

PYTHON OBJECT ORIENTED PROGRAMMING EXERCISES BECOME A PRO DEVELOPER



EDCORNER LEARNING

Python
Object Oriented
Programming Exercises Become a
Pro Developer

Python Object Oriented Programming Exercises Become a Pro Developer

Edcorner Learning

Table of Contents

[Introduction](#)

[1 Overview of Python OOPS](#)

[2 What in Python is OOPS?](#)

[3 Some of the Major Advantages of OOPS are:](#)

[4 Difference between procedural and object-oriented programming](#)

[5 Python's Class and Objects](#)

[6 What is a Class Definition in Python?](#)

[7 An __init__ method: What is it?](#)

[8 Creating a Class Object](#)

[9 What is Instance Methods with Example](#)

[10 The four core ideas of object-oriented programming are:](#)

[11 Inheritance](#)

[12 Super\(\)](#)

[13 Polymorphism](#)

[14 Class Methods and polymorphism](#)

[15 Inheritance and Polymorphism](#)

[16 Encapsulation](#)

[17 A getter and a setter](#)

18 Modifiers of Access

19 Abstraction

20 Important aspects of abstract classes

21 Certain noteworthy aspects of Abstract classes are:

22 OOPS's benefits in Python

Lets starts Python OOPS Programming Exercises

Module 1 Class Method - Decorator

Module 2 Static Method - Decorator

Module 3 Special Methods

Module 4 Inheritance

Module 5 Abstract Classes

Module 6 Miscelleanuoes Exercises

Introduction

Python is a general-purpose interpreted, interactive, object-oriented, and a powerful programming language with dynamic semantics. It is an easy language to learn and become expert. Python is one among those rare languages that would claim to be both easy and powerful. Python's elegant syntax and dynamic typing alongside its interpreted nature makes it an ideal language for scripting and robust application development in many areas on giant platforms.

Python helps with the modules and packages, which inspires program modularity and code reuse. The Python interpreter and thus the extensive standard library are all available in source or binary form for free of charge for all critical platforms and can be freely distributed. Learning Python doesn't require any pre-requisites. However, one should have the elemental understanding of programming languages.

This Book consist of Indepth Python OOPS Concepts and 73 python Object Oriented Programming coding exercises to practice different topics.

In each exercise we have given the exercise coding statement you need to complete and verify your answers. We also attached our own input output screen of each exercise and their solutions.

Learners can use their own python compiler in their system or can use any online compilers available.

We have covered all level of exercises in this book to give all the learners a good and efficient Learning method to do hands on python different scenarios.

1 Overview of Python OOPS

Python's OOPS ideas are extremely similar to how we solve problems in the real world by writing programs. The most common method in programming is to solve any problem by creating objects.

The term "object-oriented programming" refers to this method. It is simpler to write and understand code thanks to object-oriented programming, which correlates our instructions with difficulties encountered in the actual world. They represent actual people, businesses, and employees as "software objects" containing "data" and "functions" that can be performed on them.

2 What in Python is OOPS?

Object Oriented Programming System is referred to as OOPS in programming. To create a programme using classes and objects is a paradigm for or methodology for programming. Every entity is treated as an object by OOPS.

Python's object-oriented programming is focused on objects. Any OOPS-based programming that is developed solves our problem but takes the shape of objects. For a particular class, we are free to produce as many objects as we choose.

What then are things? Anything with characteristics and certain behaviours is an object. The terms "variables of the object" and "functions of the object" are frequently used to describe an object's properties and behaviours, respectively. Objects might be conceptual or actual objects.

Let's say that a pen exists in the real world. A pen's characteristics are its colour and type (gel pen or ball pen). Additionally, the pen's behaviour may include the ability to write, draw, etc.

A logical object might be any file on your computer. Files can retain data, be downloaded, shared, and have other behaviours. They have properties like file name, file location, and file size.

3 Some of the Major Advantages of OOPS are:

- Writing readable and reuseable codes helps them reduce the amount of redundant code (using inheritance).
- Because they are so closely related to real-world circumstances, they are simpler to visualise. For instance, the ideas of objects, inheritance, and abstractions have a connection to real-world situations.
- Each object in oops represents a different section of the code and has its own logic and data for inter-object communication. Therefore, there are no difficulties with the code.

