**Python**

**Exercises**

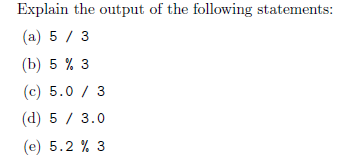
The code must be submitted under your name in GitHub in a repository called Python. Work individually.

Create one file with all your work and name it: cs361python.py or cs 631python.py.

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

You will provide a hardcopy with your code to Dr. Scharff on 12/17.

**Exercise 1**



A) Returns 1.6666666666666667 in Python 3. In Python 2 it would return 1. In Python 3, division always returns a float, while Python 2, dividing two ints would return an int, to mimic this behavior in Python 3, you would use // which is floored division.

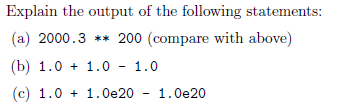
B) Returns 2. This is 5 mod 3, which is 2.

C) Returns 1.6666666666666667 in both Python 2 and 3. Since this is a float divided by an int, it returns a float in python 2. In Python 3, division always returns a float.

D) Returns 1.6666666666666667 in both Python 2 and 3. Since this is a float divided by an int, it returns a float in python 2. In Python 3, division always returns a float.

E) Returns 2.2. This is 5.2 mod 3, 5.2 divided by 3 creates a remainder of 2.2.

**Exercise 2**

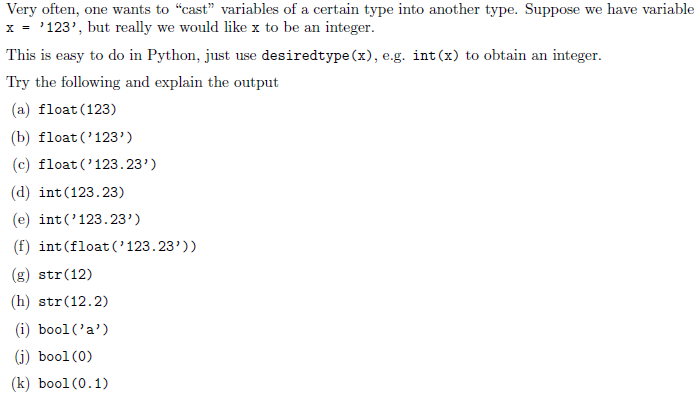


A) Causes an overflow. Floats in python do not have arbitrary precision (infinite length) like ints do. Importing the decimal library will allow floats with arbitrary precision.

B) Returns 1.0, adding and subtracting floats will return a float.

C) This returns 0. Because it first evaluates 1.0 + 1.0e20, which returns 1.0e20 because of how it is rounded. It then does 1.0e20 - 1.0e20 which is 0.0.

**Exercise 3**



A) Converts the int 123, into float: 123.0.

B) Converts the string '123' into the numeric float value: 123.0.

C) Converts the string '123.23' into the numeric float value: 123.23.

D) Since it is the float, the decimal is ignored, which returns 123.

E) Will return a value error. There is no direct integer that is equivalent to '123.23', the string '123.23' must first be converted to a float, and then to an int.

F) First converts the string '123.23' to the float 123.23, then int's that float, so the decimal is ignored. Returns 123.

G) Turns the integer 12, into the string '12'.

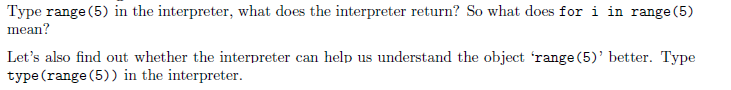
H) Turns the float 12.2, into the string '12.2'

I) All values except for 0 and empty data structures are equivalent to a boolean True.

J) 0 is equivalent to a boolean False.

K) All values except for 0 and empty data structures are equivalent to a boolean True.

**Exercise 4**



range(5)

#Returns range(0,5)

'''for i in range(5) means to iterate through the range(0,5) including 0, but not including 5.'''

for i in range(5):

print(i)

#The above will print:

#0

#1

#2

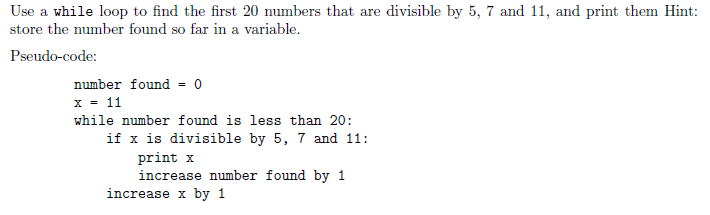
#3

#4

type(range(5))

#This is an object of type "range"

**Exercise 5**



counter = 0

x = 5 \* 7 \* 11 #This will be the first number that is divisible by all 3 numbers

while (counter < 20):

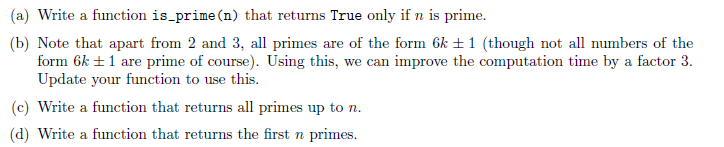
if (x % 5 == 0 and x % 7 == 0 and x % 11 == 0):

print(x)

counter = counter + 1

x += 1

**Exercise 6**



**A**

def is\_prime(x):

if (x == 2):

return True

elif (x % 2 == 0):

return False

elif (x > 2):

for n in range(3, math.floor(math.sqrt(x))+ 1, 2):

if (x % n == 0):

return False

else:

return True

else:

return False

**C**

def sieve\_of\_eratosthenes(lim):

if lim < 0:

print("Error")

Primes = []

A = [0 for x in range(lim)]

