**Object-Oriented Programming In Python**

**OOPs**

**Object-oriented programming (OOP)** is a programming paradigm that deals with various fundamentals of its concepts.

Python is the best OOPs supported programming language since its existence It can solve programming problems where computation is done by the object.

**List of OOPs concepts**

1. Class
2. Object
3. Method
4. Inheritance
5. Polymorphism
6. Encapsulation
7. Data abstraction

### **Defining a Class in Python**

Before creating a Class, we should know a few essential instructions to build an effective class.

* To create a class, use the **“class” keyword** followed by the **class name**.
* The end of the Class name must declare a **colon**.
* To declare a documentation string it is useful to get some more information about the class.
* Using **constructor** in a class, we will be smoothly handling a large program.

**Syntax**

Class classname:

   <Statement 1 >

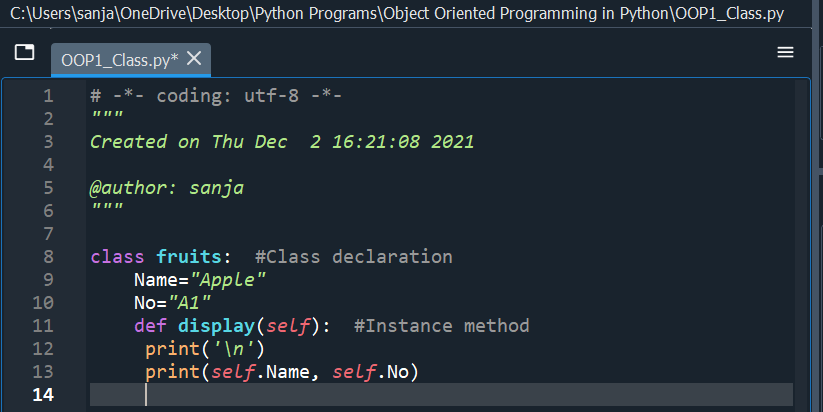
……………………………..

……………………………..

    <Statement N >

**Program 1 :-**

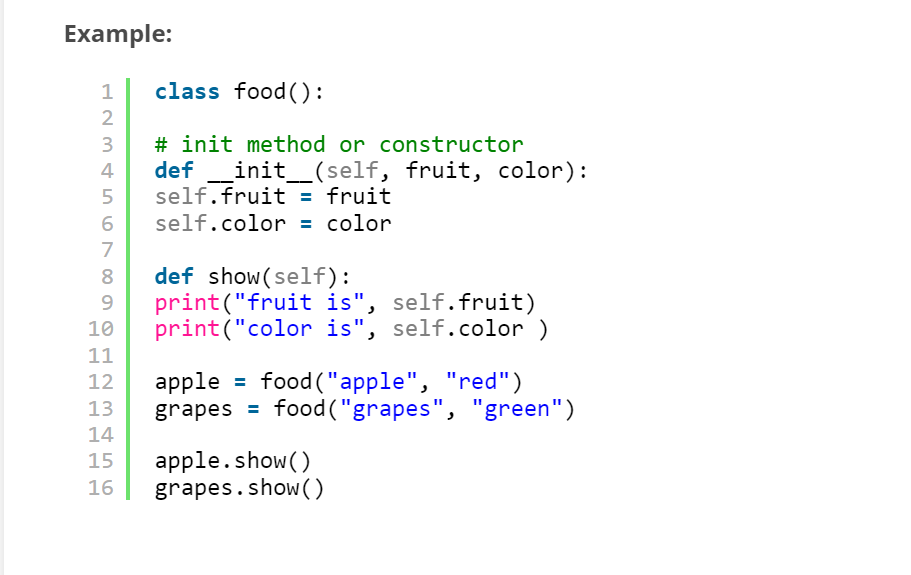
**Code :-**



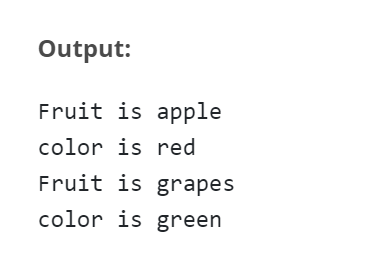
**Self :-**

* **The self** is used to represent the [**instance**](https://www.edureka.co/blog/isinstance-in-python/)**of the class**.
* With this keyword, you can **access the attributes and methods** of the [class in python](https://www.edureka.co/blog/python-class/).
* It binds the attributes with the given arguments. The reason why we use self is that **Python does not use the ‘@’ syntax to refer to instance attributes.**

**Example :-**



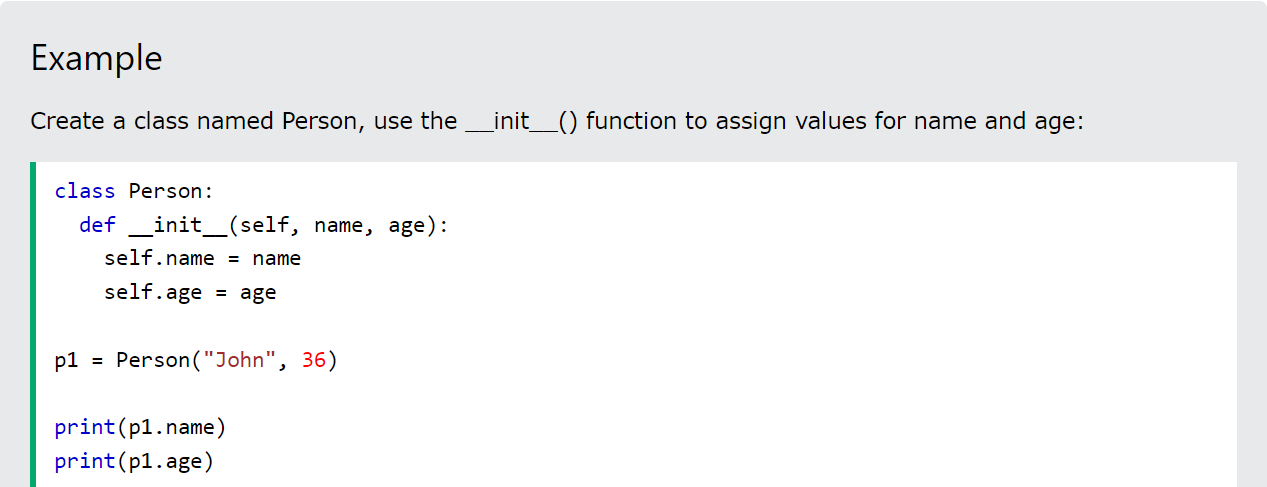
**Output :-**



**The \_\_init\_\_() Function**

To understand the meaning of classes we have to understand **the built-in \_\_init\_\_() function.**

