Artificial Intelligence Diploma

Python Session 7

NeuroTech

Agenda

NEUROTECH

Object Oriented Programming

NeuroTech

Object Oriented Programming

- OOP stands for Object-Oriented Programming. It's a programming paradigm that's based on the concept of "objects," which are instances of classes. OOP focuses on organizing code in a way that models real-world entities, their attributes, and their interactions.

- In OOP, a "class" is a blueprint for creating objects. It defines the structure and behavior of objects. An "object" is an instance of a class, and it encapsulates both data (attributes or properties) and functions (methods) that operate on that data.

Advantages of OOP ?

Object-Oriented Programming (OOP) offers numerous advantages that contribute to better code organization, reusability, and maintainability. Here are some key advantages of OOP:

1. **Modularity and Reusability:**
    1. OOP encourages breaking down complex systems into smaller, manageable modules (classes and objects).
    2. Modules can be reused in different parts of the application or in other applications, saving development time and effort.
2. **Code Organization:**
    1. OOP promotes a structured way of organizing code by encapsulating data and behavior within objects.
    2. The clear separation of concerns enhances code readability and comprehension.

Advantages of OOP ?

3. Ease of Maintenance:

1. The modular nature of OOP makes it easier to locate, update, and debug specific parts of the code without affecting the entire system.
2. Changes in one part of the code are less likely to lead to unintended consequences in other parts.

4. Flexibility and Extensibility:

1. OOP allows for extending existing classes to create new ones through inheritance.
2. New functionality can be added to a system without altering the existing codebase, promoting code extensibility.

5. Efficient Problem Solving:

1. OOP models real-world entities and interactions, making the code more intuitive and closely aligned with the problem domain.
2. This approach simplifies problem-solving and system design.

Advantages of OOP ?

6. Abstraction and Encapsulation:

1. Abstraction allows focusing on what an object does rather than how it does it, simplifying complex systems.
2. Encapsulation protects the integrity of data by restricting direct access and ensuring controlled interaction with objects.

7. Hierarchical Organization:

1. OOP supports creating hierarchies of classes, which models relationships between objects.
2. Hierarchies mimic real-world structures, making the code easier to understand and navigate.

8. Polymorphism and Flexibility:

1. Polymorphism allows objects of different classes to be treated uniformly through a common interface.
2. This enhances code flexibility by enabling the interchangeability of objects.

6.Design Patterns:

1. OOP supports the use of design patterns, which are proven solutions to recurring programming problems.
2. Design patterns promote standardized and effective approaches to coding.

Advantages of OOP ?

10. Effective Collaboration:

1. OOP encourages dividing a project into modules, making it easier for multiple programmers to work on different parts of the code simultaneously.
2. Well-defined interfaces and contracts ensure smoother collaboration.

10. Real-World Modeling:

1. OOP models entities and their interactions as objects, making the code more intuitive and closely mirroring the real world.
2. This approach improves communication between developers and stakeholders.

11. Encapsulation of State:

1. Objects encapsulate their state (data), preventing unintended interference from external sources.
2. This helps maintain data integrity and prevents unexpected changes.

12. Security and Control:

1. Encapsulation restricts direct access to object data, providing controlled access through methods.
2. This prevents unauthorized manipulation of data and maintains system security.

13. Easier Testing and Debugging:

1. Isolated objects and modules are easier to test and debug.
2. Defects can be localized and fixed without affecting the entire application.

Class and Object

OOP

- In Object-Oriented Programming (OOP), a **class** is a blueprint or template for creating objects. It defines the structure and behavior that its objects will have. A class encapsulates data (attributes) and methods (functions) that operate on that data, allowing you to create multiple instances (objects) of the same type.

- Think of a class as a blueprint for a specific type of object, like a cookie cutter that defines the shape and features of cookies. Objects are the actual cookies you create using that cutter.

- In Object-Oriented Programming (OOP), an **object** is a concrete instance of a class. It's a fundamental unit that represents a specific entity or data structure, following the blueprint defined by its corresponding class. Objects are created from classes and have attributes (data) and methods (functions) associated with them.
- Think of a class as a template or blueprint, and an object as a realization of that template, carrying the specific values and behaviors defined in the class.

OOP

In the context of Object-Oriented Programming (OOP), within a class, **variables** and **methods** play crucial roles in defining the structure and behavior of objects. Let's delve into what these terms mean within a class:

Variables in Class :

- **Variables** in a class are often referred to as **attributes**, **properties**, or **fields**. They represent data that is associated with the class and its objects. These variables define the state of an object and store specific information about that object. Each instance (object) of the class can have its own unique set of attribute values.

- Attributes can hold various types of data, including integers, strings, floats, lists, and even other objects. Attributes define the characteristics of an object, and they encapsulate the data that the object needs to maintain.

OOP

Methods in Class (Functions):

