COMP 364: Functions I

Carlos G. Oliver, Christopher Cameron

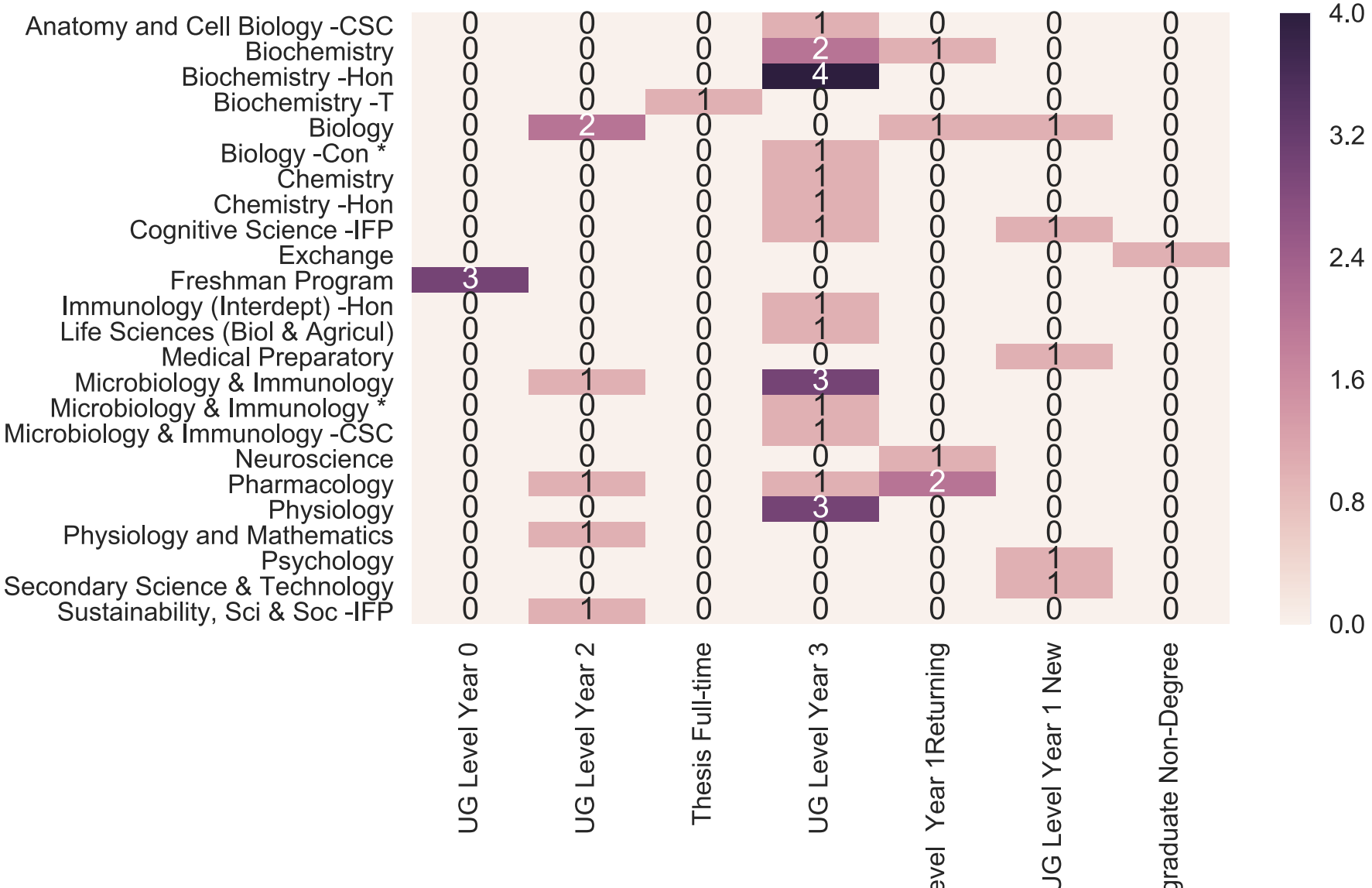
September 25, 2017

Outline

1. Tentative Final Exam Date: December 12, 2017
2. Assignment Topics
3. Recap + Warmup
4. Functions Theory
5. Functions practice: tic tac toe.