**All classes have a function called \_\_init\_\_(),** which is always **executed when the class is being initiated.**Use the \_\_init\_\_() function to **assign values to object properties**, or other operations that are necessary to do when the object is being created.**Acts like a constructor**



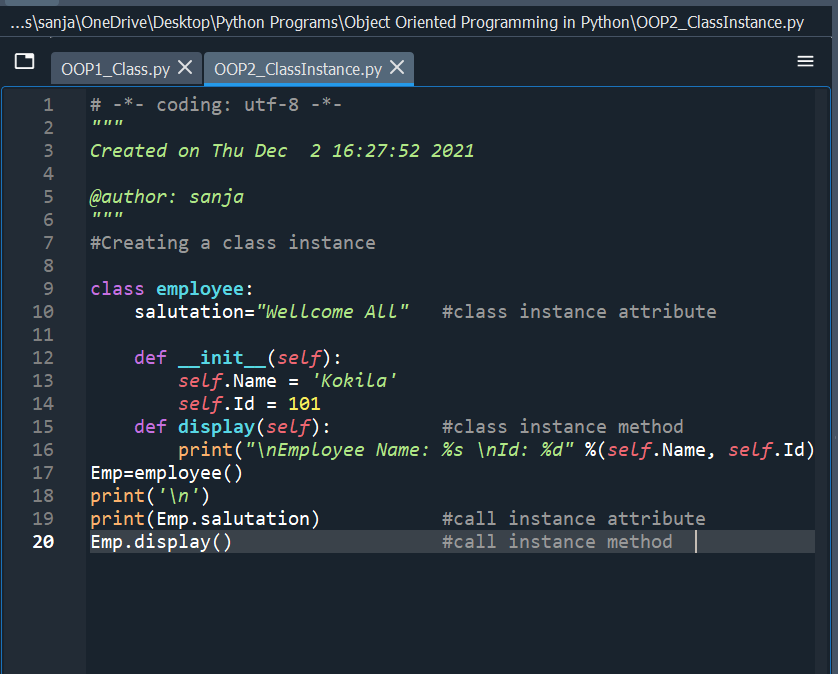
**Creating a class instance**

A **Class** is actually a blueprint or a representation of how an Object has to be instanciated. Whereas an Object is called as the **Instance of a Class**.For Eg: **Class** can be assumed as a blueprint of a car, describing how it has to look, how it has to work, so on.. Whereas the Object is the real car.

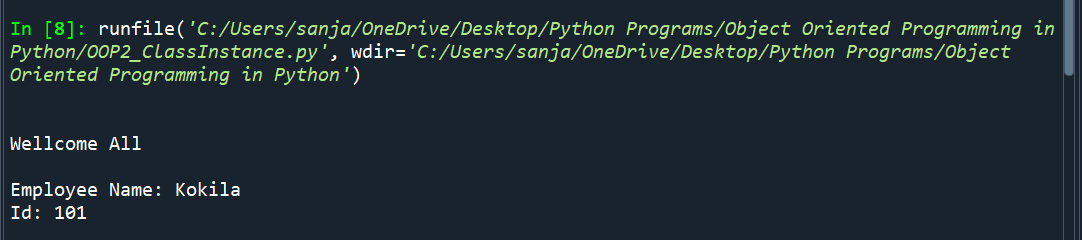
In **Python** we create **instances** in the following manner **Instance** = class(arguments)

**Program 2 :-**

**Code:**



**Output :-**



**Python Constructor**

A **constructor** is a special type of method that helps to initialize a newly created object. Depending on **\_\_init\_\_ method** we can pass any number of the arguments while creating the class object.

There are **two types of constructors** available in Python programs in order to initialize instance members of the class. Given below,

* **Non-parameterized constructer**
* **Parameterized constructor**

**A constructor can declare like "\_\_init\_\_ ()" method**. The **underscore** indicates it is a special method then followed by the **“init” is represents the acronym of initialization**.

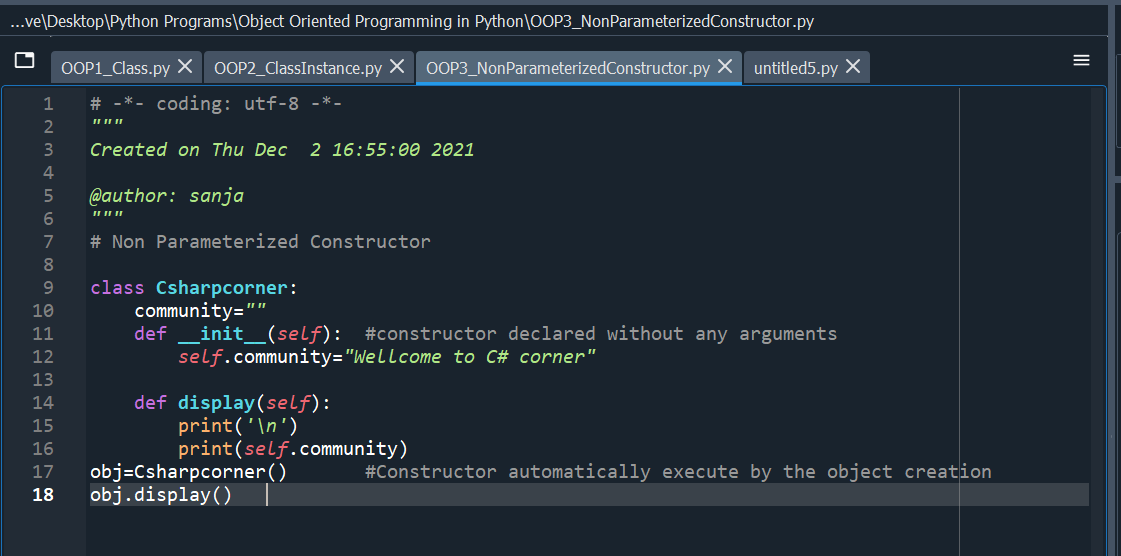
**Non-paramterized constructor:-**

**Non-parameterized constructor**, we **cannot pass any arguments** by its parameter.

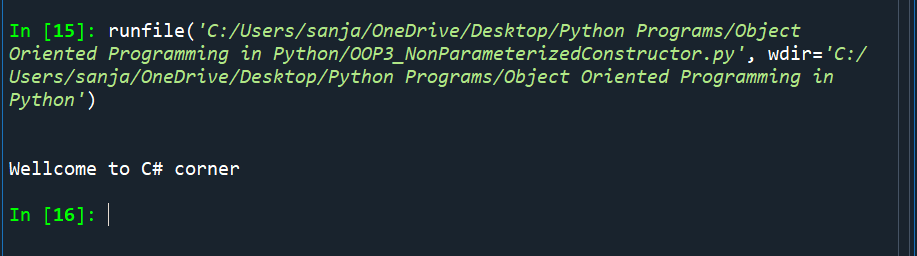
It's also known as a **simple default constructor** and has only **one argument that is a reference to the instance being constructed.**

