**pwd**: PRINT WORKING DIRECTORY

जिस folder मे है उसे दिखने के लिए

ls: list directory

folder के सभी file दिखने के लिए

clear: screen clear करने के लिए

cd: change directory

eg- cd F://

mkdir: make directory

folder create करने के लिए

eg- mkdir “python tutorials”

note: folder name always put in double corse “”

touch: file create करने के लिए

eg: touch hello.py

cd..: for go back in previous folder

rm: for remove file

eg: rm hello.py

rm –rf: for remove folder

eg: rm –rf “python tutorials”/

note: always put / in last to remove folder

cd ~ for go back to previous folder

code <file name> to start code in that file

mv: for rename file (this is also use for move file )

eg: mv <old file name> <new file name>

For move file in folder in same folder

Eg: mv file.py ./new\_folder

For in previous directory

mv file.py ..

cp: for copy file

For copy file in folder in same folder

eg: cp file.py ./new\_folder

note: up arrow also used for run written code

Print function

Eg print(“hello world”)

**Strings: collection of characters inside “Double quotes” or ‘ Single quots’**

In python we can use single or double quotes but in some of the programming language we can’t use single quotes in it to represent the string.

We can use ‘single quotes’ inside “double quotes” and vice versa. But we can’t use ‘single quotes’ inside single quotes and same apply for double quotes. This will cause syntax error.

Also we don’t use print function like **Print(‘ I’m pintu raj’)** here I’m will create syntax error

Escape Sequence

Since we can’t use ‘single quotes’ inside single quotes and similarly for double quotes. So to make it possible escape sequence will us in it

To use ‘single quotes’ inside single quotes we use with backward slace

For eg: **print (“hello \”world \” world”)**

escape sequence also use to change line by using \n

eg: **print(“ line1\nline2”)**

\t for tab means to make more space b/w them

eg: **print(“ line1\tline2”)**

Shortcut: we can use any escape sequence as a normal line by simply using r at starting

eg: **print(r“ line1\nline2”)**

To print emoji

We use website: unicode.org

From that website we copy that unique code and print in by replacing + sign three zero and back slash at starting

Eg: print(“\U0001F602”)

TO USE PYTHON AS A CALCUATOR

To use python as a calculator we use print function

|  |  |  |
| --- | --- | --- |
| Operators | Description | Example |
| + | Addition | Print(2+3)  5 |
| - | Subtraction | Print(2-3)  -1 |
| \* | Multiplication | Print(2\*3)  6 |
| / | Float division | Print(4/2)  2.0  Print(2/4)  0.5 |
| // | Integral division | Print(4/2)  2  Print(2/4)  0 |
| % | Modulo, it gives remainder | Print(6%2)  0 |
| \*\* | exponent | Print(2\*\*3)  8 |

We can use round function to round the number

Eg: print(round(2\*\*0.5,4))

Here 4 is use to round the decimal upto 4 digit

In normal we use BOADMAS Rule to solve the problem but in python it use Precedence rule

|  |  |
| --- | --- |
| OPERATORS | PRECEDECNCE AND ASSOCIATIVITY RULE |
| PARANRHESE () | HIGHEST |
| EXPONENT | RIGHT TO LEFT |
| \*,/,//,% | LEFT TO RIGHT |
| +,- | LEFT TO RIGHT |

VARIABLE

In programming we need some space to store data that’s why we use variable

For eg if we are making game in which olny 13 year old child can play then we need to ask the age of the user then we store that age in variable after that we check the condition whether is he eligible or not

So let’s us know how define variable

Eg. Number1 = 2

Print(number1)

Then the output will be 2

Similarly we can change the variable

Data that can be store in variable

We can store any data type like string, floating point number, integer number or any number and some special type of data structure (we will discuss later)

For eg. name = “Pintu”

Print(name)

The output will be Pintu

There is some rule of using variable

1. Variable can’t be start with number means can’t use like 2number = 4 but can use as n2umber = 4
2. We can’t use any special character in variable except \_

Concatenation

This is the process to adding the different value in one value

Eg: first\_name = pintu

Last\_name = raj

Full\_name = first\_ name + last\_name

Print(full\_name)

Note: here we can’t add any number to string

Eg: print(first\_name + 3)

This will give error

But we can add number by making it string

Eg: print(first\_name + ‘3’) or print(first\_name + str(3))

We can not add string with number but we can multiply string with number

Eg: first\_name = pintu

Last\_name = raj

Print(first\_name\*3)

This will gives output as pintupintupintu

Now we will know how to get input from user

To get input from user we use input function

Note: input function always takes input in string

Eg: name= input (“type your name”)

For calculation any number

eg: number\_ one = input( “enter first number”) say 4

number\_ two = input(“enter second number”) say 4

total = number\_one + number\_two

print(‘total is’ + ‘total ) -> this will give the result as 44

but we want as summation of that number but since it was a string so they simply show as 44

for make it as calculation we use int function

eg: number\_ one =int( input( “enter first number”)) say 4

number\_ two = int(input(“enter second number”)) say 4

total = number\_one + number\_two

print(‘total is’ + str(total))

this will the desired result as summation

note: here we have to use str() to make it string function.

How to make more variable in one line

To make more variable in one line we can do in following ways

Name , age = “Pintu”, “21”

Print(“your name is” + name “your age is” + age)

Note: here we use 21 in string because name is string and 21 is integer and we can’t add both so to make it string we had done.

This can also be done as

Name , age = “Pintu”, 21

Print(“your name is” + name “your age is” + str(age))

Glance at .split() function

Let’s the code is

Name, age = input(enter your name and age).split()

Print(name)

Print(age)

In a simple word we can say that .split() is splitting the input name and age into two separate input

Note : we need the put some space b/w name and age

If we want to separate by comma then we have to use .split(‘,’) function

STRING FORMATTING

Previously we add different input as

Name = input(enter your name: )

Age = input(enter your age: )

print(‘your name is’ + name + ‘your age is’ + str(age))

But we can do in another way too

Name = input(enter your name: )

Age = input (enter your age:)

Print(‘your name is {} and your age is {}’.format(name, age)) for python 3 and later

Also in python 3.6 and later we can also format as

Print(f’your name is {name} and your age is {age}’)

Here we don’t have to think about whether age is in string or in integer value

Also we can do calculation in this format as

Print (f’your name is {name} and your age is {age +3}’)

Or print(‘your name is {} and your age is {}’.format(name, age + 3)

String indexing

String indexing is basically printing the letter of any variable.

Note: In python position always start with 0. And for writing the index number we always use square brackets []

Eg: variable = ‘python’

|  |  |  |
| --- | --- | --- |
| P | 0 | -6 |
| Y | 1 | -5 |
| T | 2 | -4 |
| H | 3 | -3 |
| O | 4 | -2 |
| N | 5 | -1 |

And if we want to print the letter t then,

Print(variable[2]) this will give the outcome t

For very large word we don’t know how many world are there but I have to print last 3rd word then we can print in following way

Print(variable[-3]) this will give outcome as h.

Selecting sub sequences or slicing

In string indexing we print only one letter but if we want to print more than one letter then we use slicing

Syntax [starting argument : stoping argument -1]

Taking the previous example

Let’s suppose we want to print the “tho” in the variable python, then

Print(variable[2 : 5])

Note: we can also give the position with negative number.

If I don’t write any argument then we will get complete word

Eg: print(variable[ : ]) this will give whole word ‘python’

If we don’t give stop argument then this the result gives from where arguments start to last word of the word and similarly for the start arguments

Step argument

This is basically jumping the step in the argument

Eg: for the word

Variable = python

Print([2:5:1] here 1 is mean to jump to the 1 to the current word

For print([2:5:2] meaning jumping to the 2nd word to the current word

Print([2:5:1]) gives the outcome as ‘tho’

Print([2:5:2]) gives the outcome as ‘to’

Note : In the whole string formatting we use directly the string in print function

Eg: print(“python”[2:5])

Print(“python[2:5:2])

Note: if we put -1 to step argument then step starting going back

Eg: print(python[5:2:-1]]) This will give the output as noh

Basically negative sign for jumping backward

If we only do as

Print(python[::-1] this will gives as reverse order of name as nohtyp

String method

|  |  |
| --- | --- |
| Function | Uses |
| Len() | It count the no of character in the string |
| Lower() | For the string written in upper or lower case it finally convert into lower case |
| upper() | Convent all character into capital letter |
| Title() | Conver 1st character into capital of any word |
| Count() | This will count the no of time the character is repeated. This is case sensitive. |

Note: In method we always use dot

Eg: for len function

Print(len(‘Pintu’)) this will count the no of character in the word Pintu

And the outcome is 5

Note: len function also count space as a character

For the word Pintu Raj it will give the outcome as 9 as one character space as counted

Eg: for lower method

Name= PInTu Raj

Print(name.lower())

This will give the outcome as pintu raj

Eg: for upper method

Name= PInTu Raj

Print(name.upper())

This will give the outcome as PINTU RAJ

Eg: for title method

Name= PInTu raj

Print(name.upper())

This will give the outcome as Pintu Raj

Eg : for count method

Name= messIsSipi

Print(name.count(‘s’)) note: s should be in string

This will give the outcome as 3

Note: we can use two or more method at same time

For eg: name = input("Please enter your name: ")

char = input("Please enter any character : ")

print(name.lower().count(char))

here name. lower() first make the name input in lower case then count(char) counting the character

Problem of space solving with strip method

#This method is used for removing spaces for any word. This method is called strip method

name1 = '       Pintu          '

print(name1) #This will gives name as written in string with space

# But if we put strip() method then spaces will removes

# for left side we use lstrip()

print(name1.lstrip())

#For right side we use rstrip()

print(name1.rstrip())

#for both side we use strip() method

print(name1.strip())

#Note: strip() method can't remove spaces in the middle of the word

Find and Replace method

Replace method:

This is used to replace variable

name = 'This is my first text on replace variable and i want to change all the spaces with underscore'

print(name.replace(' ', "\_",5))# HERE  5 is to no of space from starting we want to replace

#replace can also replace word too

print(name.replace('first text', '1st TEXT'))# here if first text is written more than one time then

#we can also change the how many first text i want to change

# for eg:

variable = 'this is my first car and that is my second car. Is there any other car?'

print(variable.replace('is', 'was'))# but here outcome is

# thwas was my first car and that was my second car. Is there any other car?

#replace is case sensitive and this  is also change to thwas

Find method:

#find method used to find the position of the character in the string

#find method give the integer value

#for eg:

variable = "This is the my first car."

print(variable.find('is', 3)) #here 3 is the starting point of character from where it has to find the character.

