polymorphism

July 28, 2024

[1]:
```
#super method used
```

[2]:
```
class pwskills:
    def __init__(self,mentor):
        #__init__ i sfunc which is used to pass the data
        self.mentor=mentor
  #init is also a constructor
    def mentor_name(self):
        print(self,mentor)

class datascience(pwskills):

    def __init__(self,mentor,mentor_mail_id):
        self.mentor_mail_id=mentor_mail_id
        self.mentor=mentor

    def show_info(self):
        print(self.mentor,self.mentor_mail_id)
```

[3]:
```
python_basic=datascience("sudhanshu","sdjf@gmail.com")
```

[4]:
```
python_basic.show_info()
```

```
sudhanshu sdjf@gmail.com
```

[6]:
```
python_basic.mentor_mail_id
```

[6]:
```
'sdjf@gmail.com'
```

[7]:
```
python_basic.mentor
```

[7]:
```
'sudhanshu'
```

[24]:
```
class pwskills:
    def __init__(self,mentor):
        #__init__ i sfunc which is used to pass the data
```

```
        self.mentor=mentor
  #init is also a constructor
    def mentor_name(self):
        print(self,mentor)

class datascience(pwskills):

    def __init__(self,mentor,mentor_mail_id):

        super().__init__(mentor)
        #super keyword used when we reassign the value
        self.mentor_mail_id=mentor_mail_id


        #self.mentor=mentor

    def show_info(self):
        print(self.mentor,self.mentor_mail_id)
```

[25]:
```
python_basic=datascience("sudhanshu","sdjf@gmail.com")
```

[26]:
```
python_basic.show_info
```

[26]:
```
<bound method datascience.show_info of <__main__.datascience object at
0x7f7c48f1ccd0>>
```

[27]:
```
python_basic.mentor_name
```

[27]:
```
<bound method pwskills.mentor_name of <__main__.datascience object at
0x7f7c48f1ccd0>>
```

[28]:
```
class pwskills:
    def __init__(self,mentor):
        #__init__ i sfunc which is used to pass the data
        self.mentor=mentor
  #init is also a constructor
    def mentor_name(self):
        print(self,mentor)

class datascience(pwskills):

    def __init__(self,mentor,mentor_mail_id):
        self.mentor_mail_id=mentor_mail_id
        #self.mentor=mentor
        super().__init__(mentor)
```

```
    def show_info(self):
        print(self.mentor,self.mentor_mail_id)
```

[29]:
```
python_basic=datascience("sudhanshu","sdjf@gmail.com")
```

[30]:
```
python_basic.mentor_mail_id
```

[30]:
```
'sdjf@gmail.com'
```

[69]:
```
class human:
    def __init__(self):
        pass
    def eat(self):

        print("print the eat method from human")
```

[70]:
```
class male(human):
    def __init__(self,name):
        self.name=name

    def eat(self):
        super().eat()
        print(self.name)
```

[71]:
```
nare=male("naresh")
```

[72]:
```
nare.eat()
```

```
print the eat method from human
naresh
```

[74]:
```
nare.eat
```

[74]:
```
<bound method male.eat of <__main__.male object at 0x7f7c491b5540>>
```

[1]:
```
#destructor
```

[6]:
```
class fileopener:
    def __init__(self,filename):
        self.filename=filename

    def open_file(self):
        print("this will open the file",self.filename)
        #del is kind of dundur method it is also a destructor

    def __del__(self):
        self.filename
```

[7]:
```
f1=fileopener("f1.txt")
```

[8]:
```
f1.open_file()
```

```
this will open the file f1.txt
```

[25]:
```
import time
class timer:
    def __init__(self):
        self.start_time=time.time()

    def task(self):
        time_spent=time.time()-self.start_time
        print(time_spent)

    def __del__(self):
        print("")

    def __str__(self):
        return "this is my class timer"
```

[26]:
```
t1=timer()
```

[27]:
```
t1.start_time
```

[27]:
```
1716178348.7964313
```

[28]:
```
t1.task()
```

```
0.6354708671569824
```

[29]:
```
print(t1)
```

```
this is my class timer
```

[30]:
```
import time

class timer:
    def __init__(self):
        self.start_time=time.time()

    def task(self):
        time_spent=time.time()-self.start_time()

    def __del__(self):
        print("")
```

