Hyperiondev

TASK

Introduction to Object-Oriented Programming

Visit our website

Introduction

WELCOME TO THE INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING TASK!

Object-Oriented Programming (OOP) is a fundamental style of programming for developing larger pieces of software. Up until now, the programs you have written are simple enough to be run from just one file. In the real world of software development, multiple programmers work on large projects that may have hundreds of different files of code that implement the functionality of the project.

Your first step to building more complex programs is understanding OOP. This may be the first truly abstract concept you encounter in programming, but don't worry! Practice will show you that once you get past the terminology, OOP is very simple.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHY OOP?

Imagine we want to build a program for a university. This program has a database of students, their information, and their marks. We need to perform computations of this data, such as finding the average grade of a particular student. Here are some observations from the above problem:

- A university will have many students that have the same information stored in the database, for example, age, name, and gender. How can we represent this information in code?
- We need to write code to find the average of a student by simply summing their grades for different subjects, and dividing by the number of subjects taken. How can we only define this code once and reuse it for many students?

OOP is the solution to the above problems, and indeed many real-world implementations of the above systems will use OOP.

THE COMPONENTS OF OOP

The Class

The concept of a class may be hard to get your head around at first. A class is a specific Python file that can be thought of as a 'blueprint' for a specific data type.

We have discussed different data types in previous tasks, and you can think of a Class as defining your own special data types, with properties you determine.

A class stores properties along with associated functions called ***methods*** which run programming logic to modify or return the class properties.

In the example that we discussed earlier (building a program for a university around a database of students), we would use a class called Student to represent a student. This is perfect because we know that the properties of a Student match those stored in the database such as name, age, etc.

Defining a Class in Python

Let us assume that the database stores the age, name, and gender of each student. The code to create a blueprint for a Student, or class, is as follows:

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

This may look confusing, so let's break it down:

● Line 1:
This is how you define a class. By convention, classes start with a capital letter to differentiate them from variable names which follow camelBackNotation.

● Line 3:
This is called the constructor of the class. A constructor is a special type of function that basically answers the question 'What data does this blueprint for creating a Student need to initialise the Student?'. This is why it uses the term 'init' which is short for initialisation. As you can see, age, name, and gender are passed into the function.

● Line 4-6:
We also passed in a parameter called `self`. self is a special variable that is hard to define. It is basically a pointer to this Student object that you are creating with your Student class/blueprint. By saying `self.age = age`, you're saying "I'll take the age passed into the constructor, and set the value of the age parameter of THIS Student object I am creating to have that value". The same logic applies to the name and gender variables.

The above piece of code is more powerful than you may think. All OOP programs you write will have this format to define a class (the blueprint). This class now gives us the ability to create thousands of Student objects which have predefined properties. Let's look at this in more detail.

Creating objects from a class

Now that we have a blueprint for a Student, we can use it to create many Student objects. Objects are basically initialised versions of your blueprint. They each have the properties you have defined in your constructor. Let's look at an example. Say we want to create objects from our object representing two students, namely Philani and Sarah.

This is what it looks like in Python:

```
class Student(object):
      def __init__(self, age, name, gender, grades):
            self.age = age
            self.name = name
            self.gender = gender
            self.grades  = grades
philani  = Student(20, "Philani Sithole", "Male", [64,65])
sarah = Student(19, "Sarah Jones", "Female", [82,58])
```

We now have two objects of the class Student called Philani and Sarah.

Pay careful attention to the syntax for creating a new object. As you can see, the age, name, and gender are passed in when defining a new object of type Student.

These two objects are like complex variables. At the moment they can't do much because the class blueprint for Student just stores data, but let's extend the Student class with methods.

Creating methods for a class

Methods allow us to define functions that are shared by all objects of a class to carry out some core computations. Recall how we may want to compute the average mark of every student. The code below allows us to do exactly that:

```
class Student(object):
      def __init__(self, age, name, gender, grades):
            self.age = age
            self.name = name
            self.gender = gender
            self.grades  = grades

      def compute_average(self):
            average = sum(self.grades)/len(self.grades)
            print("The average for student " + self.name + " is " +
str(average))

philani  = Student(20, "Philani Sithole", "Male", [64,65])
sarah = Student(19, "Sarah Jones", "Female", [82,58])

sarah.compute_average()
```

First, notice that we've added a new property for each student, namely grades, which is a list of ints representing a student marks on two subjects. In our example of university students, this can most certainly be retrieved from a database.

