Chapter 3

Python Programming

Python is an open source programming language, probably the most widely used programming language for artificial intelligence and machine learning research. In this chapter we will provide an introduction to Python programming.

3.1 A simple Python program

Note: We will be using Python 3 for this course!

We start off by writing a simple Python program.

```
print("Hello World")
```

The above code in Python prints "Hello World" on the screen (without the quotes).

3.2 Variable and Simple Data Types in Python

Variables are the basic place holders for information in programming languages. Below we provide example of common types of variables in Python .

```
text="Hello World!"
print(text)
var_float=4.5
var_int=5
type(text)
```

In the above code 'text' is a variable which holds a string, 'var_float' is a variable which holds a floating type number and 'var_int' is a variable which holds an integer . Notice, in Python we do not need to specify the type of the variable during assignment and the type is automatically set depending on the value on the right hand side. We can check the data type of a variable by using the 'type' command in python.

Note: Variable names can contain only alphabets number and underscore, no spaces. Avoid using python keyword or functions names as variable names.

Good Practice: It is a good practice to use lowercase alphabets for variable names, uppercase letters are used in different names that we will see in the coming chapters.

Strings

A series of characters is called a string. In python anything inside quotes (either single or double quotes) is considered a string.

```
str1="test with double quotes"
str2='test with single quotes'
type(str1)
type(str2)
```

Some interesting combinations of the quotes:

```
str1="single quote ' ' inside double quotes"
str2='double quote " " inside single quote'
print(str1)
print(str2)
```

The following are some of the methods of the string class in Python .

Note: Methods will be formally defined in Chapter 4.

```
name="john doe"
print(name.title())
print(name.upper())
print(name.lower())
```

We can join different string and string variable in Python :

```
print("This is a test" + "to check the + operation for strings")
```

We use f-strings to add strings from string variables

```
str1="part 1"
str2="part 2"
str3="part 3"
cat_str=f"\t{str1}\t{str2}\t{str3}\n"
print(cat_str)
print(cat_str.rstrip())
print(cat_str.lstrip())
print(cat_str.strip())
```

The f_string format can be extended to multiple string variables in Python . Also notice the use of \t and \n and strip commands to add and remove white spaces in Python .

Note: The string.join(iterable) methods provides a flexible way of creting a string from an iterable (strings, lists, tuples etc). Each element of the iterable is sperated by the string. Try the example below:

```
str1='test'
print(str1)
print(','.join(str1))
```

Good Practice: "The most effective debugging tool is still careful thought, coupled with judiciously placed print statements." — Brian W. Kernighan

Integers

As the name suggests integer variables are place holders for integer data. Try the following operations for integers in Python

```
x=4
y=2
print(x+y)
print(x-y)
print(x/y)
#we can use // for integer division
print(x//y)
print(x*y)
print(x**y)
print((x*y)*(x+y)/(x**y))
```

+, -, *, /, ** represent addition, subtraction, multiplication, division, and power operation respectively. Also notice that we can group operations using (), the operation precdence order for un-grouped operations is power, division, multiplication, addition subtraction.

Float

Similar to integers all the above operations are valid for floats as well, with same operation precedence.

Note: If float(s) are involved in an operation Python casts the solution as a float as well. For division opeation the output is a float even if the operands are integers or the result is a whole number.

```
x=4
y=2.0
print(x*y)
```

Additional Features

Some interesting addition features in Python are:

- Underscore can be used in numbers for readability in Python .
- The assignment operation in Python can be used for multiple assignments.

```
x=10_1000
print(x)
x,y,z=1,2,3
print(x,y,z)
```

Good Practice: Python does not have a constant data type. Usually constant variable are labelled with all capital letters

The symbol # is used in Python for comments.

```
#This is the comment
print("The comment part is note executed by python compiler")
```

3.3 Basic Containers

Containers are used in Python to group objects together. We have already seen one type of containers in Python .

Question: What data type that we have seen so far groups together some objects? Hint: the objects that is group together are of character type.

The answer to the above question is strings. Strings are a collection of characters. The containers are grouped into two types in Python 1) mutable, 2) immutable. Mutable containers can be modified after their definition and immutable containers can not be modified after their definition.

