Modules and pip

**Modules** = Using someone else written code

2 types of modules:

1. Built in modules
2. External modules

**Built in modules** who modules hote joh phele se python ke andar hote like we don’t really need to download it from any source.

But **external modules** are those modules that we do need to install externally using pip or any other source and that code is written by some other developer.

**Pip** is used to download external modules , it is a package manager.

When you enter python or python3 in a shell, a repl of python opens up which means Read Evaluate Print Loop.

Variables and Data Types

Using type function you can extract the type of a variable. **type()**

Numeric Data: int[2,-3],float[1.4,-2.4],complex[6 + 2j] we can create complex number using **complex(6,2)**

Text Data: str[“Hello world”, “Python Programming”]

Boolean Data: [True,False]

Sequenced Data:[list,tuple] list are mutable and tuple are immutable.

Mapped Data:[dictionary] key value pairs. Int,float,bool,tuple can be the keys but not list or dictionary because keys must be a unique identifier means immutable data type we have to use and lists,dict are mutable.

**In python everything is an object**. Dict is an object Int is an object everything is an object and you can see that using type function, you will see that every data type is a class object.

Typecasting

Also known as type conversions.

Two types of typecasting:

1. Explicit Conversion: The conversion which is done via developer. Like int(a) + int(b) etc
2. Implicit Conversion: The conversion that automatically handles by python, like adding a integer and a float will be result in a float number. Like 3+3.1 will be result in a float , float + int equals to float , int + float equals to float and float + float equals to float. This happens because data types in python do not have the same level i.e. ordering of data type is not the same in python. It is handled by the interpreter automatically.

Inputs

**input() method**

**Strings**

**Single line** strings: print(“hello buddy, ‘Hows going’”)

**Multiline** strings: using triple quotes

“””” Once upon a time,

Ishowspeed died

“”””

Strings are like an array of characters like we can access its characters using indexes but its not an actual array of characters its like an array of characters.

For character in name:

Print(character) # prints out all the characters in the name container/variable

String Slicing

[start:end]

**Len()** function to print out the length of a variable

Slicing runs end-1 , start from starting point

Negative slicing: -1 [len(var) -1] etc

String methods

Set of built in methods used to modify strings.

Concept of immutability: strings are immutable. We can change them like when we use upper() function it does not change the original string infact it returns a copy of that string.

Upper(), lower(), rstrip() removes trailing characters, lstrip() removes leading characters, if you want same trailing and leading characters to be removed you can use strip function.

Replace() it replaces all occurances of a string with another string.

Split() method split the elements on the basis of what you pass in it and returns a list.

Capitalize() it capitalize the first letter and turn all other capital letters to a lower case. Mostly used while publishing a blog.

Center() method aligns the string to the center as per the parameters given by user. Center(50,”.”)

Count() method returns the number of times the given value has occurred within the given string.

endswith() method checks if the string ends with a given value. If yes then return true else return false. We can also provide start and ending index to check on a specific line like endswith(“z”,4,50) etc

Same goes for startswith() which checks if the string starts with a given value or not.

find() method searches for the first occurrence of the given value and returns the index where it is present. If it doesn’t find any then it will return -1. It will not throw any error.

Index() method is same as find but if it doesn’t find any then it will throw an error and program will quit. We can use this when we are sure that it is important to find that occurrence then we can use that.

isalnum() method returns true if the entire string only consists of A-Z, a-z, 0-9. If any other characters or punctuations are present, then it returns false.

isalpha() method returns true if the entire string only consists of A-Z or a-z. this time no 0-9

islower() returns true if all the characters in the string are lowercase, else it returns false.

isprintable() returns true if all the values within the given string are printable, if not then returns false.

isspace(): returns true if the string contains only white spaces, else return false.

istitle(): returns true only if the first letter of each word of the string is capitalized, else it returns false.

Isupper() same as islower()

Swapcase() will swap case the string like upper to lower lower to upper etc.

Title() will convert a string into a title like capitalizing each word’s first letter.

Match Case statements

Match case statements are just like switch case statements but we don’t use break in match case statements and match case statements are added towards python 3.10 so that can be a question asked in an interview.

*match x:*

*case 2:*

*print("it is 2")*

*case \_ if x>2:*

*print("number is greater than 2")*

*case \_:*

*print("Im default case")*

we can also set if statements by using \_ and also using \_ we can declare default case just like in switch case.

