Python Chapter 7 OOP Activities

Contents

Time required: 90 minutes

Videos from YouTube

- Learn Python Object Oriented Programming! (P.O.O.P)

DRY

Don't Repeat Yourself

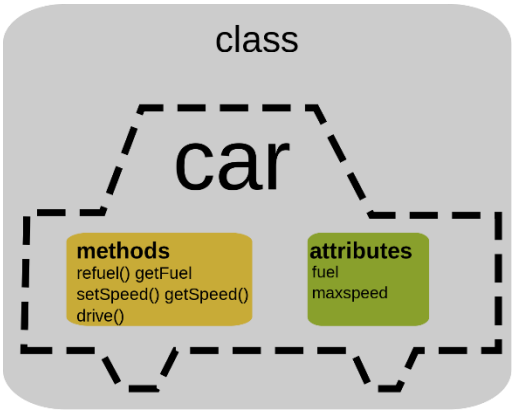
Online Tutorials

Go through the following tutorials.

- LearnPython.org Classes and Objects
- Python Classes and Objects
- Python Inheritance

Object Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which are instances of classes.



Objects and Classes:

- **Objects:** Think of them as real-world entities. For example, a car, a person, or a book.
- **Classes:** These are like blueprints or templates for creating objects. They define the properties (attributes) and behaviors (methods) that the objects will have.

Attributes and Methods:

- **Attributes:** These are the characteristics or properties of an object. For a car, attributes could be its color, model, and year.
- **Methods:** These are the actions or behaviors that an object can perform. For a car, methods could include starting the engine or honking the horn.

Encapsulation:

- This is like putting everything related to an object in one place. It keeps the data (attributes) and the methods that operate on the data together in a class. This makes the code more organized and easier to manage.

Tutorial 1: A Car with Class

Objects:

- Objects represent real-world entities.
- Example: Defining a Car class to model different cars.

Classes:

- Classes act as blueprints for objects.

Create a Python file named **car.py**

```python
# Define a class named 'Car'
class Car:
    # Constructor method to initialize object attributes
    def __init__(self, model, year):
        # Set 'model' attribute to the provided model parameter
        self.model = model
        # Set 'year' attribute to the provided year parameter
        self.year = year


# Create an instance of the 'Car' class with model "Toyota" and year 2022
my_car = Car("Toyota", 2022)

# Print a formatted string using the 'model' and 'year'
# attributes of the 'my_car' instance
print(f"My car is a {my_car.model} from {my_car.year}.")
```

- **class Car: -** Defines a blueprint for creating objects of type Car.
- **def __init__(self, model, year): -** Constructor method initializes the object with model and year attributes.
- **self.model = model** and **self.year = year -** Sets the attributes of the object to the provided values.
- **my_car = Car("Toyota", 2022) -** Creates an instance of the Car class named 'my_car' with model "Toyota" and year 2022.
- **print(f"My car is a {my_car.model} from {my_car.year}.") -** Prints a formatted string using the 'model' and 'year' attributes of the 'my_car' instance. Outputs a descriptive sentence about the car.

Example run:

```
My car is a Toyota from 2022.
```

Tutorial 2: A Person with Class

Attributes:

- Attributes store data within a class.
- Example: Extending the Person class with additional attributes.

Methods:

- Methods perform actions within a class.
- Adding a greet method to the Person class.

Create a Python program file named **person.py**

```python
# Define a class named Person
class Person:
    # Constructor method (called when creating an object)
    def __init__(self, name, age):
        # Initialize object attributes (name and age)
        self.name = name
        self.age = age

    # Method to greet and provide information
    def greet(self):
        return f"Hello, my name is {self.name} and I'm {self.age} years old."

# Create an instance (object) of the Person class with name "Alice" and age 25
individual = Person("Alice", 25)

# Call the greet method of the individual object and print the result
print(individual.greet())
```

- **class Person:** - Defines a blueprint for creating objects of type Person.
- **def __init__(self, name, age): -** Constructor method initializes the object with a name and age.
- **self.name = name** and **self.age = age:** - Sets the attributes of the object to the provided values.
- **def greet(self): -** Defines a method within the class to generate a greeting.
- **return f"Hello, my name is {self.name} and I'm {self.age} years old."** - Returns a formatted greeting string using the object's attributes.

