# **Module 11** - Python Classes

Contents

[Module 11 - Python Classes 1](#_Toc216013411)

[Different Code Structures 2](#_Toc216013412)

[Object-Oriented Programming 2](#_Toc216013413)

[Classes & Objects 2](#_Toc216013414)

[Encapsulation 4](#_Toc216013415)

[Abstraction 5](#_Toc216013416)

[Inheritance 5](#_Toc216013417)

[Polymorphism 7](#_Toc216013418)

[\_\_init\_\_() Constructor 7](#_Toc216013419)

[Self Keyword 7](#_Toc216013420)

[Class and Instance Attributes 8](#_Toc216013421)

[Method Overriding 10](#_Toc216013422)

[Setters & Getters 12](#_Toc216013423)

[Examples 14](#_Toc216013424)

[Reference and Additional Materials: 19](#_Toc216013425)

## 

## Different Code Structures

There are many types of code structures. The main type we will be dealing with in this course of Object-Oriented Programming but let's review different code structures.

[Sequential Code](https://medium.com/@gugunmfauzi96/python-programming-concepts-sequential-conditional-branching-and-looping-structures-9179565c7f2d#:~:text=A.%20Sequential%20Execution%3A%20Step%2Dby%2DStep%20Instructions): A type of programming structure where instructions are executed in a linear order, one after the other. It focuses on executing one task at a time.

[Conditional Code](https://www.w3schools.com/python/python_conditions.asp): This type of coding makes decisions based on whether a condition is true or false using [if statements](https://www.w3schools.com/python/gloss_python_if_statement.asp). It allows programs to choose different actions depending on the situation.

[Repetitive Code](https://openclassrooms.com/en/courses/6902811-learn-python-basics/7090826-easily-repeat-tasks-using-loops#:~:text=or%20log%20in-,When%20to%20Use%20Loops,-In%20programming%2C%20there): Code that reapers a set of instructions until a condition is met, using [loops](https://www.w3schools.com/python/python_for_loops.asp). Loops help reduce repetition and make code more efficient.

[Store and reuse](https://medium.com/@augeluiza/functions-reusable-blocks-of-code-eca0080037cd) (procedural programming): [Functions](https://www.py4e.com/html3/04-functions) group code into reusable blocks that can be called whenever they’re needed. They make programs easier to read, manage, and maintain.

[Object-Oriented Code](https://medium.com/data-bistrot/understanding-object-oriented-programming-in-python-47b5961d849e): Code that organizes data and behavior into objects based on real-world concepts. It promotes reusable, modular, and scalable program structures.

## Object-Oriented Programming

Python is an object-oriented programming language, which means it’s built around the idea of objects. Objects are created from something called a class, which is like a blueprint or custom data type. All the elements of python programming like variables like int, str, float, list, dict, tuples are objects. [Object-Oriented programming](https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/) (OOP) stands on [four pillars](https://youtu.be/jNocnk-lByw?si=16XMBIgFZIdYYqYo): Encapsulation, Abstraction, Inheritance and Polymorphism. These principles help organize code so that it's easier to build, maintain, and scale. With OOP, you can model real-world things in your code, like students, books, or accounts, each with their own data and actions, making your programs more logical and easier to understand.

## Classes & Objects

Classes and objects go hand-in-hand and make your code easier to maintain and build upon as your scripts start to grow. [Classes](https://docs.python.org/3/tutorial/classes.html#classes) serve as the blueprints for creating objects. A class itself doesn’t do anything on its own, it simply defines what an object should look like and what it can do. The real work happens when you create objects, also called instances, based on that class. You can create many different objects from the same class. A class acts like a capsule that holds variables, called attributes or properties, and functions, called methods, all packaged together. In fact, in Python, even built-in types like integers, floats, strings, and functions are actually created from classes behind the scenes.

[Objects](https://docs.python.org/3/tutorial/classes.html#a-word-about-names-and-objects:~:text=9.1.%20A%20Word%20About%20Names%20and%20Objects%C2%B6), as mentioned, are a specific instance of a class. If class is a blueprint for a house, an object is the house physically built from that blueprint. We've seen many objects already by using strings, lists, dictionaries, and tuples. If a student record is a class, we can create objects like Alice’s record and Bob’s record based on the student record template. The state of a student record includes information like the student's name, ID number, and GPA. The behaviors of a student record are actions it can perform, like updating a GPA or adding a new course. When an object performs a behavior, it can change its state, like updating the GPA would change the student's record.

