TASK

Introduction to Object-Oriented Programming II: Inheritance

Visit our website

Introduction

WELCOME TO THE INHERITANCE TASK!

Inheritance is one of the core pillars of OOP. Inheritance allows us to reuse code from classes in new classes that share that code, but can have additional logic or variations on the logic.

Get in touch

Connect for support

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

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team 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!

What is Inheritance?

Let's begin by examining inheritance from the natural world. A look at a child in comparison to a parent quickly reveals that the child inherits certain traits from the parents. The attributes that they might inherit are things such as eye colour, size of nose, or height. They could also inherit certain abilities such as athletic abilities.

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 properties and methods. 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 (also referred to as the parent class), and the class that inherits from the base class is called the "derived" class (also referred to as the subclass).

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 that act on base objects may also act on derived objects, a fact which is often useful when writing an object-oriented program.

Python inheritance in action

To demonstrate how inheritance works, first, we need to create a parent class, I will use a “car” class as an example:

```
# Parent class for a car from which we can extend to a subclass
class Car:
    #class variable for whether engin is running or not
    is_running = False

    # constructor that allows us to set the make and model
    # as instance variables
    def __init__(self, make, model):
        self.make = make
        self.model = model

    # this method starts the engine
    def start_car(self):
        self.is_running = True

    # this method turns off the engine the engine
    def turn_off_car(self):
        self.is_running = False

    # this method prints the make and model to console
    def show_make_and_model(self):
        print(f"This vehicle is a {self.make} {self.model}")
```

The parent class above has attributes for the make and model of the car and some actions or methods that can turn the car on and off. We can extend this class to a pickup truck class (the subclass) which will inherit the attributes and methods from the car class, but we can then add additional attributes and methods.

```
# We are inheriting all of the attributes and methods
# from the Car class by passing it as an argument to
# the PickupTruck class
class PickupTruck(Car):
    #this is an additional class variable that is specific
    # to the PickupTruck class
    is_loaded = False
    # this method loads the truck
    def load(self):
        self.is_loaded = True

    # this method removes the load from the truck
    def unload(self):
        self.is_loaded = False
```

We can then create an instance of the PickupTruck class and demonstrate how the attributes and methods are inherited from the Car class as well as new attributes and methods that we created in the subclass:

```
# we are calling a method that we created in the subclass
# this changes the variables from False to True
pickup_truck_1.load()
# we are calling a method inherited from the parent class
pickup_truck_1.start_car()

# we print out to values so that we can see that both of
# the above methods worked
print(pickup_truck_1.is_running)
print(pickup_truck_1.is_loaded)

# lastly we are calling another method that was inherited
# from the parent class
pickup_truck_1.show_make_and_model()
```

The above code will generate the following output which demonstrates that all the attributes and methods are working in the subclass:

```
True
True
This vehicle is a Toyota Hilux
```

The “super” function and its use

Python’s super() function gives you access to methods in a parent class from the subclass that inherits from it. The super() alone returns a temporary object of the superclass that allows you to call that superclass’s methods.

While referring to the parent from the subclass, we don’t need to write the name of the parent class explicitly.

Calling the methods created in the parent class with super() saves you from needing to rewrite those methods in your subclass and enables you to swap out parent classes with minimal code changes.

The super() function returns a temporary object of the parent class that allows you to call that parent class’ methods. The goal of the super() function is to provide a much more abstract and portable solution for initialising classes. Let’s see an example of the super() function in Python.

```
class Computer():
    def __init__(self, computer, ram, ssd):
        self.computer = computer
        self.ram = ram
        self.ssd = ssd

class Laptop(Computer):
    def __init__(self, computer, ram, ssd, model):
        super().__init__(computer, ram, ssd)
        self.model = model

vivobook = Laptop('Asus', 8, 512, 'Vivobook')
print('The computer make is:', vivobook.computer)
print(f"This computer has ram of {vivobook.ram}GB")
print(f"This computer has ssd of {vivobook.ssd}GB")
print('This computer model is:', vivobook.model)
```

In the above example, the Laptop class inherits from the Computer class. Using the super() function, we are able to use the constructor function of the parent class in the constructor function of the subclass so that the three attributes from the parent class (computer, ram, ssd) can be initialised together with the additional attribute of the subclass (model).
So, now, if we only create an object of the subclass class, we still have all the access to the parent class' attributes because of the super() function.

Method Overriding

Overriding is the ability of a class to change the implementation of a method provided by a parent class.

Overriding is a very important part of OOP. By using method overriding a class can make a copy of another class, but at the same time enhance or customise part of its behaviour. Method overriding is thus a part of the inheritance mechanism.

In Python, method overriding occurs by defining in the subclass a method with the same name of a method in the parent class. When you define a method in the subclass that has the same name as the method in the parent class, an instance of the subclass will execute the logic in the subclass when that method is called. If this method is not defined in the subclass, then the method in the parent class is executed.

Here is an example of a method being inherited from the parent class without method overriding, when this method is executed from an instance of the subclass, the method in the parent class is executed:

```
class Father():
    def transport(self):
        print("The transport used is a car")

class Son(Father):
    pass

son_1 = Son()
# this will output "The transport used is a car"
# because it is using the inherited method from
# the class called father.
son_1.transport()
```

Now let's override the transport() method in the subclass so that a son will be printed as using a bicycle.

```
class Father():
    def transport(self):
        print("The transport used is a car")

class Son(Father):
    def transport(self):
        print("The transport used is a bicycle")

son_1 = Son()
# this will output "The transport used is a bicycle"
# because the inherited method is being overridden
# by the subclass
son_1.transport()
```

Compulsory Task 1

In this task, we're going to be demonstrating your understanding of inheritance. Make a copy of the **task1_instructions.py** file named **compulsory_task1.py** and follow the instructions.

Compulsory Task 2

Create a file named **method_override.py** and follow the following instructions:

- Take user inputs that ask for the name, age, hair colour and eye colour of a person.
- Create an adult class with the following attributes and method:
    - name, age, eye_colour, hair_colour
    - Method called can_drive() that prints the name of the person and that they are old enough to drive.
- Create a subclass of the adult class named “Child” that has the same attributes, but overrides the can_drive method to print the persons name and that they are too young to drive.
- Create some logic that determines if the person is 18 or older and create an instance of the Adult class if this is true. Otherwise, create an instance of the Child class. Once the object has been created, call the can_drive() method to print out whether the person is old enough to drive or not.

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.