Python Classes

Python is an Object Oriented Programming (OOP) language. In Python, classes are used to represent a logical collection of fields and methods. Further, these classes are utilized to create objects.

In this article, learn how to create and use a Python class.

What is a class?

A class is a logical template to represent objects. In addition to functions or methods that perform related operations, a class can contain properties that represent state. Objects are instances of such classes representing one or more entities. The process of creating an object from a class is called instantiation. A single class can be instantiated several times representing several objects.

Example

Consider a class `PersonInformation` representing information of a person. `PersonInformation` class may contain personal details like `first_name`, `last_name`, `age` as fields and may include functions such as `print_data()` to print the information stored in those fields. Each person's details are stored in a single instance of the class.

Class definition

A class is created using the keyword `class`.

Syntax

```
class ClassName:

''' This is a docstring which briefly describes the class.'''

# fields

# functions

# more code
```

Note:

- Class names are written in camel case
  - **Example**: `ClassName`, `PersonInformation`, `Country`, `TechnicalWriter`.
- Fields and methods are written in snake_case.
- A class may contain blocks of code representing functions or fields.
- Its recommended to include a docstring in the beginning, which briefly describes the class's functionality.
- You can also create a class without any statements. However, such a class is of less use.

Sample

```
class PersonInformation

# List of statements
```

A class creates a new local `namespace` where all its attributes are defined.

Defining class attributes

In a class, class attributes are fields defined outside the methods. They are shared across instances of the class in the same process.

Sample

```
class PersonInformation:

  first_name = "Alex"
  last_name = "Sa"
  age = 10
```

Special attributes

There can also be special attributes that begin with double underscores `__`.

Example

- `__doc__` gives docstring of the class.
- `__init__()` is the constructor of the class.
- `__str__()` return human-readable representation of the class.

Creating an object

Once a class is defined, you can create an object by instantiating it. Object creation is similar to a function call.

Syntax

```
objectName = ClassName()
```

This creates a new object instance, `objectName`, which can be used to access the attributes.

Sample

```
person_information_object = PersonInformation()
print(PersonInformation.first_name)
```

Output

```
Alex
```

Defining class methods

Methods are functions you can define to access, modify and perform operations on these attributes.

Syntax

```
class ClassName:
''' This is a docstring which briefly describes the class.'''

# variables

def __init__(self, argument 1, ..., argument n):
  # more code

def function_name(self, argument 1, ..., argument n):
  # more code
```

`self` parameter

- The `self` parameter is the first parameter of any instance-level method. It's name is written in snake case by convention.
- Its a reference to the current instance of the class and is used to access instance attributes or methods. It can also be used to refer to class-level attributes or methods.

`__init__()` function

All classes have a built-in function `__int__()` that automatically executes when a class is initiated. You can use this function to assign values to class attributes at runtime. This type of function is known as a constructor in OOP.

Let's use the above sample code with `__init__()` and a method.

Sample - Try this

```
class PersonInformation:
  ''' This is a class with personal information such as first name, last name, and
age.'''

  # Define an init method with variables to obtain values at runtime.
  def __init__(self, first_name, last_name, age):
    self.first_name = first_name
    self.last_name = last_name
    self.age = age

  # Define a method to display the values.
  def print_data(self):
    print("Person information:")
    print(f'Name: {self.first_name} {self.last_name}')
    print(f'Age: {self.age}')

 # Intializing an object.
person_information = PersonInformation("Alex", "Sa", 10)

person_information.print_data()
```

Output

```
Person information:
Name: Alex Sa
Age: 10
```

Conclusion

In this article, you can learn the basic concepts of classes in Python. You can also view code samples to write a class containing methods.

Stay tuned for more informative articles.