Day 2:

**Conditional Statements:**

Python has similar loops and conditional statements as in every programming language.

Conditional statements include if, elif and else

>>> if 10 < 20:

>>> #do something

elif is like else if part in other languages.

>>>if 10>15:

>>> #do something

>>>elif 10 < 11:

>>> #do something

If no conditions meet on if/elif loops, then you might want to do something that occurs by default then we go for the else part.

>>>if 10 > 15:

>>> #do something

>>>else:

>>> print(“10 is small”)

Note that you don’t need to specify any condition on else part, whereas elif part requires a condition. You can use else and elif only after the if condition otherwise It’s a traceback error.

Also note that comparison operators (<, >, ==, >=, <=, !=) as well as logical operators are always accompanied in a loop. There are also exceptions where we use mod or boolean.

For eg: >>> if n%2: -> Goes inside if only if n%2 > 0 else it doesn’t

>>> print(“odd”)

>>>else:

>>> print(“even”)

**is operator:**

The is operator is somewhat similar to comparison operator ==. Only difference between them is (==) operator checks if value is equal while (is) checks if both have the same object.

Eg:

>>>x = 2

>>> if 5\*x == 10:

>>> print(“True”)

>>> if 5\*x is 10:

>>> print(“True)

**Loops:**

They include for and while loops. Python doesn’t have do…while loop. The for loop is used mostly when range of input is known. The range function in for loop is a useful function, it returns a list of numbers up to certain number.

For loop syntax:

>>>for i in range(5): -> where range returns the list [0,1,2,3,4] and i which is the iterator takes each one using the in operator

>>> print(i,end=””)

>>>0 1 2 3 4 -> o/p

Here you can observe three things:

range(5) – returns a list of numbers [0,1,2,3,4]

in – checks and takes each value from [0,1,2,3,4]

i – on each iteration i gets assigned (basically the iterator)

We learn more about in operator after covering list data structures.

Suppose you want to iterate in reverse order then we can do the using the range method or using the reversed method (reversed(range(5))

Eg:

>>>for i in range(5,0,-1): -> 0 denotes up to 0 and -1 denotes the amount of decrement

>>> print(i, end=" ")

>>>5 4 3 2 1 - o/p

continue & break statement:

continue statement is used to send the loop back to the iteration part (beginning of the loop). On meeting some condition, you don't need to process the rest of the logic inside the loop then we can use this continue.

Eg:

>>>sum = 0

>>>for i in range(10):

>>> if i > 5:

>>> print("values greater than 5")

>>> continue -> sends compiler to next iteration

>>> i += sum

The break on the other hand is used to exit the loop even before the last iteration is met. On meeting some condition, you don't need to process the rest of the loop and you need to exit the loop then you can go for break.

Eg:

>>>sum = 0

>>>for i in range(10):

>>> if i > 5:

>>> print(sum)

>>> break -> Exits the loop

>>> i += sum

while loop is used whenever the range is unknown. The while loop executes forever and always needs a condition on which the loop should break. Otherwise it may result in overflow/time out error.

while loop syntax:

>>>a = 5

>>>while a>0: -> condition on entering and breaking the loop

>>> print(a, end=" ")

>>> a = a - 1 -> if this line is not provided program runs forever.

>>>5 4 3 2 1 -> o/p

**Data structures – Lists:**

When you want to store a group of values in a single variable then it refers to a list. Lists in python are similar to arrays in other programming languages expect for the fact that lists are dynamic, easy to implement and more powerful.

Syntax for declaring lists:

>>> ls = [] -> empty list

>>> ls = list() -> empty list

>>> ls = [1,2,3,4] -> list with four numbers

Notice the elements in the list are comma separated. List elements can be indexed with [ ] operator. Python lists are zero-indexed.

Lists are mutable while strings are not mutable.

>>>ls[0] = 5

>>>print(ls)

>>>[5,2,3,4]

Looping through a list is similar

>>>for i in ls:

>>> print(i, end="")

>>>[5,2,3,4]

On each iteration i corresponds to ls[0],ls[1],...so on. The len function which we saw in strings can also be used in lists to find the length of list.

If we want more control over the looping process, we can do something like this

>>>for i in range(len(ls)): -> len returns the length/size of the list

>>> print(ls[i]) -> indexing based looping

>>>[5,2,3,4]

The dir(list()) function is used to show the predefined functions in list

There are many predefined helpful functions for lists. Some of them are:

* >>>ls.append(10) -> appends 10 at end of the list
* >>>ls.sort() -> used to sort the lists in ascending order
* >>>ls.pop() -> used to remove the last element from list and also returns it

The string slicing which we saw before can also be applied to lists

For eg

>>> ls[-1] -> returns last element of list

>>>ls = [5,2,3,4]

>>>ls[1:3] -> slices from 1 to 2

>>>[2,3]

**in operator:**

The in operator is one of the beautiful things in python. You can check whether an element is present in a list or not easily using the in operator. It's return type is boolean.