Using else with while loop

What it does is that as soon as the while loop condition becomes False, the interpreter comes out of the while loop and execute the else condition.

*while(x<5):*

*print(x)*

*x+=1*

*else:*

*print("im inside else")*

Now if we talk about do-while loop in python there is no such thing as do while loop in python but we can emulate this. Do while loop executes one time whether condition is true or false it doesn’t care as we know. Do while loop are also known as an exit-controlled loop.

That’s how we can emulate the do-while loop in python. Following code showcases that:

*while True:*

*number = int(input("Enter a positive number: "))*

*print(number)*

*if not number > 0:*

*break*

By doing so when we enter negative number it will print the number and then exit it like demonstrating how the do-while loop executes for at least one time.

Break and continue

We can use break statements when we want to break out of the loop, we use continue statements when we want to break out of the iteration.

We use else with while loop when we want else statement to execute only when the loop is successfully ended without breaking out of it. But when you break out of the loop, else condition below it will not execute.

x = 5

while(x<10):

    print(x)

    x+=1

    if x==8:

        break

else:

    print("Im in else")

Functions Arguments

There are four types of arguments:

1. Default arguments
2. Keyword arguments
3. Variable length arguments
4. Required arguments

Required arguments:

*def average(a,b):*

*return (a+b)/2*

Required arguments are required to must be given to a function.

Default Arguments:

*def average(a=2,b=4):*

*return (a+b)/2*

Even value is not provided while invoking a function, it will take default values to process the function on.

Keyword Arguments:

We can provide arguments with key=value, this way interpreter recognizes it by the name and it can be in any order.

*def average(a=2,b=4):*

*return (a+b)/2*

*print(average(b=22,a=21))*

Variable length Arguments:**important**

def average(\*numbers):

    return sum(numbers)/len(numbers)

def sumDatasets(\*datasets):

    print(datasets)

    for dataset in datasets:

        print(sum(dataset))

sumDatasets([1,2,3],[4,5,6])

print(average(1,4,5,2))

variable length arguments are used when you are unaware of how many arguments you will get , so using **\*varname** will accept a bunch of numbers or data in a form of iterable tuple and then operate an operation on it.

Double star argument **\*\*formData**

def formField(\*\*formData):

    print(type(formData))

    print('Second print statement: \nClient name: ',formData['name'],'$, Client Age:',formData['age'])

# formField(name='John', age=25)

formField(name='Jane', age=22, uni='Iqra University')

in here we are taking formData as a dictionary and it is used when we want to accept variables as dictionary.

List comprehension

List comprehension means creating a list on the fly like based on some conditions we are creating a list like below:

lst = [i for i in range(1,100)]

lst

lst = [i for i in range(1,100)]

lst2 = [i for i in lst if i%2==0]

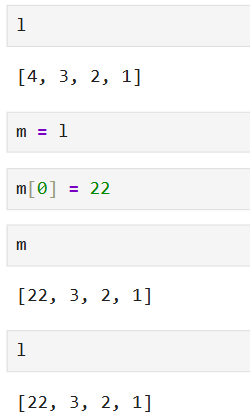
# now lst2 contains even numbers from 1-100 range

List methods

Lists are mutable sequenced data type.

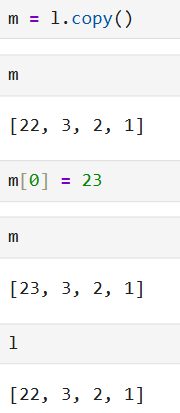
.append() , .sort(reverse=False), reverse(), .count(), copy() will be used in copying a list to another variable so it doesn’t affect the original list.

Copy() method usage:



As you can see that it affects our original list called “l”

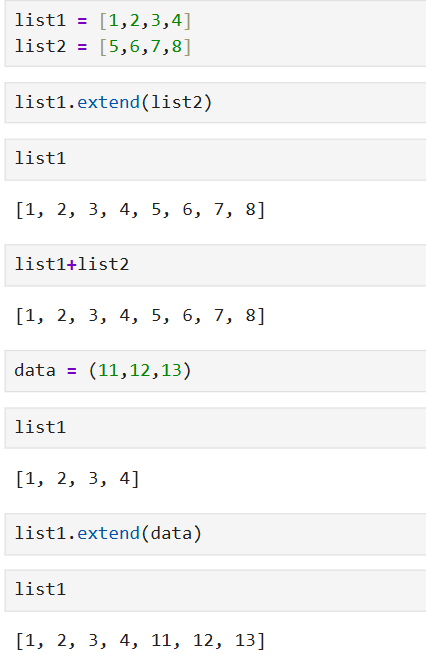
So we don’t want to interfare with original list while we want to copy it to another variable so what we can do is use a copy() function.