4 Difference between procedural and object-oriented programming

Python adheres to four different programming paradigms, did you know that?

Imperative, functional, procedural, and object-oriented programming are some of them.

Procedural Oriented Programming (POP) and Object-Oriented Programming are two of the most significant programming paradigms in Python (OOP). Let's start.

1. What Are They, First?

Let's examine the methodology each paradigm employs:

Suppose you want to prepare Maggie for dinner! Then you follow a series of stages, including —

Warm up some water in a pan.

Include Maggie in it

add masala

Prepare and serve.

Similar to this, POP needs to follow a specific set of steps in order to function. POP is made up of functions. There are different parts of a POP programme called functions, each of which is responsible for a different job. The functions are set up in a particular order, and the programme control happens one step at a time.

OOP: Object-oriented programming. The programme is divided into items. These objects are the entities that blend the traits and workings of things found in the real world.

2. Which Locations Do They Prefer?

POP is only appropriate for little jobs. Because the code becomes more difficult as the program's duration increases and becomes bloated with functions, Debugging gets even more challenging.

OOP works well for bigger issues. Recursion can be used to make the code reusable, which makes it simpler and cleaner.

3. Which Offers Greater Security?

POP offers the functions along with all the data, which makes it less secure. Thus, our data are not secret. If you wish to protect your login passwords or any other secret information, POP is not a recommended solution!

OOP offers security by data hiding, making it more secure. Encapsulation is a unique idea in OOP that grants it the ability to hide data (we will read about this further).

4. Programming approach

POP adheres to programming from the top down. The top-down method of programming focuses on dissecting a complex issue into manageable units of code. The minor portions of the problems are then resolved.

OOPS principles adhere to the Bottom-up method of programming. The Bottom-Up method prioritises resolving the smaller issues at their most fundamental level before incorporating them into a comprehensive and entire solution.

5. Utilization of Access Modifiers:

When applying the ideas of inheritance, access specifiers or access modifiers are used in Python to restrict the access of class variables and class methods outside of the class. The keywords Public, Private, and Protected can be used to do this.

No access modifiers like "public," "private," or "protected" are used in POP. To employ the aforementioned modifiers, POP lacks the concepts of classes and inheritance. Modifiers for access are supported by OOP. They understand inheritance, so they can use terms like "public," "private," or "protected."

Note:

The access modifiers in Python come in quite handy when working with inheritance ideas. Class methods can make use of this idea as well.

5 Python's Class and Objects

Let's say you want to keep track of how many books you own. You can easily do that by utilising a variable. Also possible is to compute the sum of five numbers and save the result in a variable.

Simple values are intended to be stored in a variable using primitive data structures like numbers, characters, and lists. Consider your name, the square root of a number, or the quantity of marbles (say).

But what if you need to keep a record of every employee in your business? For instance, if you try to put all of your employees in a list, you can later become confused about which index of the list corresponds to which individual details (e.g. which is the name field, or the empID or age etc.)

```
#Edcorner Learning Python OOPS

employee1 = ['Edcorner', 104120, "Developer", "Dept. 2A"]
employee2 = ['Learning', 211240, "Designer", "Dept. 3B"]
employee3 = ['John', 131124, "Manager", "Dept. 1E"]
|
```

Even if you try to keep them in a dictionary, the entire codebase will eventually become too complicated to manage. Therefore, we employ Python Classes in these situations.

To create user-defined data structures in Python, use a class. Classes set up functions, called "methods," that describe how an object made from a class can behave and what actions it can take. Classes and objects are the main topics covered by Python's OOPS ideas.

Classes simplify the code by avoiding complicated codebases. It accomplishes this by developing a model or design for how everything ought to be defined. Any object that derives from the class should have the characteristics or capabilities specified by this clause.

Note:

Simple structural definition is all that a class does. It makes no specific reference to anything or anyone. For instance, the class HUMAN has attributes like name, age, gender, and city. It does not identify any particular HUMAN, but it does describe the qualities and capabilities that a HUMAN or an object of the HUMAN class ought to have.

The term "object" refers to a class instance. It is the class's actual implementation and is real.