#the output of the above script is 5

Center method:

#this method is used to add any character to the left or right of the any string

#for eg:

name = 'Pintu'

print(name.center(len(name) + 4,"#"))

#here len(name) is used to automatically detect the length of name and then we add 4 to make the length

#of string to be print and after the comma we write the things  that we want to add in the variable name in string

Note: Strings are immutable means we can’t change the string but we can print any change in the string by replace method, [] etc.

Some more operator

name = 'Pintu'

name = name + 'Raj'

print(name) # this will give the output as PintuRaj

name = 'Pintu'

#we can also in this way

name += "Raj"

print(name)

name = 'Pintu'

#we can also in this way

name += "Raj"

print(name)

#we can add, sub,multiply

#eg:

age = 20

age += 5

age -= 4

age \*= 2

print(age)

|  |  |
| --- | --- |
| Symbol | Meaning in if else elif statement |
| > | Greater than |
| < | Smaller than |
| >=  we can’t write as => this will give syntax error | Greater than or equal to  (Similarly we can say for smaller than or equal to) |
| == | Equal to |
|  |  |
|  |  |
|  |  |

if statement

if statement is used to check the condition whether the condition is fulfilled or not.

age = int(input(('Please enter your name: ')))

if age >= 14:

    print('you are above 14')

don't forget to put colon in if conition otherwise we  will get error

#and also give some spaces( means Indentation)

#although after giving colon indendation will automatically give

Pass statements:

In case, we don’t want to write anything in the blog of if statement then we simply write pass statement

For eg:

name = input("Please enter your name: ")

if name:

    pass

here we made if statement and not given any condition then if we don’t write pass statement the then we will get error message

simply if statement is for to just pass the if condition without giving any condition to if statement

In real life we need pass statement

Else statement:  
else statement is used after if statement

Else meaning in programing is if if statement is not fulfilled then what condition do program will impliment

It is also used after elif statement

Eg:

age = int(input("Please enter your AGE: "))

if age > 14:

    print('You are eligible. Thankyou!')

else:

    print("SORRY!! You are not eligible.")

elif statement:

elif statement is basically another if statement meaning if 1st if statement is not fulfilled then the other if statement will written with elif statement

eg:

age = int(input("Please enter your AGE: "))

if 0<age<=3:

    print("Ticket Price: Free")

elif 3<age<=10:

    print('Ticket Price: 100')

elif 10<age<=60:

    print("Ticket Price: 260")

else:

    print('Ticket Price: 200')

we can also make if statement by making if statement again and again

but the problem is if we make if statement more than one time then it will check for each time meaning if statement create always separate blog and elif is under if statement if we use if statement inside else statement then it is called nested if-else statement

eg:

winnig\_number = 27

user\_input = int(input('Guess any number: '))

if user\_input == winnig\_number:

    print('You won!!')

else:

    if user\_input > winnig\_number:

        print('too high')

    else:

        print('too low')

and , or operator:

and, or operator is used to check two condition simultaneously

eg:

name = 'abc'

age = 19

if name == 'abc' and age == 19:

    print( 'conditon true')

else:

    print('condition false')

if name == 'abc' or age == 19:

in keyword

in keyword is used to check the the character in the string

for eg:  
name = 'Pintu Raj'

if 'a' in name:

    print('a is present in name')

else:

    print('not present')

note: in keyword is case sensitive

similar for not in :

not in keyword is used to check the the character does not in the string

in keyword has more use in loops data structure and many more.

Check empty or not syntax:

If input is given then the outcome will hello <name> and if not then the output will be you didn’t type anything

This syntax check with if statement

name = input('Please enter your name : ')

if name:

    print(f'Hello {name}')

else:

    print('You didn\'t enter anything')

while loop:

while loop is used to print any variable desired no of times

for eg:

i = 1

while i < 10:

    print('hello world')

    i = i+ 1

infinite loop:

while True:

    print('hello world')

for loop:  
for loop is same as while loop with simple syntax

for eg:

for i in range(10):

    print(f'hello world {i}')

this will give the output as

hello world 0

hello world 1

hello world 2

hello world 3

hello world 4

hello world 5

hello world 6

hello world 7

hello world 8

hello world 9

for i in range(5,10):

    print(f'hello world {i}')

this will give the output as

hello world 5

hello world 6

hello world 7

hello world 8

hello world 9

Break keyword:

This keyword is used to break the loop

For eg:

for i in range(1,10):

    if i == 5:

        break

    print(i)

this will give the output only

1

2

3

4

Continue keyword:

This is used for skip the thing that wanted to skip

for i in range(1,10):

    if i == 5:

        continue

    print(i)

this will give the output as

1

2

3

4

6

7

8

9

Step argument in for loop :

This meaning if we want to print like that

for i in range(0,10,2):

    print(i)

This will give output like

0

2

4

6

8

How to use for loop with string?

We already have use for loop with string but here we will see special in python

Used method:

name= 'Pintu'

for i in range(len(name)):

    print(name[i])

this method used in other programming language too

new method:

name = 'Pintu'

for i in name:

    print(i)

eg2:

num = input('Please enter any number: ')

total = 0

for  i  in num:

    total+= int(i)

print(total)

the output of this is : the sum of all the numbers

here for i in num meaning whatever the value of i in num

Function :

Function is used to define any formula to any task

For eg:

def addition(a,b):

    return a+b

#here function is created with help of def this function will take

two variable

#for eg:

num1 = int(input('Please enter any number'))

num2 = int(input('Please enter any number'))

total = addition(num1,num2)

print(f"The sum of the two number is : {total}")

print vs return:

in making any function we can use either print or return

in print if we call the fuction the function will print the output

but in return function if we call the function, the function will just do what we have define but don’t give output.

Note: we can use any predefined function inside any function.

Eg:

def addition(a,b):

    return a+b

num1 = int(input('Please enter any number'))

num2 = int(input('Please enter any number'))

num3 = int(input('Please enter any number'))

def mult(a,b):

    return a\*b

print(mult(addition(num1,num2),num3))

note: the a &b inside the addition function has nothing relation inside the function of mult

**Note: if we use comma separated in print function then it will give the output with space separated**

a = 0

b = 1

print(a)

print(a, b)

the output is 0

0 1

Fibonacci series

This is the summation of last two number of series . for eg

0 1 1 2 3 5 8 13 21 34

Default parameter:

This means setting some value in making a function

For eg:

def user\_info(first\_name, last\_name, age):

    print(f'your first name is {first\_name}')

    print(f'your last name is {last\_name}')

    print(f'your age is {age}')

user\_info('Pintu', 'Raj', 21)

#Now the default parameter meaning is to making any value to default

# like for eg let's make the age as default at 20

def user\_info(first\_name, last\_name, age = 20):

    print(f'your first name is {first\_name}')

    print(f'your last name is {last\_name}')

    print(f'your age is {age}')

# this will give the output as

# your first name is Pintu

# your last name is Raj

# your age is 20

There are some rules of default parameter:-

1. We can overwrite the default parameter in further call of the defined function

for eg if we had made age = 20 as a default parameter and we call the function and then we write the age as 21 then the output will we 21.

1. We can make default from right side parameter
2. def user\_info(first\_name, last\_name = "unknown", age):
3. print(f'Your first name is {first\_name}')
4. print(f'Your last name is {last\_name}')
5. print(f'Your age is {age}')

this will give error as output

Variable scope:

Basically meaning is we can’t use any function variable in other function as a variable

For eg:

def func():

    x = 7

    return x

def func2():

    print(x)

# meaning in func2 we can't print the value x in func2 as it defined in func()

# to use the value of x from fucn() we have to make it as a global variable

#and neither we can do like as

print(x)

x = 7

def func():

    return x

# here we can't use the value of x = 7 inside the func()

# to use the global variable we have to specify the as global x

def func2():

    global x

    return x

print(func2())

INTRODUCTION TO LIST

List is a ordered collection of items. We can store anything inside a list like string, integer, float, etc.

To make list we use square bracket []

Eg:

number = [1 ,2, 3, 4]

print(number)

word = ['word', 'word2', 'word3']

print(word)

mixed = [2, 4, 'word', 5, 'word2', 2.3]

print(mixed)

How to access the elements of list?

We can access the element with the help of square bracket

For eg:  
mixed = [2, 4, 'mango', 'apple', 5, 3.8, 'graphes']

#Now if we want to use the word mango then we do as follow:

print(mixed[2])

mixed = [2, 4, 'mango', 'apple', 5, 3.8, 'graphes']

#Now if we want to use the word mango then we do as follow:

print(mixed[2])

# for using more than one element

print(mixed[:5])

this will give the output as

[2, 4, 'mango', 'apple', 5]

print(mixed[:5:2])

this will give the output as

[2, 'mango', 5]

Basically accessing is same as the in step argument

For changing the item in the list we as follow:

mixed = [2, 4, 'mango', 'apple', 5, 3.8, 'graphes']

mixed[2] = 'Ripped Mango'

print(mixed)

this will give the output as :  
[2, 4, 'Ripped Mango', 'apple', 5, 3.8, 'graphes']

mixed = [2, 4, 'mango', 'apple', 5, 3.8, 'graphes']

mixed[2:6] = ['Ripped mango', 'guava']

print(mixed)

this will give the output as :  
[2, 4, 'Ripped mango', 'guava', 'graphes']

this simply meaning is to chaning the list items inside the list with this list item from third elements to 5th elements

but if we do direct as :

mixed = [2, 4, 'mango', 'apple', 5, 3.8, 'graphes']

mixed[2:6] ="ripped manago"

print(mixed)

this will give the output as :

[2, 4, 'r', 'i', 'p', 'p', 'e', 'd', ' ', 'm', 'a', 'n', 'a', 'g', 'o', 'graphes']

How to add data inside the list?

fruits = ['graphes', 'apple']

#To add data inside list we use append method

#this method will add data inside the list in the last

fruits.append('mango')

print(fruits)

this will give the output as :

['graphes', 'apple', 'mango']

Genererally, we don’t know in starting that what do we will add inside the list so we take empty list and adding the items inside the list with the help of append method as:

fruits = []

fruits.append('mango')

fruits.append('graphes')

print(fruits)

this will give the output as

['mango', 'graphes']

The other method to add data inside the list:

For the item to be added inside the list at specific position, we use insert method

fruits = ['mango', 'orange']

fruits.insert(1, 'garphes')  #Here 1 is the position at where graphes has to be inserted

print(fruits)

this will give the output as :

['mango', 'garphes', 'orange']

**Note: If we give the position which is not available inside the list then it will add at the last in the list.**

How to add two list?

For this we simply use plus symbol

fruits = ['mango', 'orange']

fruits1 = ['graphes', 'apple']

# so to add this two list we simply add the lists as :

fruits3  = fruits + fruits1

print(fruits3)

this will give the output as

['mango', 'orange', 'graphes', 'apple']

For extension of items inside the list we use extend method

fruits = ['mango', 'orange']

fruits1 = ['graphes', 'apple']

fruits.extend(fruits1) # This will just entend the element of list fruits with the list fruits1

print(fruits)

this will give the output as :

['mango', 'orange', 'graphes', 'apple']

For add list inside the list we use append method :

fruits = ['mango', 'orange']

fruits1 = ['graphes', 'apple']

fruits.append(fruits1)

print(fruits)

this will give the output as :

['mango', 'orange', ['graphes', 'apple']]

How to delete data inside the list?

