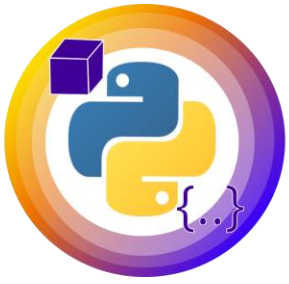


Python OOP: Instances and Instance Attributes

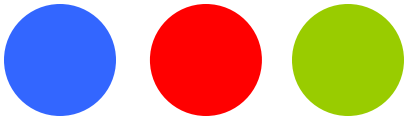
Instances

Key Takeaways

- **Instances**
  - They are **concrete** representations or example of the abstract objects that classes describe.
  - They are **created from a class** that acts like a "blueprint". Classes determine the attributes and functionality of their instances.
  - You can assign custom or predefined values for their attributes. These values are assigned in the constructor __init__(), a method that runs when an object is created.
  - They share the same "categories" of attributes, but the attributes can have different values. Changing the value of an attribute for one instance doesn't affect the other instances.

Classes act like "blueprints" to create instances

✓ For example: a class could have a "color" attribute. All the instances of that class would have this attribute, but the values can be different for each instance. One instance could have the value "blue" and another one the value "red".



Instances

Key Takeaways

- **General Syntax to Create an Instance**

```
<variable> = <ClassName>(<arguments>)
```

- **Example**

Class Name | Arguments

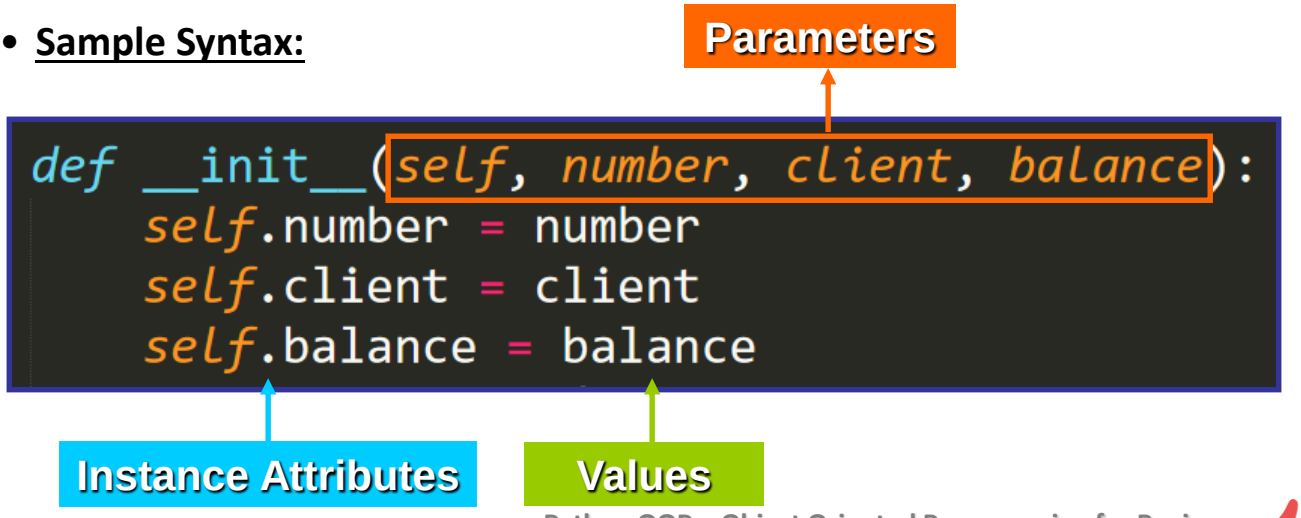
```
my_account = BankAccount("5621", "Gino Navone", 33424.4)
```

```
class BankAccount:

    accounts_created = 0

    def __init__(self, number, client, balance):
        self.number = number
        self.client = client
        self.balance = balance
        BankAccount.accounts_created += 1

    def display_balance(self):
        print(self.balance)
```

self is not included in the list of arguments

Key Takeaways

- **Constructor __init__():**
  - This is a reserved method,
  - Also called the **"constructor"** of the class.
  - It's called when an object (instance) is created.
  - Syntax:
    - ✓ Keyword "def".
    - ✓ __init__() with double underscore.
    - ✓ Within parentheses:
      - Parameter "self".
      - The formal parameters separated by a comma and a space after the comma.
    - ✓ A colon (:) at the end of the line.
- **Sample Syntax:**



Instance Attributes

Key Takeaways

- **Instance Attributes**
  - They **belong to the instances**. These attributes are relevant to the context of the program.
    - ✓ For example: bank accounts have an owner, a balance, and a number. These could be instance attributes in a BankAccount class.
  - Their **values are independent**. They are **not shared** across instances. Each instance has its own individual copy of the attribute.
  - To pass custom values for these attributes, you need to add them as parameters in the constructor __init__() and assign them using self.<attribute> = <value>
  - You can access and modify the values of these attributes after the instance has been created.
  - Changing the value of an attribute for one instance doesn't affect the value of the other instances of the class.
  - You can also set fixed values for these attributes when the object is created by assigning a value in __init__().

Instance Attributes

Key Takeaways

- **General Syntax to Assign a Value to an Instance Attribute**

```
self.<instance_attribute> = <value>
```

- **Example**

```
class BankAccount:

    accounts_created = 0

    def __init__(self, number, client, balance):
        self.number = number
        self.client = client
        self.balance = balance
        BankAccount.accounts_created

    def display_balance(self):
        print(self.balance)
```

Could be fixed values

```
self.balance = 10000
```

Instance Attributes

Key Takeaways

- **General Syntax to Get the Value of an Instance Attribute**

```
self.<instance_attribute>
```

- **Example**

```
class BankAccount:

    accounts_created = 0

    def __init__(self, number, client, balance):
        self.number = number
        self.client = client
        self.balance = balance
        BankAccount.accounts_created += 1

    def display_balance(self):
        print(self.balance)
```

We can get and use the current value

Instance Attributes

Key Takeaways

- **Instance Attributes – Default Values**
  - You can set default values for the parameters in the class constructor __init__() using <parameter>=<value>
  - If the arguments are omitted, the default arguments are used and they can be assigned to the instance attributes.
  - Warning: The parameters that have default arguments have to be located at the end of the list of parameters. Otherwise, an error will be thrown.

```
class BankAccount:

    accounts_created = 0

    def __init__(self, number, client, balance=1034.4):
        self.number = number
        self.client = client
        self.balance = balance
        BankAccount.accounts_created += 1

    def display_balance(self):
        print(self.balance)
```

Default value for balance

No argument = default value

```
myAccount = BankAccount("5621", "Gino Navone")
```