Properties

name
age
gender
city

Methods

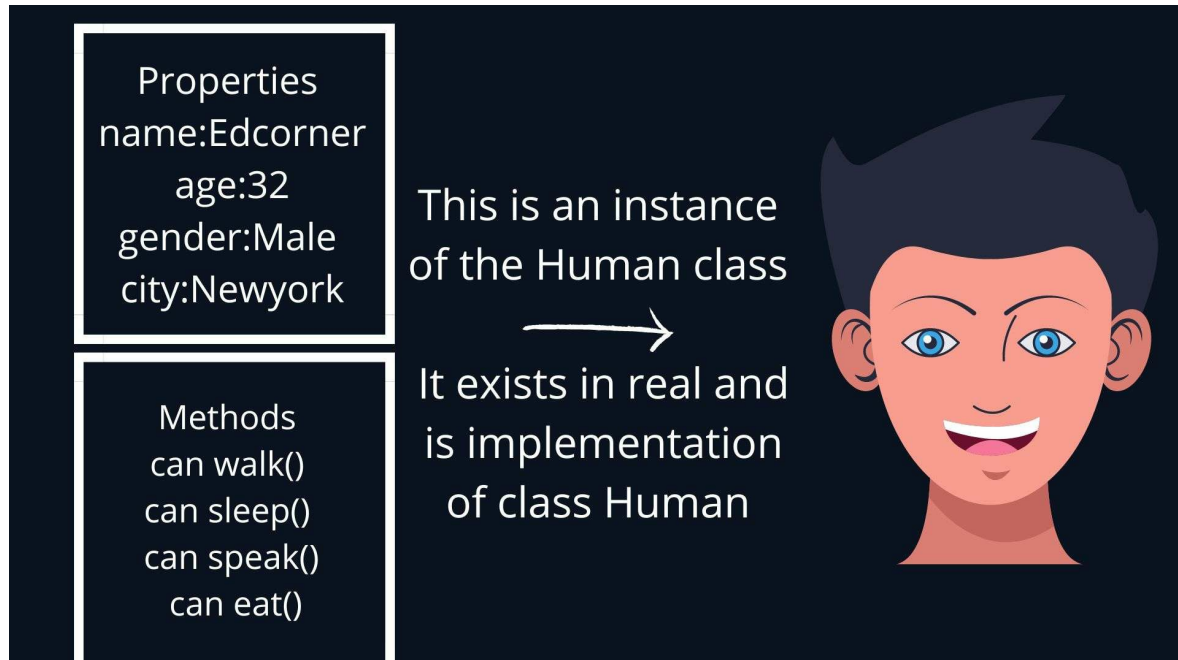
walk()
sleep()
speak()
eat()

This is class Human



It's a concept





A collection of data (variables) and methods (functions) that access the data is referred to as an object. It is the actual class implementation.

Take this example where Human is a class. This class only serves as a template for what Human should be, not an actual implementation. You could argue that the "Human" class only makes logical sense.

However, "Edcorner" is a Human class object (please refer the image given above for understanding). It follows that Edcorner was built using the blueprint for the Human class, which holds the actual data. Unlike "Human," "Edcorner" is a real person (which just exists logically). He is a genuine person who embodies all the characteristics of the class Human, including having a name, being male, being 32 years old, and residing in Newyork. Additionally, Edcorner uses all of the methods included in the Human class; for example, Edcorner can walk, speak, eat, and sleep.

And many humans can be produced utilising the class Human blueprint. For instance, by leveraging objects and the blueprint for the class Human, we could generate many more persons.

Short Tip:

class = draught (suppose an architectural drawing). The Object is a real item that was created using the "blueprint" (suppose a house). An instance is a representation of the item that is virtual but not a true copy.

No memory is allotted to a class when it is defined; just the object's blueprint is produced. Memory allocation only takes place during object or instance creation. The actual data or information is contained in the object or instance.

6 What is a Class Definition in Python?

Python classes are defined using the word `class`, which is then followed by the class name and a colon.

Syntax:

```
#Edcorner Learning Python OOPS
```

```
class Human:  
    pass
```

```
|
```

Below the class definition, indented code is regarded as a component of the class body.