Most comman method is pop method

Pop method will delete the last element, if we don’t pass any arguments in the pop method (meaning if we didn’t write anything inside the bracket of pop)

Eg:

fruits = ['apple', 'guavas', 'kivi', 'oranges']

fruits.pop()

print(fruits)

this will give the output as:

['apple', 'guavas', 'kivi']

if we pass an argument then it will delete the same as we give the position

eg:

fruits = ['apple', 'guavas', 'kivi', 'oranges']

fruits.pop(1)

print(fruits)

this will give the output as :

['apple', 'kivi', 'oranges']

Delete operator: For this operator we del statement

This will also delete the elements from the list

Eg: fruits = ['apple', 'guavas', 'kivi', 'oranges']

del fruits[1]

print(fruits)

this will give the output as:

['apple', 'kivi', 'oranges']

Remove method:

This will help when we don’t know the position of the particular elements in the list then remove method will helpful to us.

Eg:

fruits = ['apple', 'guavas', 'kivi', 'oranges']

fruits.remove('kivi')

print(fruits)

this will give the output as :  
['apple', 'guavas', 'oranges']

**Note: In remove method we give the instruction to delete the item which is not present in the list then, it will show the error message.**

In remove method if there is two same elements in the list then remove method will delete the first elements starting from left side.

Eg:

fruits = ['apple', 'guavas', 'kivi', 'apple', 'oranges']

fruits.remove('apple')

print(fruits)

this will give the output as:

['guavas', 'kivi', 'apple', 'oranges']

How to check whether the element is present in the list or not?

To check as follow:

fruits = ['apple', 'guavas', 'kivi', 'apple', 'oranges']

if 'apple' in fruits:

    print('apple is present')

else:

    print('apple is not present')

this will give the output as :

apple is present

`Some more method in list

Count method:

This method is used to count the no of times the elements present inside the list.

Eg:

fruits = ['apple', 'guavas', 'apple', 'banana', 'kivi', 'banana', 'kivi']

print(fruits.count('apple'))

this will give the output as 2

sort method:

this method is used to sort the elements in the list alphabetically

eg:  
fruits = ['apple', 'guavas', 'apple', 'banana', 'kivi', 'banana', 'kivi']

fruits.sort()

print(fruits)

this will give the output as :  
['apple', 'apple', 'banana', 'banana', 'guavas', 'kivi', 'kivi']

Note: we can’t do like this

fruits = ['apple', 'guavas', 'apple', 'banana', 'kivi', 'banana', 'kivi']

print(fruits.sort())

this will give the output as: None

for number if we use sort method then it will arranged in ascending order

eg:

numbers = [1, 85, 47, 10, 5, 11, 25, 46]

numbers.sort()

print(numbers)

this will give the output as :  
[1, 5, 10, 11, 25, 46, 47, 85]

shorted function:

this will help when we don’t want to sort the list but we want to print sorted list.

Eg:

numbers = [1, 85, 47, 10, 5, 11, 25, 46]

print(sorted(numbers))

this will give the output as:

[1, 5, 10, 11, 25, 46, 47, 85]

Clear method:

This method will empty the list.

Eg:

numbers = [1, 85, 47, 10, 5, 11, 25, 46]

numbers.clear()

print(numbers)

this will give the output as: []

copy method:

This method is used to copy the list

Eg:

numbers = [1, 85, 47, 10, 5, 11, 25, 46]

list2 = numbers.copy()

print(list2)

this will give the output as:

[1, 85, 47, 10, 5, 11, 25, 46]

List comparasion:

In python to compare the list we use two ways:

1. Double equal ==
2. Is keyword

Eg:

fruits1 = ['oranges', 'apple', 'pear']

fruits2 = ['banana', 'kiwi', 'apple', 'banana']

fruits3 = ['apple', 'oranges', 'pear']

fruits4 = ['oranges', 'apple', 'pear']

print(fruits1 == fruits2)

print(fruits1 == fruits3)

print(fruits1 == fruits4)

this will give the output as:

False

False

True

Note: order inside the list should same to get True

Is used to check whether the objects are in the same memory or not.

For check the value inside the list we will use double equal

Split method:

Split method convert string to list

Eg:

user\_info = 'Pintu 21'.split()

print(user\_info)

this will give the output as:  
['Pintu', '21']

Join method:

Join method convert list to string

Eg:

user\_info = ['Pintu', '21']

print(','.join(user\_info)) # here comma inside the quotes is to show by which way we want to join the list into string

this will give the output as: Pintu,21

generally, we less use join method

LIST VS ARRAY

Array means in English is ordered collection of items

List meaning in python is ordered collection of items

For the programming language c, c++, java we use array

In array we can use only single type of data either integer or string

In python list we can store any data.

There is also array module of python which we have to import in which we can store only fix type data type

List is more flexible. But the performance of array module in python is good still we didn’t use.

We didn’t use array module in python just because of flexibility of list.

But we use numpy array in python.

LIST VS STRING

String are immutable means we can’t change the string

List are mutable means we can change the element inside the list.

For eg:  
#for string

s = 'pintu'

# for using any method we don't do any change inside the string

# for eg:

print(s) # this will print the string

print(s.title()) #here title is just printing the  in title print

this will give the output as: pintu

Pintu

# for list

list1 = ['word1', 'word2', 'word3']

#for pop method, the method will actually delete the word from the list

# for eg:

list1.pop()

print(list1)

This will give the output as:  
['word1', 'word2']

Looping inside the list

Looping inside the list meaning is creating a loop in the list to print all the element of the list.

Eg:

# for for loop:

list1 = ['mango', 'guavas', 'apple', 'orange', 'kivi']

for fruit in list1:

    print(fruit)

# for while loop:

i = 0

while i < len(list1):

    print(list1[i])

    i += 1

output for the for loop and for while loop:

mango

guavas

apple

orange

kivi

List inside list ( also called 2d list)

List inside is like each separate list inside the is count as a separate elements

Eg:

list1 = [['apple','guavas', 'oranges', 'kivi'], [1, 2, 3, 4, 5], ['one', 'two', 'three', 'four', 4]]

# In this list each list inside the list is count as a seperate element of the list.

print(list1[2])

# Now if we want to print each element inside the each list then

# we will play loop inside the loop

#eg:

for i in list1: # here we introduce a new variable i for this loop

    for j in i: # here we also introduce a new variable j for this loop

        print(j)

the output of print(list[2]) is: ['one', 'two', 'three', 'four', 4]

and the output of print(j) is : apple

guavas

oranges

kivi

1

2

3

4

5

one

two

three

four

4

Now , for accessing the element in the list inside the list. We do as follow:

list1 = [['apple','guavas', 'oranges', 'kivi'], [1, 2, 3, 4, 5], ['one', 'two', 'three', 'four', 4]]

# for accessing the  word kivi then we do as :

print(list1[0][3])

#here in first square bracket we assign the argument of list inside the list

# In the 2nd square bracket we assign the argument of that element of that patticular list

The output of this is: kivi

**Note: “type” is a function which check the data is which type** (There is nothing relation between the list and type it is just a function)

Eg:

name = 'Pintu Raj'

print(type(name))

this will give the output as: <class 'str'> (meaning this is saying that name is a string type)

list1 = [['apple','guavas', 'oranges', 'kivi'], [1, 2, 3, 4, 5], ['one', 'two', 'three', 'four', 4]]

print(type(list1))

this will give the output as: <class 'list'>

How to generate with range function?

numbers = list(range(1, 11))

print(numbers)

this will give the output as :

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Pop method is not only remove the element but also it kept which can we use further

For eg:

number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

l = number.pop() # we make the poped number item into a variable

print(l)

this will give the outcome: 10

index method:

if we needed to find any particular element’s position from the list then we use index method.

Eg:

number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# here if we want to find the position of the 7 in the list

print(number.index(7))

this will the output as: 6 (position start from zero)

Now if we have the multiple times the same value then we specify from where we want to find the value of the elements.

Eg:

numbers = [2, 4, 5, 6, 2, 56, 44, 2, 11]

# Now if we want to find the position of the 2nd '2' then we do as  follow:

print(numbers.index(2, 1))

this will give the output as: 4

we can also give the stopping argument as number.index(2, 1, 10)

Pass list to a function:

# def negative\_list(l):

#     for i in l:

#         return list(-i)

number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# print(negative\_list(number))

def negative\_list(l):

    negative = []

    for i in l:

        negative.append(-i)

    return negative

print(negative\_list(number))

This will give the output as :

[-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]

MIN & MAX Function

Min is used for finding lowest number in the list similarly max is used for finding the maximum number in the list.

Eg:

list1 = [6, 60, 5]

print(min(list1))

print(max(list1))

the output of the function are: 5 and 5

Tuples

Tuple is also a data structure which we used to store data like list.

It can store anytype of data type like list. But tuples are immutable

In tuple we use parenthesis ()

We use tuples when we know that our data is not going to change

For eg week days Monday Tuesday wedesday

Tuples are faster than list

Method that used in tuples

1. Count
2. Index
3. Len function
4. Slicing in tuple eg list[::1]

Looping inside tuple

We can use looping in the tuple same as in list.

Eg:

mixed\_tuple = (2, 3, 5, 5.5, 2)

for i in mixed\_tuple:

    print(i)

this will give the output as:

2

3

5

5.5

2

Tuple with one elements:

For making tuple with one element we have to give comma after the element

Eg:

Num = (1,)

If we write like num = (1) then it will considered as int.

Tuple without paraenthsis:

If we write by giving comma then it will automatically considered as tuple

Eg:

guitars = 'yamaha', 'baton raouge', 'taylor'

numbers = 2, 3, 4

tuple unpacking:

for the elements inside the tuple if we want to assign each element into single variable then we can do as follow:

guitarists = ('Maneli Jamal', 'Eddie Van Der Meer', 'Andrew Foy')

guitarists1, guitarists2, guitarists3 = (guitarists)

List inside tuple:

We can add or eliminate any data from the list inside tuple.

favourites = ('southern mongnalia', ['Tokyo Ghoul Theme', 'landscape'])

# In this tuple, from the list we can add or delete the elements

favourites[1].pop()

print(favourites)

this will give the output as :

('southern mongnalia', ['Tokyo Ghoul Theme'])

In tuple we can also use min, max, sum function

Sum meaning summation of the element of the tuple

Function returning two values:

If we define any function and returning two value then the it will return in tuple.