A[0] = 1

A[1] = 1

MAX = math.floor(math.sqrt(lim))

for i in range(2,MAX+1):

if A[i] == 0.0:

for j in range(i\*i,lim,i):

A[j] = 1

for x in range(len(A)):

if A[x] == 0:

Primes.append(x)

return Primes

#Testing the function

all\_primes\_up\_to\_999 = sieve\_of\_eratosthenes(999)

print(all\_primes\_up\_to\_999)

#Test/verify with part (a)

for x in all\_primes\_up\_to\_999:

if is\_prime(x) == False:

print("Error", x, "is not prime")

**D**

def first\_n\_primes(n):

if n < 0:

print("Error")

elif n == 0:

return []

elif n == 1:

return [2]

elif n == 2:

return [2,3]

else:

result = [2,3]

counter = 2

x = 0

while (counter < n):

test = 6 \* x + 1

test2 = 6 \* x - 1

if is\_prime(test2) == True:

result.append(test2)

counter = counter + 1

if is\_prime(test) == True:

if counter < n:

result.append(test)

counter = counter + 1

x = x + 1

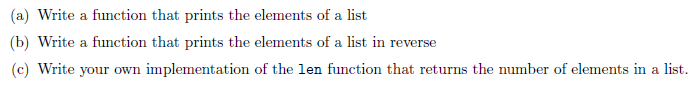
return result

#Testing the function

first\_100\_primes = first\_n\_primes(100)

print(first\_100\_primes)

**Exercise 7**



**A: Prints elements of a list**

lst = [0,1,2,3,4,5,6,7,8,9]

empty = []

#A prints elements

def iterator(List):

for element in List:

print(element)

iterator(lst)

iterator(empty)

**B**: print elements in reverse

def reverse\_iterator(List):

r = -1

for x in range(len(List)):

print(List[r])

r = r-1

reverse\_iterator(lst)

reverse\_iterator(empty)

**C:** returns number of elements in list

def Length(List):

size = 0

for element in List:

size = size + 1

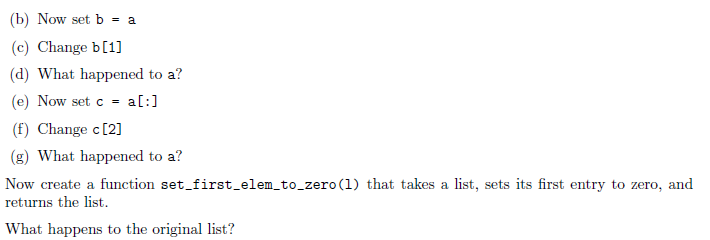
return size

Length(lst)

Length(empty)

**Exercise 8**





**A: create list with entries**

a = [0,1,2,3,4,5,6]

**B: set b = a**

b = a

**C: change b[1]**

b[1] = 42

**D: what happened to a?**

print(a) #a[1] also got changed to 42

**E: set c=a[:]**

c = a[:]

**F: change c[2]**

c[2] = 42

**G: what happened to a?**

print(a) #a remained unchanged

'''Create function that takes list, sets its first entry to zero, and returns'''

def set\_first\_elem\_to\_zero(l):

l[0] = 0

return l

test\_list = [1,1,1,1]

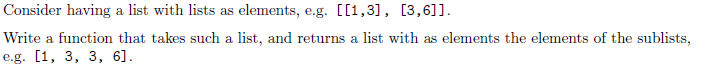
set\_first\_elem\_to\_zero(test\_list)

#What happened to the go list?

print(test\_list)

#The first element got changed to 0

**Exercise 9**



List = [[1,3],[3,6]]

def flatten(l):

flattened\_List = []

for vector in l:

for element in vector:

flattened\_List.append(element)

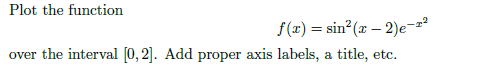
return flattened\_List

New\_List = flatten(List)

print(New\_List)

**Exercise 10**

Use mathplotlib



import matplotlib.pyplot as plt

import numpy as np

def f(x):

return (np.sin(x-2) \*\* 2) \* (np.e \*\* (-x \*\* 2))

x = np.arange(0.0, 2.0, 0.01)

y = f(x)

plt.plot(x, y)

plt.title('Exercise 10')

plt.xlabel('X')

plt.ylabel('Y')

plt.grid(True)

plt.show()

**Exercise 11**



List = [1,2,3,4,5,6,7,8,9,10]

empty = []

def iteration\_mul(List):

if len(List) == 0:

return 0

result = 1

for element in List:

result = result \* element

return result

def recursion\_mul(List):

if len(List) == 0:

return 0

if len(List) == 1:

return List[0]

else:

return recursion\_mul([List[0]]) \* recursion\_mul(List[1:])

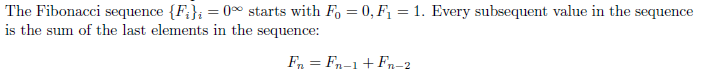
iteration\_mul(List)

recursion\_mul(List)

iteration\_mul(empty)

recursion\_mul(empty)

**Exercise 12**



No real question is asked. So, I will assume that I need to write a Fib function.

fiblist = [0 for x in range(999)] #Using a list for memoization

def fib(x):

if x == 0:

fiblist[x] = 0

return 0

elif x == 1:

fiblist[x] = 1

return 1

elif x == 2:

fiblist[x] = 2

return 2

elif fiblist[x] != 0.0:

return fiblist[x]

else:

fiblist[x] = fib(x-1) + fib(x-2)

return fiblist[x]

for x in range(50):

print(fib(x))

**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.

import re

file = open('emails.txt','r')

file = file.read()

Emails = re.findall(r'[\w\"\.-@]\*[\w\"\w.-]+@[\w\.-]+\.[\w]+', file)

print(Emails)

**References**

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