'pass' is frequently used as a stand-in for code whose implementation we can forego for the moment. We may execute the Python code without throwing an error by using the "pass" keyword.

7 An `__init__` method: What is it?

In a method named `init`, the qualities that all Human objects must possess are specified (). When a new Human object is formed, `__init__()` assigns the values we supply inside the object's properties to set the object's initial state. In other words, each new instance of the class is initialised via `__init__()`. Any number of parameters can be passed to `__init__()`, but `self` is always the first parameter.

The `self` parameter contains a reference to the active class instance. This means that we can access the data of an object's variables by using the `self` argument, which corresponds to the address of the current object of a class.

Since this `self` points to the address of each individual object and returns its corresponding value, even if there are 1000 instances (objects) of a class, we can always access each of their unique data.

Let's look at how the Human class defines `__init__()`:

```
#Edcorner Learning Python OOPS
```

```
class Human:  
    def __init__(self, name, age, gender):  
        self.name = name  
        self.age = age  
        self.gender = gender  
|
```

We use the self variable three times in the body of. `__init__()` for the following purposes:

`self.name = 'name'` creates the name attribute and sets the name parameter's value as its value.

self.age = The age attribute is created and given the value of the age parameter that was supplied.

self.gender = The gender parameter value is produced and assigned to the gender attribute.

In Python, there are two categories of attributes:

Class attribute number 1:

These variables apply to all instances of the class equally. For each new instance that is created, they do not have new values. Just after the class definition, they are defined.


```
#Edcorner Learning Python OOPS
```

```
class Human:  
    #class attribute  
    species = "Homo Sapiens"  
|
```

In this case, whatever object we create will have a fixed value for the species.

2. Instance Information:

The variables that are defined inside of any class function are known as instance attributes. Every instance of the class has a unique value for the instance attribute. These values rely on the value that was supplied when the instance was created.

```
#Edcorner Learning Python OOPS

class Human:
    #class attribute
    species = "Homo Sapiens"
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

x = Human("Edcorner", 32, "Male")
y = Human("Learning", 30, "Female")
```

The instance attributes in this case are name, age, and gender. When a new instance of the class is created, they will have different values.

8 Creating a Class Object

It is referred to as instantiating an object when a new instance of a class is created. The class name and parantheses can be used to create an object. The object of a class can be assigned to any variable.

```
#Edcorner Learning Python OOPS
```

```
x = ClassName()  
|
```

Memory is allocated to an object as soon as it is created. Therefore, employing the operator `==` to compare two instances of the same class will result in false (because both will have different memory assigned).

The values for name, age, and gender must also be supplied when creating objects of the Human class.

```
#Edcorner Learning Python OOPS

class Human:
    #class attribute
    species = "Homo Sapiens"
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

x = Human("Edcorner", 32, "Male")
y = Human("Learning", 30, "Female")

print(x.name)
print(y.name)
```

Edcorner
Learning

Here, we have created two Human class instances by passing in all the necessary inputs.

A `TypeError` will be raised if the necessary parameters are not given. `TypeError: The three necessary positional arguments for init(), "name," "age," and "gender," are missing.`

Now let's look at how to use class objects to retrieve those values. The dot notation allows us to access the instance values.

```
#Edcorner Learning Python OOPS
```

```
class Human:
    species = "Homo Sapiens"

    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender
```

```
# x and y are instances of class Human
```

```
x = Human("Edcorner", 30, "male")
```

```
y = Human("Learning", 32, "female")
```

```
print(x.species) # species are class attributes, hence will have same value for all instances
```

```
print(y.species)
```

```
# name, gender and age will have different values per instance, because they are instance attributes
```

```
print(f"Hi! My name is {x.name}. I am a {x.gender}, and I am {x.age} years old")
```

```
print(f"Hi! My name is {y.name}. I am a {y.gender}, and I am {y.age} years old")
```

```
Homo Sapiens  
Homo Sapiens  
Hi! My name is Edcorner. I am a male, and I am 30 years old  
Hi! My name is Learning. I am a female, and I am 32 years old
```

As a result, we discover that the dot operator is all we need to access the instance and class attributes.


```
#Edcorner Learning Python OOPS

class Human:
    #class attribute
    species = "Homo Sapiens"
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

Human.species = "Sapiens"
obj = Human("Edcorner",32,"male")
print(obj.species)
```

Sapiens

The class attributes in the example above have the identical values of "Homo Sapiens," but the instance attributes have various values depending on the parameter we gave when constructing our object.

