Classes

Python basics

Kunal Khurana

2023-10-06

Table of contents

Learning outcomes . . . 2
Remarks* . . . 2
Creating a Dog class . . . 3
Making an instance from a Class . . . 3
Accessing atributes . . . 4
Calling methods . . . 4
Creating multiple instances . . . 4
Example below . . . 5
creating a Restraunt class . . . 5
Making an instance from a class . . . 5
Printing individual attributes . . . 5
Calling methods . . . 5
Working with classes and instances . . . 6
Modifying an Attribute's Value Through a Method . . . 7
Inheritance . . . 8
Definig attributes and methods for child class . . . 9
Extending battery class . . . 10
Importing classes . . . 12
Importing from python's standard library . . . 12
Rolling a die 10 times and storing it in class Die . . . 13
Lottery analysis . . . 13
print four numbers or letters from a list of 10 numbers and 5 letters. . . . 13

Learning outcomes

1. a brief introduction to object-oriented programming
2. writing and storing information in classes
3. *init*() method
4. intro to python's standard library and random module

Remarks*

1. Making an object from a class is called instantiation, and you work with instances of a class.

2. we use title case to represents classes in python
3. Make sure to use two underscores on each side of **init**().
4. Any variable prefixed with self is available to every method in the class, and we'll also be able to access these variables through any instance created from the class.
5. The super() function at is a special function that allows you to call a method from the parent class.
6. Add the module's directory to the path by importing sys at the beginning of Jupyter Notebook.

```
import sys
sys.path.append("E://machine learning projects//car.py")
```

Creating a Dog class

```
class Dog:  #caplilazed name in python refers to classes

    def __init__(self, name, age):   #dockstring defining the class with 3 parameters; mak
        #initiliazing attributes; self passed automatically
        self.name = name             #argument
        self.age = age               #argument

    def sit(self):
        #simulate sitting
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        #simulate roll over
        print(f"{self.name} rolled over!")
```

Making an instance from a Class

```
my_dog = Dog('Bruno', 6)                #lowercase in my_dog refers to single instance cre

print(f"My dog's name is {my_dog.name}.")
print(f"My dog's age is {my_dog.age}.")
```

```
My dog's name is Bruno.
My dog's age is 6.
```

Accessing atributes

```
my_dog.name
```

```
'Bruno'
```

Calling methods

```
my_dog.sit()
my_dog.roll_over()
```

```
Bruno is now sitting.
Bruno rolled over!
```

Creating multiple instances

```
my_dog = Dog('Rambo', 6)
your_dog = Dog('Tyson', .5)

print(f"My dog's name is {my_dog.name}.")                    #when calling instances no roun
print(f"My dog is {my_dog.age} years old.")
my_dog.sit()                                                 #called a method 'sit' from cla

print(f"\nYour dog's name is {your_dog.name}.")
print(f"Your dog's age is {your_dog.age} years old.")
your_dog.roll_over()                                          #called a method 'roll_over' f
```

```
My dog's name is Rambo.
My dog is 6 years old.
Rambo is now sitting.

Your dog's name is Tyson.
Your dog's age is 0.5 years old.
Tyson rolled over!
```

```
my_dog.sit()
```

```
Rambo is now sitting.
```

Example below

creating a Restraunt class

```
class Restaurant:

    def __init__ (self, restaurant_name, cuisine_type):
        self.restaurant_name = restaurant_name
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        print(f"\nThe restaurant's name is {self.restaurant_name}.")   #add self to access
        print(f"The cuisine type offered is {self.cuisine_type}.")

    def open_restaurant(self):
        print(f"The {self.restaurant_name} is now open!")
```

Making an instance from a class

```
restaurant = Restaurant('Desi Dhabha', 'Vegetarian')
```

Printing individual attributes

```
print(f"Restaurant Name: {restaurant.restaurant_name}")
print(f"Cuisine Type: {restaurant.cuisine_type}")
```

```
Restaurant Name: Desi Dhabha
Cuisine Type: Vegetarian
```

Calling methods

```
restaurant.describe_restaurant()
restaurant.open_restaurant()
```

```
The restaurant's name is Desi Dhabha.
The cuisine type offered is Vegetarian.
The Desi Dhabha is now open!
```

Working with classes and instances

```
class Car:

    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
    def get_descriptive_name (self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

my_new_car = Car('audi', 'a4', 2023)
print(my_new_car.get_descriptive_name())
```

```
2023 Audi A4
```

```
# adding an attribute that changes over time
class Car:

    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name (self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

my_new_car = Car('audi', 'a4', 2023)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

```
2023 Audi A4
This car has 0 miles on it.
```

The simplest way to modify the value of an attribute is to access the attribute directly through an instance.

Here we set the odometer reading to 23 directly

