**🔮 Magic Methods in Python (Dunder Methods)**

**Magic methods** (also called **dunder methods**, short for **"double underscore" methods**) are **special methods in Python that start and end with double underscores** (\_\_). They **enable custom behavior for objects**, such as operator overloading and built-in function support.

**🔹 Why Are They Called Magic Methods?**

These methods are **not explicitly called by the user**—Python **automatically invokes them** in certain situations.

For example:

class MyClass:

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

self.name = name # Called automatically when an object is created

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

return f"MyClass({self.name})" # Called when using print(obj)

obj = MyClass("Alice")

print(obj) # Calls \_\_str\_\_ automatically: Output -> MyClass(Alice)

📌 **Here, Python automatically calls:**

* \_\_init\_\_() when creating the object.
* \_\_str\_\_() when printing the object.

**🔹 List of Common Magic Methods**

**🔸 Object Initialization & Representation**

| **Magic Method** | **Purpose** | **Example** |
| --- | --- | --- |
| \_\_init\_\_(self, ...) | Constructor (object creation) | obj = MyClass() |
| \_\_del\_\_(self) | Destructor (called when object is deleted) | del obj |
| \_\_repr\_\_(self) | Official string representation | repr(obj) |
| \_\_str\_\_(self) | Human-readable string representation | print(obj) |

**🔸 Operator Overloading**

| **Magic Method** | **Purpose** | **Example (obj1 & obj2 are objects)** |
| --- | --- | --- |
| \_\_add\_\_(self, other) | + Operator | obj1 + obj2 |
| \_\_sub\_\_(self, other) | - Operator | obj1 - obj2 |
| \_\_mul\_\_(self, other) | \* Operator | obj1 \* obj2 |
| \_\_truediv\_\_(self, other) | / Operator | obj1 / obj2 |
| \_\_floordiv\_\_(self, other) | // Operator | obj1 // obj2 |
| \_\_mod\_\_(self, other) | % Operator | obj1 % obj2 |
| \_\_pow\_\_(self, other) | \*\* Operator | obj1 \*\* obj2 |

**🔸 Comparison Operators**

| **Magic Method** | **Purpose** | **Example** |
| --- | --- | --- |
| \_\_eq\_\_(self, other) | == (Equality) | obj1 == obj2 |
| \_\_ne\_\_(self, other) | != (Not equal) | obj1 != obj2 |
| \_\_lt\_\_(self, other) | < (Less than) | obj1 < obj2 |
| \_\_le\_\_(self, other) | <= (Less than or equal) | obj1 <= obj2 |
| \_\_gt\_\_(self, other) | > (Greater than) | obj1 > obj2 |
| \_\_ge\_\_(self, other) | >= (Greater than or equal) | obj1 >= obj2 |

**🔸 Container & Iteration Methods**

| **Magic Method** | **Purpose** | **Example** |
| --- | --- | --- |
| \_\_len\_\_(self) | Returns length | len(obj) |
| \_\_getitem\_\_(self, key) | Get item (obj[key]) | obj[0] |
| \_\_setitem\_\_(self, key, value) | Set item (obj[key] = value) | obj[0] = 10 |
| \_\_delitem\_\_(self, key) | Delete item (del obj[key]) | del obj[0] |
| \_\_iter\_\_(self) | Returns an iterator | for i in obj: |
| \_\_next\_\_(self) | Next item in iteration | next(obj) |

**🔹 Example: Operator Overloading Using Magic Methods**

**🔸 Overloading + Operator**

class Point:

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

self.x = x

self.y = y

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

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

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

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

p1 = Point(2, 3)

p2 = Point(4, 5)

p3 = p1 + p2 # Calls \_\_add\_\_ method

print(p3) # Output: Point(6, 8)

📌 **Explanation:**

* p1 + p2 **calls \_\_add\_\_()**, returning a new Point object.
* print(p3) **calls \_\_str\_\_()**, printing a readable string.

**🔹 Example: Making a Custom Iterable**

Let's make a class that behaves like a list.

python

CopyEdit

class CustomList:

def \_\_init\_\_(self, items):

self.items = items

def \_\_getitem\_\_(self, index):

return self.items[index] # Allows obj[index]

def \_\_len\_\_(self):

return len(self.items) # Allows len(obj)

obj = CustomList([10, 20, 30])

print(obj[1]) # Output: 20 (calls \_\_getitem\_\_)

print(len(obj)) # Output: 3 (calls \_\_len\_\_)

📌 **Explanation:**

* obj[1] **calls \_\_getitem\_\_()** to retrieve an item.
* len(obj) **calls \_\_len\_\_()** to get the length.

**🔹 Why Use Magic Methods?**

✅ **Custom Behavior:** Modify how objects work with built-in operations.  
✅ **Readable Code:** Use obj + obj instead of obj.add(obj).  
✅ **Integration:** Works with Python’s built-in functions like len(), print(), sorted().