**Program 3**

**Code:-**



**Output:-**

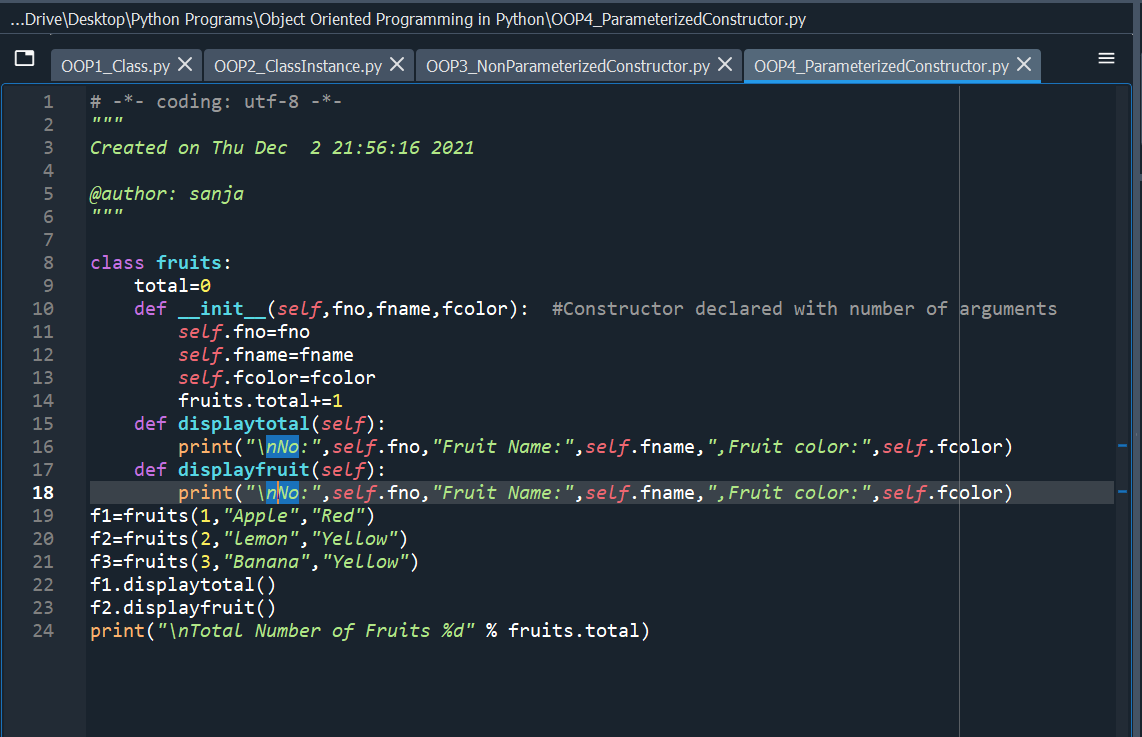


### **Parameterized constructor**

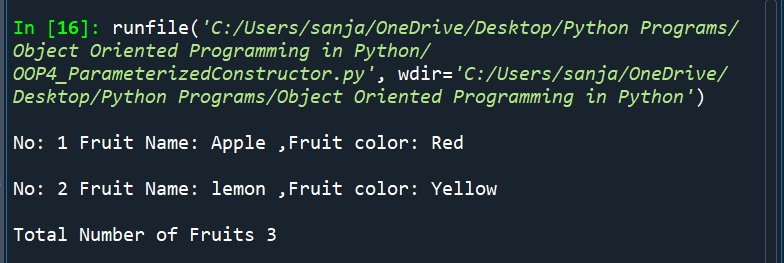
A **constructor with arguments** is known as the parameterized constructor **used for setting up properties of the object**. So, we can initialize an object with some values while declaring. In python, **the first argument considers as a reference to the instance known as the “self” keyword.**

**Program 4 :-**

**Code :-**



**Output:-**



## Objects in Python

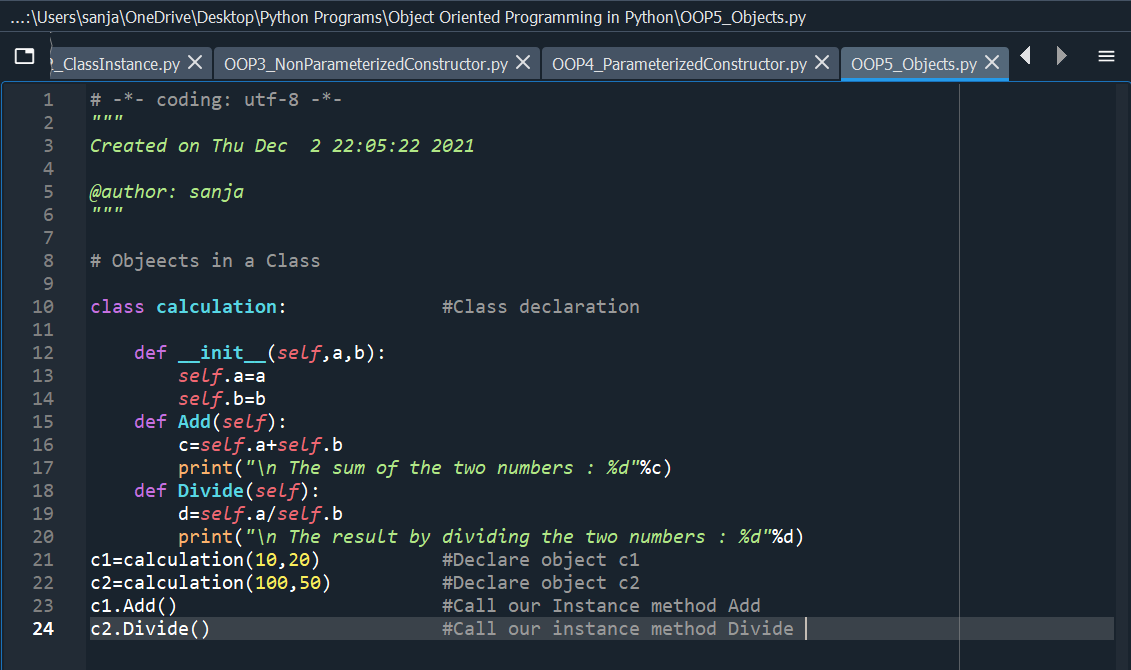
In a programming language, **objects considered as a value or variable of the Class**. It allows us to **create an instance of the class** using the class name, and it is able to **access all the methods** of the specified class like “\_\_init\_\_” method. In Python, everything consists of an Object to deal with itself.