Assignment Topics

We have a diverse group so suggest assignment topics!



Recap

- Names, Objects, Namespace: store data and give it a name (Python party)
- Conditional statments: control the execution of your program
- Lists, Tuples, Sets: containers for multiple objects
- Loops: repeat operations

Warm-up

1. `conditionals:` Tinder ® hired you to add an "age compatibility" feature to its app. They want you to use the classic unwritten rule: *if your age is x and the person you are intrested is y years old, it's weird unless* [1]

$$\frac{x}{2} + 7 < y < 2(x - 7) \tag{1}$$

   Write a program that: 1) asks the user for their age and their match's age. 2) prints to the screen whether the match is ok, too old, or too young.

2. `loops:` Write a program that prints the sum of every third number in a list. 1) only use a simple `for` loop over the list. 2) use the `range()` function.

[1] https://xkcd.com/314/

We've been dying to tell you more about functions.

Functions make programming a lot easier and a lot more fun.

[2]

[2] http://taylormarshall.com/2015/09/catholic-video-how-to-refute-mormons-in-4-points.html

Why functions?

- Let's say I come up with some super useful code that can get the minimum in a list. And I store it in `minlist.py`.

```
1 my_list = [10, 200, 1, -1, 20, 500]
2 current_min = my_list[0]
3 for val in my_list:
4     if val < current_min:
5         current_min = val
6 print(f"Smallest value is {current_min}.")
```

Life Hack 1

Does line 6 look weird? It's called an `f-string` which is super useful. Read more here

Why functions?

- What if I need the minimum of 2 lists?

```
1  my_list_one = [10, 200, 1, -1, 20, 500]
2  my_list_two = [100, -20, 100, 320, 11, 103]
3
4  current_min = my_list_one[0]
5  for val in my_list_one:
6      if val < current_min:
7          current_min = val
8  print(f"Smallest value is {current_min}.")
9  current_min = my_list_two[0]
10 for val in my_list_two:
11     if val < current_min:
12         current_min = val
13 print(f"Smallest value is {current_min}.")
```

- That was ugly...

Functions to the rescue!

- Functions let us store and re-use code.
- We can use the key word `def` to define our own functions.
- Once the function is defined, just call it using its name and its code will execute.
- Note: without a call, the function's code will not run.

```
1  def my_min(lala):
2      current_min = lala[0]
3      for val in lala:
4          if val < current_min:
5              current_min = val
6      print(f"Smallest value is {current_min}.")
7
8  my_list_one = [10, 200, 1, -1, 20, 500]
9  my_list_two = [100, -20, 100, 320, 11, 103]
10
11 my_min(my_list_one)
12 my_min(my_list_two)
```

The anatomy of a function

- Function header
  1. `def` tells Python you are defining a function
  2. `function_name`. Functions are objects so we give them names
  3. `(function_arguments)` Objects you would like the function to work on (optional)
- Function body
  - Any code that is tabbed at least once and follows the header is stored in the function.

```
1 #header
2 def my_function_name(argument_1, argument_2, ...,):
3     #function body, some code
4     ...
5     ...
6 #outside function body
```

Exercise: Drawing Flags

Write a function called `flag(n, m, c)` that prints an `n` rows by `m` columns rectangle that looks like a flag using string `c`.

```
1  def flag(n, m, c):
2      ....
3      ....
4  flag(5, 30, "*")
5  **********              **********
6  **********              **********
7  **********              **********
8  **********              **********
9  **********              **********
```

