# Learn

## example of class :

class abc:

name = "chetan"

roll = 1

def func1(self):

print("func1")

obj = abc()

obj.func1()

obj.roll = 4

## example of class with inheritance :

class parent:

def func1(self):

print("func1")

class child(parent):

def func2(self):

print("func2")

class next\_child(parent , child): or class next\_child(child):

def func3(self):

print("func3")

obj = child()

obj.func1()

obj.func2()

obj1 = next\_child()

obj1.func1()

obj1.func2()

obj1.func3()

## Decorator

Decorator are the function which accepts the function and return the a function.

Decorator are used when you want to expand the existing function or add more functionality to the existing function. Without changing the existing funtion

Example 2: without arguments

def add\_decorator(func):

    def decorator():

        print("Something is happening before the function is called. is addtion")

        func()

    return decorator

@add\_decorator

def addition():

    print(1+2)

addition()

Example 2: with arguments

Note :

\*args allows a function to accept any number of positional arguments. It collects them into a tuple.

def func(\*args):

for arg in args:

print(arg)

func(1, 2, 3) # Output: 1 2 3

Code:

def add\_decorator(func):

    def decorator(\*args):

        print("Something is happening before the function is called. is addtion")

        func(\*args)

    return decorator

@add\_decorator

def addition(a, b):

    print(a+b)

addition(1,2)

output:

Something is happening before the function is called. is addtion

3

# Interview