AI Boot Camp

Python Classes and OOP

Module 3 Day 3

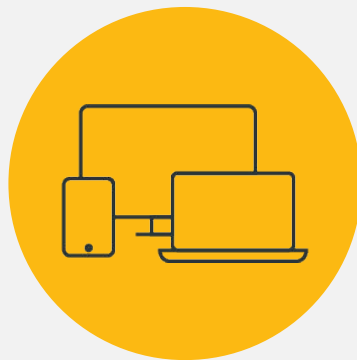# Class Objectives

By the end of class, you will be able to:

1. Understand the fundamental concepts of object-oriented programming.

2. Understand the differences between class objects, attributes, instances, and methods.

3. Write code that is reusable, modular, and readable using Python classes.

4. Understand and apply inheritance in OOP.

Instructor **Demonstration**

Introduction to Object Oriented Programming

Object-Oriented Programming

Object-oriented programming (OOP) is centered on creating objects. You can think of an object as a wrapper for a group of related code properties and behaviors. Objects allow us to model things from the real-world, as well as any relationships or interactions they have with other objects.

An object is a software entity that contains both data and procedures.

The data contained in an object is known as the object's data attributes. You can think of these as the object's properties. For example, if you were creating an object to represent a car, the object's data attributes might be its make and model.

The procedures that an object performs are known as methods. Methods are functions that are defined inside of an object and perform operations on the object's data attributes. You can think of methods as being responsible for the object's behavior.

Python Classes

Classes act as a "blueprint" that specifies the properties and behaviors of an object or instance.

In the car example, the class defines how the object will look and behave.

Class

Blueprint of a car

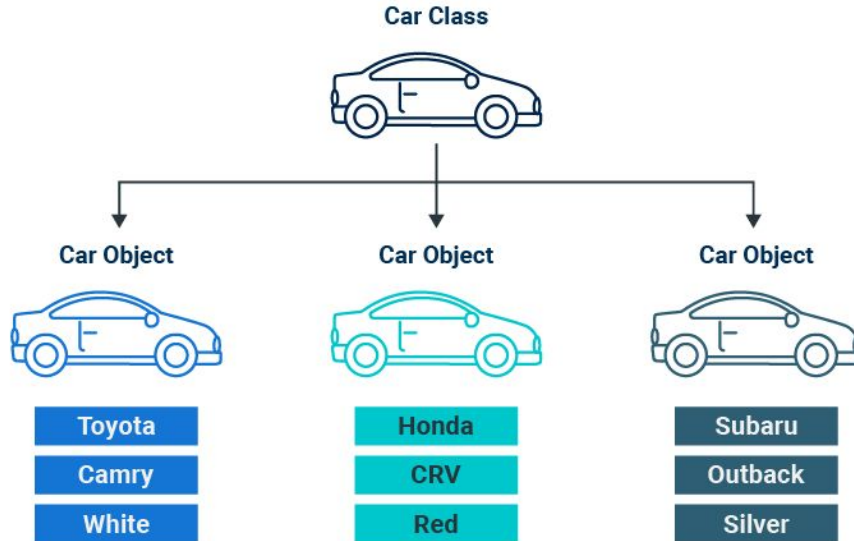The object is the completed product: the car itself.

Object

Car

Python Classes

We create functions in the classes that define the object's properties and behavior.

For example, a `Car` class can have attributes such as the make, model, and color.

Defining Classes

We define a class using the keyword `class`, followed by the name of the class and a colon.

Class names in Python are written in `UpperCamelCase`, with the first letter of every word capitalized.

All the code in the class is indented below the class name or definition, just like when adding code to a function.

```
class Car:
    < code for the class goes here >
```

Creating a Class

Classes require the following:

- A special method called the constructor, `__init__()`. This is responsible for initializing the newly created car object.

- The constructor takes parameters just like a function.
  - For the `Car` class we have entered three parameters to the constructor: `self`, `make`, and `model`.