**Syntax**

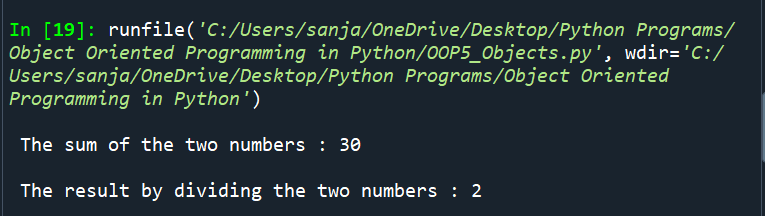
<object-name> = <class-name>(<arguments>)

**Program 5 :-**

**Code:-**



**Output:-**



**Class attributes** are everything listed in the **class Classname**: code block, including the variable declarations above the \_\_init\_\_ functions, static methods, class methods, and methods involving instances with selflink for more info: <https://docs.python.org/2/tutorial/classes.html#class-objects>  
**Instance attributes** are both:

1.) **every function listed in the class Classname**: code block so everything other than the variable declarations above the \_\_init\_\_ function &

2.) **data attributes which is when we use instance.attr** and set that equal to some value

**Example :-**

class SomeClass(object):

class\_attribute = 'This -- Defined at the class level.'

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

self.instance\_attribute = 'This -- Defined on an instance.'

def method(self):

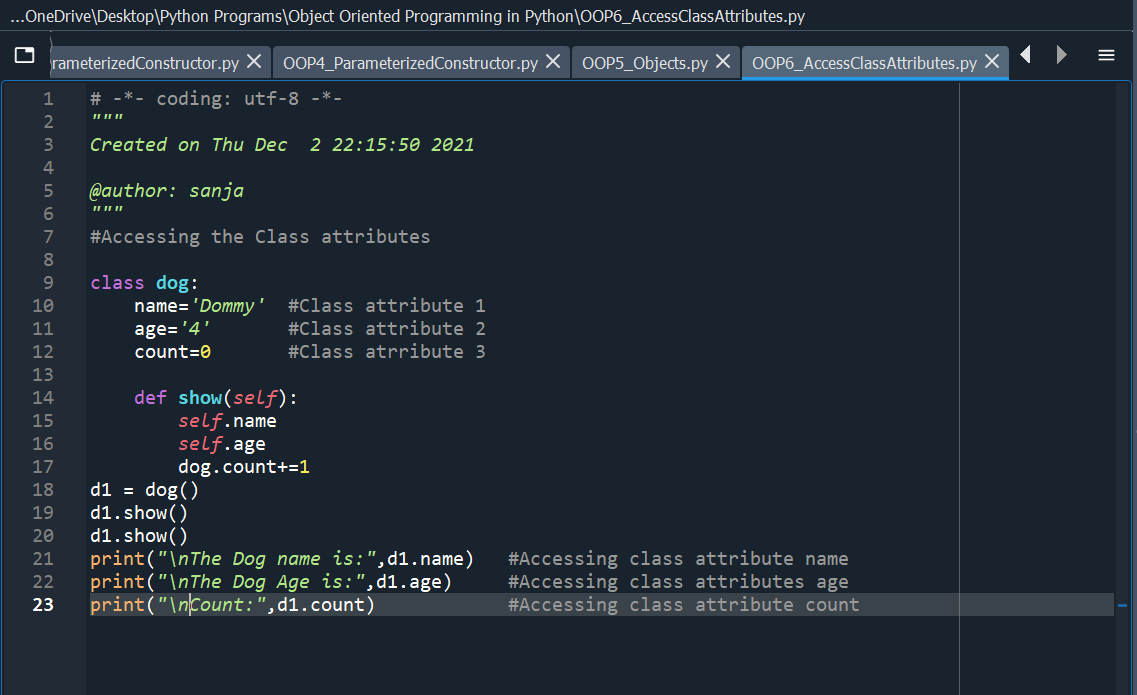
pass

instance = SomeClass()

