Lecture11

October 8, 2024

[25]:
```
class Car:
    wheels = 4

    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def start_engine(self):
        return f"The engine of the {self.year} {self.make} {self.model} is now␣
↪running."

    def stop_engine(self):
        return f"The engine of the {self.year} {self.make} {self.model} is now␣
↪off."


my_car = Car("Toyota", "Corolla", 2020)
print(my_car.make)
print(my_car.model)
print(my_car.year)

print(my_car.start_engine())
print(my_car.stop_engine())
```

```
Toyota
Corolla
2020
The engine of the 2020 Toyota Corolla is now running.
The engine of the 2020 Toyota Corolla is now off.
```

[27]:
```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_checked_out = False # be rented out
    def check_out(self):
        if not self.is_checked_out:
            self.is_checked_out = True
            return f"{self.title} has been checked out."
        else:
            return f"{self.title} is already checked out."

    def return_book(self):
        if self.is_checked_out:
            self.is_checked_out = False
            return f"{self.title} has been returned."
        else:
            return f"{self.title} was not checked out."

book1 = Book("1984", "George Orwell", "1234567890")
book2 = Book("To Kill a Mockingbird", "Harper Lee", "0987654321")

print(book1.check_out())
print(book1.return_book())
print(book2.check_out())
```

```
1984 has been checked out.
1984 has been returned.
To Kill a Mockingbird has been checked out.
To Kill a Mockingbird is already checked out.
```

[6]:
```
class Dog:
    species = "mammal"

    def __init__(self, name, age):
        self.name = name
        self.age = age

dog1 = Dog("Philo", 5)
dog2 = Dog("Mikey", 6)

print("{} is {} and {} is {}.".format(dog1.name, dog1.age, dog2.name, dog2.age))

if dog1.species == "mammal":
    print("{} is a {}!".format(dog1.name, dog1.species))
```

```
Philo is 5 and Mikey is 6.
Philo is a mammal!
```

[13]:
```
class Dog:
    species = "mammal"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def description(self):
        return "{} is {} years old.".format(self.name, self.age)

    def speak(self, sound):
        return "{} says {}".format(self.name, sound)

mikey = Dog("Mikey", 6)

print(mikey.description())
print(mikey.speak("Gruff gruff!"))
```

```
Mikey is 6 years old.
Mikey says Gruff gruff!
```

[43]:
```
class Calculate_area:
    # Instance Method
    def rectangle_area(self, w, h):
        return w * h

    @classmethod
    def triangle_area(cls, b, h):
        return 0.5 * b * h

    @staticmethod
    def circle_area(r):
        return 3.14 * r * r

cal = Calculate_area()
cal_rec = cal.rectangle_area(4, 5)
cal_tri = cal.triangle_area(4, 5)
cal_circle = cal.circle_area(5)

print('Rectangle Area =', cal_rec)
print('Triangle Area =', cal_tri)
print('Circle Area =', cal_circle)

# print('Test Triangle Area =', Calculate_area.triangle_area(4, 5))
# print('Test Circle Area =', Calculate_area.circle_area(5))
# print('Test Rectangle Area =', Calculate_area.rectangle_area(5, 6)) # Error␣
↪because it is an instance method
```

```
Rectangle Area = 20
Triangle Area = 10.0
Circle Area = 78.5
Test Triangle Area = 10.0
```

[44]:
```python
class StudentTest:
    def __init__(self, name, score1, score2, score3):
        self.name = name
        self.score1 = score1
        self.score2 = score2
        self.score3 = score3

    def sumScores(self):
        return self.score1 + self.score2 + self.score3

    def __str__(self):
        return "Name:{}, Total of Scores:{}".format(self.name, self.sumScores())

std1 = StudentTest("Jantra", 20, 35, 25)
print(std1.name, std1.sumScores())
print(std1)
```

```
Jantra 80
Name:Jantra, Total of Scores:80
```

[47]:
```python
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return "Some sound"

class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"

class Cat(Animal):
    def speak(self):
        return f"{self.name} says Meow!"

dog = Dog("Buddy")
cat = Cat("Whiskers")

print(dog.speak())
print(cat.speak())
```

```
Buddy says Woof!
Whiskers says Meow!
```

[48]:
```python
class Dog:
    species = "mammal"

    def calAge(self, age):
        print('Dog Age is {}'.format(age*3))

class SomeBreed(Dog):
    pass

class SomeOtherBreed(Dog):
    species = "reptile"
    def calAge(self, age):
        print('Dog Age is {}'.format(age*4))

frank = SomeBreed()
print(frank.species)
frank.calAge(5)

beans = SomeOtherBreed()
print(beans.species)
beans.calAge(5)
```

```
mammal
Dog Age is 15
reptile
Dog Age is 20
```

[1]:
```
class Animal:
    def speak(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Dog(Animal):
    def speak(self):
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

def make_animal_speak(animal):
    print(animal.speak())

dog = Dog()
cat = Cat()

make_animal_speak(dog)
make_animal_speak(cat)
```

```
Woof!
Meow!
```

[3]:
```python
class Shape:
    def area(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        from math import pi
        return pi * (self.radius ** 2)

shapes = [Rectangle(10, 20), Circle(5)]

for shape in shapes:
    print(f"Area: {shape.area()}")
```

```
Area: 200
Area: 78.53981633974483
```

[6]:
```python
class employee(object):
    def __init__(self):
        self.name = "Peter"
        self._age = 45
        self.__salary = 35000

object1 = employee()
print(object1.name)
print(object1._age)
print(object1.__salary)
```

```
Peter
45
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[6], line 10
      8 print(object1.name)
      9 print(object1._age)
---> 10 print(object1.__salary)
```

```
AttributeError: 'employee' object has no attribute '__salary'
```

[7]:
```
class employee(object):
    def __init__(self):
        self.__maxearn = 30000

    def earn(self):
        print("earning is:{}".format(self.__maxearn))

    def setmaxearn(self, earn):
        self.__maxearn = earn

em1 = employee()
em1.earn()

em1.__maxearn = 10000
em1.earn()

em1.setmaxearn(15000)
em1.earn()
```

```
earning is:30000
earning is:30000
earning is:15000
```

[8]:
```
from abc import ABC, abstractmethod

class employee(ABC):
    def emp_id(self, id, name, age, salary):
        pass

class childemployee(employee):
    def emp_id(self, id):
        print("emp_id is 12345")

emp1 = childemployee()
emp1.emp_id(id)
```

```
emp_id is 12345
```

[9]:
```
from abc import ABC, abstractmethod
class Absclass(ABC):
    def print(self, x):
        print("Passed value: ", x)
    @abstractmethod
    def task(self):
        print("We are inside Absclass task")
```

```python
class test_class(Absclass):
    def task(self):
        print("We are inside test_class task")

class example_class(Absclass):
    def task(self):
        print("We are inside example_class task")

test_obj = test_class()
test_obj.task()
test_obj.print(100)

example_obj = example_class()
example_obj.task()
example_obj.print(200)

print('test_obj is instance of Absclass? ', isinstance(test_obj, Absclass))
print('example_obj is instance of Absclass? ', isinstance(example_obj, Absclass))
```

```
We are inside test_class task
Passed value:  100
We are inside example_class task
Passed value:  200
test_obj is instance of Absclass?  True
example_obj is instance of Absclass?  True
```