9.1) Decorators

Decorators are a very powerful and useful tool in Python since it allows programmers to modify the behaviour of a function or class. Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it. But before diving deep into decorators let us understand some concepts that will come in handy in learning the decorators.

First Class Objects
In Python, functions are first class objects which means that functions in Python can be used or passed as arguments. Properties of first class functions:

> A function is an instance of the Object type.

> You can store the function in a variable.

> You can pass the function as a parameter to another function.

> You can return the function from a function.

> You can store them in data structures such as hash tables, lists, …

In [9]:

```
# Python program to illustrate functions
# can be treated as objects
def shout(text):
    return text.upper()

print(shout('Hello'))

yell = shout

print(yell('Hello'))
```

```
HELLO
HELLO
```

In [10]:

```
# Python program to illustrate functions
# can be passed as arguments to other functions
def shout(text):
    return text.upper()

def whisper(text):
    return text.lower()

def greet(func):
    # storing the function in a variable
    greeting = func("""Hi, I am created by a function passed as an argument.""")
    print (greeting)

greet(shout)
greet(whisper)
```

```
HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT.
hi, i am created by a function passed as an argument.
```

In [8]:

```
# Python program to illustrate functions
# Functions can return another function

def create_adder(x):
    def adder(y):
        return x+y

    return adder

add_15 = create_adder(15)

print(add_15(10))
```

```
25
```

```
Syntax for Decorator:

@gfg_decorator
def hello_decorator():
    print("Gfg")

'''Above code is equivalent to -

def hello_decorator():
    print("Gfg")
```

simply decorators decorate a function just like we decorate our house without permenantle changing the house

example1): Create a decorate that claculates the time taken to execute any function.

In [2]:

```
import time
def timer(func):  #timer function with a function as an argument
    def timer_inner():
        start = time.time()
        func()
        end = time.time()
        print(end-start)
    return timer_inner
```

In [3]:

```
def test1():
    import time
    start = time.time()
    print(88+46)
    end = time.time()
    print(end-start)
```

In [4]:

```
#Suppose we have to calculate the execution time for each
#function so instead of writing a code every time we can
# use a decorator
@timer
def test2():  #function test2() decorated with test timer
    print(88+46)
```

In [5]:

```
test1()
```

```
134
0.0
```

In [6]:

```
test2()
```

```
134
0.0
```

In [ ]:

```

```

Chaining Decorators

In simpler terms chaining decorators means decorating a function with multiple decorators.

Example:

In [7]:

```
# code for testing decorator chaining
def decor1(func):
    def inner():
        x = func()
        return x * x
    return inner

def decor(func):
    def inner():
        x = func()
        return 2 * x
    return inner

@decor1
@decor
def num():
    return 10

@decor
@decor1 #1st decorator to be executed
def num2():
    return 10

print(num())
print(num2())
```

```
400
200
```

In [ ]:

In [ ]:

9.2) Class Method

The classmethod is an inbuilt function in Python, which returns a class method for a given function.;

> Syntax: classmethod(function)

> accepts the function name as a parameter

> function returns the converted class method.

You can also use @classmethod decorator for classmethod definition.

In [21]:

```
#python program to demonstrate
# use of a class method and static method.

from datetime import date

class person:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def student_details(self):
        print(self.name,self.age)

    #a class method to create a
    #person object by birth year.
```

In [22]:

```
obj = person('asit', 'asit@gmail.com')
```

In [23]:

```
obj.student_details
#this shows that the method can be sccessed as a parameter
```

Out[23]:

```
<bound method person.student_details of <__main__.person object at 0x00000
2A08A1B0D00>>
```

In [24]:

```
obj.name #name parater
```

Out[24]:

```
'asit'
```

In [25]:

```
obj.age
```

Out[25]:

```
'asit@gmail.com'
```

From above we can see both name and age can be accessed directly but for method we have to apply () example

In [26]:

```
obj.student_details()
```

```
asit asit@gmail.com
```

To avoid using object and directly access function/method using class name we use @classmethod

In [42]:

```
#example
class person:
    def __init__(self,name,email):
        self.naam = name
        self.khat = email

    @classmethod
    def details (cls,name1,email1): #here cls is binding our entire dataset to the class
        return cls(name1,email1)

    def student_details(self):
        print(self.naam,self.khat)
```

In [43]:

```
#now we can access details method directly using class name
#without making an object
obj = person.details('asit','asit@gmail.com')
```

In [44]:

```
obj.naam
```

Out[44]:

```
'asit'
```

In [45]:

```
obj.khat
```

Out[45]:

```
'asit@gmail.com'
```

In [46]:

```
obj.student_details()
```

```
asit asit@gmail.com
```

Here we achieved Function overloading i.e this class method is able to assign the name1 and email1 arguments to the init function.

