MM220: Computational Lab

Amrita Bhattacharya

October 4, 2020

Introduction to python -I: Programming basics

Python is a high level interactive programming language. It is a free software and is used primarily for numerical computations and plotting. It host powerful data processing and plotting libraries such as scikit learn, matplotlib etc.

1 Getting started

1.1 Version

Listing 1: Python version

```
python --version
```

We are using Python 3.6.9. However, if you are using python 2.7. There is only very small difference.

1.2 First program

Type the following in the command prompt and press “Enter". Please note that strings are generally written in double or single cursors.

Listing 2: Hello

```

>>> print ("Hello, Python!")
Hello, Python!

>>> 2+3
5
>>> 2*3
6
>>> 2**3
8
>>> a=2
>>> b=3
>>> a+b
5
>>> a
2
>>> a=3
>>> a+b
6
```

1.3 Saving your program

You can use the prompt for performing preliminary calculations. But for all practical purposes, we want to save our programs, so that we can use them infuture. So, let us quit the terminal by pressing the keys ‘Ctrl’ and ‘D’. Make a new folder ‘python’ in your working directory. We will store all our programs and execute them in this location from now onwards. Please type ‘pwd’ and ensure your location as ‘/path-to-directory/python’.

We have pre written the programs in ‘vi’ editor. You may type ‘vi hello.py’ in your terminal to open a file called ‘hello.py’. Press ‘i’ (short for Insert) to enanle writing mode and once written, we will save the contents by pressing ‘Esc’ and typing by ‘wq’(short for write and quit).

Listing 3: Hello saved

```
print ("Hello, Python!")
```

To execute your saved program type ‘python hello.py’. Congratulations you executed your first program.

2 Assigning values to variables

2.1 Assignments

We use the equal sign (=) to assign values to variables. Python also allows you to assign a single value to several variables simultaneously. You can also assign multiple objects to multiple variables. Please write the following lines in a new file called ‘assign.py’.

Listing 4: Assign values to variables

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
print (counter)
print (miles)
print (name)

a = b = c = 1

print (a, b, c)

x, y , z =  1,  2, "john"

print (x, y, z)
```

Execute using ‘python assign.py’. We have printed the assigned values of our variables.

3 String, and list

3.1 String

Strings in Python are identified as a set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Please write the following lines in a file called ‘word.py’.

Remember when you write numbers in quotes python refers it as a string.

Listing 5: String

```
str = 'Hello World!'

print (str)          # Prints complete string
#print (str[0])       # Prints first character of the string
#print (str[2:5])     # Prints characters starting from 3rd to 5th
#print (str[2:])      # Prints string starting from 3rd character
#print (str * 2)     # Prints string two times
#print (str + "TEST") # Prints concatenated string

#num='3333'

#print (num * 3)
```

Execute using 'python word.py'. Please unhash the statements one by one to see how to refer to the elements in the string.

3.2 List

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). Please write and execute the following lines as 'list.py'.

Listing 6: List

```
list = ['abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)         # Prints complete list

l=[]
l.append(list[1])
print (l)

#print (list[0])       # Prints first element of the list
#print (list[1:3])     # Prints elements starting from 2nd till 3rd
#print (list[2:])      # Prints elements starting from 3rd element
#print (tinylist * 2) # Prints list two times

#print (list + tinylist) # Prints concatenated lists
#print (list[1]+ list[4]+ tinylist[0]) # adds up the elements
#print (list[1]**2)  # square of the element
#list[1]=1000 #update element
#print (list)
#print len(list) #size of list
```

Unhash the statements one by one to examine the output. Please note that any variable in a list can be updated. The “len" option gives the number of elements in the list.

4 Loops

A loop statement allows us to execute a statement or group of statements multiple times. In the examples below we will use the for and while loops to build your programs.

For loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

While loop Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

Listing 7: Loop

```
list = ['abcd', 786 , 2.23, 'john', 70.2 ]

l=[]

for i in range(len(list)): # for loop
   l.append(list[i])

print (l, l[1])

for i in range(len(list)): # for loop
   print (list[i])

i=0
while (i < len(list)):    # while loop
    print (list[i])
    i=i+1

#for num in range(2, 101, 2):
#   print(num)

#n = 2
#while n <= 100:
#    print (n)
#    n += 2
```

Now, once you have executed the current program, hash out the first example and unhash the second.

5 Decision making

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions. Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise. If the action is FALSE the content inside is not executed.

Listing 8: If statements