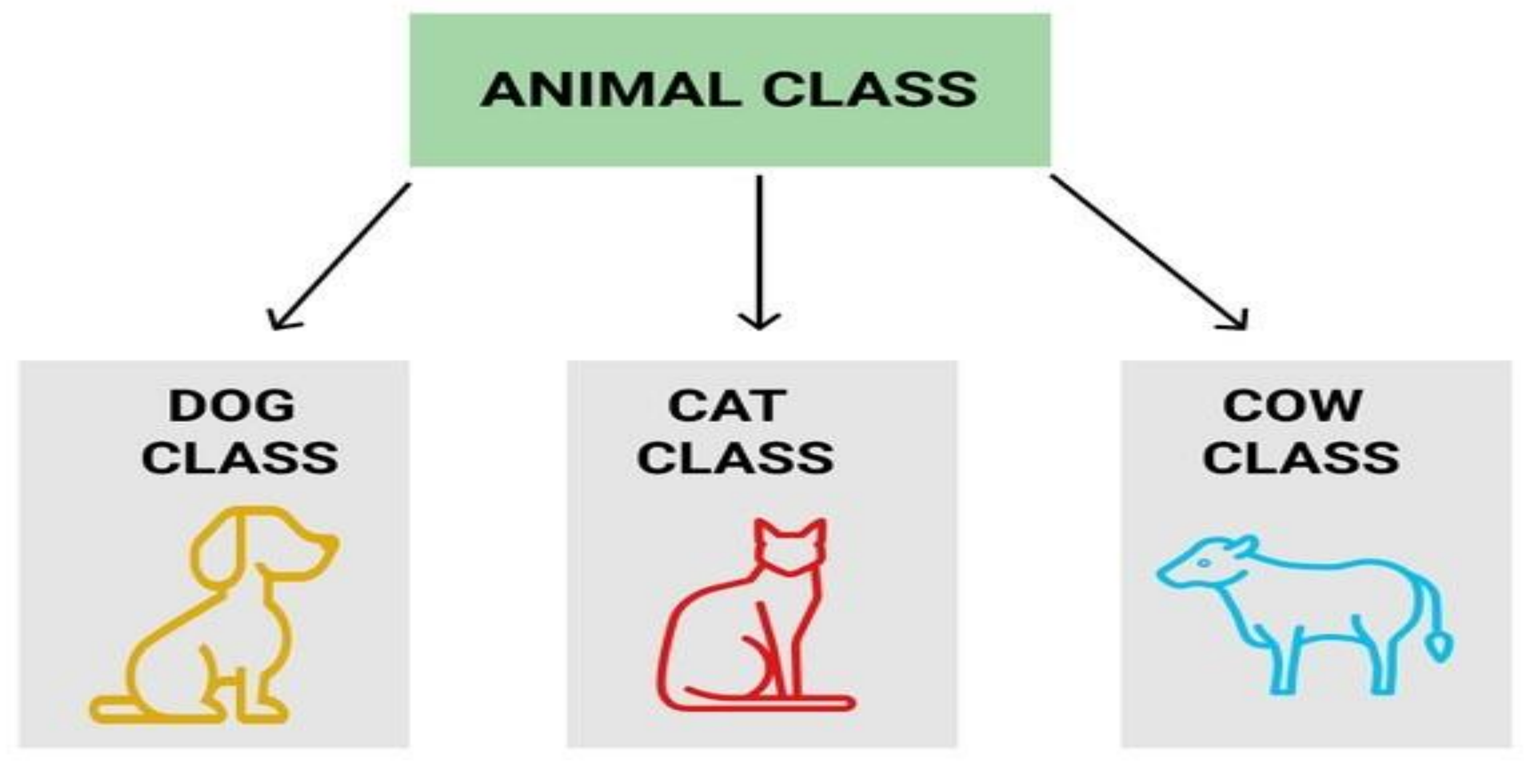
- **Methods** in a class are functions that define the actions or behaviors that objects of the class can perform. Methods operate on the attributes of the class and can interact with other objects, enabling the objects to perform specific tasks or computations.

- Methods can access and modify the attributes of an object. They encapsulate the operations that are associated with the object's behavior and provide a way to interact with the object's data.

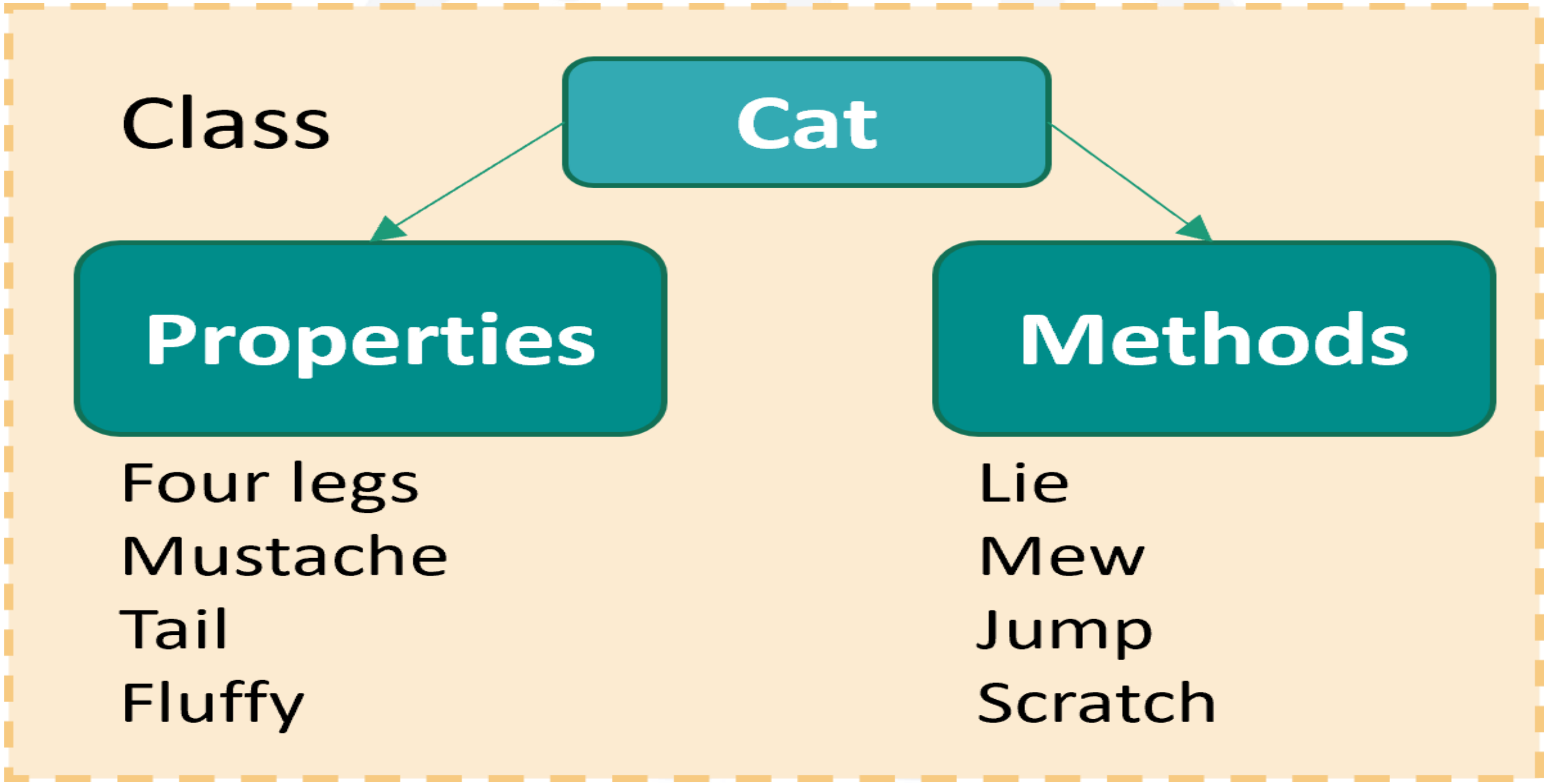
Relationship between Variables and Methods:

- Attributes (variables) define the state of an object, while methods define the behavior of that object. Methods operate on the attributes, allowing objects to perform actions or provide specific functionalities based on their state.

