Tomer Alon

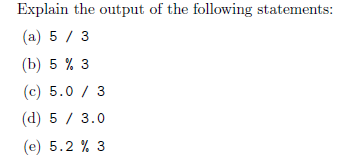
CS361

Programming Languages Principles and Implementation

Python Exercises

**Python**

**Exercise 1**



print("(a) 5 / 3: ", 5/3)

#Prints 1.6666666666666667 since Python 3's division returns a float

print("(b) 5 % 3: ", 5%3)

#Prints 2, since 2 is the remainder when 5 is divided by 3; % is the modulous operator which returns the remainder

print("(c) 5.0 / 3: ", 5.0/3)

#Prints 1.6666666666666667

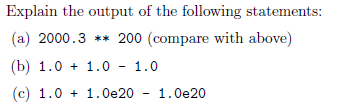
print("(d) 5 / 3.0: ", 5/3.0)

#Prints 1.6666666666666667

print("(e) 5.2 % 3: ", 5.2%3)

#Prints 2.2, since 2.2 is the remainder when 5 is divided by 3; % is the modulus operator which returns the remainder

**Exercise 2**



#print("(a) 2000.3 \*\* 200: ", 2000.3\*\*200)

#Prints an overflow error

print("(b) 1.0 + 1.0 - 1.0: ", 1.0+1.0-1.0)

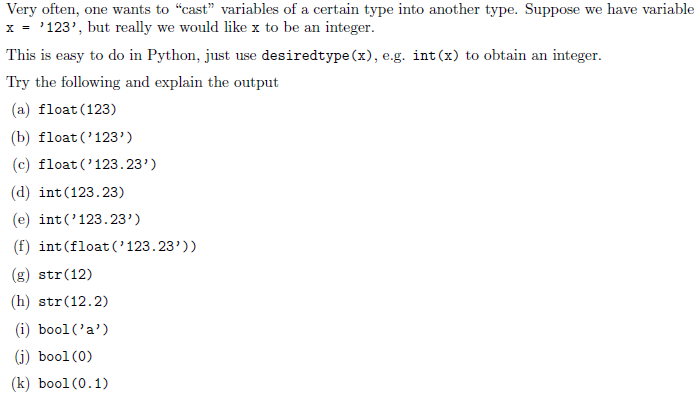
#Prints 1.0; following the operations: 1.0 is added to 1.0 and then 1.0 is subtracted, thusly returning a float

print("(c) 1.0 + 1.0e20 - 1.0e20: ", 1.0+1.0e20-1.0e20)

#Prints 0.0; 1.0 is added to 1.0e20, which is already an enormous number, so the addition of 1.0 is so minute, 1.0e20 remains the same.

#Subtracting 1.0e20 from 1.0e20 results in 0.0

**Exercise 3**



print("(a) float(123): ", float(123))

#Prints 123.0 because the int 123 is casted as a float

print("(b) float('123'): ", float('123'))

#Prints 123.0 because the string '123' is casted as a float

print("(c) float('123.23'): ", float('123.23'))

#Prints 123.23 because the string '123.23' is casted as a float

print("(d) int(123.23): ", int(123.23))

#Prints 123 because the float 123.23 is casted as an int

#print("(e) int('123.23'): ", int('123.23'))

#Prints an int error, because the casting is done on a string which should be first casted as a float

print("(f) int(float('123.23')): ", int(float('123.23')))

#Prints 123 because the string '123.23' is casted as a float and then casted as an int

print("(g) str(12): ", str(12))

#Prints 12, in the form of a string, since the int 12 is casted as a string

print("(h) str(12.2): ", str(12.2))

#Prints 12.2, in the form of a string, since the float 12.2 is casted as a string

print("(i) bool('a'): ", bool('a'))

#Prints True, since in Python only 0, as well as empty lists, empty tuplets, and empty strings, evalualte to False

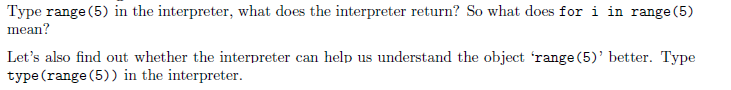
print("(j) bool(0): ", bool(0))

#Prints False

print("(k) bool(0.1): ", bool(0.1))

#Prints True

**Exercise 4**



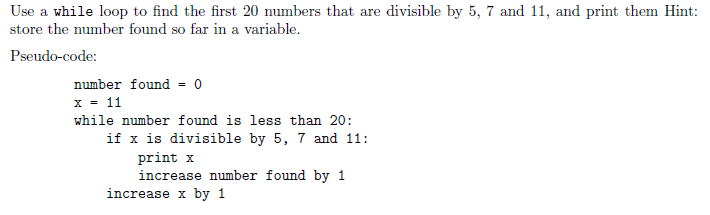
#In the interpreter:

#range(5) -> Prints range(0, 5), which is an iteration of i from start (inclusive) to stop (exclusive) by step. Start defaults to 0

#Type 'type(range(5)) in the interpreter:

#type(range(5)) -> Prints <class 'range'>

**Exercise 5**



def exe5():

nofound = 0

x = 11

ans = list()

while nofound < 20:

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

ans.append(x)