- For every parameter, except the `self` parameter, the class needs to have instance variables that store the attributes of the car.

```python
# Define the Car class
class Car:
    """ Creating a Car class with attributes and instances"""
    def __init__(self, make, model):
        self.make = make
        self.model = model
```

The `__init__()` Constructor

- The `__init__()` constructor or method is responsible for assigning the **initial** state of those attributes to any car object that you create.

- If you want a user to be able to specify the make and the model of a car object instance, you set up your `__init__()` method to receive these two attributes.

- **The first parameter in the `__init__()` method is always going to be `self`.**
  - It is a reference to the object itself.
  - The `self` parameter in the `__init__()` method is a convention in Python that references the instance of the object being created or operated upon, in this case it is the car object and its attributes.

- **The second parameter is the `make` attribute.**

- **The third parameter is the `model` attribute**

Class Instance Variables

- Instance variables store the information passed on to the parameters, they represent the attributes or properties of the Car object.
    - `self.make = make`
    - `self.model = model`

- The **dot notation**, `self.made` and `self.model`, are used to access and modify the attributes of an object being created.

- By using `self` in our dot notation, we are referring to the current object being defined or operated on.

Class Instances

When you actually create an object based on the blueprint you are creating an **instance** of that class, The instance will contain all the attributes and behaviors that you laid out in the "blueprint".

To add data to the `make` and `model` parameters of the `Car` object, we pass in data to the `my_car` instance as follows:

If we want to print out the `make` and `model` of the `car` object we reference the attributes of the `car` instance as follows:

```python
# Create an instance of the Car class
# using the Car() constructor.
my_car = Car("Toyota", "Camry")


# Print the details of the car
print(f"Make: {my_car.make}")
print(f"Model: {my_car.model}")

The output is as follows:
Make: Toyota
Model: Camry
```

Activity:
Modify the Car Class

In this activity, you'll practice adding parameters to a class, creating class methods, and creating an instance of the class that passes parameters to the class based on the inputs of a user.
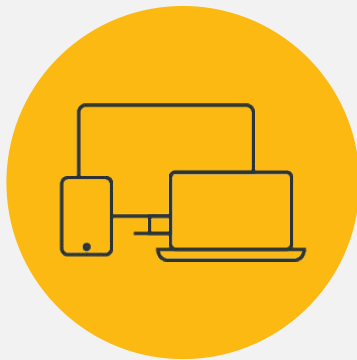
Suggested Time:
15 Minutes

Time's up!
Let's review

Questions?

Instructor **Demonstration**

Adding Methods to Classes

Instance Methods

Here are some benefits of using instance methods. They allow you to:

01 Access and manipulate the attributes of an instance

02 Reuse the same code across multiple instances of a class

03 Encapsulate related actions together

04 Modify or extend the behavior of objects by adding or modifying new behaviors to classes.

We are about to create a method that will retrieve the make of the car instance. **Why would it be useful to have a method like this?**

Creating a Method for a Class

01 First, we define the `Car` class and initialize the class with the `make` parameter.

02 Next, we add an instance method, which is a function that is defined within a class. This method will get the make of the car when it is called.

```python
# Define the Car class
class Car:
    """Creating a Car class with methods"""
    def __init__(self, make):
        self.make = make



    # Create a method to get the make of the car
    def get_make(self):
        """ Gets the make of the car"""
        return self.make
```

Creating a Method for a Class

01 To access or get the attribute from the instance `get_make()`, we create an instance of the `Car` class and pass in the argument for the make.

```python
# Create an instance of the Car class.
my_car = Car("Toyota")
```

02 Next, we use the name of the instance plus the `get_make()` method to get the make of the car.

```python
# Get the current make
current_make = my_car.get_make()
print(f"Current Make: {current_make}")
```

How does `my_car.get_make()` get the make of the car?