Example for Class

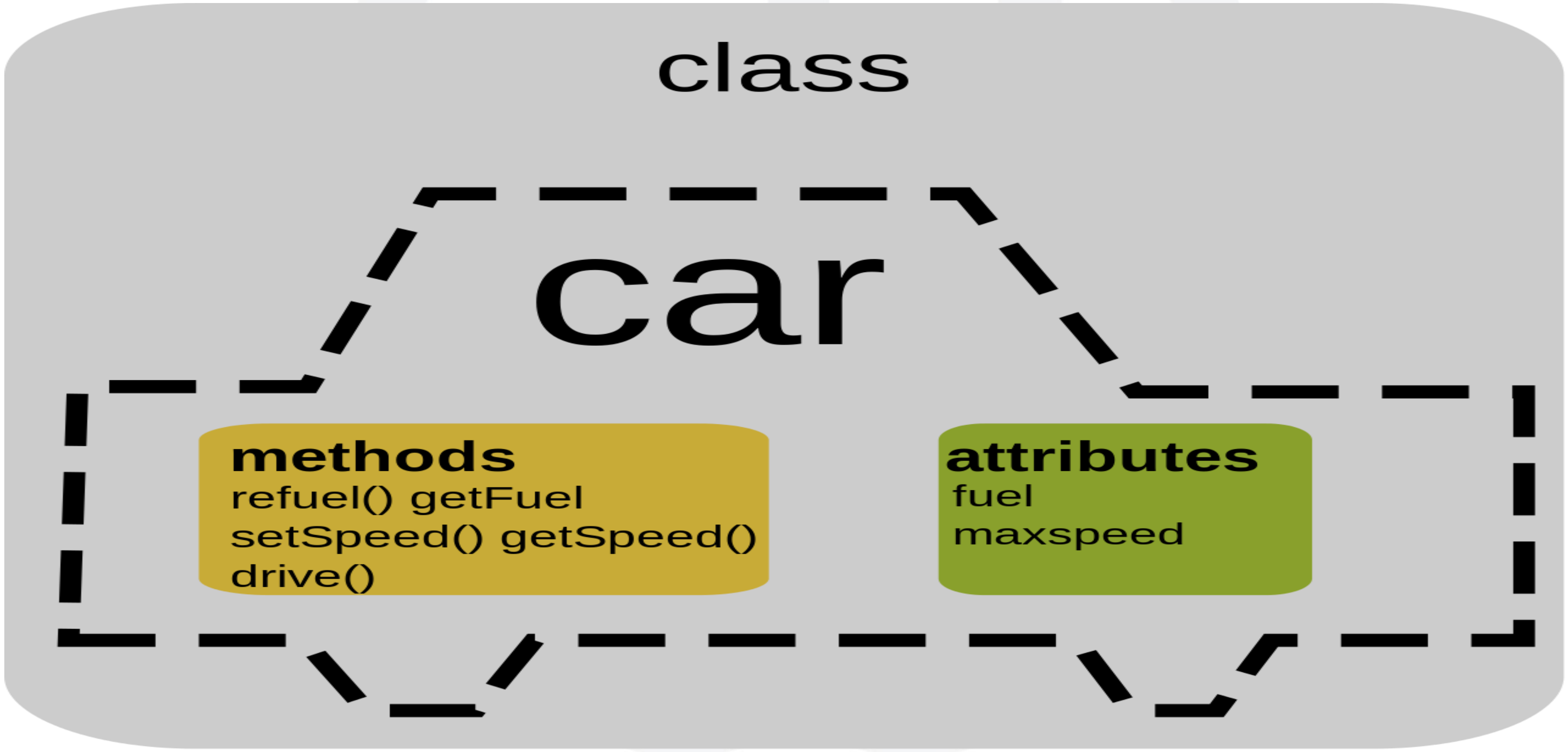


OOP



NeuroTech

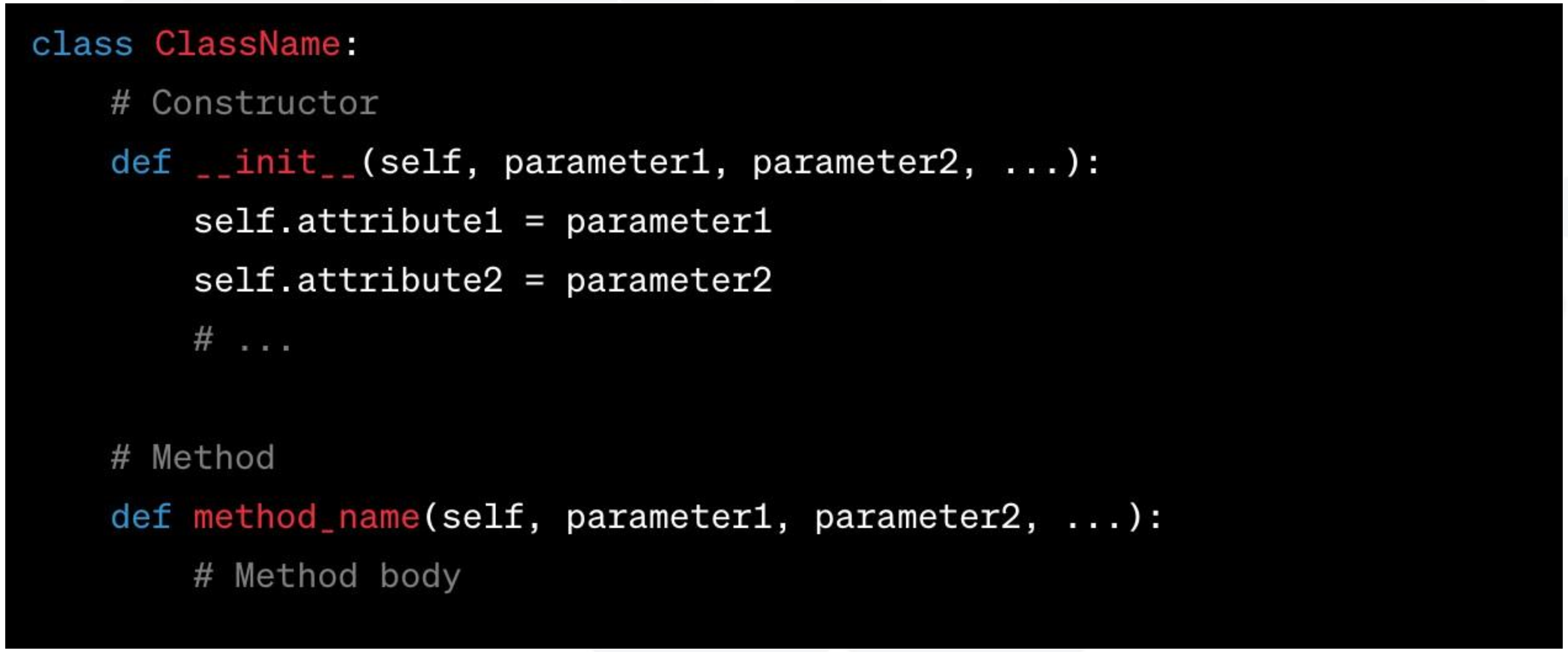
OOP



NEUROTECH

OOP

Creating a class in Python is a fundamental step in object-oriented programming. Here's how you can create a class along with its attributes and methods:



```python
class ClassName:
    # Constructor
    def __init__(self, parameter1, parameter2, ...):
        self.attribute1 = parameter1
        self.attribute2 = parameter2
        # ...

    # Method
    def method_name(self, parameter1, parameter2, ...):
        # Method body
```