[33]:
```
t2=timer()
```

[35]:
```
t2.task
```

[35]:
```
<bound method timer.task of <__main__.timer object at 0x7f1a58047190>>
```

[38]:
```
t2.start_time
```

[38]:
```
1716179020.7967763
```

[12]:
```
#decorator func
```

[13]:
```
def test(func):
    def inner_test():
        print("this is the start of my inner test")
        func()
        print("this is the end of my inner test")

    return inner_test
@test
def test1():
    print("this is my test1")

@test
def test2():
    print( "this is my test2")
```

[14]:
```
test1()
```

```
this is the start of my inner test
this is my test1
this is the end of my inner test
```

[15]:
```
test2()
```

```
this is the start of my inner test
this is my test2
this is the end of my inner test
```

[17]:
```
test(test1)
```

[17]:
```
<function __main__.test.<locals>.inner_test()>
```

[5]:
```
import time

def print_list(l):
    start_time=time.time()
```

```
    for i in l:
        print(l)

    end_time=time.time()
    total_time=end_time-start_time
    print(total_time)
```

[6]:
```
print_list([1,2,23,332])
```

```
[1, 2, 23, 332]
[1, 2, 23, 332]
[1, 2, 23, 332]
[1, 2, 23, 332]
7.033348083496094e-05
```

[7]:
```
import time
def print_key(d):
    start_time=time.time()
    print(d.keys())
    end_time=time.time()
    total_time=end_time-start_time
    print(total_time)
```

[8]:
```
def find_time(func):
    def cal_time(*args):
        start_time=time.time()
        func(*args)
        end_time=time.time()
        total_time=end_time-start_time
        print(total_time)

    return cal_time
```

[9]:
```
#@is called as decorator
```

[10]:
```
@find_time
def print_key(d):
    print(d.keys())
```

[11]:
```
print_key({"name":"sony","age":18,"clg":"NITA"})
```

```
dict_keys(['name', 'age', 'clg'])
5.459785461425781e-05
```

[12]:
```
@find_time
def print_list(l):
```

```
    for i in l:
        print(l)
```

[13]:
```
print_list([2,32,32,4334,3])
```

```
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
7.152557373046875e-05
```

[14]:
```
import logging
def log_func(func):
    def log_inner(*args):
        logging.basicConfig(filename="test.log",level=logging.INFO)
        logging.info ("this is  the start of my func")
        func(*args)
        logging.info("this is end of the func")
    return log_inner
```

[15]:
```
@log_func
@find_time
def print_list(l):

    for i in l:
        print(l)
```

[16]:
```
print_list([2,32,32,4334,3])
```

```
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
[2, 32, 32, 4334, 3]
7.128715515136719e-05
```

[27]:
```
class sony:
    def __init__(self,subject):
        self.__subject=subject
 #_subject(_)=it is protected variable

#public()=accesiable everyway
#protected(_)=accesiable in a particular place
#private(__)=restrited or in a private
```

[28]:
```
s1=sony("python")
```

[32]:
```
s1._sony__subject="datascience"
```

[34]:
```
s1._sony__subject#private variable
```

[34]:
```
'datascience'
```

[36]:
```
class sony:
    def __init__(self,subject):
        self.__subject=subject
    #property is a decorator which is already written
    @property
    def subject(self):
        return self.__subject

    @subject.setter
    def subject(self,subject):
        self.__subject=subject

    @subject.getter
    def subject(self,subject):
        return self.__subject
```

[42]:
```
s2=sony("big data")
```

[43]:
```
s2.subject
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[43], line 1
----> 1 s2.subject

TypeError: sony.subject() missing 1 required positional argument: 'subject'
```

[44]:
```
s3=sony("data science")
```

[45]:
```
s3.subject
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[45], line 1
----> 1 s3.subject

TypeError: sony.subject() missing 1 required positional argument: 'subject'
```

