Python Classes orObjects

June 5, 2024

0.1 Python Classes/Objects

Dr. Labeed Al-Saad

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class:

To create a class, use the keyword class:

Example:

Create a class named MyClass, with a property named x:

[1]:
```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
```

```
5
```

0.2 The init() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

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:

0.3 Note: The init() function is called automatically every time the class is being used to create a new object.

Example:

Create a class named Person, use the **init**() function to assign values for name and age:

[12]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)


print(p1.name)
print(p1.age)
```

```
John
36
```

0.4 The str() Function

The **str**() function controls what should be returned when the class object is represented as a string.

If the **str**() function is not set, the string representation of the object is returned:

Example:

The string representation of an object WITHOUT the **str**() function:

[13]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```

```
<__main__.Person object at 0x00000205ECBB6010>
```

Example:

The string representation of an object WITH the **str**() function:

[15]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}: {self.age}"

p1 = Person("John", 36)

print(p1)
```

```
John: 36
```

0.5 Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example:

Insert a function that prints a greeting, and execute it on the p1 object:

[18]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

0.6 Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

0.7 The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example:

Use the words mysillyobject and abc instead of self:

[19]:
```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

0.8 Modify Object Properties

You can modify properties on objects like this:

Example:

Set the age of p1 to 40:

[23]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
print(p1.name, p1.age)
p1.age = 40

print("the age is apdated to", p1.age)
```

```
John 36
the age is apdated to 40
```

0.9 Delete Object Properties

You can delete properties on objects by using the del keyword:

Example:

Delete the age property from the p1 object:

[27]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1.age

print(p1.age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[27], line 13
      9 p1 = Person("John", 36)
```

```
     11 del p1.age
---> 13 print(p1.age)

AttributeError: 'Person' object has no attribute 'age'
```

[ ]:
```
Delete Objects
--
You can delete objects by using the del keyword:

Example:

Delete the p1 object:
```

[28]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1

print(p1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[28], line 13
      9 p1 = Person("John", 36)
     11 del p1
---> 13 print(p1)

NameError: name 'p1' is not defined
```

0.10 The pass Statement

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

Example:

[29]:
```
class Person:
  pass
```