Secondly, notice a new method called compute_average has been defined under the Student class/blueprint. This method takes in self. This just means that 'this method has access to the specific Student object properties which can be accessed through `self.___`. Notice this method uses self.grades and self.name to access the properties for a particular student average calculation.

The program outputs: The average for student Sarah Jones is 70.

This is the output of the method call on line 17. Note the syntax, especially the () for calling this method from one of our objects. Only an object of type Student can call this method, as it is defined only for the Student class/blueprint.

As you can see, we can call the methods of objects that allow us to carry out present calculations. The code for this program is available in your folder in student.py. Every object we define using this blueprint will be able to run this predefined method, effectively allowing us to define hundreds of Student objects and efficiently find their averages with only 11 lines of code - all thanks to OOP!

What is Inheritance?

Inheritance is a bit more complicated than the classes and objects examined previously. Suppose you had written a class with some properties and methods, and you wanted to define another class with most of the same properties and methods, plus some additional structure. Inheritance is the mechanism by which you can produce the second class from the first, without redefining the second class completely from scratch or altering the definition of the first class itself.

Alternatively, suppose you wish to write two related classes that share a significant subset of their respective properties and methods. Rather than writing two completely separate classes from scratch, you could encapsulate the shared information in a single class and write two more classes that inherit all of that information from the first class, saving space and improving code clarity in the long run.

A class that is inherited from is called the "base" class, and the class that inherits from the base class is called the "derived" class. In most major object-oriented languages, objects of the derived class are also objects of the base class, but not vice-versa. This means that functions which act on base objects may also act on derived objects, a fact which is often useful when writing an object-oriented program. (Note: this distinction is arguably more important in languages with a

stronger type system; Python takes everything to an extreme by making all user-defined classes subclasses of a base type confusingly referred to as 'object'. If you are interested in the internals of the Python language (and you should be, because it is interesting!), master the material in this Task and then try looking online at some other material.)

To illustrate this concept better, read through the cow-themed demonstrations of composition, inheritance, and some other Python-specific object-oriented language features located in **example.py**.

Now, open **exercise.py** and fill out the logic for the method definitions. Rename this file to **exercise_completed.py** when complete. You may leave any questions you have for your mentor in comments.txt.

Instructions

First, read example.py, open it using Notepad++ (Right-click the file and select 'Edit with Notepad++') or IDLE.

Run **example.py** to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with the topic.

Compulsory Task 1

In this task, we're going to be simulating an SMS message. Some of the logic has been filled out for you in the **email.py** file.

- Open the file called **email.py**.
- Create a class definition for an `Email` which has three variables: `has_been_read`, `email_contents`, `is_spam` and `from_address`.
- The constructor should initialise the sender's email address.
- The constructor should also initialise `has_been_read` and `is_spam` to false.
- Create a function in this class called `mark_as_read` which should change `has_been_read` to true.
- Create a function in this class called `mark_as_spam` which should change `is_spam` to true.
- Create a list called `inbox` to store all emails.
- Then create the following methods:

- `add_email` - which takes in the contents and email address from the received email to make a new `Email` object.
- `get_count` - returns the number of messages in the store.
- `get_email` - returns the contents of an email in the list. For this, allow the user to input an index i.e. `GetEmail(i)` returns the email stored at position `i` in the list. Once this has been done, `has_been_read` should now be true.
- `get_unread_emails` - should return a list of all the emails that haven't been read.
- `get_spam_emails` - should return a list of all the emails that have been marked as spam.
- `delete` - deletes an email in the inbox.

Now that you have these set up, let's get everything working!

- Fill in the rest of the logic for what should happen when the user inputs send/read/quit. Some of it has been done for you.

Compulsory Task 2

In this task, we're going to be creating classes to exhibit vectors and complex numbers.

- Open the file called **exercise.py**.
- Complete the instruction indicated in Part 1

Compulsory Task 3

In this task, we're going to be creating classes to create the game tic-tac-toe, also known as noughts and crosses.

- Open the file called **exercise.py**.
- Complete the instruction indicated in Part 2

Completed the task(s)?

Ask your mentor to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.