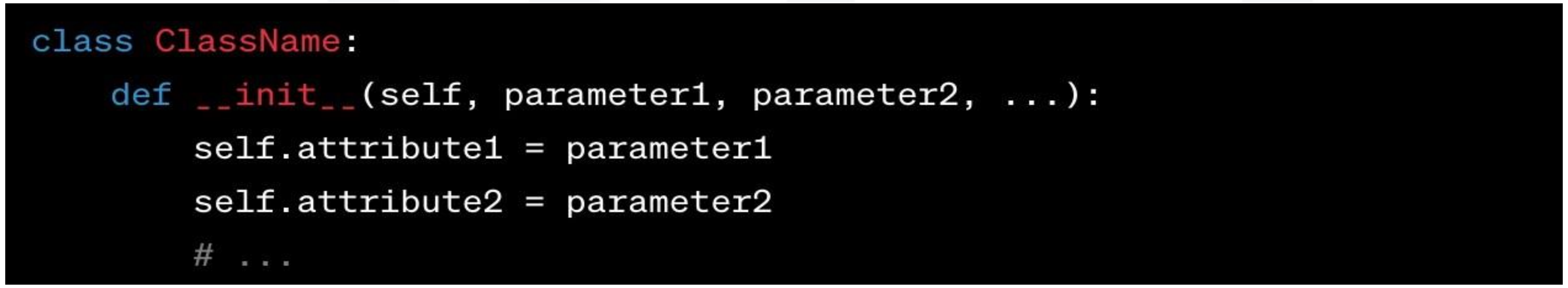
NeuroTech

OOP

- In Object-Oriented Programming (OOP), a **constructor** is a special method within a class that is automatically called when an object of that class is created. It is used to initialize the attributes of the object with the values provided during object creation. Constructors ensure that the object is properly set up and ready for use.

- In Python, the constructor method is named__init__, and it is defined within the class. The__init __ method takes at least one parameter, conventionally named self, which refers to the object being created. Additional parameters can be provided to initialize the attributes of the object.

- Here's the basic syntax of a constructor in Python:



```
class ClassName:
    def __init__(self, parameter1, parameter2, ...):
        self.attribute1 = parameter1
        self.attribute2 = parameter2
        # ...
```

OOP

Let's break down the key concepts of a constructor:

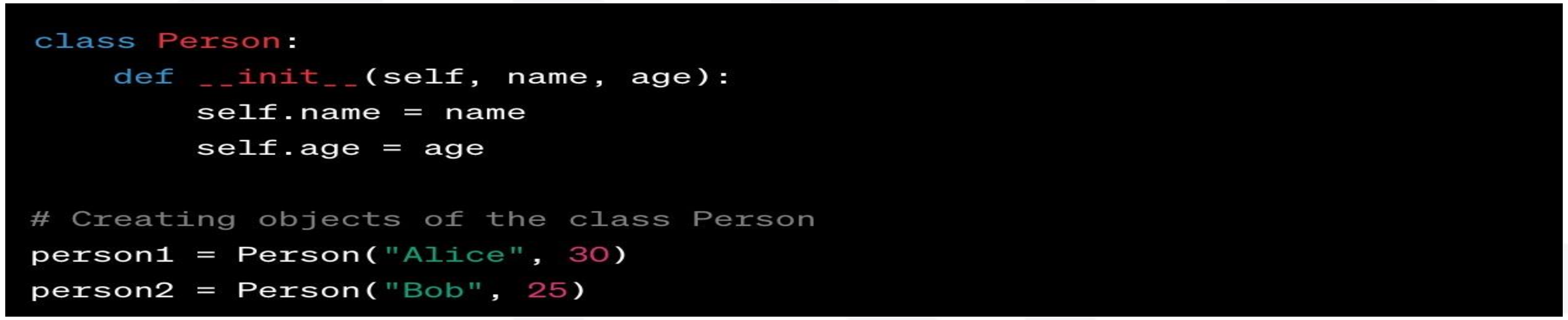
- **__init__Method:** The___init__method is a special method in Python. It is automatically invoked when an object is created from the class.

- **self Parameter:** The first parameter of the___init__method is self, which refers to the instance of the object being created. By convention, the name self is used, but you can use any valid variable name.

- **Attributes Initialization:** Inside the___init__method, you set the initial values of the object's attributes using the self.attribute_name = value syntax.

OOP

- Here's an example of a class with a constructor:



```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# Creating objects of the class Person
person1 = Person("Alice", 30)
person2 = Person("Bob", 25)
```

- In this example, the___init__method in the Person class initializes the name and age attributes of the objects being created. When you create person1 and person2, the constructor is automatically called to set their initial attributes.

- Constructors are essential for ensuring that objects are created in a valid state with the necessary attributes initialized. They help maintain the integrity of the object's data and promote consistent object creation throughout the program.

