output:

```
loop number 0
loop number 1
loop number 2
loop number 3
loop number 4
loop number 5
loop number 6
loop number 7
loop number 8
loop number 9
```

Loops: For

```
2  teletubbies = ["Tinky Winky", "Dipsy", "Laa-Laa", "Po"]
3  # Do kids nowadays know what the teletubbies are?
4
5  for creature in teletubbies:
6      print(creature + " terrifies me...")
```

You can also use for loops to iterate over a list, or the characters in a string.

```
Tinky Winky terrifies me...
Dipsy terrifies me...
Laa-Laa terrifies me...
Po terrifies me...
[Finished in 54ms]
```

Examples of for loops

```
def for_list_int(list_int):
    print("For loop through list of integers and print each element")
    for elem in list_int:
        print(f'elem is : {elem}')
```

```
for_list_int([1,2,3])
```

<- the function is called and
a list is passed through

Output:

```
For loop through list of integers and print each element
elem is : 1
elem is : 2
elem is : 3
```

Examples of for loops

```
def for_list_str(my_str):
    print("For loop through string and print each character")
    for i in my_str:
        print(f'character is : {i}')
```

```
for_list_str("Bob ?")
```

Output:

```
For loop through string and print each character
character is : B
character is : o
character is : b
character is :
character is : ?
```

Examples of for loops

```
def for_loop(my_list):
    print("For loop through list printing index and element")
    for i in range(len(my_list)):
        print(f'index {i} is: {my_list[i]}')
my_list=['Google','Bing','yahoo','You']
for_loop(my_list)
```

Output:

```
For loop through list printing index and element
index 0 is: Google
index 1 is: Bing
index 2 is: yahoo
index 3 is: You
```

Examples of for loops

```
def for_loop_r(my_list):
    print("For loop reversing index through list")
    for i in range(len(my_list)):
        print(f'index {len(my_list)-1-i} is: {my_list[len(my_list)-1-i]}')
my_list=['Google','Bing','yahoo','You']
for_loop_r(my_list)
```

Output:

```
For loop reversing index through list
index 3 is: You
index 2 is: yahoo
index 1 is: Bing
index 0 is: Google
```

Loops: While

A while loop will follow a set of instructions *while* its boolean condition holds true.

Simply put: the computer will do x instructions until y is false.

Loops: While

```
 9  if __name__ == "__main__":
10      x = 0
11
12      while x < 10:
13          print(x)
14          x+=1
```

Output:

```
0
1
2
3
4
5
6
7
8
9
```

Examples of while loops

```
def while_loop(num):
    print("While loop")
    i=0
    while(i<num):
        print(f'i is {i}')
        i+=1

while_loop(5)
```

Output:

```
While loop
i is 0
i is 1
i is 2
i is 3
i is 4
```

Examples of while loops

```
def while_loop2(time):
    while time >= 0:
        print(f'time until toby can eat: {time}')
        time -= 1
    print('yummy!')

while_loop2(10)
```

Output:

```
time until toby can eat: 10
time until toby can eat: 9
time until toby can eat: 8
time until toby can eat: 7
time until toby can eat: 6
time until toby can eat: 5
time until toby can eat: 4
time until toby can eat: 3
time until toby can eat: 2
time until toby can eat: 1
time until toby can eat: 0
yummy!
```

Loops: Nested

Nesting a loop is just a fancy way of saying putting a loop inside of another loop.

You can do this as many times as you like as long as you don’t care about your computer catching fire.

Loops: Nested

You can put a for loop within a for loop, a while loop within a while loop.

You can also get really spicy and put a for loop in a while loop, or a while loop in a for loop.

Example nested for loop

```
def nested_for(my_list):
    print("Nested for loop")
    for word in my_list:
        print(f'word is: {word}')
        for letter in word:
            print(f'letter is: {letter}')
my_list=['Google','Bing','yahoo','You']
nested_for(my_list)
```

Output:

```
Nested for loop
word is: Google
letter is: G
letter is: o
letter is: o
letter is: g
letter is: l
letter is: e
word is: Bing
letter is: B
letter is: i
letter is: n
letter is: g
word is: yahoo
letter is: y
letter is: a
letter is: h
letter is: o
letter is: o
word is: You
letter is: Y
letter is: o
letter is: u
```