For eg:

def func(int1, int2):

    multiply = int1\*int2

    add = int1 + int2

    return multiply, add

print(func(2, 4))

this will give the output as tuple : (8, 6)

we can also create tuple with range function

eg:

num = tuple(range(1, 11))

print(num)

this will give the output as :

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

We can change tuple into list, string by following ways.

num1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

num1 = list(num1)

# Now this is converterd into list

print(num1)

this will give the output as: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# for converting into string we str function

num1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

num\_list = str(num1)

print(num\_list)

This will convert into string.

Dictionaries

Why do we study dictionaries?

For real life data list is not sufficient.

Dictionaries is unordered collection of data in key : value pair

To make dictionary we can either use curly bracket or dict function

Eg:

user = {'name' : 'pintu', 'age' : 21}

user = dict(name = 'Pintu Raj', age = 21)

both way to create dictionary.

How to access data in dictionary?

To access data from the dictionary we do as follow:

user = dict(name = 'Pintu Raj', age = 21)

print(user['name'])

the output will be: Pintu Raj

we can also create dictionary as follow:

user\_info = {

    'name' : "Pintu Raj",

    "age" : 21,

    'favourite movies' : ['Iron man ', 'Avengers', 'Pirates of the carriabian'],

    "favourite song" : ['Arijit all famous', 'Arman Malik all hits'],

}

In dictionary we can represent complex data

We can create dictionary inside dictionary.

How to add data inside empty dictionary?

user\_data = {}

user\_data['name'] = 'Pintu'

print(user\_data)

this will give output as:

{'name': 'Pintu'}

In keyword:

In keyword is used to check the presence of key

Eg:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

if 'name' in user\_info:

    print('present')

else:

    print('not present')

now if we want to check for the values

the we will do as follow:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

if 'Pintu Raj' in user\_info.values():

    print('present')

else:

    print('not present')

this will give output as: present

if we want to check the complete list then we have to give the complete list

eg:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

if ['coco', 'kimino na wa'] in user\_info.values():

    print('present')

else:

    print('not present')

this will give the output as : present

Looping in dictionaries:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

for i in user\_info:

    print(i)

this will give the all keys: name

age

fav movies

fav tunes

if we want to print values of keys then we have to do as follow:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

for i in user\_info.values():

    print(i)

this will give the output as :

Pintu Raj

21

['coco', 'kimino na wa']

['awakening', 'fairy tale']

Similarly we can use key() method:

Eg:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

for i in user\_info.keys():

    print(i)

this will give output as:

name

age

fav movies

fav tunes

this is same as simply writing as:

for i in user\_info:

    print(i)

also for the value we can also write as:

for i in user\_info:

    print(user\_info[i])

item method:

in this method we use following dictionary:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

this is very important method.

Items method is basically gives us a key value pair tuple inside a list

For eg: user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

user\_items = user\_info.items()

print(user\_items)

this will give the output as:

dict\_items([('name', 'Pintu Raj'), ('age', 21), ('fav movies', ['coco', 'kimino na wa']), ('fav tunes', ['awakening', 'fairy tale'])])

why iteams method is important in looping?

For eg:

for i in user\_info.items():

    print(i)

this will give the tuple of each key value pair

output is:

('name', 'Pintu Raj')

('age', 21)

('fav movies', ['coco', 'kimino na wa'])

('fav tunes', ['awakening', 'fairy tale'])

This unpacked tuple we can use in following ways:

for i, j in user\_info.items():

    print(f'the key is {i} and its value is {j}')

this will give the output as :

the key is name and its value is Pintu Raj

the key is age and its value is 21

the key is fav movies and its value is ['coco', 'kimino na wa']

the key is fav tunes and its value is ['awakening', 'fairy tale']

Adding amd deleting data inside dictionary:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

# For adding data inside the dictionary we do as follow:

user\_info['fav\_songs'] = ['song1', 'song2']

print(user\_info)

this will give the output as :

{'name': 'Pintu Raj', 'age': 21, 'fav movies': ['coco', 'kimino na wa'], 'fav tunes': ['awakening', 'fairy tale'], 'fav\_songs': ['song1', 'song2']

Here fav\_song is added inside the dictionary

For delete we use pop method here also in following ways

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

# for deleting from the dict:

pooped\_items = user\_info.pop('fav tunes')

print(pooped\_items)

  #here pooped items will give only the value of the deleted key

  #but it will delete both key value pair

print(user\_info)

output of print(pooped items):

['awakening', 'fairy tale']

output of print(user\_info):

{'name': 'Pintu Raj', 'age': 21, 'fav movies': ['coco', 'kimino na wa']}

**Note: In pop method we cannot use empty like pop() as we can do in list, this will give an error**

Popitem method:

This will delete the last inserted item in the dictionary, in eailer then python 3.7 it remove randomly

For eg:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale']

}

popped\_items = user\_info.popitem() #This will give delete the fav tunes key value pair

print(popped\_items)

print(user\_info)

output if print(popped\_items):

('fav tunes', ['awakening', 'fairy tale'])

Output of print(user\_info):

{'name': 'Pintu Raj', 'age': 21, 'fav movies': ['coco', 'kimino na wa']}

Popitem method return tuple of key value pair and in pop method it return thing in which value is stored for string it will return string for tuple it will return tuple for list it will return list.

Update dictionary:

Update dictionary is basically update the key value pair inside the dictionary, for new key value pair it will add inside the dictionary.

Eg:

user\_info = {

    'name' : 'Pintu Raj',

    'age' : 21,

    'fav movies' : ['coco', 'kimino na wa'],

    'fav tunes' : ['awakening', 'fairy tale'],

}

more\_info = {'name' : 'Satyam Ranjan', 'State' : 'Bihar', 'Hobbies' : ['coding', 'reading', 'guitar']}

user\_info.update(more\_info)

print(user\_info)

since in dictionary key value pair of name is present so update method is just update the key value pair of name , for the key value pair of state and hobbies is not present inside the dictionary so it added in the dictionary.

The output of this are:

{'name': 'Satyam Ranjan', 'age': 21, 'fav movies': ['coco', 'kimino na wa'], 'fav tunes': ['awakening', 'fairy tale'], 'State': 'Bihar', 'Hobbies': ['coding', 'reading', 'guitar']}

Fromkeys method :

It is used for creating dictionary having different keys with same value

For eg: d = dict.fromkeys(['name', 'age', 'height',], 'unknown')

print(d)

output for this:

{'name': 'unknown', 'age': 'unknown', 'height': 'unknown'}

Fromkeys is helpful when we want to create dictionary having some of the keys in the dictionary have same value

Then we can use fromkeys to create dictionary

For the keys we can also create tuple rather than list.

We can use range function too.

Eg:

d = dict.fromkeys(range(10), 'unknown')

print(d)

this will output as :

{0: 'unknown', 1: 'unknown', 2: 'unknown', 3: 'unknown', 4: 'unknown', 5: 'unknown', 6: 'unknown', 7: 'unknown', 8: 'unknown', 9: 'unknown'}

Eg:

d = dict.fromkeys(['name', 'age'], ['unknown', 'unknown'])

print(d)

this will give the output as:

{'name': ['unknown', 'unknown'], 'age': ['unknown', 'unknown']}

Get method:

get method is used for accessing the keys inside the dictionary.

We can also accessing the value of key inside the dictionary as

d = {'name': 'Pintu Raj', 'age': 21, 'height': 5.11}

print(d['name'])

this will give the output as :

Pintu Raj

d = {'name': 'Pintu Raj', 'age': 21, 'height': 5.11}

print(d.get('name'))

this will also give the same output as:

Pintu Raj

But in get method if the key value pair is not found then it will not show any error and give the output as None

But in print(d[name]) if name is not found then it will show the error in output.

So get method is good for accessing the key value pair

We can use get method in also other ways too

For eg:

d = {'name': 'Pintu Raj', 'age': 21, 'height': 5.11}

if d.get('name'):

    print('present')

else:

    print('not preesent')

#this is same as accessing the data as in check method as follow:

if 'name' in d:

    print('present')

else:

    print('not present')

output of both will be : present

in if condition None meaning is False ( meaning this condition is not true)

clear method:

this method is used for clearing the data inside the dictionary

eg:

d = {'name': 'Pintu Raj', 'age': 21, 'height': 5.11}

d.clear()

print(d)

this will give the output as : {}  
copy method:

this method is used for copying the dictionary

d = {'name': 'Pintu Raj', 'age': 21, 'height': 5.11}

d1 = d.copy() #This will actually copy the dictionary meaning the we can use for separate work

print(d1)

# But assinging the in new variable is not copying the dictionary

d2 = d   # this will just making another name of the same dictionary

In get method we can assign another output instead of None for the key value pair not found

Eg:

d = {'name': 'Pintu Raj', 'age': 21, 'height': 5.11}

print(d.get('Height', 'Not found!!'))

this will give the output as : Not found!!

In a dictionary we can not use two same key for different value, the close to end entered value will overwrite the previous value of that keys

Eg:

d = {'name': 'Pintu Raj', 'age' : 63, 'age' : 73, 'height': 5.11,}

print(d)

this will give the output as :

{'name': 'Pintu Raj', 'age': 73, 'height': 5.11}

Set

Set is a unordered collection of unique items

To create set we use curly brackets as we use in dictionary

But In set there is no key value pair

Inside there is always unique items

For eg:

s = {1,3,45,5,6,5,3,44,5,5,43, 4}

print(s)

this will give the output as : {1, 3, 4, 5, 6, 43, 44, 45}

meaning here set is removing the duplicate inside it

so we can use set for removing the duplicate inside the list too

for eg:

s = {1,3,45,5,6,5,3,44,5,5,43, 4}

print(list(s))

this will the output as : [1, 3, 4, 5, 6, 43, 44, 45]

similary, we can do for tuple too.

Eg:

s = {1,3,45,5,6,5,3,44,5,5,43, 4}

print(tuple(s))

this will give the output as:

(1, 3, 4, 5, 6, 43, 44, 45)

Or we can make a list into a set to remove the dupliacates.

Eg:

list1 = [1, 3, 4, 5, 6, 7, 8, 7, 6, 9, 20, 4]

print(set(list1))

this will give the output as: {1, 3, 4, 5, 6, 7, 8, 9, 20}

and then we can change into list again

How to add items into set?

Add method

s = {1, 3, 5, 2, 4, 2, 5, 6, 5,}

# Now if we want to add 10 inside the set the we do as follow:

s.add(10)

print(s)

#this will give the output as {1, 2, 3, 4, 5, 6, 10}

#for delete into set we can use remove method:

s. remove(5)

print(s)

# This will give the output as : {1, 2, 3, 4, 6, 10}

In remove method we can remove only those things which are present inside the set else it will give the error message. Error से बचने के लिए discard method का उसे कर सकते है वो output मे none print करेगा In discard method if the item which we want to delete is present then it will delete otherwise it will give the output as None.