Insert() method is used to insert an element at specific index and pushes other elements to forward.

Insert (index,element)

Extend() method adds an entire list or any other collection data type(set, tuple, dictionary) to the existing list.



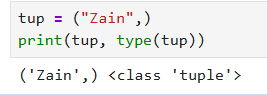
By using extend method you don’t need to create another list it modifies the original list on which you are calling the extend function, also you can pass tuple or dictionary but on dictionary when we pass it in extend it will take only key and store it inside a list. Value will be dumped.



Pop() method also used to remove the elements, in that we pass the index of element to remove from list.

Tuples

In tuples when you add one element like (1) , you will see its output that its type will be considered as int. same goes for one string when you add in a tuple it will be considered as string when you check the type. So it always be a good practice to leave single element with a comma like (1,) or (“zain”,)

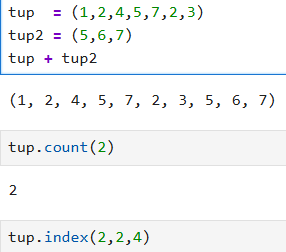


We can access elements using indices and same slicing method can be applied on tuple as we applied on list.

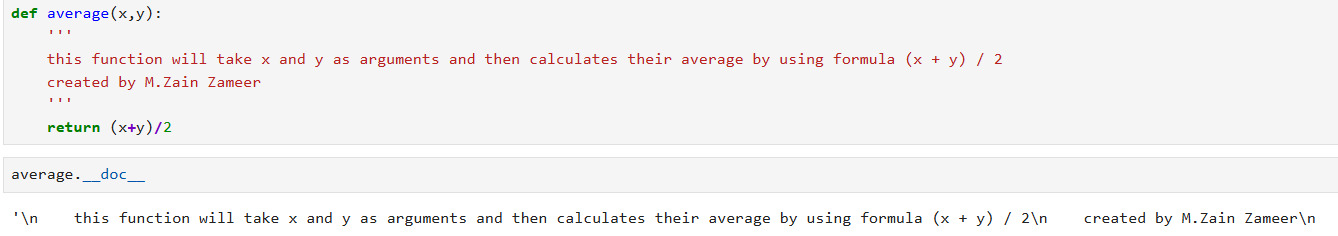
Concatenation is same as we did for list.

Count() method is used to check occurannces of element in a tuple.

Index() method will give first occurrence of an element from a tuple , if it doesn’t find any then it will raise a value error. We can also give starting and ending index to index function to look for a target in a particular range.



Doc-strings



Doc strings are used to write description in functions to describe what the function is doing.

PEP-8

It provides best practice on how to write clean code for readability. Python Enhancement Proposal and Pep-8 is basically a document and it was written in 2001.

The ZEN of Python

By using “import this” you can see a poem of python on what goods for python etc.

Recursion in python

Function inside a function.

Sets

Sets are unordered collection of data items. Sets are immutable. Sets only contain unique values.

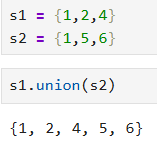
{1,2,3}

Set can contain multiple data types elements.

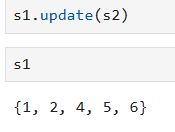
To create an empty set, use set() not {} as it will consider it as dictionary not set that’s why.

Sets methods

.unique() method

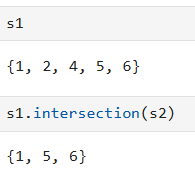


When I applied union function, s1 and s2 remain untouched so if I want to really update that s1 I can use .update() method.

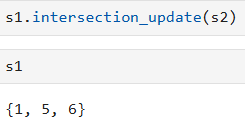


Now my s1 got affected.