Classes are created by using the [class keyword](https://www.w3schools.com/python/ref_keyword_class.asp). In our example, we are going to build and manage student records. In this scenario, the class would be “Student” and the object would be individual students.

class Student:

name = “ ”

gpa = 0.0

When naming a class, like we named “Student” above, there are a few simple [syntax rules](https://medium.com/@krisbredemeier/class-and-object-in-python-2d5980eeb92c#:~:text=In%20proper%20python%20code%2C%20your,normal%20variable%20or%20function%20name.). Your class name should be capitalized and as short and descriptive as possible. Inside the class, you would also define your method. A method is a function that is defined inside a class and is used to describe the behavior of an object. It allows objects to perform actions or interact with their data using the structure defined in the class. It would look similar to this below:

class Student:

name = “ ”

gpa = 0.0

def display\_info(self):

print("Displaying student info")

After defining a class, you can assign attributes.

student1.name = "Ravi"

student1.gpa = 3.7

The object will represent a specific implementation of the class and hold its own data. You can only [create an object](https://www.w3schools.com/python/trypython.asp?filename=demo_ref_keyword_class2) after you define a class as it is an instance of a class.

Next, create an object:

#create object from class

student1 = Student()

# access the class attribute

print(student1.name)

Each object created from the class can store different data, but will have access to the same methods and structure as it was defined in the class that accompanies it. This will allow you to work with multiple objects in your script, each with its own rules defined in the class but with their own unique information. When you put it all together, it would look like this:

#Class definition

class Student:

name = "" #attribute

gpa = 0.0 #attribute

def display\_info(self):

print("Displaying student info")

# Object instantiation

student1 = Student()

# Method calling

student1.display\_info()

# Attribute initialization

student1.name = "Alice"

student1.gpa = 3.9

print(student1.name)

print(student1.gpa)

## Encapsulation

[Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-python/) is the idea of wrapping variables and methods (functions) together into a single unit called a class. It’s like putting all the data and actions related to something in one place so you can manage it more easily. When using encapsulation, the internal details are hidden. You only interact with the class through its functions without needing to know how everything works behind the scenes. This is similar to how we've been using built-in Python methods. We provide the input and get the output without seeing the internal code. [Encapsulation](https://codesarray.com/view/Understanding-Encapsulation-in-Python) helps protect the data and keeps things organized, making it easier to build secure and reliable programs.

In Python, two types of encapsulation are possible: composition and dynamic extension. Composition means describing a model in terms of simpler components. For example, we can say a car is composed of parts like tires, an engine, and a body, and we can keep breaking those down into even smaller parts until we reach the basic data. In code, this is done by creating and using other classes, then passing their objects as attributes inside a larger class. It’s a powerful way to build complex systems from simpler, reusable pieces.

## Abstraction

[Abstraction](https://www.tutorialspoint.com/python/python_abstraction.htm) is the process of hiding unnecessary details from the user to make a program easier to use and understand. It allows users to focus on what an object does rather than how it does it. For example, when driving a car, you care about pressing the gas pedal or turning the wheel, not the internal mechanics that make it work. This is helpful for managing complexity and making programs easier to maintain since changes to internal code don’t affect how others use it. In programming, this is often managed through access control. Most languages have access modifiers like public, protected, and private to control what can be accessed and from where. These are represented using naming conventions rather than keywords. A public attribute has no underscore, a protected attribute starts with a single underscore, and a private attribute starts with two underscores. There's also a special attribute called [\_\_dict\_\_](https://www.executeprogram.com/courses/python-in-detail/lessons/__dict__), which holds all the attributes of an object and can be accessed with object.\_\_dict\_\_ to see the stored data. [Abstraction](https://www.geeksforgeeks.org/data-abstraction-in-python/) helps keep your code clean and safe by exposing only what’s necessary and hiding the rest.