Eg:

s = {1, 3, 5, 2, 4, 2, 5, 6, 5,}

s.discard(12)

# in set there is not 12 but it is not showing the error message

#but in remove method it show the error message

s.discard(5)  #This will remove the integer 5

print(s)

clear method:

this method is used to clear all the items inside the set.

Eg:

s = {1, 3, 5, 2, 4, 2, 5, 6, 5,}

s.clear()

print(s)

this will give the output as: set()

copy method:

this method is used to copy the set

eg:  
s = {1, 3, 5, 2, 4, 2, 5, 6, 5,}

s1 = s.copy()

print(s1)

**Note: Inside the set we can’t store list, tuple or dictionary. Inside the set we can store string, int, floating**

For check items inside the set we can use in keyword

Eg:

s = {'a', 'b', 'c'}

 # To check the elements:

if "a" in s:

     print('Present')

else:

    print('Not Present')

for loop inside the set:

here’s for loop is same as in the list

eg:

s = {'a', 'b', 'c'}

for i in s:

    print(i)

here the output will all elements but in random mannar

In set we can perform union and intersection operation

**For union we use | (pipe) symbol and for intersection we use & (and) symbol.**

Eg:

s1 = {1, 3, 5, 6, 7, 4}

s2 = {4, 6, 8, 9, 10}

# for union

d = s1 | s2

print(d)

output is: {1, 3, 4, 5, 6, 7, 8, 9, 10}

s1 = {1, 3, 5, 6, 7, 4}

s2 = {4, 6, 8, 9, 10}

# for intersection

d = s1 & s2

print(d)

output is: {4, 6}

List Comprehension ( This is python special)

List comprehension is used to create long code in short line

For Eg:

# In normal ways to create loop inside the list

list1 = []

for i in range(1, 11):

    squares\_num = i\*\*2

    list1.append(squares\_num)

print(list1)

# Now in list comprehension method:

list3 = [i\*\*2 for i in range(1, 11)]

print(list3)

list1 and list3 will give the same output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

list comprehension is work as

list1 = [ <things that should to be printed in list> <defining the loop or anything else>]

eg:

list1 = [-i for i in range(1, 11)]

print(list1)

this will give the output as : [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]

How to use if statement inside the list comprehension?

If statement will written at last

Eg:

list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

#making the list of even numbers:

list2 = [i for i in list1 if i%2 == 0]

print(list2)

This will give the output as: [0, 2, 4, 6, 8, 10]

We can also write as

list2 = [i for i in list(range(10)) if i%2 == -0]

print(list2)

this will give the same above output.

Using list comprehension if else statement

In case if statement using only then we used in last but in case we want to use if else statement both we do as follow

list3 = [i\*\*2 if i%2 == 0 else -i for i in range(1, 11)]

print(list3)

this will give the output as:

[-1, 4, -3, 16, -5, 36, -7, 64, -9, 100]

Meaning for if else both statement we write if statement in starting and then the loop function

How to print nested list?

Nested list meaning list inside the list.

nessted\_comp = [[i for i in range(1, 4)] for j in range(1, 4)]

print(nessted\_comp)

this will give the output as : [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

Dictionary Comprehension

Dictionary comprehension is same as the list comprehension the only difference is here is key value pair

Eg:

dict1 = {i : i\*\*2 for i in range(1, 11)}

print(dict1)

this will give the output as: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

eg:

dict1 = {f'The square of {i} is' : i\*\*2 for i in range(1, 11)}

print(dict1)

this will give the output as :

{'The square of 1 is': 1, 'The square of 2 is': 4, 'The square of 3 is': 9, 'The square of 4 is': 16, 'The square of 5 is': 25, 'The square of 6 is': 36, 'The square of 7 is': 49, 'The square of 8 is': 64, 'The square of 9 is': 81, 'The square of 10 is': 100}

dict1 = {f'The square of {i} is' : i\*\*2 for i in range(1, 11)}

# We can also print key vlaue pair in each single line:

for j, k in dict1.items():

    print(f'{j}: {k}')

this will give the output as :

The square of 1 is: 1

The square of 2 is: 4

The square of 3 is: 9

The square of 4 is: 16

The square of 5 is: 25

The square of 6 is: 36

The square of 7 is: 49

The square of 8 is: 64

The square of 9 is: 81

The square of 10 is: 100

Eg:

# For counting the chacter using the dictionary comprehension:

name = "missisppi"

word\_counting = {char: name.count(char) for char in name }

print(word\_counting)

this will give the output as : {'m': 1, 'i': 3, 's': 3, 'p': 2}

How to use if else in dictionary?

We write if else condition inside the parenthesis then we use loop function

Eg:

odd\_even = {i : ("even" if i%2 ==0 else "odd") for i in range(1, 11)}

print(odd\_even)

this will output as :

{1: 'odd', 2: 'even', 3: 'odd', 4: 'even', 5: 'odd', 6: 'even', 7: 'odd', 8: 'even', 9: 'odd', 10: 'even'}

Set Comprehension

Generally set comprehension is not used.

Here we also do as we done in the list comprehension

Eg:

s = {i\*\*2 for i in range(1, 11)}

print(s)

this will give the output as:

{64, 1, 4, 36, 100, 9, 16, 49, 81, 25}

(here this is not in proper order as set is unordered collection of data)

Introduction to \*args

This is called star operator. Here we write star args just for conventions we can write anything like defining variable

Star args is just making the given input into a tuple which we can use in desired ways

For eg if we want to define the function which will add two number then we do as follow

def add(a, b):

    return a+b

this function will just take two argument at time but we want to give more argument to this function then it will show error.

To solve this problem we use star operators method

def all\_total(\*args): #here \* operator will just changing all the input into a tuple

    total = 0

    for i in args: # from here we use tuple to do summation

        total += i

    return total

print(all\_total(1,3,4,5,5,6,6,7))

this will give the output as : 37

here args is not any defined function we just do because convention is. We can use any other things to it too.

Eg:

def all\_total(\*num): #here \* operator will just changing all the input into a tuple

    total = 0

    for i in num: # from here we use tuple to do summation

        total += i

    return total

print(all\_total(1,3,4,5,5,6,6,7))

this will give the same output as previously = 37

args with normal parameter

we use star args with normal parameter as follow:

def multiply\_nums(num1, \*args):

    multiply = 1 # don't put zero because this will make total value to be zero.

    for i in args:

        multiply \*= i

    return multiply

print(multiply\_nums(3,4,4))

here in function num1 will be 3 then the rest will convert into tuple

so for this the output will be : 16 ( as loop is working only in args)

similary if we give num1, num2, \*args then 1st two number will be num1 and num2 repectively and the rest will be converted into tuple .

eg:

def multiply\_nums(num1, num2, \*args):

    multiply = 1 # don't put zero because this will make total value to be zero.

    for i in args:

        multiply \*= i

    return multiply

print(multiply\_nums(3,4,4))

here the output will be 4 and the value for the num1 = 3 for the num2 = 4 and in the tuple as create by star operator is 4

the output will be 4

in defining the function if no argument is given to the args then there will no error as it created the empty tuple but for num1 and the num2 we have to give the arguments otherwise it will give an error message.

We can’t use args in starting and num1 as in the last like

def multiply\_nums(\*args, num1)

because \* args will create all the element to tuple and nothing will left for the num1 and since there is no argument left for the num1 so it will show error.

How to use star args as a arguments?

This will use to just unpack the element inside the list or tuple

For eg:

def multiply\_nums(\*args):

    multiply = 1 # don't put zero because this will make total value to be zero.

    for i in args:

        multiply \*= i

    return multiply

#Now let  suppose we have a list of number in which we want to multiply all the number inside the list

nums = [2, 34, 55, 3, 23]

# Now if we do direct as follow then our funtion will not work:

print(multiply\_nums(nums)) # This will give the output as list

# To acces the number inside the list we have to unpack the list

# we can unpack the list with the help of star operator by putting the star mark as follow:

print(multiply\_nums(\*nums)) # this will give the output as : 258060

Kwargs (keyword arguments)

Denoted as double star \*\*

This will collected all arguments as dictionary

Eg:

def func(\*\*kwarg):

    print(kwarg)

func(first\_name = "Pintu", last\_name = "Raj", age = 21)

this will give the output as a dictionary

output is : {'first\_name': 'Pintu', 'last\_name': 'Raj', 'age': 21}

dict1 = {'name' : 'Pintu Raj', 'age' : 21, 'gender' : 'male'}

def fun(\*\*kwargs):

    for i, j in kwargs.items():  # here we start for loop inside the dictionary

        print(f'Your {i} is {j}')  # here return is not working insteed of print why?

fun(\*\*dict1) #Here to access the dictionary we use double star

this will give the output as :

Your name is Pintu Raj

Your age is 21

Your gender is male

Eg:

dict1 = {'name' : 'Pintu Raj', 'age' : 21, 'gender' : 'male'}

def fun(num, \*\*kwargs):

    print(num)

    for i, j in kwargs.items():  # here we start for loop inside the dictionary

        print(f'Your {i} is {j}')  # here return is not working insteed of print why?

fun('hello world', \*\*dict1) #Here to access the dictionary we use double star

this will give the output as:

Your name is Pintu Raj

Your age is 21

Your gender is male

Hello world

**Note: The order to write the parameter, default parameter, star args and double star kwargs as follow:**

**Parameter 🡪\*args 🡪 default parameter 🡪\*\*kwargs**

Eg:

def func(name, \*args, last\_name = 'unknown', \*\*kwargs):

    print(name)

    print(args)

    print(last\_name)

    print(kwargs)

func('Pintu', 1, 2, 3, 4, age = 21, gender = 'male')

this will give the output as:

Pintu

(1, 2, 3, 4)

unknown

{'age': 21, 'gender': 'male'}

Lambda Expression

Lambda expression is a function which we define in one line. Lambda expression has no name.

For eg: def add(a,b):

    return a + b

#The above function we can define using lambda expression as follow:

# In actually we didn't assigning the lambda expression in the variable

add2 = lambda a,b  : a+b  # but here we are assinging just for learning

# we use  in built in function like map, reduce, filter, etc

print(add2(2,3))

This will give the output as : 5

Eg:

variable = lambda a,b : a\*b

print(variable(2,3))

this will give the output as: 6

eg:

# for the function like below:

def is\_even(a):

    return a%2 == 0  # This will give output as either true or false

# Now we can do the same above using the lambda expression:

is\_even2 = lambda a : a%2 == 0

print(is\_even2(3))

this will give the output as : False

How to use lambda expression in if else statement?