NEUROTECH

Attributes in Class

NeuroTech

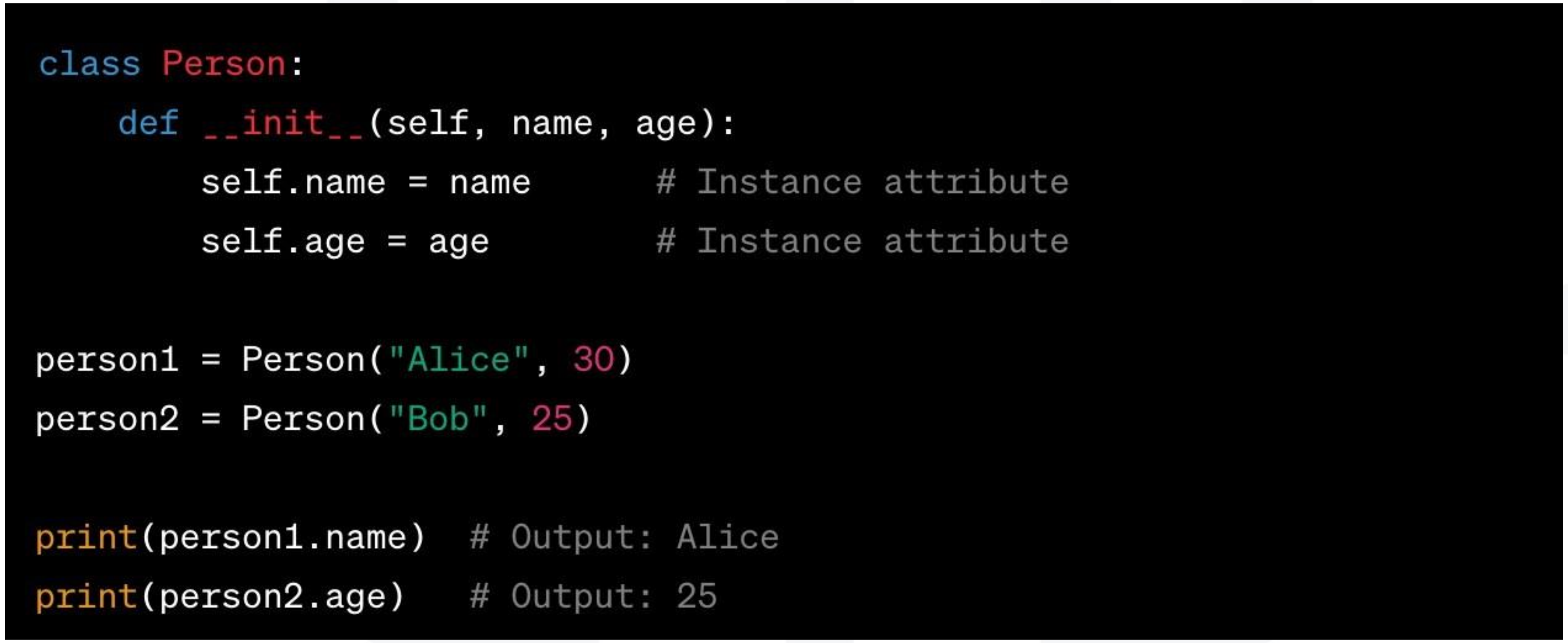
Instance Attribute Vs Class Attribute

In Object-Oriented Programming (OOP), both **instance attributes** and **class attributes** are used to store data within a class. However, they have different scopes and purposes. Let's explore the differences between these two types of attributes:

Instance Attributes:

1. **Scope:**
   1. Instance attributes are specific to each individual object (instance) of a class. Each object can have its own set of instance attributes.
2. **Initialization:**
   1. Instance attributes are typically initialized within the constructor (___init___) of the class. Each object can have different values for these attributes.
3. **Access:**
   1. Instance attributes are accessed and modified using the dot notation (object.attribute). Each object accesses its own instance attributes.
4. **Use Cases:**
   1. Instance attributes represent the unique state or characteristics of individual objects.
   2. They hold data that varies from one object to another.

Instance Attribute Vs Class Attribute



```python
class Person:
    def __init__(self, name, age):
        self.name = name      # Instance attribute
        self.age = age        # Instance attribute

person1 = Person("Alice", 30)
person2 = Person("Bob", 25)

print(person1.name)  # Output: Alice
print(person2.age)   # Output: 25
```

NEUROTECH

Instance Attribute Vs Class Attribute

Class Attributes:

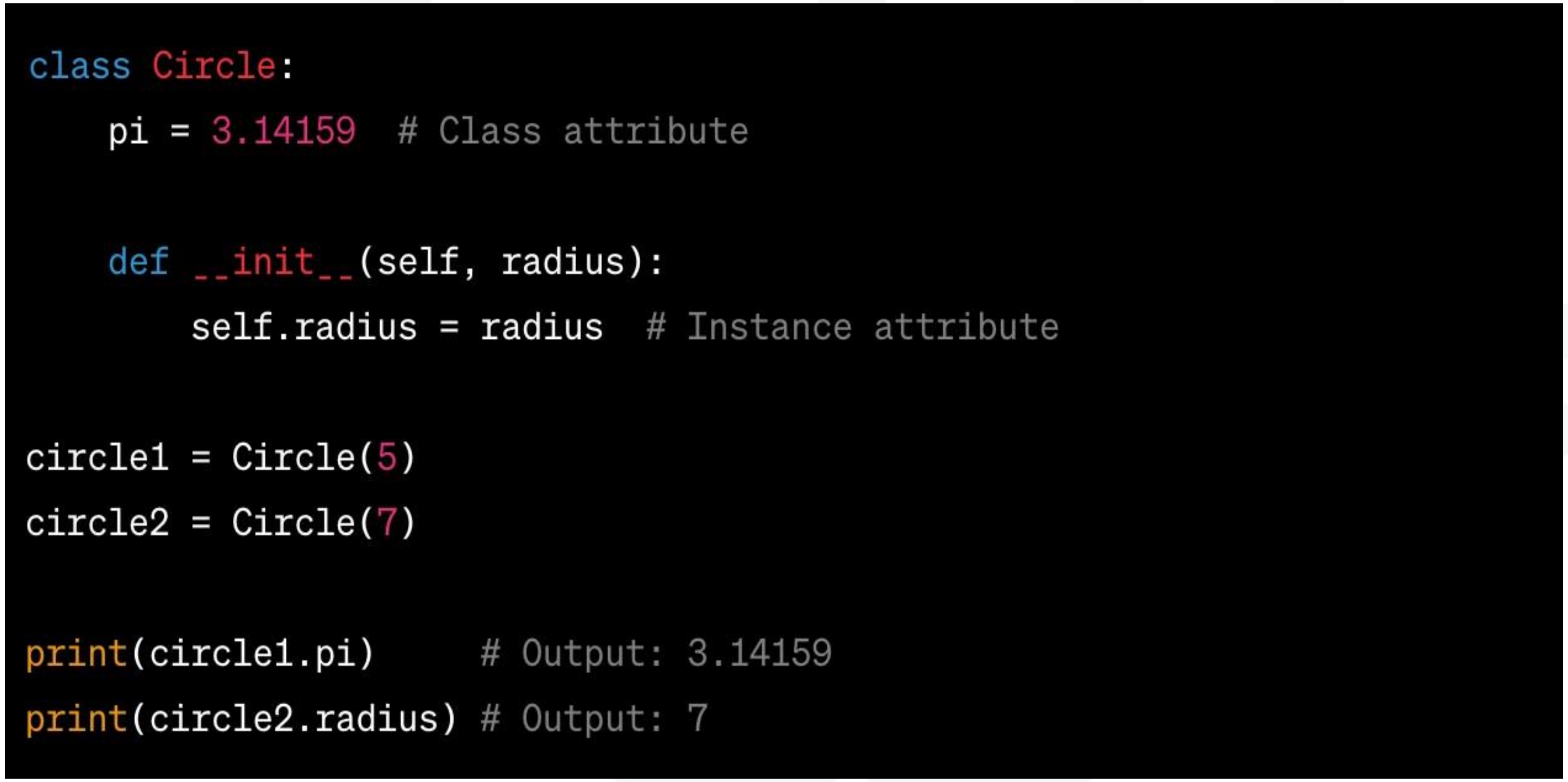
1. **Scope:**
    1. Class attributes are shared among all instances of a class. They are the same for every object created from that class.