## Inheritance

Just like classes and objects help organize your code into reusable pieces, inheritance helps you extend those pieces without having to start from scratch. [Inheritance](https://youtu.be/u1be7Vele5o?si=fdB-juAW0Ks7fOAD) allows you to create a new class based on an existing class, so the new class automatically gets all the attributes and methods of the original. The class that is being inherited from is called the [parent class](https://www.w3schools.com/python/python_inheritance.asp#:~:text=Create%20a%20Parent%20Class), and the class that inherits from it is the child class. The [child class](https://www.w3schools.com/python/python_inheritance.asp#:~:text=Create%20a%20Child%20Class) can access everything from the parent class and can also add its own unique features. You can even modify or override parts of the parent class if needed. This gives you the power to reuse existing functionality while still customizing the behavior for different use cases.

For example, if you already have a Student class that contains attributes like name, student\_id, and gpa, you might want to create a more specific type of student — like a GraduateStudent. Instead of rewriting everything, you can make GraduateStudent inherit from Student, and just add any new details you need, such as a thesis\_topic.

Here’s how that could look in code:

# Parent class

class Student:

def \_\_init\_\_(self, name, student\_id, gpa):

self.name = name

self.student\_id = student\_id

self.gpa = gpa

# Child class

class GraduateStudent(Student):

def \_\_init\_\_(self, name, student\_id, gpa, thesis\_topic):

super().\_\_init\_\_(name, student\_id, gpa)

self.thesis\_topic = thesis\_topic

There are multiple types of inheritance: single, multilevel, multiple, hierarchical, and hybrid.

A [single inheritance](https://www.geeksforgeeks.org/types-of-inheritance-python/#:~:text=inheritance%20in%20Python%3A-,Single%20Inheritance,-%3A%C2%A0) is when one class inherits from one parent class. For example, a Person class would store basic information like name and age. A Student class could inherit from the Person class to access those same attributes while also adding specific student details like GPA or major.

A [multiple inheritance](https://www.geeksforgeeks.org/multiple-inheritance-in-python/) is where a class can inherit from more than one parent class. A StudentAthlete class would inherit from both Student and Athlete classes, combining the academic and athletic information into one object.

A [multilevel inheritance](https://www.geeksforgeeks.org/multilevel-inheritance-in-python/) is inheritance that happens in a chain. For example, if Class A is inherited by Class B, and then Class B is inherited by Class C, Class C has access to features of both Class A and Class B.

A [hierarchical inheritance](https://openstax.org/books/introduction-python-programming/pages/13-4-hierarchical-inheritance) is when multiple child classes inherit from a single parent class. Each child class can use the shared attributes and methods of the parent but also define its own unique behavior.

A [hybrid inheritance](https://www.geeksforgeeks.org/what-is-hybrid-inheritance-in-python/) is a combination of two or more types of inheritance, such as mixing single and multiple inheritance in a single design. It allows for more complex class relationships and is useful for building flexible systems with overlapping features.

## Polymorphism

[Polymorphism](https://www.w3schools.com/python/python_polymorphism.asp) means “many forms,” and refers to the idea that different classes can define methods with the same name, but each class can implement that method differently. This lets you use the same method call across multiple objects and get different results depending on which class the object belongs to.

Python makes [polymorphism](https://youtu.be/tHN8I_4FIt8?si=W-FsZABuETGkNNoP) easy to use because of its [dynamic typing](https://builtin.com/articles/python-duck-typing). Polymorphism works hand-in-hand with method overriding. When a child class overrides a method from the parent class, it allows each object to respond in its own way while still using the same method name. You can call the same method on different types of objects, and Python will figure out which version of the method to use at runtime.

Here’s a simple example of [polymorphism](https://comp.mga.edu/learning/python/module/14#:~:text=April%2022%2C%202025-,14%2D3.%20Class%20Polymorphism,-Polymorphism%20in%20object) in action using a function and different data types. The function uses the same name and structure, but the result changes based on the type of data passed to it:

def summer(a, b):

return a + b

print(summer(1, 1)) #Output: 2

print(summer(["a", "b", "c"], ["d", "e"])) #Output: ['a', 'b', 'c', 'd', 'e']

print(summer("abra", "cadabra")) #Output: 'abracadabra'

Even though we’re calling the same function with the same number of arguments, the results are different depending on the type of data.