[54]:
```
class sony:
    def __init__(self,subject):
        self.subject=subject

    @property
    def subject(self):
        return self.__subject

    @subject.setter
    def subject(self,subject):
        self.__subject=subject

    @subject.getter
    def subject(self):
        return self.__subject
```

[55]:
```
s4=sony("bigdata")
```

[56]:
```
s4.subject
```

[56]:
```
'bigdata'
```

[57]:
```
#polymorphism
#ploy=many,morphism=different
```

[58]:
```
def test(a,b):
    return a+b
```

[60]:
```
test(3,4)
```

[60]:
```
7
```

[63]:
```
test("sony","jarupula")
#in this it acts concatination operation
```

[63]:
```
'sonyjarupula'
```

[64]:
```
test("[1,2,3,4]","[443,54,5]")
```

[64]:
```
'[1,2,3,4][443,54,5]'
```

[69]:
```
class pwskills:
    def students(self):
        pass

class datascience(pwskills):
    def student(self):
```

```
        print("this will give me a datascience student")

class bigdata(pwskills):
    def student(self):
        print("this will give me a detais about big data")

sony=datascience()
naresh=bigdata()
sony.student()
naresh.student()
#it is polymorphism by method of over writting
```

```
this will give me a datascience student
this will give me a detais about big data
```

[1]:
```
#method over loading
```

[19]:
```
class bigdata:
    def __init__(self,num_of_class,num_of_stud):
        self.num_of_class=num_of_class
        self.num_of_stud=num_of_stud

    def __add__(self,other):
        return bigdata(self.num_of_class+other.num_of_class,self.
 ↪num_of_stud+other.num_of_stud)
   #add func is avalible in the python ,by this we can perform addition
  #other is not a keyword ,w ecan give our name also
c1=bigdata(1,212)
c2=bigdata(2,12)
c3=bigdata(3,2323)
result=c1+c2+c3
print(result.num_of_class,result.num_of_stud)
```

```
6 2547
```

[20]:
```
print(c1)
```

```
<__main__.bigdata object at 0x7fe5c91b12a0>
```

[21]:
```
print(c2)
```

```
<__main__.bigdata object at 0x7fe5c91b38e0>
```

[22]:
```
print(result)
```

```
<__main__.bigdata object at 0x7fe5c91b1e10>
```

[17]:
```
class datascience:
    def student(self):
        print("this will give me a details about data science student")
```

[18]:
```
i1=datascience()
i2=datascience()
i3=datascience()
i=i1+i2+i3
print(i)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[18], line 4
      2 i2=datascience()
      3 i3=datascience()
----> 4 i=i1+i2+i3
      5 print(i)

TypeError: unsupported operand type(s) for +: 'datascience' and 'datascience'
```

[23]:
```
class datascience:
    def student(self):
        print("this will give me a datascience student")

class bigdata:
    def student(self):
        print("this will give me a detais about big data")

def output_class(class_obj):
    return class_obj.student()
```

[33]:
```
sony=datascience()
naresh=bigdata()
```

[34]:
```
sony.student()
```

[34]:
```
'this will give me a datascience student'
```

[35]:
```
naresh.student()
```

```
this will give me a detais about big data
```

[25]:
```
output_class(sony)
```

```
this will give me a datascience student
```

[26]:
```
output_class(naresh)
```

```
this will give me a detais about big data
```

[27]:
```
#the above one is ducktile polymorphsim
```

[29]:
```
class datascience:
    def student(self):
        return "this will give me a datascience student"

class bigdata:
    def student(self):
        print("this will give me a detais about big data")
```

[36]:
```
kamala=datascience()
papa=bigdata()
```

[39]:
```
kamala.student()
```

[39]:
```
'this will give me a datascience student'
```

[38]:
```
naresh.student()
```

```
this will give me a detais about big data
```

[40]:
```
class pwskills:
    def students(self):
        return "this is a pwskills student"

class datascience(pwskills):
    def student(self):
        print("this will give me a datascience student")
```

[41]:
```
sony=datascience()
sony.student()
```

```
this will give me a datascience student
```