However, we can alter the value of a class attribute by setting a new value to classname.classAttribute.

9 What is Instance Methods with Example

A function inside a class that can only be called from instances of that class is termed an instance method. An instance method's first parameter is always self, much as init().

Let's take an example and implement some functions

```
#Edcorner Learning Python OOPS

class Human:
    #class attribute
    species = "Homo Sapiens"
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

    #Instance Method
    def speak(self):
        return f"Hello everyone! I am {self.name}"

    #Instance Method
    def eat(self, favouriteDish):
        return f"I love to eat {favouriteDish}!!!"

x = Human("Edcorner", 30, "female")
print(x.speak())
print(x.eat("Salad"))
```

```
Hello everyone! I am Edcorner
I love to eat Salad!!!
```

Two instance methods exist for the Human class:
speak() produces a string containing the Human's name.

`eat()` returns a string with the Human's favourite food and accepts the single input `"favouriteDish"`.

Now that we have a solid understanding of what Python classes, objects, and methods are, it is time to turn our attention to the foundational ideas of OOP.

10 The four core ideas of object-oriented programming are:

- Inheritance
- Encapsulation
- Polymorphism
- Abstraction of data.

Let's take a closer look at each of the Python OOPS ideas.

11 Inheritance

People frequently tell newborns that they have face features that resemble those of their parents or that they have inherited particular traits from their parents. It's possible that you've also observed that you share a few characteristics with your parents.

The real-world situation is fairly similar to inheritance as well. However, in this case, the "parent classes'" features are passed down to the "child classes." They are referred to as "properties" and "methods" here, along with the qualities they inherit.

A class can derive its methods and attributes from another class's by using the process known as inheritance. Inheritance is the process of a child class receiving the properties of a parent class.

```
#Edcorner Learning Python OOPS

class parent_class:
#body of parent class

class child_class( parent_class): # inherits the parent
class
#body of child class
|
```

The Parent class is the class from which the properties are inherited, and the Child class is the class from which the properties are inherited.

As we did in our previous examples, we define a typical class. After that, we may declare the child class and provide the name of the parent class it is descended from in parenthesis.

```
#Edcorner Learning Python OOPS
```

```
class Human:      #parent class
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

    def description(self):
        print(f"Hey! My name is {self.name}, I'm a {self.gender} and I'm {self.age} years old")
```

```
class Boy(Human):  #child class
    def schoolName(self, schoolname):
        print(f"I study in {schoolname}")
```

```
b = Boy('Edcorner', 32, 'male')
b.description()
b.schoolName("MIT")
```



```
Hey! My name is Edcorner, I'm a male and I'm 32 years old  
I study in MIT
```

The parent class Human is inherited by the child class Boy in the example above. Because Boy is inheriting from Human, we can access all of its methods and properties when we create an instance of the Boy class.

In the Boy class, a method called schoolName has also been defined. The parent class object is unable to access the method schoolName. The schoolName method can, however, be called by making a child class object (Boy).

Let's look at the problem we run into if we try to use the object from the parent class to invoke the methods of a child class.

```
#Edcorner Learning Python OOPS

class Human:
    def __init__(self,name,age,gender):
        self.name = name
        self.age = age
        self.gender = gender
    def description(self):
        print(f"Hey! My name is {self.name}, I'm a {self.gender} and I'm {self.age} years old")

class Girl(Human):
    def schoolName(self,schoolName):
        print("I study in {schoolName}")

h = Human('Edcorner',20,'girl') # h is the object of the
parent class - Human
h.description()
h.schoolName('ABC Academy') #cannot access child class's
method using parent class's object
```

```
Traceback (most recent call last):  
  File "./prog.py", line 22, in <module>  
AttributeError: 'Human' object has no attribute 'schoolName'
```

Therefore, the `AttributeError: 'Human' object has no attribute 'schoolName'` is returned in this case. because it is not possible for child classes to access parent class data and properties.