## \_\_init\_\_() Constructor

To really understand the meaning of classes, you’ve got to know the [\_\_init\_\_](https://www.geeksforgeeks.org/__init__-in-python/) function that is built in. The [\_\_init\_\_](https://youtu.be/mNpCPgdb2Jg?si=b-vWVvqI6BgJtJQE) function is a [special method](https://medium.com/data-bistrot/special-methods-in-python-oop-3b99585ee29c) used inside classes. It is called automatically every time you make a new object. Its job is to set up the object with initial values like assigning name or age. The \_\_init\_\_ function is short for initialize and always begins with def \_\_init\_\_(self, …).

## Self Keyword

The word [self](https://www.geeksforgeeks.org/self-in-python-class/) is used to refer to the specific object that is calling a method with a class. It is always the first [parameter](https://youtu.be/z8BOM0adRN0?si=Gqhw49UrHudiXWTJ) of an instance method. This is how the method knows which object it’s working with. By using self, you can access and change an object’s attributes, like updating a name for example. It makes it possible for each object to maintain its own independent data.

self.name = name

When you use it like we did right above you are basically saying “Take the name that was passed and store it in this object's name attribute.”

Even though it’s not a keyword, it is a widely used convention and makes your code easier to read and easier for others to understand. You don’t necessarily have to name the first parameter self, you could name it [anything](https://www.w3schools.com/python/python_classes.asp#:~:text=to%20the%20class.-,The%20self%20Parameter,-The%20self%20parameter) you want, but it makes everything much simpler if you just use self. If you forget to include it, your script will raise an error because Python still tries to send the instance to the method.

## Class and Instance Attributes

Classes have attributes and methods that help them define exactly what they have and what they can do, respectively. They make classes easily reusable. [Methods](https://www.w3schools.com/python/python_classes.asp#:~:text=Try%20it%20Yourself%20%C2%BB-,Object%20Methods,-Objects%20can%20also) are functions that are defined inside of a class. They are meant to be used by the objects created from that class. Methods are attached to specific objects and usually operate on that object's data. Methods are also called functions and will be used interchangeably. Attributes are the variables that belong to a class. Attributes will always be public and can be accessed using the [dot (.) operator](https://builtin.com/data-science/dot-notation) followed by your attribute name.

There are two types of attributes:

1. [Class Attributes](https://www.tutorialspoint.com/python/python_class_attributes.htm)
2. [Instance Attributes](https://www.geeksforgeeks.org/class-instance-attributes-python/#:~:text=1%20%20%20%20%20%20%20%20%20%20%20%20%20%20%0A2%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%0A2-,Instance%20Attributes,-Unlike%20class%20attributes)

In this setting, the word attribute and variable are used interchangeably so you may see a class attribute called a class variable; same goes for instance attributes.

Here are some of the main differences between class and instance attributes:

|  |  |
| --- | --- |
| **Class Attribute** | **Instance Attribute** |
| It is defined inside the class but outside of any method | It is defined inside the class and within methods (usually [\_\_init\_\_](https://www.geeksforgeeks.org/__init__-in-python/)) |
| It is shared across all instances of the class | It is unique to each individual instance |
| It is used for constants or default values common to all objects | It is used for storing data specific to each object |
| It can be modified for the class or overridden per object. Changes will affect all the objects of the chosen class. | It can be modified per object. Changes only affect the object that it is defined within. |
| It is accessed using the class name or object | It is accessed using a specific object |

Here is an example of using instance attributes. Notice how all instances attributes are specific to each defined object:

class Student:

def \_\_init\_\_(self, name, student\_id, gpa):

self.name = name #instance attribute

self.student\_id = student\_id #instance attribute

self.gpa = gpa #instance attribute

student1 = Student("Patrick", "900123", 3.8)

student2 = Student("Evelyn", "900124", 3.5)

print(student1)

print("%s (ID: %s) has a GPA of %.2f." % (student1.name, student1.student\_id, student1.gpa))

print("%s (ID: %s) has a GPA of %.2f." % (student2.name, student2.student\_id, student2.gpa))