[44]:
```
len("osny")#len is also a polymorpshim
```

[44]:
```
4
```

[45]:
```
len([12,2,233,3])
```

[45]:
```
4
```

[46]:
```
#encapsulation
#hidding a detail
#example is the capsule which we will take if we are sick
```

[48]:
```
class test:
    def __init__(self):
        self.__x=='sony'
```

[50]:
```
t1=test()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[50], line 1
----> 1 t1=test()

Cell In[48], line 3, in test.__init__(self)
      2 def __init__(self):
----> 3     self.__x=='sony'

AttributeError: 'test' object has no attribute '_test__x'
```

[51]:
```
t1.__x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[51], line 1
----> 1 t1.__x

NameError: name 't1' is not defined
```

[4]:
```
class test:
    def __init__(self):
        self.x="sony"
```

[5]:
```
t1.test()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 1
```

```
----> 1 t1.test()

NameError: name 't1' is not defined
```

[1]:
```
class test:
    def __init__(self):
        self.__x="sony"
        self.y="naresh"
        self.z="kamala"

    def __test_meth(self):
        return "this is just a test"

    def acess_var(self):
        return self.__x
    def update_var(self,data):
        self.__x=data
```

[2]:
```
s1=test()
```

[3]:
```
s1.update_var(324344)
```

[4]:
```
s1.acess_var()
```

[4]:
```
324344
```

[5]:
```
s1.y
```

[5]:
```
'naresh'
```

[6]:
```
s1._test__test_meth()
```

[6]:
```
'this is just a test'
```

[7]:
```
class bank:
    def __init__(self,account_number,balance):
        self.account_number=account_number
        self.__balance=balance

    def check_balance(self,password):
        if password =="mysecurepass":
            return self.__balance
        else:
            return "incorrect password"

sony=bank(3323,565)
sony.check_balance("mysecurepass")
```

[7]:
```
565
```

[9]:
```
sony.account_number
```

[9]:
```
3323
```

[11]:
```
sony.check_balance("jndfsjh")
```

[11]:
```
'incorrect password'
```

[12]:
```
class queue:
    def __init__(self):
        self.__queue=[]


    def enqueue(self,data):
        self._queue.append(data)

    def dequeue(self):
        if self._queue:
            return self._queue.pop(0)
        else:
            print("its empty")
```

[1]:
```
class test:
    def __init__(self):
        self.__x="sony"
        self.y="naresh"
        self.z="kamala"

    def __test_meth(self):
        return "this is just a test"

    def acess_var(self):
        return self.__x

    def update_var(self,data):
        self.__x=data
```

[2]:
```
t1=test()
```

[5]:
```
t1._test__x
```

[5]:
```
'sony'
```

[6]:
```
t1._test__test_meth()
```

[6]:
```
'this is just a test'
```

[8]:
```
t1.update_var
```

[8]:
```
<bound method test.update_var of <__main__.test object at 0x7fbe2465d660>>
```

[9]:
```
t1.acess_var()
```

[9]:
```
'sony'
```

[11]:
```
class bank :
    def __init__(self,account_num,balance):
        self._account_num=account_num
        self.__balance=balance

    def check_balance(self,password):
        if password =="mysecurepass":
            return self.__balance
        else:
            return "incorrect password"

sony=bank(21333,2332121)
sony.check_balance("mysecurepass")
```

[11]:
```
2332121
```

[21]:
```
class queue:
    def __init__(self):
        self._queue=[]

    def enqueue(self,data):
        self._queue.append(data)

    def dequeue(self):
        if self._queue:
            return self._queue.pop(0)
        else:
            print("its empty")

    def showdata(self):
        return self._queue
```

[22]:
```
q=queue()
```

[23]:
```
q.dequeue()
```

```
its empty
```

[30]:
```
q.dequeue
```

[30]:
```
<bound method queue.dequeue of <__main__.queue object at 0x7fbe1459bb50>>
```

[31]:
```
q.enqueue(8970)
```

[32]:
```
q.dequeue()
```

[32]:
```
8970
```

[33]:
```
q._queue
```

[33]:
```
[]
```

[34]:
```
q.enqueue(32)
```

[37]:
```
q._queue
```

[37]:
```
[32]
```

[38]:
```
l=[2,21,3,32,212]
```

[40]:
```
l.pop()
```

[40]:
```
212
```

[42]:
```
#also is about creating a skeleton
#abstraction=creation of out lone
```