- **individual = Person("Alice", 25)** - Creates an instance of the Person class named "Alice" with an age of 25.
- **print(individual.greet()) -** Calls the greet method of the individual object and prints the result. Outputs the personalized greeting.

Example run:

```
Hello, my name is Alice and I'm 25 years old.
```

Getters and Setters

Getters and Setters allow the program to access the data attributes on the class. Getters and setters allow control over the values. You may validate the given value in the setter before setting the value.

- **getter:** returns the value of the attribute.
- **setter**: takes a parameter and assigns it to the attribute.

Tutorial 3: Encapsulation by Book

Encapsulation organizes code by bundling data and methods. This prevents direct access to the attributes.

Example: Implementing encapsulation in a **Book** class.

Access Modifiers:

- Public and private access modifiers.
- Example: Creating private attributes in a **Book** class.

Create a Python program named **book.py**

```python
# Define a blueprint for creating Book objects
class Book:
    # Constructor method initializes object with title and author parameters
    def __init__(self, title, author):
        # Each attribute use a single underscore for a private attribute
        # Set '_title' attribute to the provided title parameter
        self._title = title
        # Set '_author' attribute to the provided author parameter
        self._author = author

    # Method to get the title attribute
    def get_title(self):
        return self._title

    # Method to get the author attribute
    def get_author(self):
        return self._author

    # Method to set the title attribute
    def set_title(self, title):
        self._title = title

    # Method to set the author attribute
    def set_author(self, author):
        self._author = author

    # Method to return a formatted string of the book's title and author
    def __str__(self):
        return f"{self._title} by {self._author}"


# Create an instance (object) of the Book class with
# title "To Kill a Mockingbird" and author "Harper Lee"
book = Book("To Kill a Mockingbird", "Harper Lee")

# Print a formatted string using the 'get_title'
# and 'get_author' methods of 'book'
print(f"{book.get_title()} by {book.get_author()}")

book.set_title("The Great Gatsby")
book.set_author("F. Scott Fitzgerald")

# Print a formatted string using the 'get_title'
# and 'get_author' methods of 'book'
print(f"{book.get_title()} by {book.get_author()}")

# Print the 'book' object using the '__str__' method
print(book)
```

- **class Book: -** Defines a blueprint for creating objects of type Book.
- **def __init__(self, title, author): -** Constructor method initializes the object with title and author attributes.
- **self._title = title** and **self._author = author -** Sets the attributes of the object to the provided values, using a single underscore _ to indicate that these attributes are intended to be private.
- **def get_title(self): -** Method to retrieve the title attribute of the book.
- **def get_author(self): -** Method to retrieve the author attribute of the book.
- **def set_title(self): -** Method to set the title attribute of the book.
- **def set_author(self): -** Method to set the author attribute of the book.
- **Def __str__(self):** - Method to return a formatted string representation of the object.
- **book = Book("To Kill a Mockingbird", "Harper Lee") -** Creates an instance of the Book class with title "To Kill a Mockingbird" and author "Harper Lee".
- **print(f"{book.get_title()} by {book.get_author()}") -** Prints a formatted string using the 'get_title' and 'get_author' methods of 'book'. Outputs the title and author of the book. Note the addition of parentheses to correctly call the methods.

Example run:

```
To Kill a Mockingbird by Harper Lee
The Great Gatsby by F. Scott Fitzgerald
```

Assignment 1: Online Video Game Store

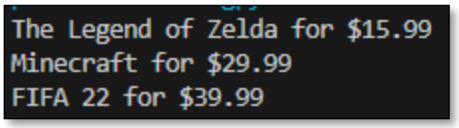
Objective: Build a basic online video game store using Python. Create three game objects within the class. This is a proof of concept.

Requirements:

- Create a Store class to represent the online store.
- Attributes: _title, _price
- Method: __str__ returns a string representation of the object.
- Create getters and setters for each attribute.

- Create three sample game titles.
- Display the available games using the __str__ method.

Example run:



Assignment Submission

1. Attach the pseudocode.
2. Attach the program files.
3. Attach screenshots showing the successful operation of the program.
4. Submit in Blackboard.