Eg:

def func(s):

    if len(s) > 5:

        return True

    return False

# Now the above fucntion can be done by lambda expression

func1 = lambda s : True if len(s) > 5 else False

# this above example can be also done without if else statement:

fucn2 = lambda s : len(s) > 5

print(fucn2("hello"))

this will give the output as : False

ENUMERATE FUNCTION

We use enumerate function to track the position of our items

We can also track the function without the enumerate function as follow :

Eg:

name = ['pintu', 'satyam', 'ashutosh', 'harshit', 'aman']

pos = 0

for i in name:

    print(f'{pos} ---> {i}')

    pos += 1

this will give the output as :

0---> pintu

1 ---> satyam

2 ---> ashutosh

3 ---> harshit

4 ---> aman

Using enumerate function :

Eg:

name = ['pintu', 'satyam', 'ashutosh', 'harshit', 'aman']

for i, j in enumerate(name):  # Here i always give the position of the variable in j(name)

    print(f'{i} ----> {j}')

this will give the output as:  
0 ----> pintu

1 ----> satyam

2 ----> ashutosh

3 ----> harshit

4 ----> aman

Map function

Map function is basically take any iterable (like list, tuple, string) and a function, then it passing all the iterable from the function after that that output we can map inside the list, tuple or in anythings.

Eg:

nums = [1, 2, 3, 4, 5, 6]

# Now if we want to get the square of that items inside the list

# then we can do as follow:

def squares(a):

    return a\*\*2

#inside the map we first pass the function then we pass the iterable

new\_list = list(map(squares, nums)) #map is basically working like the loop function

print(new\_list)

this will give the output as: [1, 4, 9, 16, 25, 36]

but in practical we use map function with lambda function

eg:

nums = [1, 2, 3, 4, 5, 6]

squares = list(map(lambda a: a\*\*2, nums))

#here lambda is defined the function in one line inside the map function

# we can also create the tuple instead of list by just replacing list from tuple

print(squares)

this will give the output as: [1, 4, 9, 16, 25, 36]

filter function

Zip function

Zip function is making the two or more list into tuple itetator which can be converted into list, tuple, dictionary, etc.

Eg:

user\_id = ['user1', 'user2', 'user3']

names = ['Pintu', 'harshit', 'rohit']

# Now the zip function will convert the two list into the tuple iterator

users = zip(user\_id, names)

# This iterator can be convert into dict, list

print(dict(users))

this will give the output as: {'user1': 'Pintu', 'user2': 'harshit', 'user3': 'rohit'}

**Note: for 3 list Zip function can’t be converted into dictionary**

Decorators

Before learning the decorators we have to know the following things

Some more about function , first class function ( closures)

# earlier we define the function as :

def square(a):

    return a\*\*2

# Now this is a square function is ready

 # We can assign the function in any variable

 s = square # this is assigned the function

# this s is same as the function square

# Meaning we can either use the s or square for the defined operation

# both square and s will be at the same location of the memory

How to pass the function as a argument ?

#here we will create the function which will take fucntion as a argument

def my\_map(func, l):

    new\_list = []

    for i in l:

        new\_list.append(func(i))

    return new\_list

# now let us define the new function which we will use in my\_map function

def square(a):

    return a\*\*2

list1 = [1,2,3,4]

# We can also use the lambda expression in it:

# We can also use the list comprehension to define the list:

def my\_map2(func, l):

    return [func(i) for i in l] # this return statement is using the list comprehension

print(my\_map2(square, list1))

print(my\_map(square, list1))

print(my\_map(lambda a : a\*\*3, list1)) #here lambda is working same as the square funcion was doing in the My\_map function

this will give the output as: [1, 4, 9, 16]

[1, 4, 9, 16]

[1, 8, 27, 64]

How to return the function from the function?

def square(a):

    return a\*\*2  # here we are returning the value from the function

# Now we will learn how to return the value from the  function

def outer\_func():

    def inner\_func():

        print('inside inner func')

    return inner\_func

# here we have defined outer\_func which is returning the inner function

# but the inner function is executed

# to execute the inner func we have to use the paranthesis

outer\_func() # this is giving the us the inner function

# Now if we use again the paranthesis then the inner function will be excute

outer\_func()()

# we can assign inside any variable to this outer\_func() which is actually giving us the inner\_func

var = outer\_func()

var()  # this will give the same output as the outer\_func()() giving us.

This will give the output as: inside inner func

Eg:

def outer\_func(msg):

    def inner\_func():

        print(f'The given message is {msg}')

    return inner\_func

outer\_func("hi there")()

# we can use call above function as follow to :

var = outer\_func('hello there')

var()

the output is : The given message is hi there

The given message is hello there

We can make many function inside a single function and according to use we can use each separate function using function returning the function as follow;  
 eg:

def to\_power(x):

    def to\_number(n):

        return n\*\*x

    return to\_number

# Now if we want to get the cube for any number then we can just do as follow:

# there will no need for again defining the function

cube = to\_power(3) # this is taking the argument as what power is

 # now the cube will take to which no we want to cube

print(cube(2))

this will give the output as : 8

finally from here we will start decorator

Decorators is used to enhance the functionality of other functions

Why do we need to define the decorator?

We use the decorator to add extra feature without changing inside the function.

Eg:

def func1():

    print('This is func1')

def func2():

    print('This is func2')

def decorator(any\_function):

    print('This is awesome function')

    any\_function()

# This will actually decorate the function but can not assign this into any other variable

# but the following function will return us the function which is now decorated

def decorator\_function(any):

    def wrapper\_function():

        print('This is awesome function')

        any()

    return wrapper\_function

@ is the symbol is used to call the decorator function in python. This is called syntactic sugar.

Eg:

def decorator\_function(any\_function):

    def wrapper\_function():

        print('This is awesome function')

        any\_function()

    return wrapper\_function

#now if we want to enhance the func1 without editing the code of the func1

# then we can use syntactic sugar as shortcut

@decorator\_function

def func1():

    print('This is func1')

func1()

this will give the output as :

This is awesome function

This is func1

Eg:

def decorator\_function(any\_function):

    def wrapper\_function(\*args, \*\*kwargs):

        print('This is awesome function')

        return any\_function(\*args, \*\*kwargs) # to get outcome from any function we have to add return here

    return wrapper\_function

@decorator\_function

def func1(a, b):

    return a + b  # now to get this function output we have to get some return from the wrapper function

print(func1(3, 4))

this will give the output as : This is awesome function

7

In above example if we write the doc string (which is used to tell about the what the function can do) we import a module from the functools called wraps

Eg:

from functools import wraps

def decorator\_function(any\_function):

    @wraps(any\_function)   # This is prebuilt function in python which we have imported above

    def wrapper\_function(\*args, \*\*kwargs):

        ''' this is wrapper function'''

        print('This is awesome function')

        return any\_function(\*args, \*\*kwargs)

    return wrapper\_function

@decorator\_function

def add(a, b):

    '''this function is used for the addition the two numbers'''

    return a + b

print(add.\_\_doc\_\_)

this will give the output as : this function is used for the addition the two numbers

**Note :**  \_\_doc\_\_ is used for calling the function doc written inside triple quotes

\_\_name\_\_ is used to call the function name

These above are the method so it is used with the function name with dot

Eg:

from functools import wraps

def print\_decorator(any\_function):

    @wraps(any\_function)

    def wrapper\_function(\*args, \*\*kwargs):

        print(f'You are calling {any\_function.\_\_name\_\_} function')

        print(f'{any\_function.\_\_doc\_\_}')

        return any\_function(\*args, \*\*kwargs)

    return wrapper\_function

@print\_decorator

def add(a, b):

    '''This function take two numbers as arguments and return their sum'''

    return a + b

print(add(3, 4))

this will give the output as :

You are calling add function

This function take two numbers as arguments and return their sum

7

Decorator with arguments

Best explanation for the decorators with arguments

Eg:

def allowing\_entities(data\_types):

    def decorator\_function(any\_function):

        def wrapper\_function(\*args):

            if all([type(i) == data\_types for i in args]):

                print('All the elements inside the list is valid')

                return any\_function(\*args)

            else:

                print('There is some invalid elements inside the list')

        return wrapper\_function

    return decorator\_function

def add\_func(a, b):

    return a  + b

add\_func = allowing\_entities(int)(add\_func)  # here if we put one more paranthsisis pair then it will just call the decorated add\_func as follow

# add\_func = allowing\_entities(int)(add\_func)(2,3) <--- this is meaning we have given the arguments in the decorated add\_func

# now for print the  we just have to use print in this way --->  print(add\_func)

print(add\_func(2,3))

Generators

Generators are iterators. This will also generate list like sequence but it only generate when we demand for that items from the sequence. We use the generator just because it is use less memory and it is comparatively fast than list and tuple

To use the generator we use **yield** keyword

Eg:

# in normal

def num1(n):

    for i in range(1, n + 1):

        print(i)   # or simply we can use the return keyword too.

# Now if we want to print the number from 1 to 10

# them the following way to generate using the generator

def num(n):

    for i in range(1, n+1):

        yield i    # here yield keyword is same as the return keyword.

# Now this is generator is created.

# Now for calling the generators. we use the for loop.

for i in num(10):

    print(i)

# Generators are iritator so we can use it only one time as after calling the number it remove from the memory.

This will give the output as:

1

2

3

4

5

6

7

8

9

10

Eg:

def even\_sequence(n):

    for i in range(1, n+1):

        if i%2 == 0:

            yield i

        else:  #We can also define this generator without else statement

            continue

even = even\_sequence(50)  #Here we have generate the sequence which will we obtain using the for loop

for i in even:  # this loop is using for getting the numbers

    print(i)

print(even)

# now if generate inside the loop  then we can use it every time

# like as follow:

for i in even\_sequence(40):  # This will generate the sequence also

    print(i)

for i in even\_sequence(40):  # This will generate the sequence also

    print(i)

this will each time output.

Generator comprehension

Generator comprehension is same as the list comprehension just we have to replace the square bracket with the parenthsis.

Eg:

list1 = [i\*\*2 for i in range(1, 11)]

print(list1)   # this will print the list which is created above

# similarly we can create the generators

gen1 = (i\*\*2 for i in range(1, 11))

print(gen1)  # this will create generators

# to call the generator we can use the loop method

for i in gen1:  # this will call the generated generators.

    print(i)

# since the generator is already created so we can not use the for loop to create the generator again.

Object Oriented Programming ( OOP)

This is most common topic in almost all the popular programming language (Python, JAVA, etc.)

What is object oriented programming?

OOP is just a way/style to write the code in programming language.

It is very useful in creating the real world programming.

The most common word we will see in OOP 🡪 class, object and method

**Note: object is also called as the instance**

Class meaning in which category does the object is categories.

For eg list is a class

And list1 = [2,4,6,7] 🡪 this is called the object of class list

And any action do we perform inside the object is called its method

Like for adding the elements inside the list we use append method.

Now in OOP we will learn how to make our own class.