```
var = [100, 150, 200, 250]
for i in range(len(var)): # for loop
    print var[i]
    if var[i] == 200:
      print ("No")
    elif var[i] == 150:
      print ("Sorry")
    elif var == 100:
      print ("Got it")
    else:
      print ("You are not there")
```

6 Functions

Functions begin with the keyword def (short for definition) followed by the function name and parentheses. Any input parameters or arguments should be placed within the parentheses. A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. The results of the functions are returned to the program itself using 'return'. The functions are executed using objects. Please have a look at the programs below;

6.1 Simple examples

Listing 9: function-1

```
def sum(a, b):          #function1
  summation = a +b
  return summation

def mult(a,b):          #function2
  multiplication= a*b
  return multiplication

def division(a,b):      #function3
  division= a/b
  return division

summation1 = sum(4, 2)     # object of function 1
summation2 = sum(6, 2)     # another object of function 1
multiplication = mult(4,2) # object of function 2
division = division(4,2)   # object of function 3
print (summation1, summation2, multiplication, division)
```

The 'return' functionality returns the desirable output to the program.

Functions can be accessed by creating objects. You can create multiple objects for each of the functions.

Listing 10: function-2

```
def math(a,b):
  add= a+b
  sub= a-b
  mult=a*b
  div= a/b
  return add, sub, mult, div

add, sub, mult, div = math(6,2)

print (add, sub, mult, div)
```

You can return multiple values through each of your methods. Your object will be able to access the values from the method in only the given order.

7 Module

If we quit from the Python interpreter, the definitions are lost. As your program gets longer, we may want to split it into several files for easier maintenance. We may also want to use a function that we have written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

A module is a file containing Python definitions and statements. The file name is the module name with the suffix '.py'.

Listing 11: Fibo

```
def fib(n):    # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

You can import 'fibo.py' as a module if it is in the same path. Else you can also add it to your python libraries to make it available globally (beyond the scope of this class).

Listing 12: module

```
import fibo
print (fibo.fib(1000))
print (fibo.fib(100))
```

8 class

To define a *class* in Python, you can use the class keyword, followed by the class name and a colon. A function defined within the class is called *method*. The methods can be called by the creating objects.

Listing 13: method

```
class MEMS:
    def year(self): # method
        print(2020)

student1 =MEMS() # Instantiating objects
student2 =MEMS() # Instantiating objects

MEMS.year(student1)
MEMS.year(student2)

student1.year()
student2.year()
```

The __init__ is the initializer that you can later use to instantiate objects. It's similar to a constructor in Java. It takes the first argument as *self*, which refers to the object itself.

Listing 14: Initiate

```
class MEMS:
    def __init__(self, stream, batch):
        self.stream = stream # attribute
        self.batch = batch   # attribute

    def identity(self):
        print ('student of', self.stream, self.batch)

student1=MEMS('physical_metallurgy', 'Btech')
student2=MEMS('process_metallurgy', 'Mtech')

student1.identity()
student2.identity()

print (student1.stream())
```

In the following, we learn on how to control the attributes in init, from the different methods in the program.

Listing 15: more

```
class MEMS:
    def __init__(self):
        self.batch = "BTECH"
        self.year = 2008
        self.stream ="corrosion"
        self.credit =48

    def update(self):
        if self.stream == "Process_metallurgy":
            print ('course credit is', self.credit)

    def compare(self, other):
        if self.credit == other.credit:
            return True

student1=MEMS()
student2=MEMS()

student1.batch = "MTECH"
print (student1.batch)

student2.batch = "MTECH"
student2.year = 2020
student2.stream = "Process_metallurgy"

student2.update()

if student1.compare(student2):
    print("Same")
```

In the following, we will learn about the class variables and the instance variables. We will figure out how to control the class variables from the different methods within the program.

Listing 16: class-variable

```
class MEMS:
    year =2020           # class variable

    def __init__(self):
        self.roll=10957          #instance variable
        self.name= "Arpit"

student1 =MEMS()
student2 =MEMS()

student2.name ="Rohit"
student2.roll = 10958

print(student1.roll)
print(student2.roll)

print(MEMS.year)
MEMS.year = 2018

print (MEMS.year)
```

In the following, we will learn on how to perform mathematical operations on the attributes of the class;

Listing 17: operations

```
class student:
    dept = 'MEMS'
    def __init__(self, m1, m2, m3, m4):
        self.m1=m1
        self.m2 =m2
        self.m3=m3
        self.m4 =m4

    def avg(self):
        return (self.m1+self.m2+ self.m3+self.m4)/4

S1= student(36, 28, 39, 47)
S2=student(87,98, 99, 92)

print (S1.avg())
print (S2.avg())
```

In the following, we learn about sub classes (class within a class). We also learn on how to control the attributes within the sub class.