Accessing Attributes of an Instance

01

When the `get_make()` method is called the data returned is the `self.make` attribute, which was initialized when we created the `Car` class.

02

The data returned from the `self.make` attribute is the make of the car.

Manipulating the Attributes of an Instance

01

We can change change the attribute of an instance by creating new methods.

02

Remember the class serves as a blueprint for creating car objects. We can create a new methods to change the car's object.

Manipulating the Attributes of an Instance

We can create a new method, **set_make()**, to change the make of the car when we call this method.
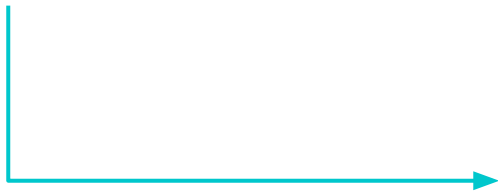
```python
# Define the Car class
class Car:
    """Creating a Car class with methods"""
    def __init__(self, make):
        self.make = make


        # Create a method to get the make of the car
    def get_make(self):
        """ Gets the make of the car"""
        return self.make


    # Create a method to change the make of the car
    def set_make(self, new_make):
        """ Sets the make of the car"""
        self.make = new_make
```

Activity:

Car Class Methods

In this activity, you'll practice creating class methods to modify an instance of the `Car` class. You'll prompt the user to input the three new parameters of the car. Next, you'll pass each parameter to the new methods you created. Finally, you'll print out the updated information about the car. This activity allows you to demonstrate how class methods can be used to interact with and modify an instance of the `Car` class.

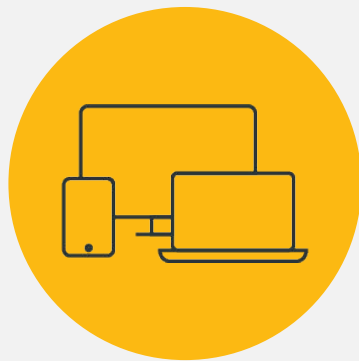**Suggested Time:**

15 Minutes

Time's up!
Let's review

Questions?

Break

15 mins

Instructor **Demonstration**

Creating Modules from Classes

Storing Classes as Modules

- By storing the classes in modules you are able to practice better programming by promoting better code organization, encapsulation, and reusability

- When importing a class or a function from another Python file we don't need to use the file extension, `.py`.

The `Car.py` file.

```python
# Define the Car class
class Car:
    """Creating a Car class with methods"""
    def __init__(self, make, model, body, engine,
year, color):
        self.make = make
        self.model = model

    # Create a method to get the make of the car
    def get_make(self):
        """ Returns the make of the car"""
        return self.make

    # Create a method to get the model of the car
    def get_model(self):
        """ Returns the model of the car"""
        return self.model
```

The `car_data.py` file.

```python
# Import the Car class from the Car.py file.
from Car import Car

# Create an instance of the Car class.
car = Car("Subaru", "CrossTrek Limited")

# Get the current make using the getter methods.
print('Here are the details of the car.')
print(f"Make: {car.get_make()}")
print(f"Model: {car.get_model()}")
```

Activity:

Create and Import a Bank Account Module

In this activity, you'll import a class from a Python file and use its methods to interact with user input in a separate Python file.
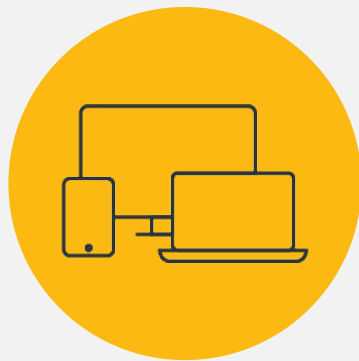
Suggested Time:

20 Minutes

Time's up!
Let's review

Questions?

Instructor **Demonstration**

Inheritance

What Is Class Inheritance?

Class inheritance is a concept in OOP where a new class can inherit the properties and behaviors of an existing, or main, class.