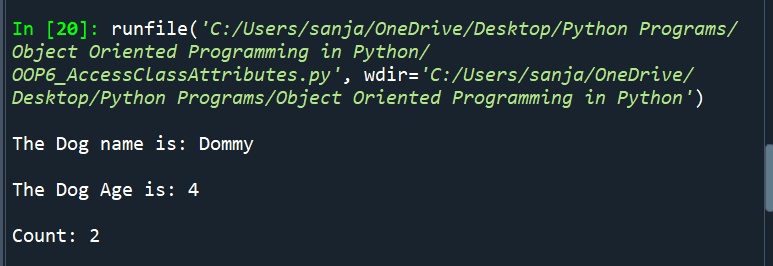
**Accessing the class Attributes**

**Program 6 :-**

**Code :-**



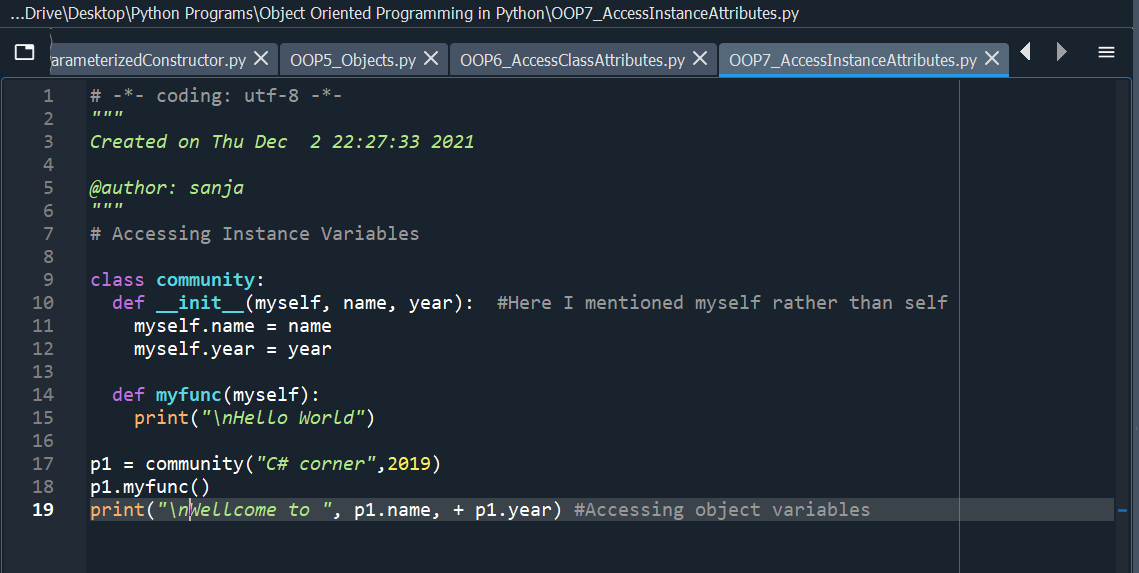
**Output :-**



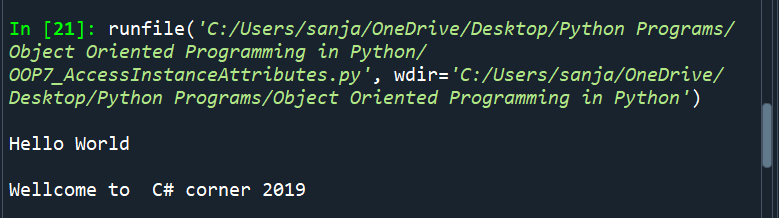
**Accessing Instance Variables**

**Program 7 :-**

**Code :-**



**Outputs:-**

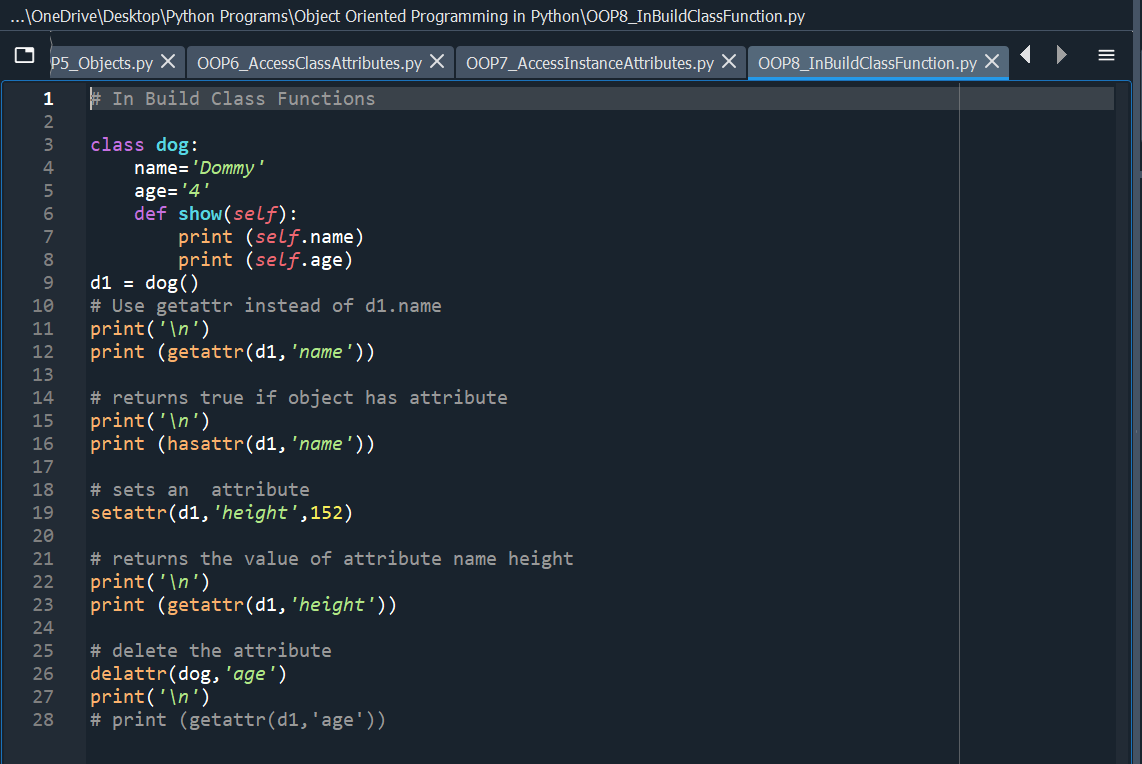


**Accessing attributes (In build class function)**

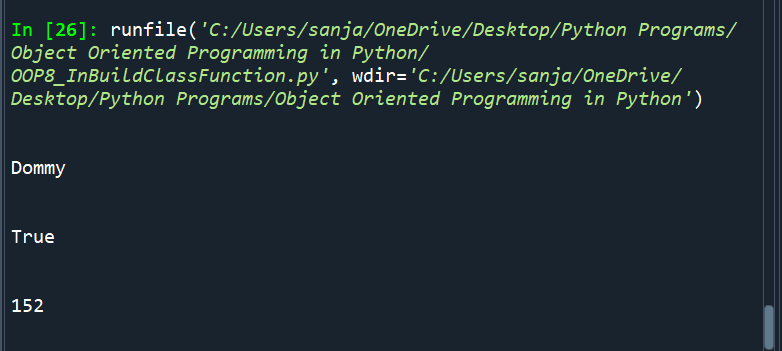
* **Getattr(obj, name, [,default])** – it allows us to access attribute of an object.
* **Setattr(obj, name) –** this method used to set an attribute. If it created when it does not exist.
* **Hasattr(obj, name) –** to check the attribute Exist or Not
* **Deleteattr(obj, name) –** Delete an attribute

