30 Days Of Python: Day 21 - Classes and Objects

Classes and Objects

Python is an object oriented programming language. Everything in Python is an object, with its properties and methods. A number, string, list, dictionary, tuple, set etc. used in a program is an object of a corresponding built-in class. We create class to create an object. A class is like an object constructor, or a "blueprint" for creating objects. We instantiate a class to create an object. The class defines attributes and the behavior of the object, while the object, on the other hand, represents the class.

We have been working with classes and objects right from the beginning of this challenge unknowingly. Every element in a Python program is an object of a class. Let us check if everything in python is a class:

```
$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> num = 10
>>> type(num)
<class 'int'>
>>> string = 'string'
>>> type(string)
<class 'str'>
>>> boolean = True
>>> type(boolean)
```

```
<class 'bool'>
>>> lst = []
>>> type(lst)
<class 'list'>
>>> tpl = ()
>>> type(tpl)
<class 'tuple'>
>>> set1 = set()
>>> type(set1)
<class 'set'>
>>> dct = {}
>>> type(dct)
<class 'dict'>
```

Creating a Class

To create a class we need the key word **class** followed by the name and colon. Class name should be **CamelCase**.

```
# syntax
class ClassName:
  code goes here
```

Example:

```
class Person:
  pass
print(Person)
```

```
<__main__.Person object at 0x10804e510>
```

Creating an Object

We can create an object by calling the class.

```
p = Person()
print(p)
```

Class Constructor

In the examples above, we have created an object from the Person class. However, a class without a constructor is not really useful in real applications. Let us use constructor function to make our class more useful. Like the constructor function in Java or JavaScript, Python has also a built-in **init**() constructor function.
The **init** constructor function has self parameter which is a reference to the current instance of the class **Examples:**

```
class Person:
      def __init__ (self, name):
        # self allows to attach parameter to the class
          self.name =name

p = Person('Suniksha')
```

```
print(p.name)
print(p)

# output
Suniksha
<__main__.Person object at 0x2abf46907e80>
```

Let us add more parameters to the constructor function.

```
class Person:
      def __init__(self, firstname, lastname, age, country, city):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city


p = Person('Suniksha', 'Patel', 20, 'Finland', 'Helsinki')
print(p.firstname)
print(p.lastname)
print(p.age)
print(p.country)
print(p.city)

# output
Suniksha
Patel
20
Finland
Helsinki
```

Object Methods

Objects can have methods. The methods are functions which belong to the object.

Example:

```
class Person:
      def __init__(self, firstname, lastname, age, country, city):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city
      def person_info(self):
        return f'{self.firstname} {self.lastname} is {self.age} years old.
He lives in {self.city}, {self.country}'

p = Person('Suniksha', 'Patel', 20, 'Finland', 'Helsinki')
print(p.person_info())

# output
Suniksha Patel is 20 years old. He lives in Helsinki, Finland
```

Object Default Methods

Sometimes, you may want to have a default values for your object methods. If we give default values for the parameters in the constructor, we can avoid errors when we call or instantiate our class without parameters. Let's see how it looks:

Example:

```
class Person:
      def __init__(self, firstname='Suniksha', lastname='Patel', age=20,
country='Finland', city='Helsinki'):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city

      def person_info(self):
        return f'{self.firstname} {self.lastname} is {self.age} years old.
He lives in {self.city}, {self.country}.'

p1 = Person()
print(p1.person_info())
p2 = Person('John', 'Doe', 30, 'Nomanland', 'Noman city')
print(p2.person_info())
```

```
# output
Suniksha Patel is 20 years old. He lives in Helsinki, Finland.
John Doe is 30 years old. He lives in Noman city, Nomanland.
```

Method to Modify Class Default Values

In the example below, the person class, all the constructor parameters have default values. In addition to that, we have skills parameter, which we can access using a method. Let us create add_skill method to add skills to the skills list.

```
class Person:
      def __init__(self, firstname='Suniksha', lastname='Patel', age=20,
country='Finland', city='Helsinki'):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city
          self.skills = []

      def person_info(self):
        return f'{self.firstname} {self.lastname} is {self.age} years old.
He lives in {self.city}, {self.country}.'
      def add_skill(self, skill):
          self.skills.append(skill)

p1 = Person()
print(p1.person_info())
p1.add_skill('HTML')
p1.add_skill('CSS')
p1.add_skill('JavaScript')
p2 = Person('John', 'Doe', 30, 'Nomanland', 'Noman city')
print(p2.person_info())
```

```
print(p1.skills)
print(p2.skills)

# output
Suniksha Patel is 20 years old. He lives in Helsinki, Finland.
John Doe is 30 years old. He lives in Noman city, Nomanland.
['HTML', 'CSS', 'JavaScript']
[]
```

