In Python, special methods (also known as **magic methods** or **dunder methods**) are used to define how objects of a class behave in different operations. They are always surrounded by double underscores (\_\_), such as \_\_repr\_\_, \_\_str\_\_, \_\_add\_\_, \_\_sub\_\_, and \_\_mul\_\_. Let's explore these methods:

**1. \_\_repr\_\_ (Representation Method)**

* This method is meant to provide an **unambiguous** string representation of an object, mainly for debugging.
* It should return a string that, ideally, can be used to recreate the object.

**Example:**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_repr\_\_(self):

return f"Person('{self.name}', {self.age})"

p = Person("Alice", 30)

print(repr(p)) # Output: Person('Alice', 30)

**2. \_\_str\_\_ (String Method)**

* This method defines the **human-readable** string representation of an object.
* It is called by str() or when print() is used on an object.
* If \_\_str\_\_ is not defined, Python falls back to \_\_repr\_\_.

**Example:**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_str\_\_(self):

return f"{self.name} is {self.age} years old"

p = Person("Alice", 30)

print(str(p)) # Output: Alice is 30 years old

print(p) # Output: Alice is 30 years old

**3. \_\_add\_\_ (Addition Operator Overloading)**

* Used to define the behavior of + between objects.
* Often used in numeric classes or when concatenating objects.

**Example:**

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_add\_\_(self, other):

return Vector(self.x + other.x, self.y + other.y)

def \_\_repr\_\_(self):

return f"Vector({self.x}, {self.y})"

v1 = Vector(2, 3)

v2 = Vector(4, 5)

print(v1 + v2) # Output: Vector(6, 8)

**4. \_\_sub\_\_ (Subtraction Operator Overloading)**

* Used to define the behavior of - between objects.

**Example:**

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_sub\_\_(self, other):

return Vector(self.x - other.x, self.y - other.y)

def \_\_repr\_\_(self):

return f"Vector({self.x}, {self.y})"

v1 = Vector(5, 7)

v2 = Vector(2, 3)

print(v1 - v2) # Output: Vector(3, 4)

**5. \_\_mul\_\_ (Multiplication Operator Overloading)**

* Defines how objects behave when multiplied using \*.

**Example:**

class Vector:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_mul\_\_(self, scalar):

return Vector(self.x \* scalar, self.y \* scalar)

def \_\_repr\_\_(self):

return f"Vector({self.x}, {self.y})"

v = Vector(3, 4)

print(v \* 2) # Output: Vector(6, 8)

**Summary:**

| **Method** | **Purpose** |
| --- | --- |
| \_\_repr\_\_ | Provides an unambiguous string representation of an object (used for debugging). |
| \_\_str\_\_ | Provides a user-friendly string representation of an object (used by print()). |
| \_\_add\_\_ | Defines behavior for + (addition of objects). |
| \_\_sub\_\_ | Defines behavior for - (subtraction of objects). |
| \_\_mul\_\_ | Defines behavior for \* (multiplication of objects). |