12 Super()

The parent class is referred to in the inheritance-related method `super()`. It can be used to locate a certain method in a superclass of an object. It serves a very important purpose. Let's examine its operation —

```
#Edcorner Learning Python OOPS
```

```
super().methodName()  
|
```

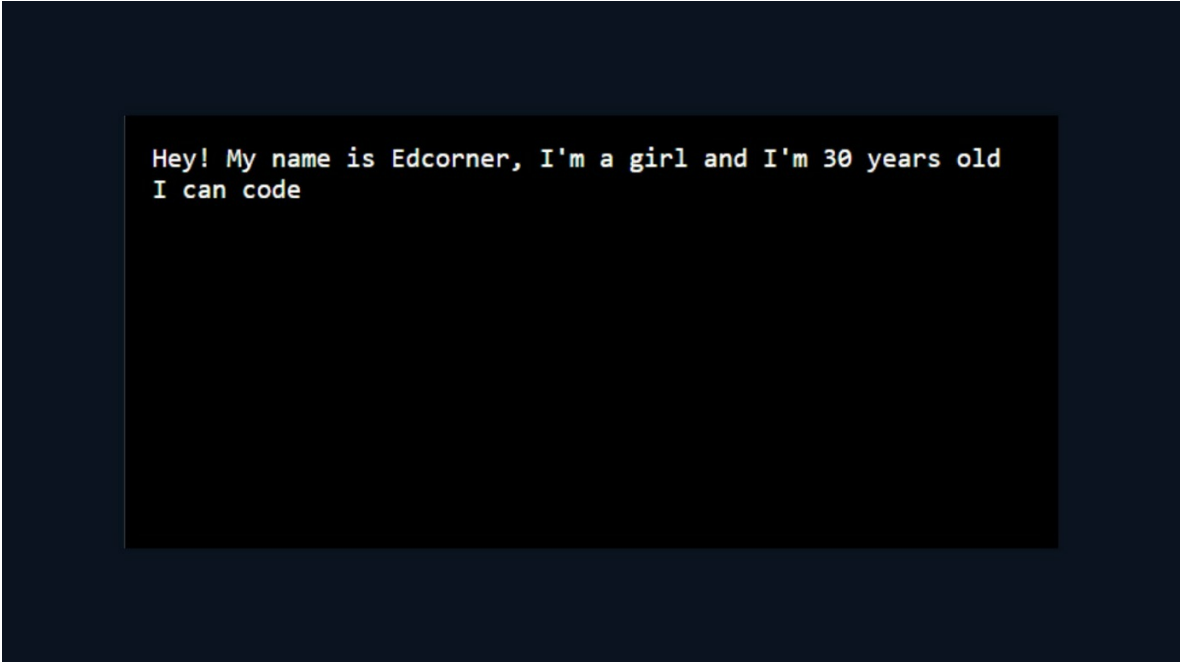
The super function's syntax is as follows. After the super() keyword, we write the name of the parent class function we want to refer to.

```
#Edcorner Learning Python OOPS

class Human:
    def __init__(self,name,age,gender):
        self.name = name
        self.age = age
        self.gender = gender
    def description(self):
        print(f"Hey! My name is {self.name}, I'm a {self.gender} and I'm {self.age} years old")

    def code(self):
        print("I can code")

class Girl(Human):
    def codecode(self):
        print("I can teach")
    def activity(self):
        super().code()
g = Girl('Edcorner', 30, 'girl')
g.description()
g.activity()
```



```
Hey! My name is Edcorner, I'm a girl and I'm 30 years old
I can code
```

Here, the classes Human and Girl have definitions for the Code() method. But as you can see, each method has a separate implementation. The Code technique in Human class reads "I can Code," whereas the Code method in Girl class reads "I can teach." So let's call the Code technique from the child class to the parent class.

As you can see, we are using `super` to call the `Code` method at line 19. `()`.

`Code()`. This will invoke the Human class' `Code` method. Thus, "I can Code" is printed. Nevertheless, `Code()` was already implemented in `Girl` (at line 15).

If a method with the same name already exists in the subclass, `super()` will still call the method in the superclass.