So Classmethod works as an alternative for intit method

In [ ]:

```

```

In [104]:

```
class person:

    mobile_number = 9134534535 #class variable can be accessed without creating an object

    def __init__(self, name , email):

        self.name = name
        self.email = email

    @classmethod  #ClassMethod
    def change_number(cls, mobile):
        person.mobile_number = mobile

    @classmethod  #ClassMethod
    def details(cls , name1 , email1):
        return cls(name1 , email1)

    def student_details(self): #instance method
        print(self.name , self.email,person.mobile_number)
```

In [105]:

```
#Class veriable can be accessed without mking a class object
#example
person.mobile_number
```

Out[105]:

```
9134534535
```

student_details() can be accessed in 2 ways:-

In [108]:

```
#1 By creating veriable of Classmethod

a = person.details('asit','asit@gmail')
a.student_details()
```

```
asit asit@gmail 9134534535
```

In [111]:

```
#2 By creating a class object(for __init__ method):

class_obj = person("asit","gmail.com")
class_obj.student_details()
```

```
asit gmail.com 9134534535
```

In [112]:

```
#Accessing and changing Mobilbe no. using ClassMethod
person.change_number("99889988")
person.mobile_number
```

Out[112]:

```
'99889988'
```

Class method

> Class method is a global method which will be accessable by all the object.It only tries to create one instances.

Instance/init method

> Creates multiple insatance for multiple objects. Therefore takes lot of memory.

In [127]:

```
#example
class person2:

    mobile_number = 9134534535 #class variable can be accessed without creating an object

    def __init__(self, name , email):

        self.name = name
        self.email = email

    @classmethod  #ClassMethod
    def change_number(cls, mobile):
        person2.mobile_number = mobile

    @classmethod  #ClassMethod
    def details(cls , name1 , email1):
        return cls(name1 , email1)

    def student_details(self): #instance method
        print(self.name , self.email,person2.mobile_number)
```

In [135]:

```
#passing details through instance method
person2_obj = person2('asit','asit@gmail')
```

In [136]:

```
#passing details using classmethod
a = person2_obj.details("Euphor","Euphor@gmail")
```

In [137]:

```
#Now lets see the above passed data
a.student_details()
#only once instance created
```

```
Euphor Euphor@gmail 9134534535
```

In [138]:

```
person2_obj.student_details()
#multiple objects there fore multiple instance
```

```
asit asit@gmail 9134534535
```

In [ ]:

```

```

Adding an External Function to a class

In [154]:

```
class person3:

    mobile_number = 9134534535 #class variable can be accessed without creating an object

    def __init__(self, name , email):

        self.name = name
        self.email = email

    @classmethod  #ClassMethod
    def change_number(cls, mobile):
        person3.mobile_number = mobile

    @classmethod  #ClassMethod
    def details(cls , name1 , email1):
        return cls(name1 , email1)

    def student_details(self): #instance method
        print(self.name , self.email,person3.mobile_number)
```

In [155]:

```
#Function to be added
def course_details(cls, cousre_name):
    print("course details:- ", cousre_name)
```

In [156]:

```
#adding Function
person3.course_details = classmethod(course_details)
```

In [157]:

```
#now course deatail is added:
person3.course_details("data science")
```

```
course details:-  data science
```

In [159]:

```
#accessing course details using object
person3_obj = person3("asit","asit2gmail")
```

In [163]:

```
person3_obj.course_details("Big Data")
```

```
course details:-  Big Data
```

In [ ]:

```

```

Deleting Function inside Class

In [165]:

```
#Deleting Change no Classmethod
class person4:

    mobile_number = 91345345345 #class variable can be accessed without creating an object

    def __init__(self, name , email):

        self.name = name
        self.email = email

    @classmethod  #ClassMethod
    def change_number(cls, mobile):
        person4.mobile_number = mobile

    @classmethod  #ClassMethod
    def details(cls , name1 , email1):
        return cls(name1 , email1)

    def student_details(self): #instance method
        print(self.name , self.email,person4.mobile_number)
```

In [167]:

```
#1st method
del person4.change_number
person4.change_number()
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_17316\1056426125.py in <module>
      1 #1st method
----> 2 del person4.change_number
      3 person4.change_number()

AttributeError: change_number
```

In [171]:

```
#2nd Method
#delattr("class name","method/variable")
person4.details("asit","@gmail")
```