Strings

Let's start with strings. As we have mentioned before strings are collection of characters. Strings is an immutable container in Python , that is a string can not be modified after it has been created. Strings are indexed by integers, that is, if you want to access an element (character) from the string you can access by using it's index. Notice that the first element of strings (as well as other indexable containers in Python) start with index 0.

```
str1='test'
str1[0]
str1[1]
str1[3]
str1[4]
str1[1]='b'
```

The last two lines of above code will produce errors. The 5th line is trying to access the 5th element of the string, where the string only has 4 elements. The 6th line tries to modify the 2nd element of the string to a new value and will results in an error as strings are immutable container in Python .

Lists

Lists are the most versatile datatype in Python . A list is defined by comma separating it's elements within square brackets. Elements of a list do not need to be of the same datatype.

```
my_list=[1,2,3,'a','dog',4.5]
```

The above code will create a Python list called my_list which has 6 elements. The first three elements are of integer datatype, followed by a character, a string and a float. Python list are mutable objects and can be modified after definition.

```
my_list=[1,2,3,'a','dog',4.5]
my_list[4]='cat'
del my_list[1]
print(my_list[3])
```

The above is a valid code and the second line will replace the 5th element of the list (dog) with cat. The third line of the code will delete the 2nd element of the list. The fourth line of the code will print the value of the 4th element of the list.
Some common operation and methods for lists are:

```
lst1=[1,2,3]
lst2=['a','b','c']
lst_cat=lst1+lst2
lst_rep=lst1*6
#to check if an element is a member of a list
#we can do x in list, it will return true or false value
print(3 in lst1)
#lists are iteratable objects
for x in lst1:
    print(x)

lst1.append(4)
print(lst1.count(3))
```

[append insert, del, pop, pop position, removing by values] Run the above command and discuss the results with the instructor to make sure that you understand what each command is doing.

Tuples

Tuples is an immutable Python container. Tuple structure is similar to lists with the exception that tuples can not be modified. Tuples can be defined by separating it's elements by comma, inside parenthesis.

```
my_tuple=(1,2,3'a','dog',4.5)
my_tuple[0]
my_tuple[3]=1
```

The first line produces a tuple called my_tuple, notice that similar to lists elements of tuple can be of different datatypes. The 2nd line of code will access the first element of the tuple. The 3rd line will produce and error as we are trying to modify the 4th element of the tuple and tuples are immutable objects.

```
tup1=(1,2,3)
tup2=('a','b','c')
tup_cat=tup1+tup2
tup_rep=tup1*4
del(tup_cat)
print(tup_cat)
len(tup_rep)

#to check if an element is a member of a tuple
#we can do x in tuple, it will return true or false value
print(3 in tup1)
#lists are iteratable objects
for x in tup1:
    print(x)
```

Dictionary

Dictonary object in Python is a collection of elements, where each element has a key and a value. The key is used to access the value. A simple way to understand the concept of dictionaries is to think about real dictionaries, where we can look up the meaning of words. In Python keys correspond to words and the values correspond to meaning of the words. Similar to a dictionary one key can only have one value (like one word has one meaning*). Dictionary are defined with curly brackets.

```
my_dict={'key1':'some value'}
my_dict['key2']='test1'
my_dict['key3']='test2'
print(my_dict)
```

Notice that in the definition of the dictionary, the key and the value are separated by a colon. We can also create an empty dictionary by leaving the curly brackets empty, and we can also create a dictionary with more elements in the definition by separating the elements by comma. Dictionaries are mutable objects and the values assigned to keys can be changed. Notice that by the construction of a dictionary the keys are immutable objects, we can't change a key once it is defined, although we can remove it and add new key values pair.

```
my_dict={'first_name':'N', 'last_name':'K', 'age':'xy'}
print(my_dict)
del(my_dict['age'])
my_dict['AGE']='xy'
my_dict.clear()
print(my_dict)
my_dict['name']='NK'
del(my_dict)
print(my_dict)
```

3.4 Loops

Loops is a programming concept where we want to repeat an action a certain number of times, or until a certain condition is met. For example we might be asked to practice a problem is class until we solve it correctly. Or we might be asked to exercise three times a week. In both these cases an action has to be repeated in the first case until we achieve a condition and in the second case until we achieve a certain number of repetitions.