**Program 8 :-**

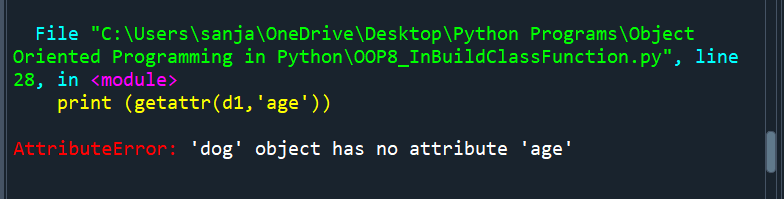
**Code :-**



**Ouput :-**



**On uncommenting line 28**



**This shows that the age attribute got deleted.**

### **Built-in class Attributes**

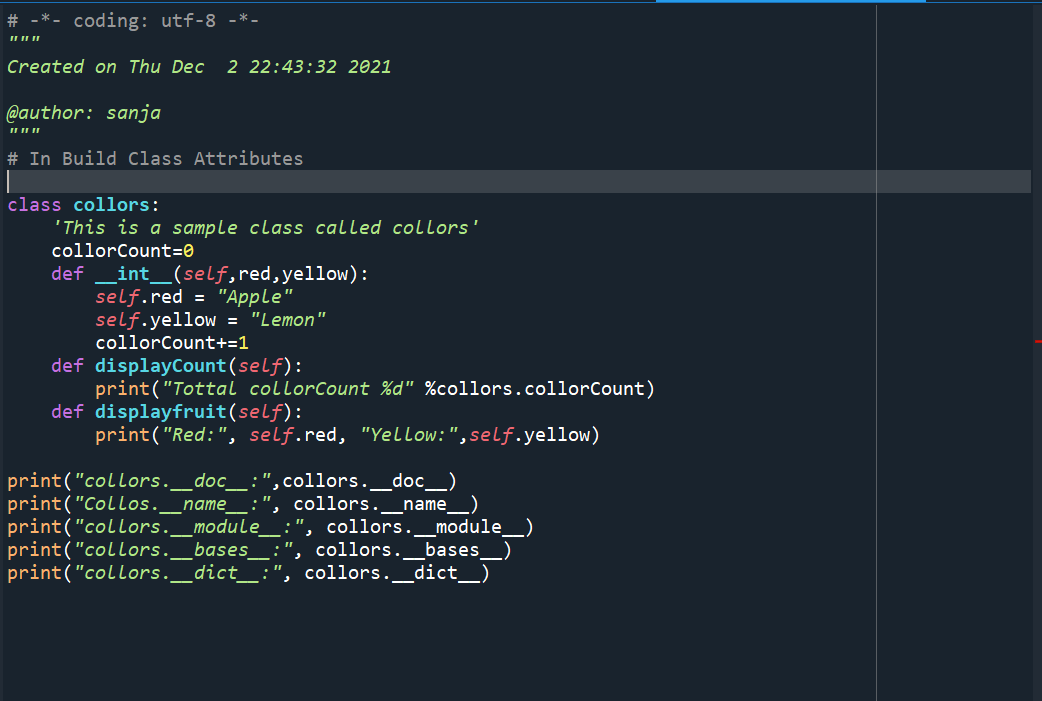
Python built-in class attributes give us some information about the class.

Every class in python can retain below an attribute and **access by using the dot operator**.

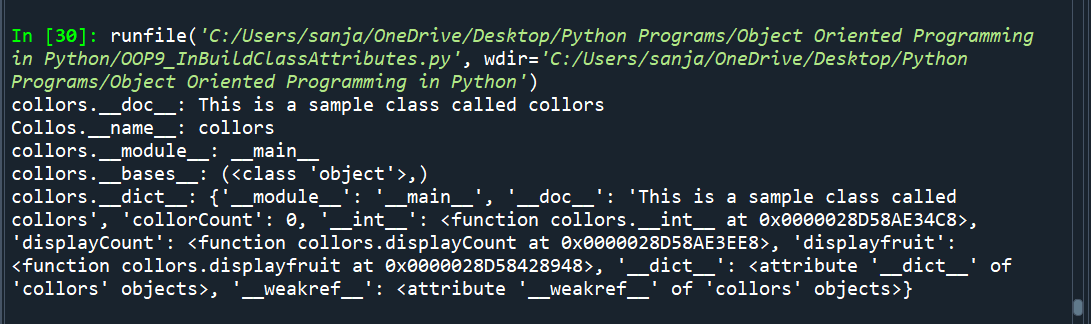
1. **\_\_dict\_\_ :**To holding the class namespace
2. **\_\_doc\_\_:** To give the class documentation string or None, if #ff0000
3. **\_\_name\_\_ :**It gives the class name
4. **\_\_module\_\_ :**Itprovide module name in which the class is defined
5. **\_\_bases\_\_ :** To give the bases of the class

**Program 9 :-**

**Code :-**



**Output :-**



**Python Method**

A **method** is a **function wrapped with a collection of statements** handled inside the body of the class. It will do some specific tasks and return a result to the requester.

**Benefits of methods**

* Code reusability
* Wrapping and protecting the code
* Determine code accessibility