According to convention for making the class we should use the first letter of our class in capital letter. (This is not any rule but this convention is just for the ease)

Some of the predefined class in the python also have their first letter of the class in small letter like ‘list’

In making the class we direct use the colon without giving the parenthesis.

Like for defining the function we use **def** keyword, similarly here we use the **class** keyword.

Eg:

class Person:  # <--- here we had made a class

class Person():  #<--- Don't do like this

Inside the class we define the special type of method we called it as init method. Also known as constructor method.

Inside the class if we define any function the we call it as a method.

That’s why here we call init as a method.

To defining the init method we do as follow.

\_\_init\_\_ (double underscore init double underscore) then inside the parenthesis we have to write a special type of word called ‘self’ after that in same parenthesis we write attribute that object will have like first\_name, last\_name, age.

Now in second line we have to declare the object variable

Like

For eg:

class Person:

    def \_\_init\_\_(self, first\_name, last\_name, age):

        # here inside the init method by default the first word will be the object

        # like here self will be the object meaning it will represent the object.

        print('init method called')

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.age = age

p1 = Person('Pintu', 'Raj', 21)

p2  = Person('Harshit', 'Vasistha', 21)

# whenever we create any object, our init method is called

# Now if we want to know the first name of the object we call it as :

print(p1.first\_name)

# here p1 is represent the self of our class

# since the p1.first\_name is the variable so we don't use the parenthesis to call the object's first name

The output will be:

init method called

init method called

Pintu

Since we have created two object so two times the init method is called that’s why the method is printing the ‘init method called’ print twice.

Instance method or object method:

class Person:

    def \_\_init\_\_(self, first\_name, last\_name, age):

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.age = age

    # Now defining the instance method inside the class

    def full\_name(self):

        return f'{self.first\_name} {self.last\_name}' # Now we have created the instance / object method:

# Now we will define the instance of this class:

p1 = Person('Pintu', 'Raj', 21)

p2 = Person('Harshit', 'Vashistha', 24)

# to call the instance method we do as follow:

print(Person.full\_name(p1))  # we can call in either this way or the following way

print(p1.full\_name())

# both will give us the same output

The output will be: Pintu Raj

Here full\_name method is function inside the class so we call it with parenthesis

Person.full\_name(p1) is same as the following eg

list1 = [1,2,3,4,5]

# to clear the list we can do as too

list.clear(list1) # this is same as the following

list1.clear()  # this will also clear the list

l2 = [7,8,9,10]

# to append inside the list we can do as following too

list.append(l2, 11) # here the l2 is the object like in our case self

# above is same as the following:

l2.append(12)   # this will also append

eg:

class Person:

    def \_\_init\_\_(self, first\_name, last\_name, age):

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.age = age

    def full\_name(self):

        return f'{self.first\_name} {self.last\_name}'

    # Now we will define the method which check above 18

    def is\_abov\_18(self):

        if self.age > 18:  # this same can be done as follow

            return True

    def is\_abov\_18(self):

        return self.age > 18  # this will either evaluated as True or False

Class variable or Class attributes:

We define the class variable which is same for the all of the method which we define in that class.

Eg:

def Circle:

    def \_\_init\_\_(self, radius,pie):

        self.raidus = radius

        self. pie = pie

    def circle\_circumference(self):

        return 2\*self.pie\*self.radius    # here we use the circumference formula

    def circle\_area(self):

        return self.pie\*self.radius\*\*2    # here we use the area formula

c1 = Circle(3, 3.14)

c2 = Circle(5, 3.14)   # here for  each instance / object we have to give the value of pie

# if we use the class variable then we don't have to give the value of pie each time

Without class variable we have to give the common variable each time which also cause the extra memory uses.

Now with using the class variable

class Circle:

    pie = 3.14  # This is class variable, this will same for the all method which will define in this class

    def \_\_init\_\_(self, radius):

        self.radius = radius  # This is instance/object variable

    def circle\_circumference(self):  # here we don't have to give the value of pie

        return 2\*Circle.pie\*self.radius  # Now instead of naming pie as self.pie we do as  ---> Circle.pie as pie is the class variable

    def circle\_area(self):

        return Circle.pie\*self.radius\*\*2    # similarly here to we write Circle.pie as pie is now class variable

c1 = Circle(5)

print(c1.circle\_circumference())

print(c1.circle\_area())

another example for the class variable.

Eg:

# let's suppose we have to give the offer of 10 percent discount on the laptops

# then we can use the class variable as follow:

class Laptops:

    discount\_percent = 10

    def \_\_init\_\_(self, brand\_name, model\_name, price):

        self.brand\_name = brand\_name

        self.model\_name = model\_name

        self.price = price

    def apply\_dicount(self):  # Now here there is no need of using the number of what percent we want to discount

        dicounted\_price = self.price\*(Laptops.discount\_percent/100)

        return self.price - dicounted\_price

l = Laptops('Apple', 'Macbook Pro', 130000)

print(l.apply\_dicount())

now here is problem that there is not all the laptops having the same discount.

So we do some change in it as follow

1. We change the Laptops.discount\_percent with self.discount\_percent
2. For any specific object / instance, to change the discount percent we add the line as
3. l2.discount\_percent = 50 (This is written for the instance l2)
4. step 3 will create the variable inside that object with the name of discount\_price then in apply discount will check whether any discount\_price name variable is present in that instance or not. If it present then it will use the value of that variable, and in case the variable is not present then it will check for the class variable.

Eg:

# let's suppose we have to give the offer of 10 percent discount on the laptops

# then we can use the class variable as follow:

class Laptops:

    discount\_percent = 10

    def \_\_init\_\_(self, brand\_name, model\_name, price):

        self.brand\_name = brand\_name

        self.model\_name = model\_name

        self.price = price

    def apply\_dicount(self):  # Now here there is no need of using the number of what percent we want to discount

        dicounted\_price = self.price\*(self.discount\_percent/100)

        return self.price - dicounted\_price

l = Laptops('Apple', 'Macbook Pro', 130000)

l2 = Laptops('Lenevo', 'v110',30000)

print(l.apply\_dicount())  # this will give the discount accordingly to class variable as there is not any discount\_percent name variable in l

l2.discount\_percent = 50  # this create the new instance variable in the instance l2 like brand\_name

print(l2.\_\_dict\_\_)  # This will print the instances in the form of dictionary.

print(l2.apply\_dicount())

Class method:  
To make the our class we use a decorator 🡪 @classmethod

In gernerally very less use the class method

Eg:

class Person:

    total = 0

    def \_\_init\_\_(self, first\_name, last\_name, age):  # when ever the instance is created the class is called from here

        Person.total += 1

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.age = age

# Now we will create the class method:

    @classmethod

    def count\_instances(cls):

        return f'You have created {cls.total} instances in the {cls.\_\_name\_\_} class'

p1 = Person("Pintu", 'Raj', 21)

p2 = Person('Aman', 'Kumar', 21)

p3 = Person('Aditya', 'Raj', 21)

print(Person.count\_instances())  # here we can do with using the name of the objects like as follow

print(p1.count\_instances())  # this will also give the same output as the above one will give

# This is happening just because python will first check in the that object if that variable is not found then it will check into the class variable.

Class method as a constructor:  
Class method is used for creating the object in different style then we use the class method as a constructor.

That type of constructor we define with the help of classmethod decorator.

Eg:

class Person:

    total = 0

    def \_\_init\_\_(self, first\_name, last\_name, age):

        Person.total += 1

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.age = age

    @classmethod

    def from\_string(cls, string):

        first, last, age = string.split(',')

        return cls(first, last, age)  # this will return the all variable

p1 = Person("Pintu", 'Raj', 21)

p2 = Person('Aman', 'Kumar', 21)

p3 = Person('Aditya', 'Raj', 21)

p4 = Person.from\_string('Pintu,Raj,21')

print(p4.first\_name)

Static method:

To define the static method we use the static method decorator 🡪 @staticmethod

There is no relation with the class method or instances method

This is define just for the normal function in the class.

Eg:

class Person:

    total = 0

    def \_\_init\_\_(self, first\_name, last\_name, age):

        Person.total += 1

        self.first\_name = first\_name

        self.last\_name = last\_name

        self.age = age

    @classmethod

    def from\_string(cls, string):

        first, last, age = string.split(',')

        return cls(first, last, age)  # this will return the all variable

    # Now defining the static method:

    @staticmethod

    def func():

        print('hello, the static method is called')

p1 = Person("Pintu", 'Raj', 21)

p2 = Person('Aman', 'Kumar', 21)

p3 = Person('Aditya', 'Raj', 21)

p4 = Person.from\_string('Pintu,Raj,21')

print(p4.first\_name)

# to call the static method we do as follow

Person.func()

Some useful meaning of the following term:

Encapsulation: encapsulation meaning writing the data of class or the function (so called method in class) which we have define in the class at one place.

In short, we encapsulate the useful data at one place.

Abstraction: Abstraction meaning hiding the complexities of any method from the user called abstraction like in following example:

l = [3,4,1,6,5]

# Now for arranging the element inside the list we use the sort method

l.sort() #here sort method is called tim sort method

in this example we don’t know what is the algorithm of the sort method.

Now let’s talk about some name convention.

In python using the single underscore before any name of in the method of class, its indicating that this is private(in actually there is nothing private in the python) meaning don’t do any kind of editing in that method.

Eg:

class Phone:

    def \_\_init\_\_(self, brand\_name, model\_name, price):

        self.brand\_name = brand\_name

        self.model\_name = model\_name

        self.\_price = price

    def make\_a\_call(self):

        print(f'calling {Phone\_number} ...')

    def full\_name(self):

        return f'{self.brand\_name} {self.model\_name}'

now here underscore in the price method we want to say to the other developer that please don’t do anything with this method. However they can do. But the convention is to just telling the intension to it.

Double underscore name double underscore called dunder method like init method.

Sometime it is called magic magic method.

Now let’s talk about name mangling

In name mangling we use the double underscore name. name mangling is not a convention

By using the name mangling python will change the name of that attributes or variable with a new name as underscore class name underscore that variable name.

Eg

class Phone:

    def \_\_init\_\_(self, brand\_name, model\_name, price):

        self.brand\_name = brand\_name

        self.model\_name = model\_name

        self.price = price

    def make\_a\_call(self):

        print(f'calling {Phone\_number} ...')

    def full\_name(self):

        return f'{self.brand\_name} {self.model\_name}'

# in the above class if we use the self.price with self.\_\_price

# then it will change that attribute with self.\_Phone\_price

In the following example

class Phone:

    def \_\_init\_\_(self, brand\_name, model\_name, price):

        self.brand\_name = brand\_name

        self.model\_name = model\_name

        self.price = price

        self.complete\_specification = f'{self.brand\_name} {self.model\_name} and the price is {self.price}'

    def make\_a\_call(self):

        print(f'calling {Phone\_number} ...')

    def full\_name(self):

        return f'{self.brand\_name} {self.model\_name}'

phone1 = Phone('Nokia', '1100', 1000)

print(phone1.brand\_name)

print(phone1.model\_name)

print(phone1.complete\_specification)

There three major problem.