Inheritance

Using inheritance we can reuse parent class code. Inheritance allows us to define a class that inherits all the methods and properties from parent class. The parent class or super or base class is the class which gives all the methods and properties. Child class is the class that inherits from another or parent class. Let us create a student class by inheriting from person class.

```
class Student(Person):
    pass


s1 = Student('Eyob', 'Yetayeh', 30, 'Finland', 'Helsinki')
s2 = Student('Lidiya', 'Teklemariam', 28, 'Finland', 'Espoo')
print(s1.person_info())
s1.add_skill('JavaScript')
s1.add_skill('React')
s1.add_skill('Python')
print(s1.skills)

print(s2.person_info())
s2.add_skill('Organizing')
s2.add_skill('Marketing')
s2.add_skill('Digital Marketing')
print(s2.skills)

output
Eyob Yetayeh is 30 years old. He lives in Helsinki, Finland.
['JavaScript', 'React', 'Python']
Lidiya Teklemariam is 28 years old. He lives in Espoo, Finland.
['Organizing', 'Marketing', 'Digital Marketing']
```

We did not call the **init**() constructor in the child class. If we didn't call it then we can still access all the properties from the parent. But if we do call the constructor we can access the parent properties by calling *super*.
We can add a new method to the child or we can override the parent class methods by creating the same method name in the child class. When we add the **init**() function, the child class will no longer inherit the parent's **init**() function.

Overriding parent method

```
class Student(Person):
    def __init__ (self, firstname='Suniksha', lastname='Patel',age=20,
country='Finland', city='Helsinki', gender='male'):
        self.gender = gender
        super().__init__(firstname, lastname,age, country, city)
```

```
    def person_info(self):
        gender = 'He' if self.gender =='male' else 'She'
        return f'{self.firstname} {self.lastname} is {self.age} years old.
{gender} lives in {self.city}, {self.country}.'

s1 = Student('Eyob', 'Yetayeh', 30, 'Finland', 'Helsinki','male')
s2 = Student('Lidiya', 'Teklemariam', 28, 'Finland', 'Espoo', 'female')
print(s1.person_info())
s1.add_skill('JavaScript')
s1.add_skill('React')
s1.add_skill('Python')
print(s1.skills)

print(s2.person_info())
s2.add_skill('Organizing')
s2.add_skill('Marketing')
s2.add_skill('Digital Marketing')
print(s2.skills)
```

```
Eyob Yetayeh is 30 years old. He lives in Helsinki, Finland.
['JavaScript', 'React', 'Python']
Lidiya Teklemariam is 28 years old. She lives in Espoo, Finland.
['Organizing', 'Marketing', 'Digital Marketing']
```

We can use super() built-in function or the parent name Person to automatically inherit the methods and properties from its parent. In the example above we override the parent method. The child method has a different feature, it can identify, if the gender is male or female and assign the proper pronoun(He/She).

😍 Now, you are fully charged with a super power of programming. Now do some exercises for your brain and muscles.

💻 Exercises: Day 21

Exercises: Level 1

1. Python has the module called *statistics* and we can use this module to do all the statistical calculations. However, to learn how to make function and reuse function let us try to develop a program, which calculates the measure of central tendency of a sample (mean, median, mode) and measure of variability (range, variance, standard deviation). In addition to those measures, find the min, max, count, percentile, and frequency distribution of the sample. You can create a class called Statistics and create all the functions that do statistical calculations as methods for the Statistics class. Check the output below.

```
ages = [31, 26, 34, 37, 27, 26, 32, 32, 26, 27, 27, 24, 32, 33, 27, 25, 26,
38, 37, 31, 34, 24, 33, 29, 26]

print('Count:', data.count()) # 25
print('Sum: ', data.sum()) # 744
print('Min: ', data.min()) # 24
print('Max: ', data.max()) # 38
```

```
print('Range: ', data.range() # 14
print('Mean: ', data.mean()) # 30
print('Median: ', data.median()) # 29
print('Mode: ', data.mode()) # {'mode': 26, 'count': 5}
print('Standard Deviation: ', data.std()) # 4.2
print('Variance: ', data.var()) # 17.5
print('Frequency Distribution: ', data.freq_dist()) # [(20.0, 26), (16.0,
27), (12.0, 32), (8.0, 37), (8.0, 34), (8.0, 33), (8.0, 31), (8.0, 24),
(4.0, 38), (4.0, 29), (4.0, 25)]

# you output should look like this
print(data.describe())
Count: 25
Sum:  744
Min:  24
Max:  38
Range:  14
Mean:  30
Median:  29
Mode:  (26, 5)
Variance:  17.5
Standard Deviation:  4.2
Frequency Distribution: [(20.0, 26), (16.0, 27), (12.0, 32), (8.0, 37),
(8.0, 34), (8.0, 33), (8.0, 31), (8.0, 24), (4.0, 38), (4.0, 29), (4.0,
25)]
```

Exercises: Level 2

1. Create a class called PersonAccount. It has firstname, lastname, incomes, expenses properties and it has total_income, total_expense, account_info, add_income, add_expense and account_balance methods. Incomes is a set of incomes and its description. The same goes for expenses.

🎉 CONGRATULATIONS ! 🎉