2. **Initialization:**
    1. Class attributes are defined outside any methods of the class. They are typically initialized at the class level.

3. **Access:**
    1. Class attributes are accessed using the class name or an instance of the class (ClassName.attribute or object.attribute). All instances share the same value for a class attribute.

4. **Use Cases:**
    1. Class attributes represent properties or characteristics that are common to all objects of a class.
    2. They hold data that is constant or shared across instances.

Instance Attribute Vs Class Attribute



```python
class Circle:
    pi = 3.14159  # Class attribute

    def __init__(self, radius):
        self.radius = radius  # Instance attribute

circle1 = Circle(5)
circle2 = Circle(7)

print(circle1.pi)     # Output: 3.14159
print(circle2.radius) # Output: 7
```

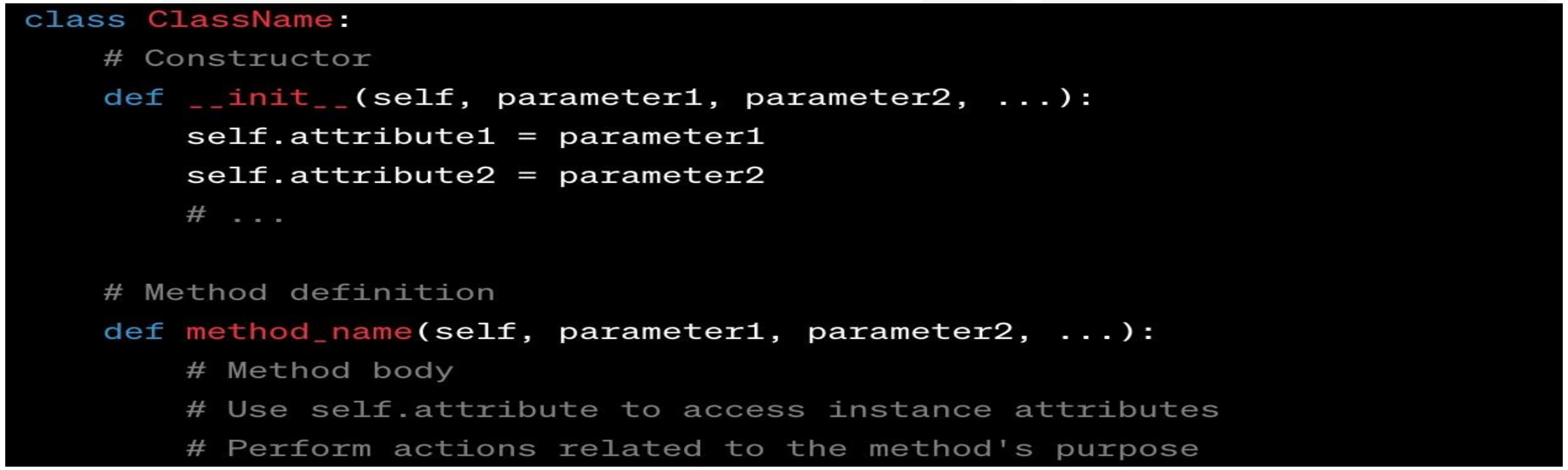
NEUROTECH

Methods in Class

OOP

- In Object-Oriented Programming (OOP), **methods** in a class are functions that define the behaviors and actions that objects of that class can perform. Methods are associated with the class and its instances (objects) and operate on the attributes of those instances. They encapsulate the operations that objects can perform and provide a way to interact with the object's data.

- Defining a method in a class involves creating a function within the class's code block. Methods in Python classes are functions that operate on the attributes of the class or perform specific actions related to the class's purpose. Here's how you define a method in a class:



```
class ClassName:
    # Constructor
    def __init__(self, parameter1, parameter2, ...):
        self.attribute1 = parameter1
        self.attribute2 = parameter2
        # ...

    # Method definition
    def method_name(self, parameter1, parameter2, ...):
        # Method body
        # Use self.attribute to access instance attributes
        # Perform actions related to the method's purpose
```

NEUROTECH

OOP

Let's break down the steps to define a method in a class:

1. **Class Definition:** Begin by defining the class using the class keyword and the desired class name.
2. **Constructor (__init__):** If needed, define the class constructor (__init__) to initialize the instance attributes. The constructor is not a method but a special method to set up the object's initial state.
3. **Method Definition:** Within the class block, define a method by using the def keyword, followed by the method name. You can use any valid function name.
4. **Parameters:** Like regular functions, methods can accept parameters. The first parameter is conventionally named self, which refers to the instance of the object the method is called on. Other parameters can be added as needed.
5. **Method Body:** Inside the method, write the code that defines the behavior of the method. This code can involve operations on the instance attributes, calculations, conditionals, loops, and more.
6. **Accessing Attributes:** To access instance attributes within the method, use the self keyword followed by the attribute name (e.g., self.attribute_name).
7. **Method Purpose:** Write the code that performs the specific action or behavior the method is meant to achieve. This can involve modifying attributes, interacting with other objects, or simply providing information.

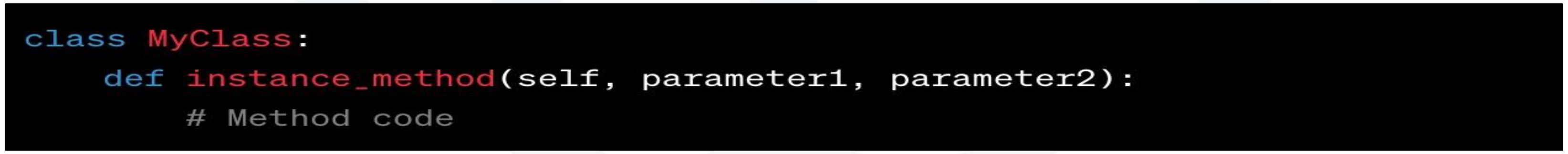
NeuroTech

Instance Methods vs Class Methods vs Static Methods

- In Python's Object-Oriented Programming (OOP), there are three types of methods you can define within a class: **instance methods**, **class methods**, and **static methods**. Each type of method serves a different purpose and has different ways of interacting with class attributes and instances. Let's explore the differences:
- **Instance Methods:**

➢ **Instance methods** are the most common type of methods in classes. They operate on instance-specific data (attributes) and can access and modify the object's state. These methods take the instance itself as their first parameter (usually named self), allowing them to interact with the specific instance they are called on.

