Classes and Objects in Python

Connect with me: Youtube | LinkedIn | WhatsApp Channel | Web | Facebook | Twitter

- Download PDF
- To access the updated handouts, please click on the following link: https://yasirbhutta.github.io/ms-excel/docs/classes.html

Classes ans Objects

In Python, you create a class using the class keyword. A class is a blueprint for creating objects (instances).

Objects are instances of a class that have attributes (data) and methods (functions).

What are instance attributes?:

- Unique to each instance (object) of a class.
- Store data specific to that object.
- Defined within the **init**() constructor method, using the self parameter.

Python Class Example: Video: How to Create a Class

Example #: How to create a Class

```python
# Class Definition
class Student:
    # Constructor
    def __init__(self, name, age, grade): # self refers to the current object being created.
        self.name = name
        self.age = age
        self.grade = grade
    # Method
    def info(self):
        print(f"Name = {self.name} Age = {self.age} Grade = {self.grade}")

# Object Creation

student1 = Student("Hamza", 8, 3)
student2 = Student("Muhammad", 15, 10)

# Accessing Attributes and Methods

print(student1.name)
student1.info()
student2.info()
```

Video: Python Classes - What is Class Constructor

Key Points:

- Classes act as blueprints for creating objects.
- Objects are instances of classes, each with their own attributes (data) and methods (behaviors).
- The `__init__()` method initializes objects when they're created.
- Methods are functions defined within a class that operate on the object's data.
- `self` is used to access the object's attributes and methods within its methods.

Example:

class Dog: def **init**(self, name, breed): self.name = name self.breed = breed

```
def bark(self):
    print(f"{self.name} says woof!")
```

Creating an object (instance) of the Dog class

my_dog = Dog("Buddy", "Golden Retriever") print(my_dog.name) # Accessing an attribute my_dog.bark() # Calling a method

The **init** method is the constructor. It's called when you create a new object and initializes the object's attributes.

Example #:

```
class Student:
    """Represents a student with their name, age, and grade."""

    def __init__(self, name, age, grade):
        """Initializes a Student object with the given attributes."""
        self.name = name
        self.age = age
        self.grade = grade

    def get_name(self):
        """Returns the student's name."""
        return self.name

    def get_age(self):
        """Returns the student's age."""
        return self.age

    def get_grade(self):
        """Returns the student's grade."""
        return self.grade

    def set_grade(self, new_grade):
        """Updates the student's grade."""
        self.grade = new_grade

    def introduce(self):
        """Prints a self-introduction message."""
        print("Hello, my name is", self.name, "and I'm in grade", self.grade)

# Example usage
student1 = Student("Hamza", 8, 3)
student2 = Student("Muhammad", 16, 10)

student1.introduce()  # Output: Hello, my name is Alice and I'm in grade 9
print(student2.get_name())  # Output: Bob
student2.set_grade(11)
print(student2.get_grade())  # Output: 11
```

Class and Instance Attributes in Python:

- In Python, class attributes are the variables defined directly in the class that are shared by all objects of the class.
- Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor using the `self` parameter.

The following table lists the difference between class attribute and instance attribute:

Class Attribute	Instance Attribute
Defined directly inside a class.	Defined inside a constructor using the `self` parameter.
Shared across all objects.	Specific to object.
Accessed using class name as well as using object with dot notation, e.g. `classname.class_attribute` or `object.class_attribute`.	Accessed using object dot notation e.g. `object.instance_attribute`.
Changing value by using `classname.class_attribute = value` will be reflected to all the objects.	Changing value of instance attribute will not be reflected to other objects.

Python Class Example: Video: How to Create a Class and Instance Attributes in Python

Anoter Example Example:

class Car: def **init**(self, make, model, year): self.make = make self.model = model self.year = year

```
def describe(self):
    print(f"{self.year} {self.make} {self.model}")
```

my_car = Car("Toyota", "Corolla", 2020) my_car.describe() # Output: 2020 Toyota Corolla

1. Inheritance

Inheritance allows you to create a new class based on an existing class. The new class (child class) inherits attributes and methods from the parent class.

The child class can also add its own attributes and methods or override methods from the parent class.

Example:

class Animal: def **init**(self, name): self.name = name

```
def speak(self):
    print(f"{self.name} makes a sound")
```

class Dog(Animal): def speak(self): print(f"{self.name} barks")

my_dog = Dog("Buddy") my_dog.speak() # Output: Buddy barks

4. Polymorphism

Polymorphism allows different classes to have methods with the same name but different behavior.

In the example above, both Animal and Dog have a speak() method, but they behave differently based on the class.

5. Encapsulation

Encapsulation is the concept of hiding the internal details of a class and providing methods to interact with the data. This is often achieved using private and public attributes.

Attributes that start with an underscore (e.g., _age) are conventionally considered private.

Example:

class Person: def **init**(self, name, age): self.name = name self._age = age # _age is considered private

```
def get_age(self):
    return self._age

def set_age(self, age):
    if age > 0:
        self._age = age
```

person = Person("Alice", 30) person.set_age(35) print(person.get_age()) # Output: 35

6. Abstraction

Abstraction involves hiding the complex implementation details and exposing only the necessary functionality. This can be achieved using abstract classes (via the abc module), but we won't go too deep into that for now.

Would you like to practice creating classes, inheritance, and encapsulation, or move on to something else, like exceptions and error handling?

Key Terms

True/False (Mark T for True and F for False)

Multiple Choice (Select the best answer)

> What keyword is used to define a class in Python?

1. ☐ object
2. ☐ class
3. ☐ define
4. ☐ declare

Watch this video for the answer: https://youtu.be/zVYzk_gnTY4

What is the correct way to create a class in Python? a) class MyClass: b) create MyClass: c) define MyClass: d) new MyClass:

Answer: a) class MyClass:

> What is the correct way to create an object instance of a class?

1. ☐ Calling the class definition directly
2. ☐ Assigning the class name to a variable
3. ☐ Using the new keyword
4. ☐ Calling the class name with parentheses

What will be the output?

```python
class Dog:
  name = "Unknown"

  def bark(self):
    print("Woof!")

dog1 = Dog()
dog1.name = "Buddy"
dog2 = Dog()

print(dog1.name, dog2.name)
```

1. ☐ Buddy Unknown
2. ☐ Unknown Unknown
3. ☐ Buddy Buddy
4. ☐ It depends on the dog breed

> What is the purpose of the self parameter in a method?

1. ☐ To store the method name
2. ☐ To refer to the current object instance
3. ☐ To pass data to other methods
4. ☐ All of the above

> What is the primary purpose of a class constructor?

1. ☐ To define the name of the class
2. ☐ To initialize the object's data members
3. ☐ To allocate memory for the object
4. ☐ All of the above

> What is the purpose of the **init** method in a Python class?

1. ☐ To define static properties
2. ☐ To store the object's type
3. ☐ To initialize the object's attributes
4. ☐ To compare objects for equality

Fill in the Blanks

Exercises

Review Questions

References and Bibliography

- Classes - Python documentation
- Python Attributes – Class and Instance Attribute Examples - freecodecamp.org