Listing 18: subclass

```
class student:
    def __init__(self, name, roll):
        self.name =name
        self.roll= roll
        self.stream = self.stream()

    def show(self):
        print(self.name, self.roll)

    class special:
        def __init__(self):
            self.stream = 'corrosion'
            self.credit = 48

        def show(self):
            print(self.stream, self.credit)

S1 =student("Manish", 10456)

S1.special.stream ="physical_metallurgy"

S1.show()
S1.special.show()

print (S1.special.credit)
```

9 Assignments

1. In this exercise, we will write a python script to plot the trajectory of a projectile without inclusion of air resistance.

To start describing the problem, we need to define the parameters in the problem.

- the mass of the projectile
- acceleration due to gravity
- initial velocity (magnitude and direction)

We also set the time step to track the trajectory. We start by putting in the initial velocity components v_x= V$\cos\theta$ and v_y= V$\sin\theta$ at t=0.

In the absence of the air drag, the x and y components of the acceleration are given as, $a_x = 0$, and a_y = -g. The velocity is given by the following equation;

$$v(t_{n+1}) = v(t_n) + a(t_n)\Delta t \tag{1}$$

Thus, the x and y can be obtained from the equations;

$$dy(t)/dt = v_y - gt \tag{2}$$

$$dx(t)/dt = v_x \tag{3}$$

Listing 19: proj

```python
import numpy as np
from matplotlib.pylab import plt

def projectile(angle, v0, time):
    t = np.linspace(0,time,300)
    dt = t[1] - t[0]   #s
    g = 9.8 # m/s2
    m = 1   # kg
    # Distribute initial velocity to x and y.
    v0 = v0    # m/s
    vx = v0*np.cos(angle)
    vy = v0*np.sin(angle)
    y0 = 0
    x0 = 0
    vg = 0
    y_list = [y0]
    x_list = [x0]
    ax=0
    ay= -g

    for i in range(1, len(t)):
        # Update y.
        vg += ay * dt
        y0 += (vg + vy)* dt
        # Update x.
        x0 += vx * dt

        # Stop updating when it hits ground at y=0.
        if y0 > 0:
            y_list.append(y0)
            x_list.append(x0)
        else:
            break

    # (max x, max t)
    return x_list, y_list

x_list1, y_list1= projectile(np.pi/3, 100, 30)

print (x_list1, y_list1)
plt.plot(x_list1, y_list1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

We obtained the following plot for $\theta = 60$ degrees, initial velocity = 100 m/s and time = 30 s.

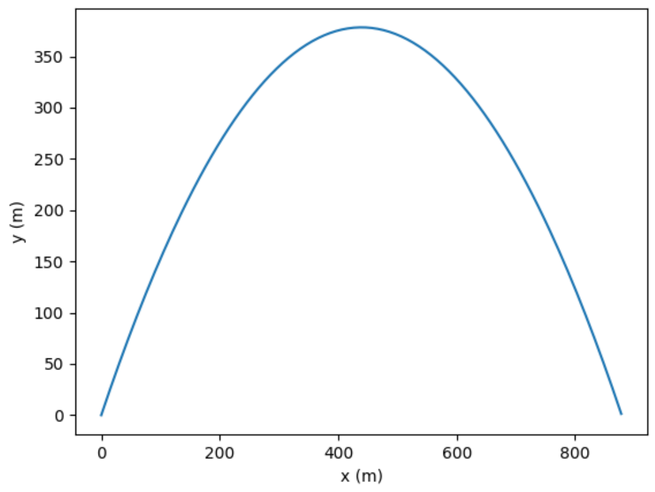

Figure 1: Trajectory of the projectile.

(i) Plot the projectile motion with $\theta = 45$ degree, initial velocity = 60 m/s and time = 25 s in the same plot. Find the range of the projectile in this case.
(ii) The air resistance drag force is approximated by the relation, $F_d = C_d v^2$, where, v is the magnitude of velocity in m/s and C_d is the coefficient of drag. Once the drag is included, the x and y components of the resultant force will be given by

$$F_x = -F_d cos\theta \tag{4}$$

$$F_y = -Mg - F_d sin\theta \tag{5}$$

(a) Find the trajectory of a projectile with $\theta = 60$ degrees, initial velocity = 30 m/s and time = 30 s with the drag force of coffecient $C_d = 0.005$ acting on it. What is the range of the projectile?
(b) Plot v_x and v_y as a function of time.

(iii) Write the above program using class and methods within the class.