Here's what it would look like if you added a class attribute. Notice that it will apply to all objects defined:

class Student:

school\_name = "Greenwood High" # class attribute (shared by all students)

def \_\_init\_\_(self, name, student\_id, gpa):

self.name = name # instance attribute

self.student\_id = student\_id # instance attribute

self.gpa = gpa # instance attribute

student1 = Student("Patrick", "900123", 3.8)

student2 = Student("Evelyn", "900124", 3.5)

print("%s (ID: %s) goes to %s and has a GPA of %.2f." % (student1.name, student1.student\_id, student1.school\_name, student1.gpa))

print("%s (ID: %s) goes to %s and has a GPA of %.2f." % (student2.name, student2.student\_id, student2.school\_name, student2.gpa))

## Method Overriding

You’ll often find yourself working with parent and child classes that share methods with the same name but need to behave differently. This is where [method overriding](https://youtu.be/hx_DRanBRRM?si=6_rNxqRXSQgVhVVP) comes into play. [Method overriding](https://www.tutorialspoint.com/python/python_method_overriding.htm) happens when a subclass provides its own version of a method that already exists in the parent class. The child class redefines the method using the same name and parameters, but with a new body that performs a different task. This allows the child class to customize or completely change how it handles that method while keeping the same method name.

Overriding a [method](https://www.geeksforgeeks.org/method-overriding-in-python/) is simple. You just define the method again in the child class using the same name and structure. When the method is called on an object of the child class, Python uses the child’s version instead of the parent’s. This is especially useful when different types of objects need to respond differently to the same action. It’s one of the key ideas in object-oriented programming because it lets you reuse and modify code efficiently.

Let’s return to our student record system. Suppose we have a Student class with a method called display\_info() that prints the name, ID number, and GPA of a student. If we later create a GraduateStudent class that inherits from Student, we might want its display\_info() method to also include the student’s thesis topic. To do this, we override the method in the child class by rewriting it.

class Student:

def \_\_init\_\_(self, name, student\_id, gpa):

self.name = name

self.student\_id = student\_id

self.gpa = gpa

def display\_info(self):

print(f"Name: {self.name}, ID: {self.student\_id}, GPA: {self.gpa}")

class GraduateStudent(Student):

def \_\_init\_\_(self, name, student\_id, gpa, thesis\_topic):

self.name = name

self.student\_id = student\_id

self.gpa = gpa

self.thesis\_topic = thesis\_topic

def display\_info(self):

print(f"Name: {self.name}, ID: {self.student\_id}, GPA: {self.gpa}, Thesis: {self.thesis\_topic}")

student1 = Student("Ashley", 1001234, 3.6)

grad\_student1 = GraduateStudent("Trey", 1002456, 3.95, "AI in Education")

student1.display\_info()

grad\_student1.display\_info()

## Setters & Getters

[Getters and setters](https://realpython.com/python-getter-setter/) are special methods used to control how values are accessed or updated within a class. While Python doesn’t enforce private variables like some other object-oriented languages, you can still use these methods to follow the principles of encapsulation. This will keep internal data safe and protected from unwanted changes. Instead of allowing direct access to class attributes, getters and setters offer a controlled way to read and modify them. This is especially helpful when you want to add validation or extra processing logic before setting a value or returning one.

Using [getters and setters](https://www.geeksforgeeks.org/getter-and-setter-in-python/) supports important programming concepts like encapsulation, abstraction, and consistency. It allows you to hide implementation details, keep your class’s data safe, and maintain consistent behavior throughout your program. You can even create attributes that are read-only or write-only, depending on your needs.

class Student:

def \_\_init\_\_(self):

self.name = ""

self.student\_id = ""

self.gpa = 0.0

# setter method for name

def set\_name(self, name):

self.name = name

# getter method for name

def get\_name(self):

return self.name

# setter method for student ID

def set\_student\_id(self, student\_id):

self.student\_id = student\_id

# getter method for student ID

def get\_student\_id(self):

return self.student\_id

# setter method for GPA

def set\_gpa(self, gpa):

if 0.0 <= gpa <= 4.0:

self.gpa = gpa

else:

print("Invalid GPA. Must be between 0.0 and 4.0.")