❑ To define an instance method, use the def keyword within the class definition and include the self parameter.



```python
class MyClass:
    def instance_method(self, parameter1, parameter2):
        # Method code
```

❑ Instance methods can access instance attributes and other instance methods.

❑ They are called using the dot notation on an instance of the class: object.instance_method(parameter1, parameter2).

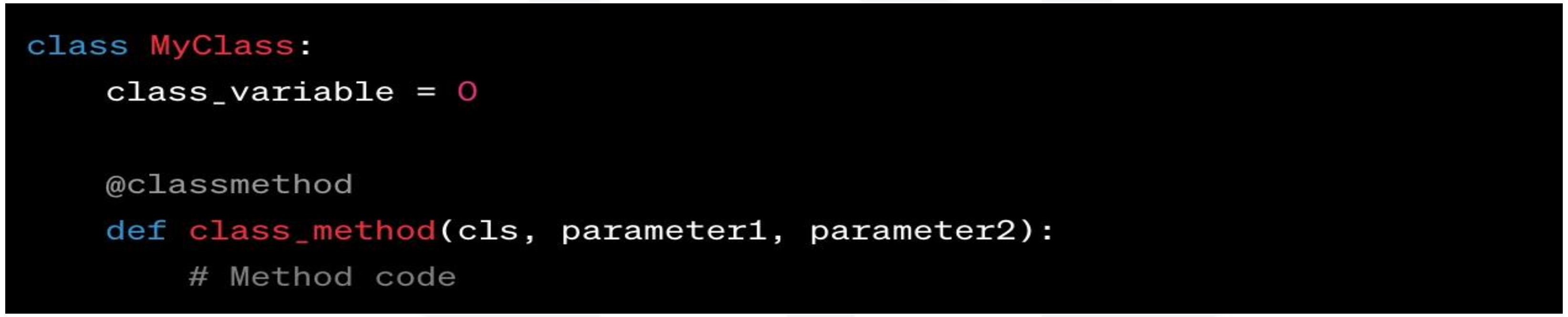
NEUROTECH

Instance Methods vs Class Methods vs Static Methods

Class Methods:

Class methods are methods that are bound to the class itself rather than to instances. They can access and modify class-level attributes and perform operations that are relevant to the entire class. Class methods are defined using the @classmethod decorator and take the class as their first parameter (usually named cls).

- ❑ To define a class method, use the @classmethod decorator before the method definition.



```
class MyClass:
    class_variable = 0

    @classmethod
    def class_method(cls, parameter1, parameter2):
        # Method code
```

- ❑ Class methods can access and modify class attributes and perform class-wide operations.
- ❑ They are called using the dot notation on the class itself: MyClass.class_method(parameter1, parameter2).

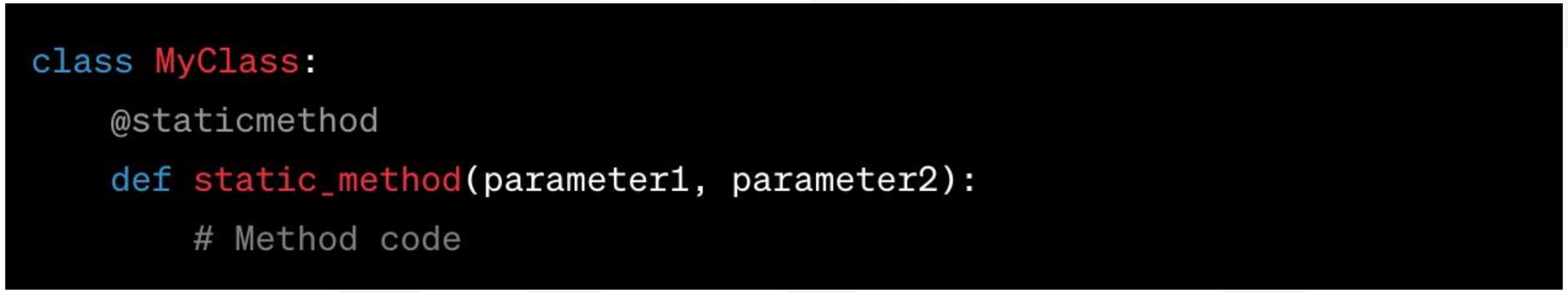
NeuroTech

Instance Methods vs Class Methods vs Static Methods

Static Methods:

Static methods are methods that are not bound to class attributes or instances. They are utility methods that don't depend on instance-specific or class-specific data. Static methods are defined using the @staticmethod decorator and do not require any special parameters (like self or cls).

- ❑ To define a static method, use the @staticmethod decorator before the method definition.



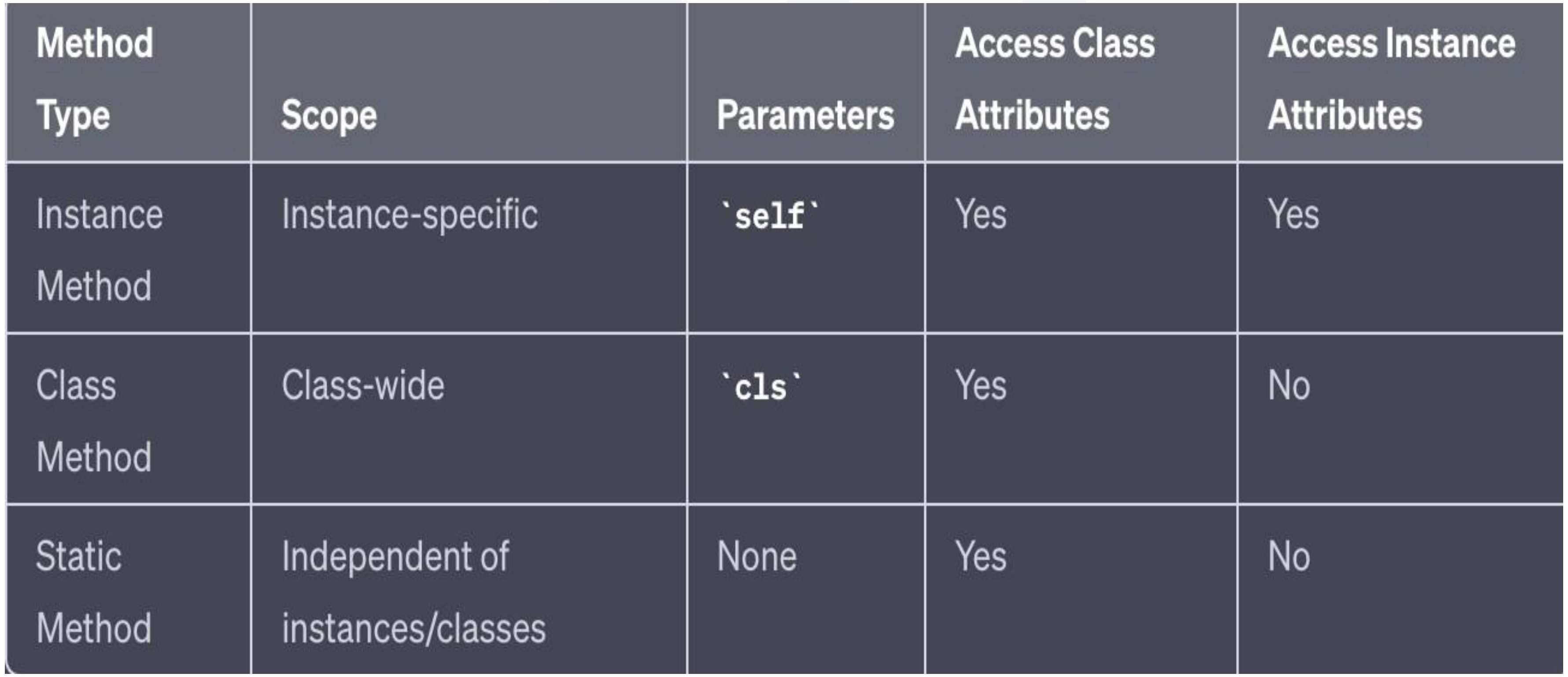
```python
class MyClass:
    @staticmethod
    def static_method(parameter1, parameter2):
        # Method code
```