**Syntax**

*Def functionname (arguments):*

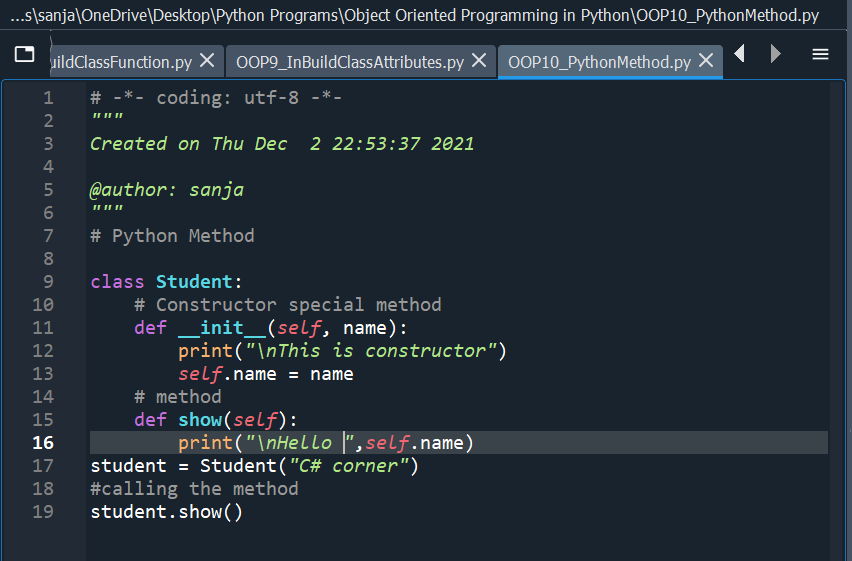
*“““Function docstring”””*

*Statements*

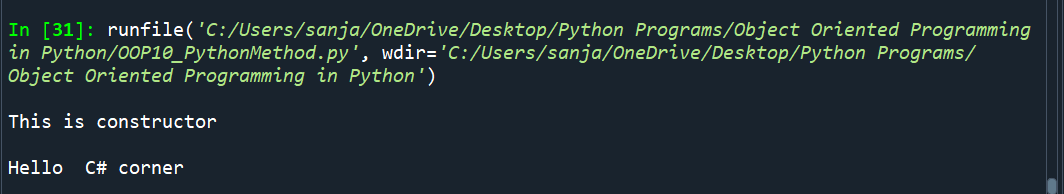
*Return[expression]*

**Program 10 :-**

**Code :-**



**Output :-**



**Inheritance :-**

**Inheritance** allows us to define a class that **inherits all the methods and properties from another class.**

**Parent class** is the class being inherited from, also called **base class**.

**Child class** is the class that inherits from another class, also called **derived class**.

Any class can be a parent class, so the syntax is the same as creating any other class.

**To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:**

**Example:-**

If the parent class is Person.

class Student(Person):  
  pass

Use the pass keyword when you do not want to add any other properties or methods to the class.

The child's \_\_init\_\_() function **overrides** the inheritance of the parent's \_\_init\_\_() function.

To keep the inheritance of the parent's **\_\_init\_\_()** function, add a call to the parent's **\_\_init\_\_()** function:

**Example:-**

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    Person.\_\_init\_\_(self, fname, lname)

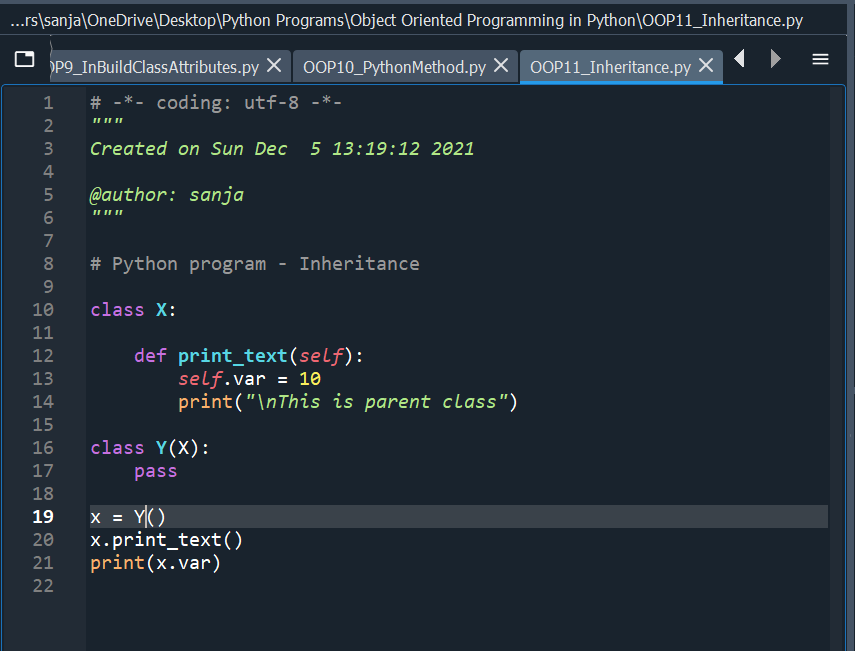
Python also has a **super()** function that **will make the child class inherit all the methods and properties from its parent:**

**Example:-**

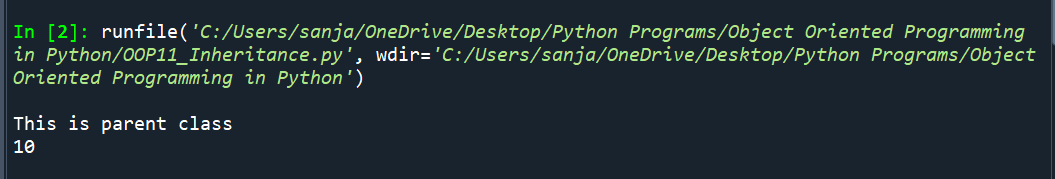
class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    super().\_\_init\_\_(fname, lname)