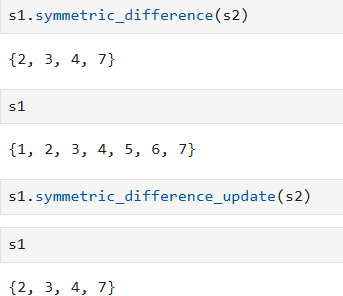
.intersection() method:



It will return the intersection but not gonna update s1 here , so if we wanna update s1 we can use .intersection\_update() method



Symmetric difference means A U B – A N B, like extract all those elements that are unique in both sets not intersect elements.



.difference method:

It will return all those elements that are not same in both sets. It will return only those elements that are not in set B , what symmetric was doing is it was making a union of set and then subtracting the intersection.

s1 = {1,2,7,8}

s2 = {10,2,12}

print(s1.symmetric\_difference(s2))

print(s1.difference(s2))

run this to see the difference.

There are several built in methods used for the manipulation of sets.

**isdisjoint()**: two sets are going to be said disjoint when there are no elements are common between them.

s1 = {13,33}

s2 = {1,2}

print(s1.isdisjoint(s2))

**.issuperset()**: it checks if all the elements of a particular set are present in the original set.

s1 = {1,2,3,4}

s2 = {1,2}

print(s1.issuperset(s2)) #true

s1 = {1,2,3,4}

s2 = {1,2,44}

print(s1.issuperset(s2)) #false

**issubset():** Reverse of superset ,

s1 = {1,2,3,4}

s2 = {1,2}

print(s2.issubset(s1)) # True

**.add():** It lets you add a single item to a set.

s1 = {1,2,3,4}

s1.add(5)

print(s1) #{1,2,3,4,5}

.**update():** If you want to add more than one item to a set you can use update method.

s1 = {1,2,3,4}

s1.update({1,2,3,41})

print(s1) # {1,2,3,4,41}

**.remove() / discard() :** It lets you delete an element from a set.

s1 = {1,22,3,4}

# s1.discard(22) # discard will not raise an error when element is not found

s1.remove(22) # remove will raise an error when element is not found

print(s1)   # {1, 3, 4}

**.pop()** will remove the element from end but we don’t know which element it will remove because it is unordered data type.

.**del keyword:** which deletes the set entirely

del s1

print(s1) # throw an error that s1 is not defined.

What if we don’t want to delete the set we just wanted to delete the elements in it we can use **clear()** function.

s1 = {1,2,22,3,4}

s1.clear()

print(s1,type(s1))

Dictionaries

Dictionaries are ordered.

Accessing element in dictionary:

d = {

    "name": "John",

    "age": 30,

}

print(d["naame"]) # it will throw a keyerror if key not found

print(d.get("age")) # it will not throw any error if key not found

Accessing multiple values:

Like keys:

d = {

    "name": "John",

    "age": 30,

}

# getting keys

print(d.keys()) # dict\_keys(['name', 'age'])

also what can I do is following iterating over keys:

d = {

    "name": "John",

    "age": 30,

}

# iterating over keys

for element in d.keys():

    print(element)

same as keys() I can use values() function to get the values or iterate over the values only.

Another way of accessing is using .items() method , what it will does is it will return you the key-value pairs

d = {

    "name": "John",

    "age": 30,

}

print(d.items()) #dict\_items([('name', 'John'), ('age', 30)])

now when it comes to iterating you can do something like this, accessing key and value at the same time,

d = {

    "name": "John",

    "age": 30,

}

for key,value in d.items():

    print(key, value)

'''

name John

age 30

'''

Dictionary Methods

Update(): it will update the elements of ep2 in ep1 like adding all the elements of ep2 to ep1 and which one is overlapping just replace it with the new one.

ep1 = {1:10, 2:20, 3:30, 4:40, 5:50, 6:60}

ep2 = {1:70, 8:80, 9:90, 10:100, 11:110, 12:120}

ep1.update(ep2)

print(ep1)

# {1: 70, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60, 8: 80, 9: 90, 10: 100, 11: 110, 12: 120}

Clear(): to clear a dictionary

ep1 = {2:20, 3:30, 4:40, 5:50}

print(ep1) # {2: 20, 3: 30, 4: 40, 5: 50}

ep1.clear()

print(ep1) # {}

pop(): removes the key-value pair whose key is passed as a parameter.

ep1 = {2:20, 3:30, 4:40, 5:50}

ep1.pop(3)

print(ep1) # {2: 20, 4: 40, 5: 50}

popitem(): removes the last key-value pair in the dictionary.

We can also use the **del** keyword to remove a dictionary item.

Day 35 continue