- ❑ Static methods cannot access instance attributes or class attributes directly. They are similar to regular functions but are defined within a class for organization.
- ❑ They are called using the dot notation on the class itself: MyClass.static_method(parameter1, parameter2).

NEUROTECH

Instance Methods vs Class Methods vs Static Methods

- Here's a summary of the differences between these three types of methods:



Method Type	Scope	Parameters	Access Class Attributes	Access Instance Attributes
Instance Method	Instance-specific	`self`	Yes	Yes
Class Method	Class-wide	`cls`	Yes	No
Static Method	Independent of instances/classes	None	Yes	No

NeuroTech

Quizzes

NEUROTECH

OOP

- **Question 1:** What is the primary purpose of encapsulation in OOP?
  a) Grouping related attributes and methods together.
  b) Creating objects from classes.
  c) Performing inheritance.
  d) Defining class attributes.

- **Question 2:** Which special method is automatically called when an object is created from a class?
  a) create()
  b) init()
  c) new()
  d) __init__()

OOP

- **Question 1:** What is the primary purpose of encapsulation in OOP?
  a) Grouping related attributes and methods together.
  b) Creating objects from classes.
  c) Performing inheritance.
  d) Defining class attributes.

- **Question 2:** Which special method is automatically called when an object is created from a class?
  a) create()
  b) init()
  c) new()
  d) __init
  ____()

NEUROTECH

OOP

- **Question 3:** What is the significance of the self parameter in instance methods?
  a) It refers to the class itself.
  b) It refers to the current method being executed.
  c) It refers to the instance of the object on which the method is called.
  d) It refers to a parent class.

- **Question 4:** Which type of method is defined using the @classmethod decorator?
  a) Instance method
  b) Class method
  c) Static method
  d) Constructor method

OOP

- **Question 3:** What is the significance of the self parameter in instance methods?
  a) It refers to the class itself.
  b) It refers to the current method being executed.
  [✓ c) It refers to the instance of the object on which the method is called.]
  d) It refers to a parent class.

- **Question 4:** Which type of method is defined using the @classmethod decorator?
  a) Instance method
  [✓ b) Class method]
  c) Static method
  d) Constructor method

NeuroTech

OOP

Question 5: What is the key difference between a class attribute and an instance attribute?

a)Class attributes are accessed using the class name, while instance attributes are accessed using an object.

b)Class attributes are specific to a particular object, while instance attributes are shared among all objects of a class.

c)Class attributes can only be modified within class methods, while instance attributes can be modified in instance methods.

d) Class attributes are initialized using the constructor, while instance attributes are set using methods.

Question 6: Which type of method does not require access to instance or class attributes?

a) Instance method

b) Class method

c) Static method

d) Constructor method

OOP

Question 5: What is the key difference between a class attribute and an instance attribute?

a) Class attributes are accessed using the class name, while instance attributes are accessed using an object.

b)Class attributes are specific to a particular object, while instance attributes are shared among all objects of a class.

c)Class attributes can only be modified within class methods, while instance attributes can be modified in instance methods.

d) Class attributes are initialized using the constructor, while instance attributes are set using methods.

Question 6: Which type of method does not require access to instance or class attributes?

a) Instance method

b) Class method

c) Static method

d) Constructor method

OOP

Question 7: What is the purpose of the super() function in a subclass?

a) It creates a new instance of the subclass.
b) It initializes the attributes of the subclass.
c) It calls a method from the parent class.
d) It sets the value of the superclass attribute.

Question 8: Inheritance allows a subclass to:

a) Access private methods of the superclass.
b) Override methods of the superclass.
c) Hide methods of the superclass.
d) Use instance attributes of the superclass directly.

OOP

Question 7: What is the purpose of the super() function in a subclass?

a) It creates a new instance of the subclass.

b) It initializes the attributes of the subclass.

c) It calls a method from the parent class.

d) It sets the value of the superclass attribute.

Question 8: Inheritance allows a subclass to:

a) Access private methods of the superclass.

b) Override methods of the superclass.

c) Hide methods of the superclass.

d) Use instance attributes of the superclass directly.

OOP

Question 9: What does the term "polymorphism" refer to in OOP?

a) The ability of a class to inherit from multiple parent classes.
b) The ability to create multiple instances of a class.
c) The ability of a method to have multiple implementations based on the input.
d) The ability to create objects with different attributes.

Question 10: What is the purpose of the @staticmethod decorator in Python?

a) It indicates that the method is a constructor.
b) It indicates that the method is a class method.
c) It indicates that the method is an instance method.
d) It indicates that the method is a static method.

OOP

Question 9: What does the term "polymorphism" refer to in OOP?

a) The ability of a class to inherit from multiple parent classes.

b) The ability to create multiple instances of a class.

c) The ability of a method to have multiple implementations based on the input.

d) The ability to create objects with different attributes.

Question 10: What is the purpose of the @staticmethod decorator in Python?

a) It indicates that the method is a constructor.

b) It indicates that the method is a class method.

c) It indicates that the method is an instance method.

d) It indicates that the method is a static method.

Any Question

NeuroTech

Thanks

NeuroTech