The `return` statement

- The `return` statement is a special word that lets the function "spit out" an object i.e. output.
- This is useful because it lets the person who called the function store the output in memory and perform operations with it later on.
- `return` is NOT the same as print()
- When Python reaches a `return` statement it automatically exits the function.

```
1 def square(x):
2     return x*x
3     print("Hellooo") #this is  never reached
4 squared = square(5)
```

The `return` statement

- A function body can have multiple `return` statements.
- Exercise: write a function that **returns** `True` if it is given a prime number and `False` otherwise.

```
1  def is_prime(x):
2      for t in range(2, x):
3          if x % t == 0:
4              return False
5      return True
6  num = 15
7  result = is_prime(num)
8  print(result)
```

The `import` statement

- `import` statements let you include code written by other people into your program. This is incredibly powerful.
- You don't have to reinvent the wheel each time you code. If someone else wrote a function that does what you need, just import it.

```
1 import antigravity
2 import this
```

The `import` statement

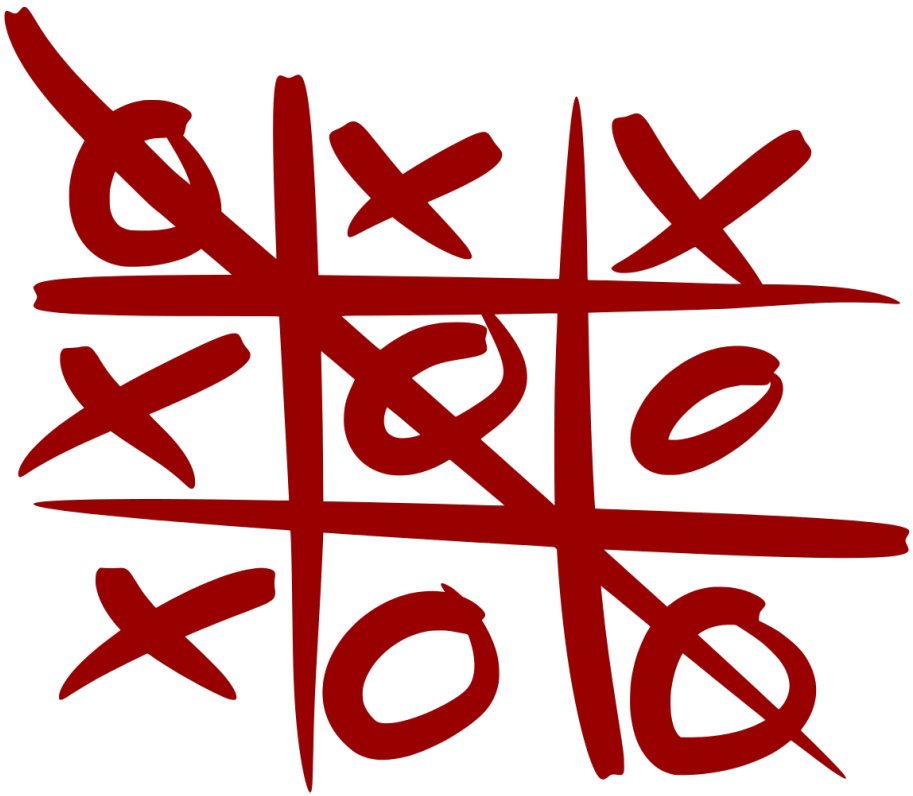
- At the top of your code say `import [package name]`

```
1 import numpy as np
2 gpa = [2.1, 4.0, 3.0, 3.3]
3 class_avg = np.mean(gpa)
4 class_std = np.std(gpa)
```

- `numpy` is a **package** (a collection of python code) that we store in our code as an object and give it the name `np` (optional)
- Once it's imported we can use its attributes just like any other object.
- Don't reinvent the wheel.

Practice: `tictactoe.py`

- Write a program that lets two people play tic tac toe.
- The program prompts the user for the index of the row and column where they would like to place their `X` or `O`
- After each turn, it prints the board and declares as winner if there is one.



[3]

[3] Wikipedia