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

Explain the output of the following statements:

a) **5 / 3**

   The output of this is 1. By default, these numbers are integers and will return an integer result. 5 divided by 3 is 1.6667 but when the computer converts it to an integer, it gets rid of the decimal and gives the integer whole number no matter what the decimal is.

b) **5 % 3**

   The output of this is 2. The percent sign signifies the modulo function which gives the remainder of your function. 5 divided by 3 is going to be 1 with a remainder of 2, so 5 modulo 3 is 2.

c) **5.0 / 3**

   The output of this is 1.66666666667. We specify that 5.0 is not an integer, rather a float type by putting ".0" afterwards. Because of this, the output can have decimals in it as well, so it shows the correct float with decimals.

d) **5 / 3.0**

   The output of this is 1.66666666667. Like in the above example, we specify that one of our terms is a float we are doing float division, not integer division so the answer returned is a float.

e) **5.2 / 3**

   The output is 1.73333333333. Like in the above 2 examples, we see a float and an integer so the division returns a float.

Exercise 2

Explain the output of the following statements:

a) **2000.3 ** 200 (compare with above)**

   The output of this is an overflow error. The '' operator means x^n with x being 2000.3 and n being 200 in this case and 2000.3^{200} is a huge number that goes over the limit accepted by a float**

b) **1.0 + 1.0 - 1.0**

   The output of this is 1.0. Here, we are dealing with floats. Even though these numbers could be stored as integers, they are specified as and returned as floats

c) **1.0 + 1.0e20 - 1.0e20**

   The output of this is 0.0. Like above, these are float numbers and they are returned as floats as well. The "e20" that follows the last two numbers signifies that the number is 1e20 or 1 followed by 20 zeros

Exercise 3

Try the following and explain the output

a) **float(123)**

   The output is 123.0. This is because a float is a data type that accounts for numbers with decimal places. 123 is displayed as 123.0 because that is the same and it is now a float.

b) **float('123')**

   The output is 123.0 again. This does the same thing as the above and turns 123 into a float by specifying the decimal place.

c) **float('123.23')**

   The output is 123.23. 123.23 is already a float, so casting it to a float doesn't change the value at all.

d) **int(123.23)**

   The output is 123. An integer type is a number that doesn't have decimal places. So the int(123.23) gets rid of the decimals and just shows the whole number integer.

e) **int('123.23')**

   The output is ValueError: invalid literal for int(). The quotes around 123.23 mean it can't be changed, it has to keep it's decimal places, because of that it can't be changed into an integer

f) **str(12)**

   The output is 12. This 12 is a string however. Strings can be made up of numbers, not just letters.

g) **str(12.2)**

   The output is 12.2. This is a string because, once again, strings can hold numbers and symbols, not just letters.

h) **bool('a')**

   The output is True. Anything that isn't 0 is automatically true.

i) **bool(0)**

   The output is False. In binary thinking, 0 represents false and 1 represents true.

j) **bool(0.1)**

   The output is True. Even though this starts with 0, it is not assigned 0 therefore it is True.

Exercise 4

Type `range(5)` in the interpreter, what does the interpreter return? So what does `for i in range(5)` mean?

Let's also find out whether the interpreter can help us understand the object '`range(5)`' better. Type `type(range(5))` in the interpreter.

Range(5) yields the result [0, 1, 2, 3, 4]. This is an list with 5 values stored in it. For i in range(5) would mean for every i type in this 0-4 5 digit range, so any one of those numbers. Type(range(5)) returns type list, meaning that range(x) makes a list with x values in it.

Exercise 5 - (full code can be found on GitHub)

Use a `while` loop to find the first 20 numbers that are divisible by 5, 7 and 11, and print them Hint: store the number found so far in a variable.

Pseudo-code:

```
number found = 0
x = 11
while number found is less than 20:
    if x is divisible by 5, 7 and 11:
        print x
        increase number found by 1
    increase x by 1
```

```
def main():
    number = 0
    x = 11

    while (number < 20):
        if (x % 5) == 0 or (x % 7) ==0 or (x % 11) == 0:
            print x
            number += 1
        x += 1

if __name__ == "__main__":
    main()
```

Exercise 6 - (full code can be found on GitHub)

(a) Write a function `is_prime(n)` that returns `True` only if n is prime.

(b) Note that apart from 2 and 3, all primes are of the form $6k \pm 1$ (though not all numbers of the form $6k \pm 1$ are prime of course). Using this, we can improve the computation time by a factor 3. Update your function to use this.

(c) Write a function that returns all primes up to n.