1. By mistake if we put the price of phone in negative then it will show the negative which practically should not to be happen.
2. Now for any phone if we change the price of the phone later then in calling phone price it will show the changed price but in the complete specification it will show the original price that we had set earlier.
3. In setting the new price if we put the negative price in that new price but it should not to be happen.

To solve the 1st problem we use if else statement like as follow:  
class Phone:

    def \_\_init\_\_(self, brand\_name, model\_name, price):

        self.brand\_name = brand\_name

        self.model\_name = model\_name

        if price > 0:

            self.price = price

        else:

            self.price = 0

        self.complete\_specification = f'{self.brand\_name} {self.model\_name} and the price is {self.price}'

    def make\_a\_call(self):

        print(f'calling {Phone\_number} ...')

    def full\_name(self):

        return f'{self.brand\_name} {self.model\_name}'

phone1 = Phone('Nokia', '1100', 1000)

print(phone1.brand\_name)

print(phone1.model\_name)

print(phone1.complete\_specification)

instead using the if else statement we can use the max function trick.

self.price = max(price, 0)

this max function will choose the greater from the price and the 0.

Now for the complete specification price problem we can do as follow.

Instead of making the instance variable we can make the function of complete specification. That function will take the self as a argument.

class Phone:

    def \_\_init\_\_(self, brand\_name, model\_name, price):

        self.brand\_name = brand\_name

        self.model\_name = model\_name

        self.price = max(price, 0) # this will help us from giving the negative price.

    def complete\_specification(self):

        return f'{self.brand\_name} {self.model\_name} and the price is {self.price}'

    def make\_a\_call(self):

        print(f'calling {Phone\_number} ...')

    def full\_name(self):

        return f'{self.brand\_name} {self.model\_name}'

phone1 = Phone('Nokia', '1100', 1000)

print(phone1.brand\_name)

print(phone1.model\_name)

print(phone1.complete\_specification()) # this will give the price as 1000

phone1.price = 500

print(phone1.complete\_specification())  # this will give the price as 500

now if we write the predefine decorator called property decorator in the complete specification then we don’t need to call the that method as a function. We can treat it as the attribute.

Eg:

@property

    def complete\_specification(self):

        return f'{self.brand\_name} {self.model\_name} and the price is {self.price}'

Now the third problem is to new set price can be negative as well

To solve this in other programming language they use the method called getter, setter etc.

Similarly we can use the property decorator instead of getter and instead of setter we can use the setter decorator

Using the property decorator we define a function inside the class which return us that price which we have define in the attribute.

Then we define the another function using setter decorator as follow:

    @property

    def price(self):

        return self.\_price

    @price.setter

    def price(self, new\_price):

        return self.\_price = max(new\_price, 0)

**Note: we first define the property decorator then we define the setter decorator otherwise it will show the error.**

Inheritence:

Inheritance is the property of python which allow us to use the property any predefined class in defining the new class.

Eg:

class Phone:

    def \_\_init\_\_(self, brand, model\_name, price):

        self.brand = brand

        self.model\_name = model\_name

        self.price = price

    @property

    def complete\_specification(self):

        return f'{self.brand} {self.model\_name} and the price is {self.price}'

    @property

    def full\_name(self):

        return f'{self.brand} {self.model\_name}'

class Smartphone(Phone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera):

        Phone.\_\_init\_\_(self, brand, model\_name, price)

        self.ram = ram

        self.internal\_memeory = internal\_memeory

        self.rear\_camera = rear\_camera

phone1 = Phone('Nokia', '1100', 1000)

phone2 = Smartphone('Oneplus', 'nord', 28000, '4gb', '128gb', '48mp')

print(phone2.full\_name)

print(phone2.complete\_specification)  # This will give the brand model\_name and the price, not giving the ram internal etc.

In this example we write the Phone class inside the smartphone class and define the init function inside the Smartphone class two times.

Another way to use the inhertence which is most commonly used is :

class Phone:

    def \_\_init\_\_(self, brand, model\_name, price):

        self.brand = brand

        self.model\_name = model\_name

        self.price = price

    @property

    def complete\_specification(self):

        return f'{self.brand} {self.model\_name} and the price is {self.price}'

    @property

    def full\_name(self):

        return f'{self.brand} {self.model\_name}'

class Smartphone(Phone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera):

        super().\_\_init\_\_(brand, model\_name, price)  # This is the most common ways to use the inhertance

        self.ram = ram

        self.internal\_memeory = internal\_memeory

        self.rear\_camera = rear\_camera

phone1 = Phone('Nokia', '1100', 1000)

phone2 = Smartphone('Oneplus', 'nord', 28000, '4gb', '128gb', '48mp')

print(phone2.full\_name)

print(phone2.complete\_specification)  # This will give the brand model\_name and the price, not giving the ram internal etc.

we can do multilevel inheritance in the class like the following example:

class Phone:

    def \_\_init\_\_(self, brand, model\_name, price):

        self.brand = brand

        self.model\_name = model\_name

        self.price = price

    @property

    def complete\_specification(self):

        return f'{self.brand} {self.model\_name} and the price is {self.price}'

    @property

    def full\_name(self):

        return f'{self.brand} {self.model\_name}'

class Smartphone(Phone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera):

        super().\_\_init\_\_(brand, model\_name, price)  # This is the most common ways to use the inhertance

        self.ram = ram

        self.internal\_memeory = internal\_memeory

        self.rear\_camera = rear\_camera

class Flagship\_phone(Smartphone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera, front\_camera):

        super().\_\_init\_\_(brand, model\_name, price, ram, internal\_memeory, rear\_camera)

        self.front\_camera = front\_camera

phone1 = Phone('Nokia', '1100', 1000)

phone2 = Smartphone('Oneplus', 'nord', 28000, '4gb', '128gb', '48mp')

phone3 = Flagship\_phone('Samsung', 's21 ultra', 106000, '12gb', '512gb', '108mp', '32mp')

print(phone2.full\_name)

print(phone2.complete\_specification)  # This will give the brand model\_name and the price, not giving the ram internal etc.

print(phone3.complete\_specification)

In this example we use the Phone attribute in the smartphone class and the smartphone class attributes in the Flagships phones class .This is called the multilevel inheritance.

Method resolution order:

This is the order in which python will check the method or the instance variable

We can check the method resolution order for the any class by printing the help function as 🡪 print(help(Flagship\_phone)

For eg: in the above example if we give the command like print(phone3.fullname) then python will search accordingly to method resolution order. Meaning it will first check in first in Flagship\_phone then in Smartphone and then in Phone and lastly in python builtin.objects if it found in any of them then it will execute from there.

Method overriding:

This is meaning if we can change the any method for any other class in class inheritance . As the python search the method according to the method resolution order so we can define the same method in first order so it excute from there.

Eg:

class Phone:

    def \_\_init\_\_(self, brand, model\_name, price):

        self.brand = brand

        self.model\_name = model\_name

        self.price = price

    @property

    def complete\_specification(self):

        return f'{self.brand} {self.model\_name} and the price is {self.price}'

    @property

    def full\_name(self):

        return f'{self.brand} {self.model\_name}'

class Smartphone(Phone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera):

        super().\_\_init\_\_(brand, model\_name, price)  # This is the most common ways to use the inhertance

        self.ram = ram

        self.internal\_memeory = internal\_memeory

        self.rear\_camera = rear\_camera

    @property

    def complete\_specification(self):

        return f'{self.brand} {self.model\_name} its RAM is {self.ram}, its Storage capacity of {self.internal\_memeory} '

class Flagship\_phone(Smartphone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera, front\_camera):

        super().\_\_init\_\_(brand, model\_name, price, ram, internal\_memeory, rear\_camera)

        self.front\_camera = front\_camera

    # Now if we want to add the camera's details for the flagships phone in the complete specification method then we can use the overriding method

    # by defining eailer

    @property

    def complete\_specification(self):

         return f'{self.brand} {self.model\_name} its RAM is {self.ram}, its Storage capacity of {self.internal\_memeory} and the camera deails as rear: {self.rear\_camera} front: {self.front\_camera}'

phone1 = Phone('Nokia', '1100', 1000)

phone2 = Smartphone('Oneplus', 'nord', 28000, '4gb', '128gb', '48mp')

phone3 = Flagship\_phone('Samsung', 's21 ultra', 106000, '12gb', '512gb', '108mp', '32mp')

print(phone3.complete\_specification)

In this example complete specification for the flagship phones will find in the flagship phone class so it will execute from there.

This is called the overriding method.

Now let’s talk about the builtin function.

Isinstance():

Isinstance function is used to check whether the objects is belonged from that class or not

Isinstance function take two argument first one will be the object name and 2nd will be the class name.

And if we print isinstance then it will give us the True or False boolam value

Eg:

class Phone:

    def \_\_init\_\_(self, brand, model\_name, price):

        self.brand = brand

        self.model\_name = model\_name

        self.price = price

phone = Phone('nokia', '1100', 1000)

# Now to check the whether this function is belonged to this class or not we do as follow:

print(isinstance(phone, Phone))

this will give the output as : True

**Note: if we inherit any class in other class then isinstance will show True for the both class.**

Issubclass():

This function will check whether the class is inherited in other class or not.

This will also take two variable 1st will be subclass and the 2nd will be the mainclass.

Eg:

class Phone:

    def \_\_init\_\_(self, brand, model\_name, price):

        self.brand = brand

        self.model\_name = model\_name

        self.price = price

class Smartphone(Phone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera):

        super().\_\_init\_\_(brand, model\_name, price)  # This is the most common ways to use the inhertance

        self.ram = ram

        self.internal\_memeory = internal\_memeory

        self.rear\_camera = rear\_camera

class Flagship\_phone(Smartphone):

    def \_\_init\_\_(self, brand, model\_name, price, ram, internal\_memeory, rear\_camera, front\_camera):

        super().\_\_init\_\_(brand, model\_name, price, ram, internal\_memeory, rear\_camera)

        self.front\_camera = front\_camera

phone1 = Phone('Nokia', '1100', 1000)

phone2 = Smartphone('Oneplus', 'nord', 28000, '4gb', '128gb', '48mp')

phone3 = Flagship\_phone('Samsung', 's21 ultra', 106000, '12gb', '512gb', '108mp', '32mp')

# now if we want to check whether smartphone is the subclass of Phone or not then we can do as follow:

print(issubclass(Smartphone, Phone))

this will give the output as : True

Multiple Inheritance:

Special magic method or Dunder method:

Dunder are those method which is written between double underscore on the both side. Like \_\_init\_\_