**Program 11 :-**

**Code :**

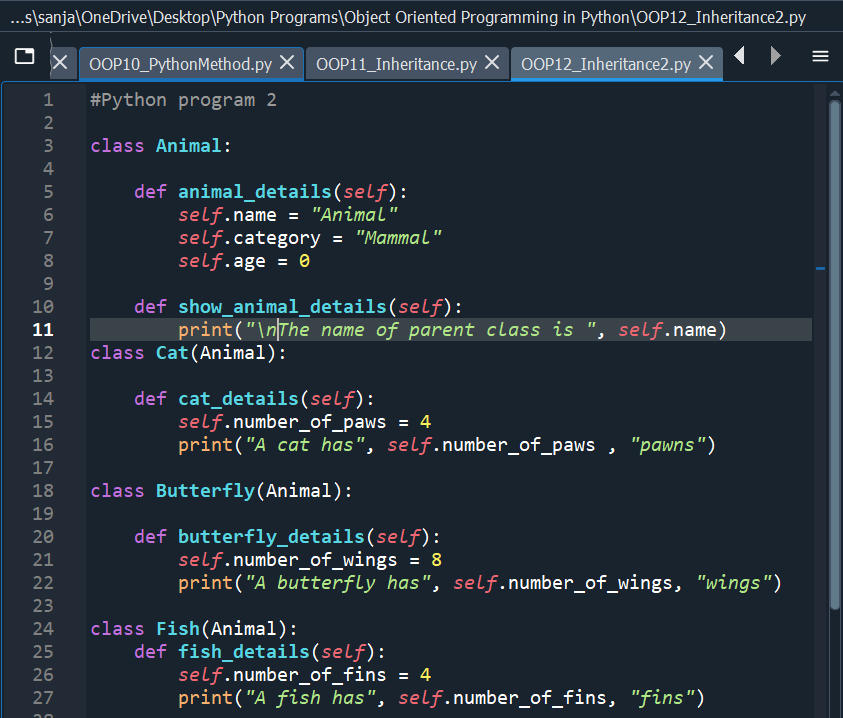
****

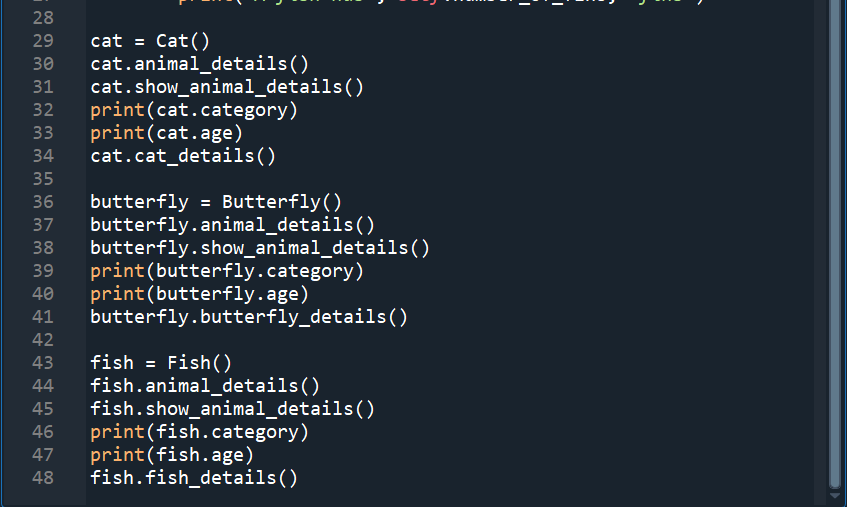
**Output :-**



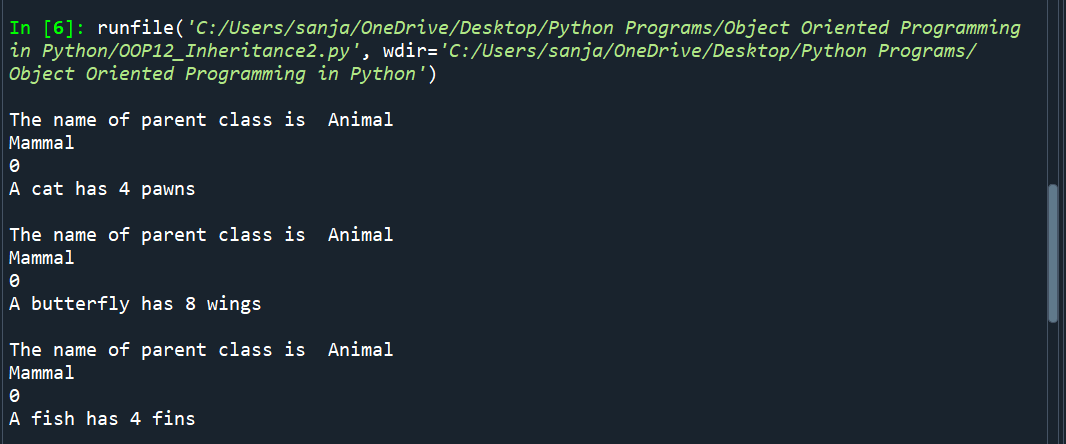
**Program 12 :-**

**Code :-**





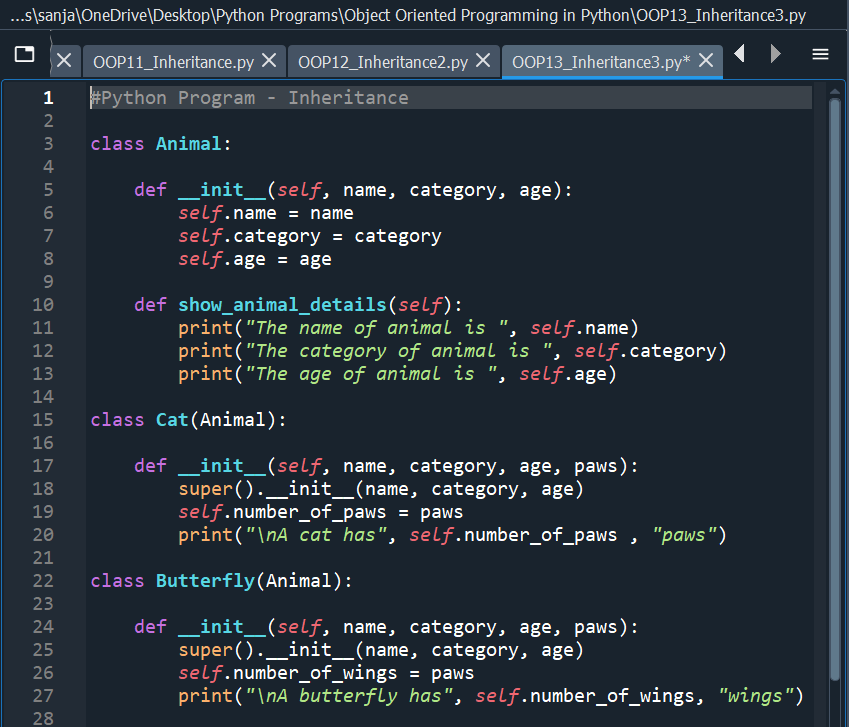
**Output :-**

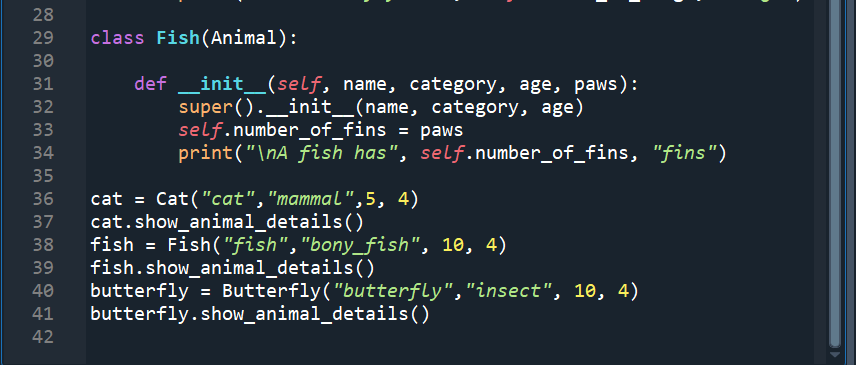


## Accessing Parent Class Constructors via Child Classes

**Program 13 :-**

**Code :-**





**Output :-**