(d) Write a function that returns the first n primes.

```
def main():
    n = 10
    print(is_prime(n))
    print(six_prime(n))
    print(up_to_prime(n))
    print(first_prime(n))

def is_prime(n):
    prime = False
    if (n%2 == 0 & n != 2):
        prime = False
    elif (n%3 == 0 & n != 3):
        prime = False
    elif (n%5 == 0 & n != 5):
        prime = False
    elif (n%7 == 0 & n != 7):
        prime == False
    elif (n%11 == 0 & n != 11):
        prime == False
    elif (n%13 == 0 & n != 13):
        prime = False
    else:
        prime = True

    if n < 2:
        prime = False
    elif n == 2 or n == 3 or n == 5 or n == 7 or n == 11 or n == 13:
        prime = True
    return prime

def six_prime(n):
    prime = False

    if (n+1)%6 == 0:
        if n%2 == 0:
            prime = False
        else:
            prime = True
```

```
    if (n-1)%6 == 0:
        if n%2 == 0:
            prime = False
        else:
            prime = True

    if n < 2:
        prime = False
    elif n == 2 or n == 3:
        prime = True

    return prime

#I'll put my items in a list because a loop with return in it
#wouldn't work for multiple values. Print would return these,
#but I don't want print in any other method than main

def up_to_prime(n):
    primes = []
    f = 0

    while (f <= n):
        if (is_prime(f) == True):
            primes.append(f)
            f = f + 1
        else:
            f = f + 1

    return primes

def first_prime(n):
    primes = []
    y = 0
    count = 1

    while (count <= n):
        if (is_prime(y) == True):
            primes.append(y)
            y = y + 1
            count = count + 1
        else:
            y = y + 1

    return primes


if __name__ == "__main__":
    main()
```

Exercise 7 - (full code can be found on GitHub)

(a) Write a function that prints the elements of a list

(b) Write a function that prints the elements of a list in reverse

(c) Write your own implementation of the `len` function that returns the number of elements in a list.

```
def main():
    a = [0, 1, 2, 3]
    elements(a)
    reverse(a)
    print(length(a))

def elements(a):
    for i in a:
        print i

def reverse(a):
    for i in reversed(a):
        print i

def length(a):
    x = 0
    for i in a:
        x += 1
    return x

if __name__ == "__main__":
    main()
```

Exercise 8 - (full code can be found on GitHub)

A) When changing b[1], a[1] also changed. This is because we set b = a. If we set it like we did to c, c = a[:], the value will change only in that list rather than both.

(b) Now set `b = a`

(c) Change `b[1]`

(d) What happened to `a`?

(e) Now set `c = a[:]`

(f) Change `c[2]`

(g) What happened to `a`?

Now create a function `set_first_elem_to_zero(l)` that takes a list, sets its first entry to zero, and returns the list.

What happens to the original list?

```
def main():
    a = [12, 14, 16]
    b = a
    c = a[:]

    print a
    print b
    print c

    c[2] = 10

    print a
    print b
    print c

    l = [1, 1, 1, 2, 3, 5]
    print(set_first_elem_to_zero(l))


def set_first_elem_to_zero(l):
    new = l[:]

    new[0] = 0

    return new

if __name__ == "__main__":
    main()
```

Exercise 9 - (full code can be found on GitHub)

Consider having a list with lists as elements, e.g. `[[1,3], [3,6]]`.

Write a function that takes such a list, and returns a list with as elements the elements of the sublists, e.g. `[1, 3, 3, 6]`.

```
def main():
    a = [[1,3], [3,6], [8,10,12]]

    print(a)
    print(sublist(a))

def sublist(a):
    s = []

    length = len(a)

    for i in range(0, length):
        templist = a[i] #when the item is a list, we separate it and make it it's own
list.
        templength = len(a[i]) #this is the length of that new list

        for x in range(0, templength):
            s.append(templist[x]) #puts the items in the sublist into this new list of
elements.
            x = x + 1

    return s

if __name__ == "__main__":
    main()
```

Exercise 10 - (full code can be found on GitHub)

Plot the function

$$f(x) = \sin^2(x-2)e^{-x^2}$$

over the interval $[0, 2]$. Add proper axis labels, a title, etc.

```
import math
import numpy as np
import matplotlib.pyplot as plt

def main():
    x = np.linspace(0, 2, 100)
    y = np.linspace(0, 2, 100)
    formula = (np.sin(pow((x - 2), )) * (np.e ** (-1 * pow(x, 2)))
    plt.title("(sin(x - 2))^2 * e^(-x^2)")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.plot(x, formula)
    plt.show()
```

Exercise 11 - (full code can be found on GitHub)

Write two functions, one that uses iteration, and the other using recursion, that achieve the following: The input of the function is a list with numbers. The functions return the product of the numbers in the list.

```
def main():
    n = [1, 4, 9, 16]
    print(iteration(n))
    print(recursion(n))

def iteration(n):
    product = 1
    for i in n:
        product *= i
    return product

def recursion(n):
    length = len(n)
    newlist = n
    product = 1
    if len(newlist) == 0:
        return product
    else:
        product = newlist.pop(0)
        return product * recursion(newlist)

if __name__ == "__main__":
    main()
```

Exercise 12 - (full code can be found on GitHub)

The Fibonacci sequence $\{F_i\}_i = 0^\infty$ starts with $F_0 = 0, F_1 = 1$. Every subsequent value in the sequence is the sum of the last elements in the sequence:

$$F_n = F_{n-1} + F_{n-2}$$

```
def main():
    print("Fibonacci 8: ", fibonacci(8));
    print("Fibonacci 12: ", fibonacci(12));

def fibonacci(x):

    if x <= 1:
        return x;

    else:
        return fibonacci(x-1) + fibonacci(x-2);



if __name__ == "__main__":
    main()
```

Exercise 13 - (full code can be found on GitHub)

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 parser
import re

def main():
    read = open("emails.txt", "r")
    text = read.read()
    read.close()

    email = re.findall(r'[\w\"\'\.\@]*[\w\"\'\.]+@[\w\.]+\.[\w\.]+[\w]+', text)
    print(email)

if __name__ == "__main__":
    main()
```

References

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