```
my_new_car.odometer_reading = 45
my_new_car.read_odometer()
```

```
This car has 45 miles on it.
```

Modifying an Attribute's Value Through a Method

```
# rest of the code same
class Car:

    def __init__(self, make, model, year, odometer_reading):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = odometer_reading
    def get_descriptive_name (self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
    def update_odometer(self, odometer_reading):
        mileage = self.odometer_reading
        return mileage

my_new_car.update_odometer(52)
my_new_car.read_odometer()
```

```
AttributeError: 'Car' object has no attribute 'update_odometer'
```

```
class Car:

    def __init__(self, make, model, year, odometer_reading):
```

```
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = odometer_reading

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def update_odometer(self, odometer_reading):
        self.odometer_reading = odometer_reading  # Update the odometer reading
        print(f"Odometer reading set to {self.odometer_reading}")

# Create an instance of the Car class
my_new_car = Car("honda", "Accord", 2023, 150)

# Update the odometer reading
my_new_car.update_odometer(52)

# Get the descriptive name of the car
car_name = my_new_car.get_descriptive_name()
print(f"Car name: {car_name}")

# Get the current odometer reading
print(f"Odometer reading: {my_new_car.odometer_reading}")
```

```
Odometer reading set to 52
Car name: 2023 Honda Accord
Odometer reading: 52
```

Inheritance

```
class Car:
    def __init__(self,make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name (self):
```

```
        long_name = f"{self.year}{self.make}{self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles

class ElectricCar(Car): #specific to electric vehicles

    def __init__(self, make, model, year):
        super().__init__(make, model, year)      #initializing attributes of parent class

my_tesla = ElectricCar(' tesla', ' model s', 2019)
print(my_tesla.get_descriptive_name())
```

```
2019 Tesla Model S
```

Definig attributes and methods for child class

```
class Car:
    def __init__(self,make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name (self):
        long_name = f"{self.year}{self.make}{self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")
```

```
    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles

class ElectricCar(Car): #specific to electric vehicles

    def __init__(self, make, model, year):
        super().__init__(make, model, year)      #initializing attributes of parent class
        self.battery = Battery()

class Battery:
    def __init__(self, battery_size= 75):
        self.battery_size = battery_size       #initializaing attributes

    def describe_battery(self):
        print(f"The car has a {self.battery_size}-kWh battery.")


my_tesla = ElectricCar(' tesla', ' model s', 2019)
print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
```

```
2019 Tesla Model S
The car has a 75-kWh battery.
```

Extending battery class

```
class Car:
    def __init__(self,make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0
```

```
    def get_descriptive_name (self):
        long_name = f"{self.year}{self.make}{self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles

class ElectricCar(Car): #specific to electric vehicles

    def __init__(self, make, model, year):
        super().__init__(make, model, year)      #initializing attributes of parent class
        self.battery = Battery()

class Battery:
    def __init__(self, battery_size= 100):
        self.battery_size = battery_size       #initializaing attributes

    def describe_battery(self):
        print(f"The car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        print(f"This car can go about {range} miles on a full charge.")


my_tesla = ElectricCar(' tesla', ' model s', 2019)
print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
```

```
my_tesla.battery.get_range()
```

```
2019 Tesla Model S
The car has a 100-kWh battery.
This car can go about 315 miles on a full charge.
```

Importing classes

```
from car import Car

my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())

my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

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

```
from car import Car
from electric_car import ElectricCar
```

Importing from python's standard library

before we learned, how to create a class and import stuff from there

```
from random import randint
randint(1,6)
```

```
from random import choice
players = ['charles', 'martina', 'michael', 'florence', 'eli']
first_up = choice(players)
first_up
```

```
first_up
```

```
from random import randint
randint(1,7)
```

Rolling a die 10 times and storing it in class Die

```
import random

class Die:
    def __init__(self, sides = 6):
        self.sides = sides

    def roll_die(self):
        result = random.randint(1, self.sides)
        return result
# create a 6-sided die
six_sided_die = Die()

# Roll the die 10 times
for _ in range(10):
    roll_result = six_sided_die.roll_die()
    print(f"The die rolled: {roll_result}")
```

Lottery analysis

print four numbers or letters from a list of 10 numbers and 5 letters.

-print if the person has won the lottery or not.

```
import random

# Create a list with numbers and letters
ticket_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'A', 'B', 'C', 'D', 'E']

# Randomly select four elements from the list
winning_combination = random.sample(ticket_list, 4)

# Print the winning combination
```

```
print("Winning Combination:", winning_combination)

# Check if the winning combination matches
if all(item in winning_combination for item in ['A', 'B', 'C', 'D']):
    print("Congratulations! You've won a prize!")
else:
    print("Sorry, your ticket did not match the winning combination.")
```

```
Winning Combination: ['E', 10, 9, 1]
Sorry, your ticket did not match the winning combination.
```