# getter method for GPA

def get\_gpa(self):

return self.gpa

# create a student object

student1 = Student()

# set values using setter methods

student1.set\_name("Sofia")

student1.set\_student\_id("900345")

student1.set\_gpa(3.7)

# get and print values using getter methods

print("%s (ID: %s) has a GPA of %.2f." % (

student1.get\_name(),

student1.get\_student\_id(),

student1.get\_gpa()))

## 

## Examples

1. Defining a Class

class Counting:

count = 0 # class attribute

def increase(self): #method

Counting.count += 1

# Object instantiation

c = Counting()

c.count = 10

#method calling

c.increase()

print(c.count)

1. Empty Class Example

#empty class definition

class Person:

pass

#Instantiating the object

person1 = Person()

#assigning attributes

person1.name = "Meena"

person1.surname = "Cruz"

person1.year\_of\_birth = 2002

person1.place="Savannah"

person2 = Person()

person2.name = "Vimal"

person2.surname = "Cruz"

person2.year\_of\_birth = 2002

person2.place="Savannah"

print(person1)

print("%s %s was born in %d." %

(person1.name, person1.surname, person1.year\_of\_birth))

print(person2.name, person2.surname)

1. Constructor example 1

class Person:

def \_\_init\_\_(self, name):

self.name = name

def talk(self):

print(f"{self.name} can talk")

person1 = Person("Riley")

print(person1.name)

person1.talk()

1. Default Constructor

class Car:

def \_\_init\_\_(self):

#Initialize the Car with default attributes

self.make = "Tesla"

self.model = "x1"

self.year = 2023

# Creating an instance using the default constructor

car = Car()

print(car.make)

print(car.model)

print(car.year)

1. Str method to print object

#string method to print object

class Person3:

def \_\_init\_\_(a, name, surname, year\_of\_birth):

a.name = name

a.surname = surname

a.year\_of\_birth = year\_of\_birth

def age(a, current\_year):

return current\_year - a.year\_of\_birth

def \_\_str\_\_(a):

return "%s %s was born in %d ." % (a.name, a.surname, a.year\_of\_birth)

p3 = Person3("Riley", "Smith", 1998)

print(p3)

print(p3.age(2022))

p4=Person3("Greg","Johnson",2004)#object instantiation

print(p4)

p4.\_\_dict\_\_

1. Abstraction: Public, Protected, and Private Methods

class ABC:

def \_\_init\_\_(self, a , b ,c,d) :#constructor

self.a = a

self.b = b

self.c = c

self.d=d

def test(self):

return "this is a public method"

def \_test1(self): # protected method

print("this is a protected method")

def \_\_test2(self): # private

print("this is a private method")

abc = ABC(3,4,5,6)#object instantiation

abc.\_ABC\_\_test2() # calling private method

abc.\_test1()

abc.\_\_dict\_\_ # helps us to access storage information

1. Getters and Setters

class Age:

def \_\_init\_\_(self):# default constructor

self.age = 10

# setter method

def set\_age(self, x):

self.age = x

# getter method

def get\_age(self):

return self.age

age1 = Age()#object instantiation

user\_age = int(input())

age1.set\_age(user\_age)#setter to pass the input

# retrieving age using getter

print("The entered age is",age1.get\_age())#getters to print the output

1. Inheritance

class Mammal:# parent class definition

def \_\_init\_\_(self,name):#constructor

self.name = name

def walk(self):

print(f"{self.name} can walk")

class Dog(Mammal):# child 1 definition

pass

class Cat(Mammal):# child 2 definition

def jump(self):

print(f"{self.name} can Jump")

dog1 = Dog("Dog") # child 1 object instantiation

dog1.walk()

#dog1.jump()

cat1 = Cat("Cat") # child 2 object instantiation

cat1.jump()

cat1.walk()

1. Method Overriding

class Employee:

def calculate\_pay(self):

return "Base salary calculation"

class Manager(Employee):

def calculate\_pay(self):

return "Base salary + bonus"

class Intern(Employee):

def calculate\_pay(self):

return "Stipend calculation"

# Usage

employees = [Employee(), Manager(), Intern()]

for emp in employees:

print(emp.calculate\_pay())

## Reference and Additional Materials:

<https://medium.com/@augeluiza/functions-reusable-blocks-of-code-eca0080037cd>

<https://codesarray.com/view/Understanding-Encapsulation-in-Python>

<https://www.executeprogram.com/courses/python-in-detail/lessons/__dict__>

<https://youtu.be/u1be7Vele5o?si=fdB-juAW0Ks7fOAD>