Day-9 Morning Assessment

OOPs

1. Self is a keyword used to initialize/refer to instance variables or instance methods whereas cls is a keyword used to refer the cls attributes and cls methods. These class methods and attributes are directly associated with the class but not with any instance.
2. Inheritance is a concept in python where child class inherits all the attributes and methods from the parent class and also add their own attributes or methods if any. It is used to reuse the code.

Ex of Method Overriding:

class Shape(self):

def area(self):

print(“Cannot determine the area of the shape”)

class Circle(Shape):

def area(self,radius):

print(f“Circle area is {2\*3.14\*radius\*radius}”)

c = Circle()

c.area(5)

1. Method Overloading is a concept in Polymorphism, where multiple functions are created with same method name but with different parameters which allows them to execute based on the users input parameters. However, this concept is not natively supported in Python.
2. Constructor : used to create or initiliaze the object. \_\_init\_\_()

Destructor: used to delete the object. \_\_del\_\_()

1. Instance Method: The method which is associated the instance of the class. It can be defined with self parameter in it.

Class Method: The method which is directly associated with class. It can be defined with cls parameter in it.

Static Method: The method which is not associated with anything and defined by nothing. It is only used as an utility function.

1. We can restrict access to class variables by adding access modifiers to the variable. We can use private access modifier to the variable(\_\_).

Eg: \_\_is\_access\_granted = False.

1. class Student:

def get\_name(self):

print(f”{self.\_\_name}”)

def set\_name(self,name):

self.\_\_name = name

s = Student()

s.set\_name(“Keerthi”)

s.get\_name()

1. Polymorphism : Polymorphism is a concept in OOPs. Poly means many and morph means form. So, polymorphism usually means a method with different forms i.e., many methods use same method name in different classes. So, based on the class and object we call, the method gets executed.

Eg:

class Dog:

def speak(self):

print(“Dog Barks”)

class Cat:

def speak(self):

print(“Cat Meows”)

def sound(Animal):

animal.speak()

d = Dog()

c = Cat()

lis = [d,c]

for i in lis:

sound(i)

1. Magic methods are special methods defined to customize the behavior of the objects.

These are represented by double underscores and also known as dunder methods. Eg: \_\_init\_\_(), \_\_str\_\_(), \_\_del()\_\_, etc..

1. Isinstance() is used as isinstance(obj, class)

Issubclass() is used as issubclass(subclass, baseclass)

Decorators

1. Decorators are used to change the behavior of the function.
2. def logger(func):

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

print(“calling the function now…!”)

return func(\*args,\*\*kwargs)

return wrapper

@logger

def func():

pass

func()

1. Yes, we can apply more than one decorator to the function in bottom to top order.
2. Functools.wraps() preserves the function’s attributes data.
3. def decorator(func):

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

print(“calling the function now…!”)

return func(\*args,\*\*kwargs)

return wrapper

@decorator

def func():

pass

func()

1. def decorator(func):

def wrapper(logged\_in,\*args,\*\*kwargs):

if logged\_in:

return func(\*args,\*\*kwargs)

else:

print(“Please login to enter into the function”)

return wrapper

@decorator

def func():

pass

func(logged\_in)

1. Property decorator allows you to create methods in a class that can just be accessed like attributes.

Ex:

class Square:

def \_\_init\_\_(self,side):

self.side =side

@property

def area(self):

print(f”area of the square is {side\*side}”)

1. def decorator(func):

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

try:

return func(\*args,\*\*kwargs)

except Exception as e:

print(“Exception occurred: ”,e)

return wrapper

@decorator

def func():

pass

func()

1. Function Decorator is used to modify the function’s behavior without changing its code/implementation. Whereas Class Decorator is used to modify the class behavior without changing its implementation.
2. Yes, we can use decorators with both class methods and static methods.

Generators

1. Generators are used to yield and return the iterator and pauses the function. It yields one by one. Whereas normal functions returns the total output at a time and it doesn’t pause the function.
2. def gen():

for i in range (1,21):

if i%2 == 0:

yield i

1. If we call next() on the exhausted generator, it returns StopIteration Exception.
2. As yield returns value one by one and pauses the function, it saves the memory.
3. Generator expression is (x for i in range(n) if i%2==0)

List comprehension expression is [x for i in range(n) if i%2 == 0)

1. def gen():

for i in range(20):

yield (i)

1. def gen():

with open(“file.txt”,’r’) as f:

for line in file:

yield (line)

1. So, the generator records the position between yields automatically.
2. So, return comes at the last of the function, prints total output at a time, runs only one time and ends the function, whereas yield doesn’t come at the end of the function, yields output one by one, pauses the function and yields numerous times.
3. In list(generator\_function), fetching of the total result might be slow as it yields one by one, where as in list- returning function, the total result is fetched immediately.

Iterators

1. Iterable is the one gets iterated like list, string,tuple etc…whereas the iterator is the one iterates the iterable like for loop, \_\_iter\_\_(), etc..
2. class Example:

def \_\_init\_\_(self):

self.num = 1

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.num <=10:

print(f’{self.num})

self.num += 1

else:

raise StopIteration

1. StopIteration is raised as to declare the end of the function.
3. For loop internally calls iter() and then next()
4. Map(), filter(), reduce(), zip(),etc..
5. it = iter([‘a’,’b’,’c’,’d’])

While it:

print(next(it))

1. it = iter([1,2,3,4,5])

while it:

r= next(it)

print(r\*r)

1. StopIteration Exception raises if we try to iterate over exhausted iterator.
2. In python, itertools module consists many functions such as count(),combinations(),permutations(), etc..which makes writing code easier.