01

The new class is called either a **subclass** or **child class**, and the existing class is called either a **base class**, **superclass**, or the **parent class**.

02

The subclass it will **inherit** all the attributes and methods from the base class. The subclass, however, may have other methods defined inside it that give it special functionality.

Storing Classes as Modules

The `Car` class

```python
# Define the main or parent class
class Car:
    """Creating a Car class with methods"""
    def __init__(self, make, model, body, engine,
year, color):
        self.make = make
        self.model = model

    # Create a method to get the make of the car
    def get_make(self):
        """ Returns the make of the car"""
        return self.make

    # Create a method to get the model of the car
    def get_model(self):
        """ Returns the model of the car"""
        return self.model
```

The `CarExtras` class

```python
# Define the subclass or child class
class CarExtras(Car):
    """Creating a CarExtras class with methods"""
    def __init__(self, make, model, anti_theft,
ext_warranty):
        # Call the parent class's __init__ method and
        pass the required parameters.
        Car.__init__(self, make, model)
        # Initialize the attributes for the CarExtras
class.
        self.anti_theft = anti_theft
        self.ext_warranty = ext_warranty

    def get_anti_theft(self):
        """ Sets the antitheft for the car"""
        return self.anti_theft

    def get_ext_warranty(self):
        """ Sets the extended warranty for the car"""
        return self.ext_warranty
```

Why did we not include **instance methods** to retrieve the make and model of the car in the subclass?
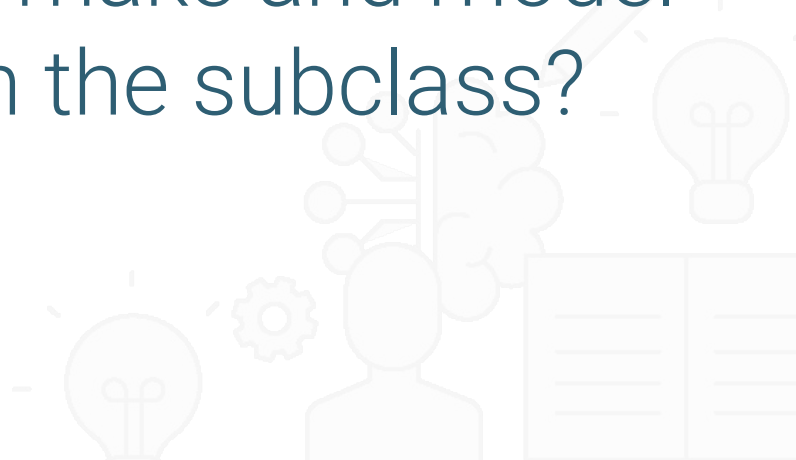
Activity:
Account Inheritance

In this activity, you'll simulate a basic banking system by creating a `SavingsAccount` and a `CD` class. The `SavingsAccount` class will be our parent class, with attributes for balance and interest rate. The `CD` class will be the child class inheriting from the `SavingsAccount` class, with an additional attribute for the length of time of the CD.
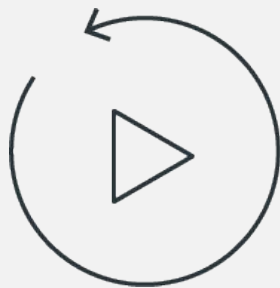
Suggested Time:
15 Minutes

Time's up!
Let's review

Questions?

Let's **recap**

Review the Class Objectives

In this lesson you learned how to:

1   Understand the fundamental concepts of object-oriented programming.

2   Understand the differences between class objects, attributes, instances, and methods.

3   Write code that is reusable, modular, and readable using Python classes.

4   Understand and apply inheritance in OOP.

Challenge

You'll be creating a customer banking system that allows users to calculate and track interest earned on savings and CD accounts. By running this application, users will be able to interactively enter their savings and CD account information, see the interest earned, and view the updated balances after a specified number of months.

Questions?

The End