For eg:

>>>if 2 in ls:

>>> print("present")

>>>if 10 in ls:

>>> print("Not present)

>>>if 7 not in ls: #not in checks if it's not in list

>>> ls.append(7) #inserting 7 inside list

Initializing a list with fixed value can be done by : >>> ls = [2] \* 5 ->o/p [5,5,5,5,5]

For building a 2d list in python:

>>>rows , cols = 5,5

>>>ls = [[0]\*rows]\*cols

>>>#indexing

>>>ls[0][1] = 2

[0]\*5 - >results in [0,0,0,0,0]

[[0]\*5]\*5 ->results in [ [0,0,0,0,0], [0,0,0,0,0] ,[0,0,0,0,0] ,[0,0,0,0,0] ,[0,0,0,0,0]]

There are several ways of constructing lists ,I have specified only one type

Python lists by default can have any data type inside it, it can also have a list inside a list as we saw above with 2d lists.

>>>ls = [1,2,"pineapple", "1" ,[1,2]]

Notable tips:

* max(ls) gives the max element in a list and vice versa for min(ls)
* sum(ls) gives the sum of all elements in a list

**Data Structures - Dictionary:**

Dictionaries are a special kind of data structures that follow key-value pattern. In list we used indexing to get the value of a particular item in the list whereas in dictionary we have a key which maps to a value. Using the key we can retrieve the value of it.

This is similar to Hashmaps in java and map in c++. The order inside a dictionary can be any order.

Eg:

>>>dt = {} #different ways of declaring a dictionary

>>>dt = dict()

>>>dt["basketball"] = "NBA" #here "basketball" is the key it's value is "NBA"

The key in dictionary should be unique. The datatype of key/value can be anything. Dictionaries are widely used data structures for various problems.

Similar to lists, dictionaries have variety of predefined functions like

dt.keys() - returns the keys present in dictionary as a list

dt.values() - returns the values as a list

dt.items() - returns both keys and values in the form of tuples

Values of dictionaries are mutable. The values can be accessed in O(1) time.

get() method:

Since dictionaries require unique keys, we can easily encounter a traceback error whenever we try to find a key that doesn't exist. To avoid such scenarios we use the get method.

Eg:

>>>dt[element] = dt.get(element,0) -> searches for a key named element,if not there then assigns its value to be 0

The below code is similar to the one return above

>>>if element is not in dt.keys():

>>> dt[element] = #some value

Looping through a dictionary:

Similar to what we saw in lists

>>>dt = {"basketball":"NBA","cricket":"ICC","football":"FIFA"} ->declaration of dict

>>>for i in dt: -> by default i corresponds to keys in dictionary

>>> print(dt[i]) -> if we index i (keys) then we get the values

If we want to loop through values then,

>>> for value in dt.values():

>>> print(value) -> each value corresponds to values

One feature that no language supports is the double iteration

>>>for key,value in dt.items(): -> Here you have two iterators

>>> print(key, value) one for iterating keys and the

other for iterating values

sorted method is used for sorting , by default it sorts in ascending order

>>>for key in sorted(dt.keys()): -> keys are sorted in ascending order

>>> print(key,dt[key])

Notable tip:

* If you need a dictionary with duplicate keys (more than one distinct key) you can use defaultdict from collections package.

**Data structures - Tuples:**

Tuples are sequence of values much like lists . The values in a tuple can indexed using the indexing operator [ ] similar to lists . The only difference between lists and tuples is that tuples are immutable.

Why use them ?

Tuples are much more efficient than lists and the space taken by tuples are also less than lists.

A tuple can be declared as

>>> t = ()

>>> t = (1,2,3) ->notice the parentheses

>>>t[0] ->indexing

>>>1

>>>t[0] = 2 ->error cannot be changed

We can compare tuples using comparison operators

>>>( 1, 10 , 20) < (2 , 3, 4)

>>> True -> this is true since 1 < 2 and it doesn't check for other values

Suppose you want to sort the dictionary according to value then

>>>dt = {"a":1,"b":2,"c":3}

>>>ls = list()

>>>for key,value in dt.items():

>>> ls.append((value,key)) #(value,key) tuple is formed

>>>ls =ls.sorted(ls,reverse=True) #sorting the list of tuples in descending order

Notable points:

* You can't add or delete elements from a tuple.
* Tuples can be used as dictionary keys.
* Tuples are faster than lists.
* If you write protected data that needs to be changed, you can go for tuples since it's constant.

Assignments for day 2:

* Given two numbers a and b , you can add a by 1 in each move . Find the minimum number of moves required to make a divisible by b.

i/p: o/p:

a = 10 b = 5 2

a = 13 b = 9 5

a = 92 b = 46 0

* Given a email string like "From: jack@lntinfotech.com Mon 17:58:00" extract the day of the week
* Given a list ls = [2, 1, 5, 8, 9, 3] calculate the sum of elements between given range i and j where ans = sum[elements up to j] - sum[elements up to i]

i/p : the first line contains the size of the list

the second line contains the list

the third line contains i and j and j>i

i/p :

6

[2, 1, 5, 8, 9, 3]

2 4

o/p:

17 *(sum of elements up to j is 25) - (sum of elements up to i is 8)*

* Given a string , sort the words inside the string from longest to shortest

i/p: "Those days were bright and shiny"

o/p: ["bright","shiny","Those","were","days","and"]

* Given a text file find the top 5 uncommon words in it