Out[171]:

```
<__main__.person4 at 0x2a08a889760>
```

In [172]:

```
delattr(person4,"details")
```

In [173]:

```
person4.details()
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_17316\3240978607.py in <module>
----> 1 person4.details()

AttributeError: type object 'person4' has no attribute 'details'
```

In [ ]:

```

```

In [174]:

```
#deleting varible
person4.mobile_number
```

Out[174]:

```
9134534535
```

In [175]:

```
delattr(person4,"mobile_number")
```

In [176]:

```
person4.mobile_number
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_17316\2283090502.py in <module>
----> 1 person4.mobile_number

AttributeError: type object 'person4' has no attribute 'mobile_number'
```

In [ ]:

```

```

In [178]:

```
#deleting Instance method
delattr(person4,"student_details")
```

In [179]:

```
person4.student_details
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_17316\5600445.py in <module>
----> 1 person4.student_details

AttributeError: type object 'person4' has no attribute 'student_details'
```

In [ ]:

```

```

In [ ]:

```

```

9.3) Static Method

Static Vs Class method

Static Method	Class Method
needs no specific parameters	class method takes cls as the first parameter

Static Method	Class Method
static methods know nothing about the class state	class methods must have class as a parameter
use @staticmethod decorator to create a static method in python	use @classmethod decorator in python to create a class method

In [189]:

```
class information:
    def student_details(self , name , mail_id , number) :
        print(name , mail_id, number)
    #the above function depends on an object
    #So if we have millions of object then we have to create
    #millions of object which increases memory utilisation



    @staticmethod
    def mentor_mail_id(mail_id):
        print(mail_id)

    @staticmethod
    def mentor_class(list_mentor) : #no self/cls is claaed in @staticmethod
        print(list_mentor)
        information.mentor_mail_id(["krish@gmail.com" , "sudh@gmail.com"])

    @classmethod
    def class_name(cls,class_name):
        cls.mentor_class(["sudh" , "krish"])

    def mentor(self , mentor_list) : #can only be accessed htrough an onject
        print(mentor_list)
        self.mentor_class(["krish" , "sudh"])
```

In [190]:

```
#Object Creation
info_obj = information()
info_obj.student_details('asit','mail@mail',1769)

#the above function depends on an object
#So if we have millions of object then we have to create
#millions of object which increases memory utilisation
```

```
asit mail@mail 1769
```

In [194]:

```
# calling mentor class
information.mentor_class(["asit","krish","another sir"])
#shows that static method can be called directly
```

```
['asit', 'krish', 'another sir']
['krish@gmail.com', 'sudh@gmail.com']
```

In [198]:

```
#Accessing mentro():
info_obj_mentor = information()
info_obj_mentor.mentor(["asit","krish","sudh"])
```

```
['asit', 'krish', 'sudh']
['krish', 'sudh']
['krish@gmail.com', 'sudh@gmail.com']
```

9.4) Magic/Dunder Method

Its is not adviciable to use this method directly

In [199]:

```
#How to Know Dunder methods?
dir(str)
```

Out[199]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
```

In [200]:

```
#using dunder methods.
a = 10
a.__add__(6)
```

Out[200]:

```
16
```

In [201]:

```
#Getting an output from a class object
```

In [205]:

```
# by default a class objects does print an output
#example
class phone():
    def __init__(self):
        self.phone_no = 9944668834
        #return self.phone_no- not possible as __init__ should return none
```

In [207]:

```
a = phone()
print(a)
```

```
<__main__.phone object at 0x000002A0861A6CA0>
```

```
 'partition',
 'removeprefix',
 'removesuffix',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

In [209]:

```python
class phone1():
    def __init__(self):
        self.phone_no = 9946843218
    def __str__(self):
        return "This is method which will print something for object"
```

In [211]:

```python
obj = phone1()
print(obj)
```

```
This is method which will print something for object
```

In [ ]:

```python

```

In [ ]:

```python

```

9.5) Property Decorators

In [241]:

```python
class property_access():

    def __init__(self, course_price , course_name):
        self.__course_price = course_price
        self.course_name = course_name

    #helps in exposing our class property (i.e private veriable) to outer world.
    def course_price_access(self):
        return self.__course_price


    def course_price_set(self , price):
        if price <= 3500:
            pass
        else :
            self.__course_price = price


    def course_price_del(self):
        del  self.__course_price
```

In [242]:

```python
p_obj = property_access(3500,'Data')
```

In [243]:

```
p_obj.course_price_access()
```

Out[243]:

```
3500
```

In [244]:

```
p_obj.course_price_set(3600)
```

In [245]:

```
p_obj.course_price_access()
#Here we can see we can't change the value of the Private veriable
```

Out[245]:

```
3600
```

In [ ]:

```

```