For Loops

For loops in Python slightly different form other programming languages like C++, basically iterate over a sequence. Rather then being controlled by a counter, they are controlled by the size of the sequence.

```
for(int i=0; i<10; i++)
{
    std::cout<<i<<std::endl;
}
```

The above is a typical for loop in C++, where we start with a varibale i set to 0, i is incerase by 1 every time the loop is run and the opertion is terminated once i is greater then or equal to 10, so the loop will run 10 times.
By contrast in Python the simplest for loop expression will be something like this:

```
for i in range(10):
    print(i)
```

A few important points to notice here, before we start looking into the working of for loop. First is the range function in Python range returns a sequence of number. The more general use of range is range(start, stop, step), where the sequence starts from 'start' the seqeunce stops at 'stop' and the step size is 'step', by default start is 0, and step is 1.
Next, associated with every loop is the concept of the things or action that have be repeated a certain number of times or until a condition is met. The are generally specified in programming language as scope, i.e. scope for a loop is the thing or action that needs to be repeated. Scope in Python is represented by indentation. Hence, it is very important to keep track of indentations in Python .

Note: In Python codes indentation defines the scope of loops. **!**

Now we come back to the for loop, the i variable of the for loop will iterate over (go over) each element of the seqeunce defined by range(10) (numbers from 0 to 9). For each i, we will print the value of i, since it is the scope of the for loop.
Because for loops are iterators in Python they can iterate over containers in Python which is a very desirable property and makes the manipulation of containers very easy.

```
lst1=[1,2,3]
tup1=('a','b','c')
dict1={'fn':'N', 'ln':'K'}

for element in lst1:
    print(element)
```

```
for element in tup1:
    print(element)

for key in dict1:
    print(key , dict1[key])


for c,element in enumerate(lst1):
    print(element)

for c, element in enumerate(tup1):
    print(element)

for c, key in enumerate(dict1):
    print(c, key , dict1[key])
```

While loop

The other type of loop in Python is while loop. While loops correspond to repetition while a condition is true. The syntax of while loop is explained in the example below

```
x=10
while (x>1):
    print(x)
    x=x-1
```

The while loop is run until the condition following the while key word is true. In the above example we have set the value of x to 10 in line 1. so the condition x>1 is true so the commands inside the while loop are performed, the values of x will be printed by line 3 and x will be decreased by 1 on line 3. The process will be repeated until we reach x=1 where the condition in no longer true and we will exit the while loop. Notice that like the for loop, the scope of the while loop is defined by indentation.

A side point to consider here is that with while loop we have to be careful that the conditions we specify are such that we don't get stuck in the loop indefinitely (unless of course this is something that we want in our code).

```
x=10
while (x>1):
    print(x)
    x=x+1
```

The above code will result in an infinite loop since the condition on line 2 is always true and as a result the code will run indefinitely.

Else with loops

Optionally we can use else with for and while loop and the else part of the code is executed after the loops are exited.

```
for x in range(10):
    print(x)
else:
    print('exited for loop')

x=10
while (x>1):
    print(x)
    x=x-1
else:
    print('x is 1')
```

In both the cases of for loop and while loop the else command is executed once the loop block is exited. Notice that else part of the code runs only once!

3.5 Python Function

One of the Python commands that we have used most frequently so far is print(). We provide some input to the print() command and it is displayed on the screen. But what exactly is this print() command doing? In programming languages such commands are called a functions. So one of the things that we notice right away is that the function print() is **reusable** i.e. we can use it to display different output at different points in the code. Another aspect to note is that use of function renders the code to be more **modular** i.e. different blocks of code do different tasks. In Python we have built in function like print() etc (built in function are those function which we don't define and come with standard Python installation). We can also define custom function in Python . Next we will see an example of a Python function to explore how to use functions in Python .

```
def square(x):
    "this function returns the square of a number"
    return x**2

#Now that the function is defined we can call it
print(square(3))
print(square(4))
```

Notice the definition of a Python function starts with the keyword def, followed by the function name and parentheses. Python function names, like other Python identifiers, can be a combination of letters (both upper and lower case), numbers and underscore. Spaces and special characters(!,*,&) are not allowed in the function name. Function name can not start with a number. Apart from these rules, special care must be observed to not use the names of the built-in functions for user-defined functions, e.g. we should not define a function with name print, since print is already a built-in function in Python .

Insider the parentheses are the parameters or arguments that are passed to the function, i.e. the values on which the function needs to perform certain operations. Parameters can also be defined inside the parentheses.

```
def square(x=3):
    "this function returns the square of a number"
    return x**2

#Now that the function is defined we can call it
print(square())
print(square(3))
print(square(4))
```

In the above example the parameter x is defined inside the parenthese of the function, when no input arguments are passed the value of x is set to 3. When the value of x is passed from the main the value is over-written inside the the function call and the function returns the square of the number passed from main.

The first line of the functions above has a string (which is called a docstring) and is optional. Docstring, if used, should contain some useful information about the function, i,e. what does the function do, what are the input parameter, what is returned from the function etc. The user can access the docstring by either using help() function or __doc__ attribute of the object (the definition of attributes and object will follow in the next chapter).

```
def square(x=3):
    "this function returns the square of a number"
    return x**2

#Now that the function is defined we can call it
square.__doc__
help(square)
```

The code block of the function start with a colon sign and the scope of the function is defined by indentation. The output of the function is passed back through the return statement. If the return is used without any arguments in front of it the function does not return any value. Return keyword also indicates that the function has ended it's execution.

Note: To quit Python help for a function press q.

Python **Recursive Function**

A recursive function is a function that call's itself. This is helpful in problem's when there is an inherent structure in the function that we can exploit. Let's take the example of factorial function to explain recursion. The factorial function of a natural number n is defined as $f(n) = n*(n-1)*(n-2)*...*3*2*1 = n*f(n-1)$.

```
def fact(n):
  if n==1:
    return 1
  else:
    return n*fact(n-1)
```

List Comprehension

In Python list comprehension provide a concise way to apply operations to elements of an iterable object. For example if want to square each element of a list or pass it to a function or to check if they meet certain conditions:

```
x=[1,2,3,4,5]
y=[i*i for i in x]
print(y)
print(x)
#print(i)
```

The above code performs operation on each element of "x" in a concise manner.

Dictionary Comprehension

Similar to list comprehension Python also provides dictionary comprehension to perform operations on dictionary elements in a concise manner.

```
dict1={'key1':'val1', 'key2':'val2'}

new_dict={key:val+' new' for (key,val) in dict1.items()}
print(new_dict)
```

The above code access each "key" and "value" in a Python dictionary and appends " new" to the each value of the dictionary.

Scope and Namespace

Try the below code and discuss the concept of scope and namespace with the instructor.

```
x="global"
def fun_scope():
    def fun_change_local():
        x = "local changed"

    def fun_change_nonlocal():
        nonlocal x
        x = "nonlocal changed"

    def fun_change_global():
        global x
        x = "global changed"

    x = "scope_fun scope"
    fun_change_local()
    print("After local assignment:", x)
    fun_change_nonlocal()
    print("After nonlocal assignment:", x)
    fun_change_global()
    print("After global assignment:", x)

fun_scope()
print(x)
print("In global scope:", x)
```

The nonlocal keyword is used to work with variables inside nested functions, where the variable should not belong to the inner function.

Exercises

Exercise 3.1: Write a program that asks the user to input, their name, title, age and occupation and outputs a string, "[title] [name] is [age] years old and works as a [job]."

Exercise 3.2: Write a program that takes a positive integer as an input and return's it's length. Return 0 if the number is not a positive integer.

Exercise 3.3: In a single line of code extract all numbers from a list that are greater than 100.

Exercise 3.4: In a single line of code capitalize the first character of the value strings in a `Python` dictionary.

Exercise 3.5: Write a program that checks if a 5 digit positive integer is palindrome. Bonus: extent it to any positive integer.

Exercise 3.6: Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 4, between 100 and 1500 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line.

Exercise 3.7: Write a code that inputs two integers and returns True if they are anagram of each other, else returns False.

Exercise 3.8: Write a program that takes a number and it's base (<36) and returns the value in decimal system. Assume number greater than 9 are represented by alphabets [a-z].

Exercise 3.9: Text based calculator: Write a program that take's in a string "No1 operator No2". and returns the output of the operation in string. Allowed operations are {+,-,*,÷}.

Exercise 3.10: Write a program that takes an input N and returns fibonacci sequence to N^{th} number in the sequence with and without using recursion.

Exercise 3.11: Write a program that takes a positive integer input N and returns all palindrome's uptill and including N.

Exercise 3.12: Write a program that return's the smallest of the integers in a list.

Exercise 3.13: Write a program that return's the largest number in a list.

Exercise 3.14: Write a program that removes all vowels from a string.

Exercise 3.15: Write a program that take's an input *N* (odd integer) and drawn an isosceles triangle with *N* stars on the base and the other sides of equal size.
Example: N=5

```
  *
 ***
*****
```

Exercise 3.16: Write a program that removes repeated words next to each other from a string.