[1]:
```
class pwskills:
    def student_details(self):
        return "this will give u a student details"
```

[9]:
```
from abc import ABC,abstractmethod
class pwskills:
    @abstractmethod
    def databaseconnect(self):
        pass
    @abstractmethod
    def checkuserenrollment(self,user_mailid):
        pass
    @abstractmethod
    def check_completed_lecture(self,user_id,class_id):
        pass
    @abstractmethod
    def check_lab_usases(self,user_id):
```

```
        pass
    @abstractmethod
    def check_intership(self,user_id):
        pass
```

[10]:
```
class databaseconnect(pwskills):
    def databaseconnect(self):
        print("this is a implementation of database connect")
```

[11]:
```
db1=databaseconnect()
```

[8]:
```
db1.
```

```
  Cell In[8], line 1
    db1.
        ^
SyntaxError: invalid syntax
```

[12]:
```
class queue:
    def __init__(self):
        self._queue=[]
       #using a private variable
    def enqueue(self,data):
        self._queue.append(data)

    def dequeue(self):
        if self._queue:
            return self._queue.pop(0)
        else:
            print("its empty")
```

[13]:
```
q=queue()
```

[14]:
```
q.enqueue(23)
```

[15]:
```
q._queue
```

[15]: [23]

[16]:
```
q.dequeue()
```

[16]: 23

[18]:
```
l=[1,2,23]
```

[19]:
```
l.pop()
```

[19]:
```
23
```

[21]:
```
l.pop(1)
```

[21]:
```
2
```

[22]:
```
#abstraction=which creates a skeleton

class pwskills:
    def student_details(self):
        return "this will give u a student details"
```

[27]:
```
from abc import ABC,abstractmethod

class pwskills:
    @abstractmethod
    def databaseconnect(self):
        pass
    @abstractmethod
    def checkuserenrollment(self,user_mailid):
        pass
    @abstractmethod
    def check_completed_lecture(self,user_id,class_id):
        pass
    @abstractmethod
    def check_lab_usases(self,user_id):
        pass
    @abstractmethod
    def check_internship(self,user_id):
        pass
```

[28]:
```
class databaseconnect(pwskills):
    def databaseconnect(self):
        print("this is a implementation of database connect")
    def checkuserenrollment(self,user_mailid):
        return "test"

    def check_completed_lecture(self,user_id,class_id):
        return "test"
    def check_lab_usases(self,user_id):
        return "test"
```

```
    def  check_internship(self,user_id):
        return "test"
```

[29]:
```
db1=databaseconnect()
```

[30]:
```
db1.databaseconnect()
```

```
this is a implementation of database connect
```

[31]:
```
db1.check_internship("sony")
```

[31]:
```
'test'
```

[4]:
```
class calculation:
    def add(x,y):
        return x+y
    def sub(x,y):
        return x-y

    def div(self,x,y):
        return x/y
```

[5]:
```
a=calculation()
```

[6]:
```
a.add(2,2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 a.add(2,2)

TypeError: calculation.add() takes 2 positional arguments but 3 were given
```

[7]:
```
a.div(2,3)
```

[7]:
```
0.6666666666666666
```

[9]:
```
b=calculation()
```

[10]:
```
b.add(self,2,2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[10], line 1
```

```
----> 1 b.add(self,2,2)

NameError: name 'self' is not defined
```

[14]:
```
class calculation:
    @staticmethod
    def add(x,y):
        return x+y
    @staticmethod
    def sub(x,y):
        return x-y
    def div(self,x,y):
        return x/y
```

[19]:
```
class cal(calculation):
    def add(x,y):
        return x*y
```

[20]:
```
cal.add(4,5)
```

[20]: 20

[21]:
```
a=calculation()
```

[22]:
```
a.add(2,23)
```

[22]: 25

[23]:
```
c1=ca1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 c1=ca1

NameError: name 'ca1' is not defined
```

[ ]: