**Python**

**Exercises**

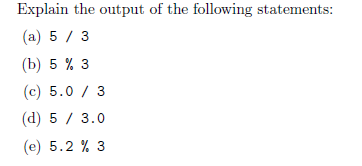
The code must be submitted under your name in GitHub in a repository called Python. Work individually.

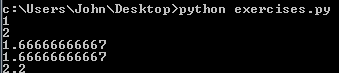
Each file will have the name: exerciseX.py where X is the exercise number. You will have 13 files at most.

Do not commit code that does not compile. The code that you commit should have been tested. -10 points for code that does not compile on the top of your grade.

You will provide a hardcopy with your code to Dr. Scharff on 12/6.

**Exercise 1**





(a) 1

int divided by int. 3 goes into 5 1.666666… times, however as both numbers are ints, the decimal is truncated. Therefore 1 is the output.

(b) 2

int modulo int. Modulo displays the remainder when the two numbers are divided. When 5/3, there are two left over. Therefore two is the output.

(c) 1.66666666667

float divided int. 5.0/3 is 1.66666… therefore since one of the numbers is float, the answer is outputted as float.

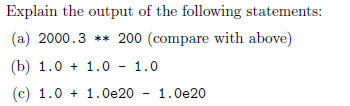
(d) 1.66666666667

Int divided by float. 5/3.0 is 1.66666… therefore since one of the numbers is a float, the answer is outputted as a float.

(e) 2.2

float modulo int. remainder of 5.2 / 3 is 2.2. Since one of the numbers is a float, answer is stored and outputted as float.

**Exercise 2**







(a) OverflowError: <34, ‘Result too large’>

2000.3 is 2000.3 to the 200th power, and cannot be held within a float

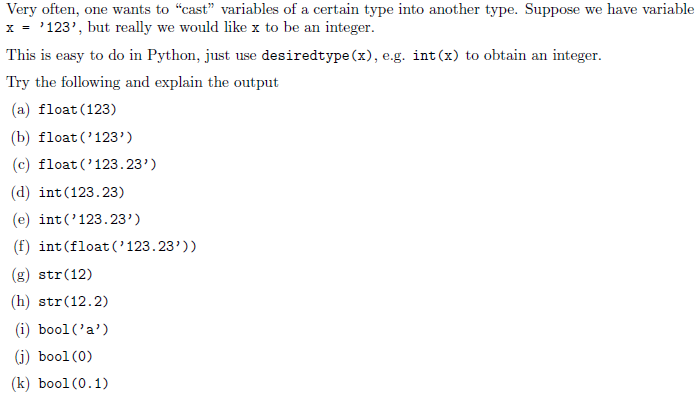
(b) 1.0

1.0+ 1.0 = 2.0, then 2.0-1.0= 1.0. The problem is read from left to right, and then outputted.

(c) 0.0

After playing with this problem, I see that whatever I change the first 1.0 to, the answer is always 0.0. Since large numbers are being used, (1.0e20) the mathematics behind it get misinterpeted. According to resources online, floats store values as approximations, then later scale with the exponent. In this problem, these values are too large, causing the incorrect output.

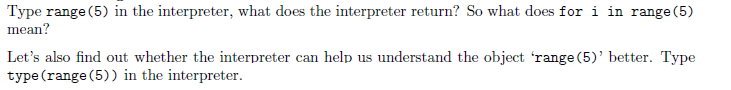
**Exercise 3**



(a) 123 is an int, but is casted as a float, therefore outputted as 123.0  
(b) a literal ‘123’ is casted as a float, outputted as 123.0  
(c) a literal ‘123.23’ is casted as a float, and outputted the same, 123.23  
(d) 123.23 is a float, but casted as an int, therefore the decimal is truncated.  
(e) Commented out in photo below due to error-ValueError: invalid literal for int() with base 10  
 since the literal (a type of string) is a float, and casted as an int, an error is thrown as the value cannot be manipulated.  
(f) In this case, the literal is converted to a float, then to an int. This bypasses the error seen in the previous example. Python cannot skip conversions. in (e) it tried to take a literal (resembling a float), and make it a number, and also an int. In this example, it goes from a literal to a float, then a second cast is called, which converts that to an int, resulting in the output, 123.  
(g) The int 12 is converted to a string, ‘12’, and outputted as such.  
(h) Similar to above, the float 12.2 is converted to a string, ‘12.2’, and thus outputted.  
(i) After toying with the bool() cast, it appears that any value besides 0 returns true. therefore, (‘a’) returns true  
(j) As mentioned before, since the value in bool() is 0, false is returned.  
(k) 0.1 is not 0, therefore it returns true



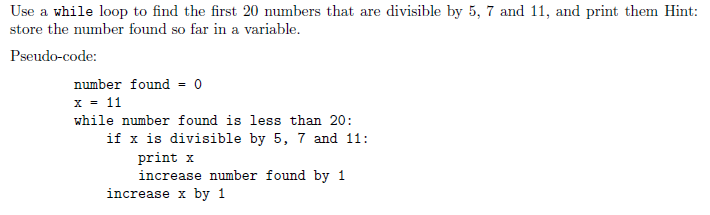
**Exercise 4**



****

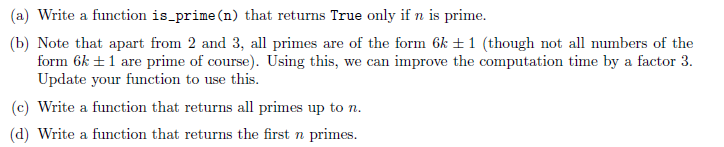
range(5) generates a list for 5 incrementations, starting at 0. i is outputted, then incremented by 1, outputted, etc.   
Evidence that it is a list is displayed when we use type(). <type ‘list’> is outputted.

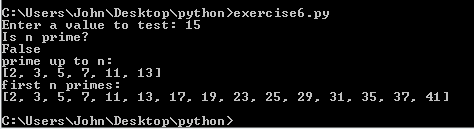
**Exercise 5**



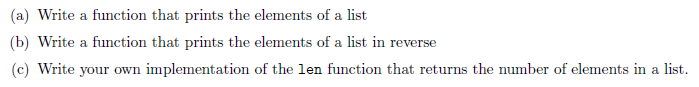


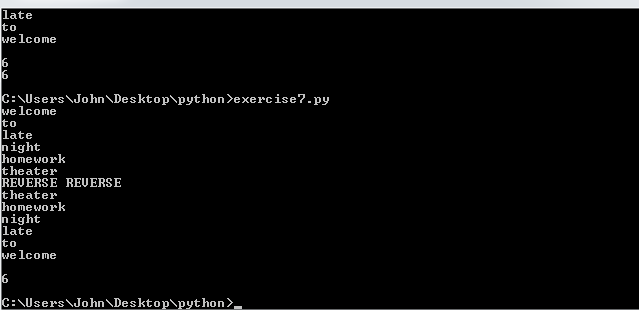
**Exercise 6**





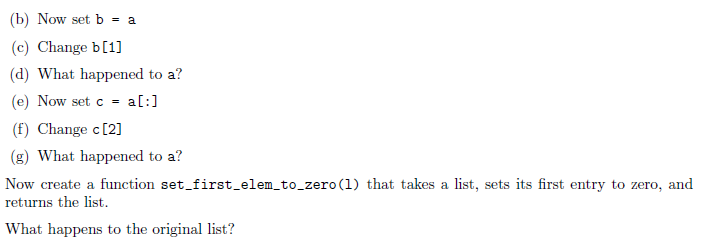
**Exercise 7**



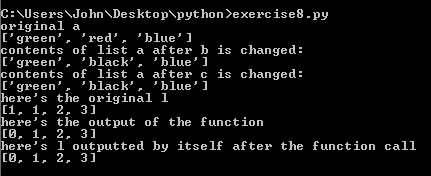


**Exercise 8**

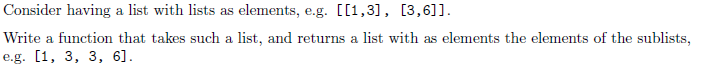


(d) b[1] is simply a pointer to a[1]. Therefore, when b[1] is changed, it points back to a[1], where the value is then changed.  
(g) a is left unchanged. a[:] “slices” the list, or copies it, over to c. Now, if c is changed, a is left unscafthed.

similar to what happened in (d), the function points to the original list, so when the list is changed in the function, it is changed in the original. if it was necessary to avoid this, you would simply use the slice feature and copy it over like so list= l[:]



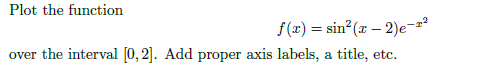
**Exercise 9**



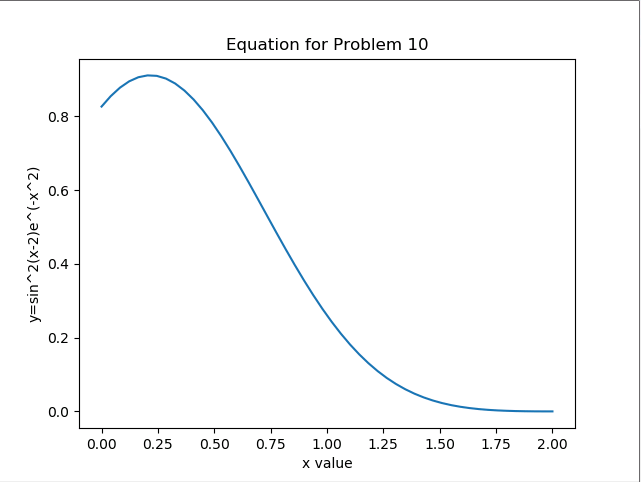


**Exercise 10**

Use mathplotlib

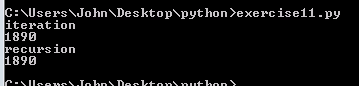


(graphed online for reference)

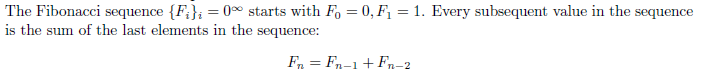


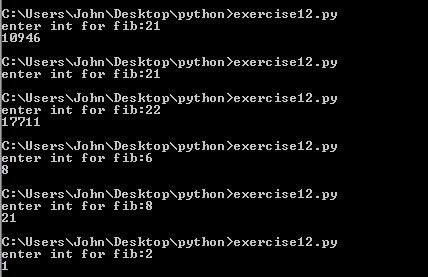
**Exercise 11**





**Exercise 12**





**Exercise 13**

Write a Python program that extracts the email addresses of a file. An email file emails.txt is provided to test your program.

<http://rubular.com/> is a site that can be useful to get familiar with regular expressions.



-see code for notes on this issue

**References**

Stanford courses on Python <https://web.stanford.edu/~schmit/cme193/exercises.html>