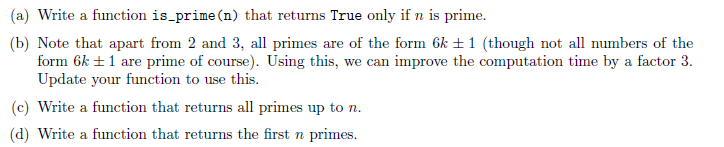
nofound += 1

x += 1

return(ans)

print(exe5())

**Exercise 6**



print("6a:", end=' ')

def is\_prime(n):

if n%2 == 0 or n < 2:

return False

for i in range(3, int(n\*\*0.5)+1,2):

if n%i == 0:

return False

return True

print(is\_prime(7))

print("\n6b:", end=' ')

def is\_prime2(n):

if n==2 or n==3:

return True

if n%2 == 0 or n < 2:

return False

for i in range(3, int(n\*\*0.5)+1,2):

if n%i == 0:

return False

return True

print(is\_prime2(7))

print("\n6c:") #https://stackoverflow.com/questions/11619942/print-series-of-prime-numbers-in-python for efficient method

def nPrimes(n):

solution = list()

sieve = [True] \* (n+1)

for p in range(2, n+1):

if (sieve[p]):

solution.append(p)

for i in range(p, n+1, p):

sieve[i] = False

return solution

print(nPrimes(30))

print("\n6d:")

def firstNprimes(n):

primes = []

count = 2

while len(primes) != n:

for i in range(2, count // 2 + 1):

if count % i == 0:

break

else:

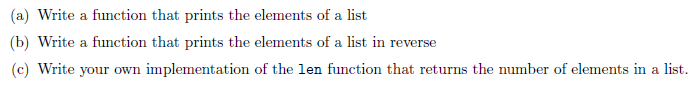
primes.append(count)

count += 1

return primes

print(firstNprimes(100))

**Exercise 7**



print("7a:")

def printList():

list = [1,2,3,4,5]

for i in (list):

print (i, end=' ')

#print(list)

printList()

print("\n7b:")

def printListbk():

list = [1,2,3,4,5]

for i in reversed(list):

print(i, end=' ')

printListbk()

print("\n7c:")

def listSize():

list = [1,2,3,4,5]

length = 0

for i in (list):

length += 1

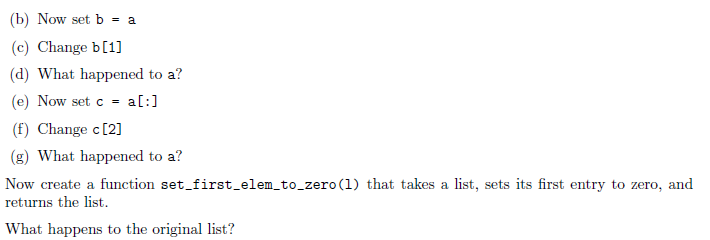
print(length)

print("Number of elements in the list: ")

listSize()

**Exercise 8**





#8a:

a = [15,63,77,34,23,444,1,100]

print(a)

#8b:

b = a

#8c:

b[1] = 2

#8d:

print(a) #a changed, just as b did, in index 1

#8e:

c = a[:]

#8f:

c[2] = 99

#8g:

print(a) #a did not change in index 2, but c did

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

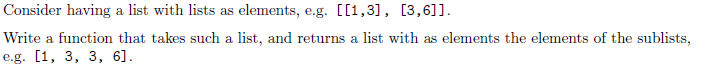
l[0] = 0

return l

lTest = [1,1,2,3,4]

print(set\_first\_elem\_to\_zero(lTest))

**Exercise 9**



exList = [[1,3],[3,6]]

def merge\_list(l):

finalList = list()

for sublist in l:

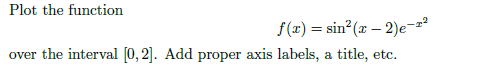
for i in sublist:

finalList.append(i)

return finalList

print(merge\_list(exList))

**Exercise 10**



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.xlabel('x-axis')

plt.ylabel('y-axis')

plt.title("f(x) = sin^2(x-2) \* e^-x^2")

plt.grid(True)

plt.show()

**Exercise 11**



prodList = [1,2,3,4,5]

def iterative\_list(l):

if len(l) == 0:

return "List is empty"

product = 1

for i in l:

product \*= i

return product

print("Iterative: ")

print(iterative\_list(prodList))

def recursive\_list(l):

if len(l) == 1:

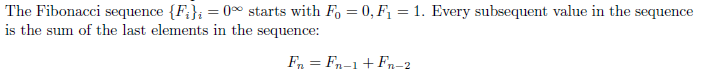
return l[0]

return recursive\_list([l[0]]) \* recursive\_list(l[1:])

print("Recursive: ")

print(recursive\_list(prodList))

**Exercise 12**



def recFib(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

return recFib(n-1) + recFib(n-2)

return n

print("Recursive Fibonacci: ")

print(recFib(29))

def memoFib(n, \_cache={}):

if n in \_cache:

return \_cache[n]

elif n == 0:

return 0

elif n == 1:

return 1

elif n > 1:

return \_cache.setdefault(n, memoFib(n-1) + memoFib(n-2))

return n

print("Memoized Fibonacci: ")

print(memoFib(29))

**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.

import re

#Define path of file 'emails\_ex13.txt'

file = open('emails\_ex13.txt', 'r')

file = file.read()

emails = re.findall(r'([^ ]+[@][^ ]+[.][a-z]+)', file)

print(emails)

**References**

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