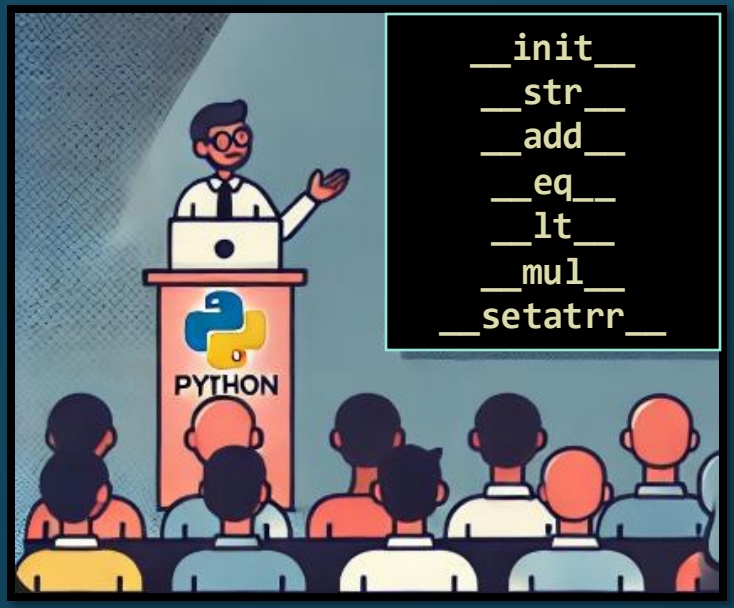


MAGIC METHODS

ENHANCE YOUR PYTHON CLASSES

Jaume Boguñá

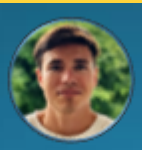
Dive into Python

What are Magic Methods?

1. They are defined within **classes** in **Python**.
2. They are special **double underscore** (__) or **dunder methods**.
3. They **control object behavior** with operations like **+**, **==**, or **print()**.

Examples:

- ✓ __init__
- ✓ __str__
- ✓ __add__
- ✓ __format__
- ✓ __setattr__



Magic Methods

1. `__init__(self, …)`

Initializes a **new instance** of the **class** (it's the **constructor**).

```
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

earth: Planet = Planet("Earth", 150)

print(earth.name, earth.distancefromSun, sep='\n')
Earth
150
```

Magic Methods

2. `__str__(self)`

Returns a **user-friendly string representation** of the **class**.

```
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __str__(self) -> str:
        return f"Planet {self.name} is {self.distancefromSun}
                million km from the Sun."


earth: Planet = Planet("Earth", 150)

print(earth)
Planet Earth is 150 million km from the Sun.
```

Calls earth.__str__()

Magic Methods

3. `__eq__(self, other)`

Defines how **two objects** of the **class** are **compared** for **equality (==)**.

```
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __eq__(self, other) -> bool:
        if isinstance(other, Planet):
            return self.distancefromSun == other.distancefromSun
        return False

earth: Planet = Planet("Earth", 150)
mars: Planet = Planet("Mars", 228)

print(earth == mars)  Calls earth.__eq__(mars)
False
```

Magic Methods

4. `__lt__(self, other)`

Defines behavior for the **less-than operator** (<).

```
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __lt__(self, other) -> bool:
        if isinstance(other, Planet):
            return self.distancefromSun < other.distancefromSun
        return NotImplemented

earth: Planet = Planet("Earth", 150)
mars: Planet = Planet("Mars", 228)

print(earth < mars)
True
```

Calls earth.__lt__(mars)

Magic Methods

5. `__add__(self, other)`

Defines behavior for the **addtion operator** (+).

```
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __add__(self, other) -> int:
        if isinstance(other, Planet):
            return self.distancefromSun + other.distancefromSun
        return NotImplemented

earth: Planet = Planet("Earth", 150)
mars: Planet = Planet("Mars", 228)

print(distance_sum := earth + mars)
378
```

Calls earth.__add__(mars)

Magic Methods

6. `__radd__(self, other)`

Defines behavior for the **right-hand addition** (+).

```python
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __radd__(self, other) -> int | float:
        if isinstance(other, (int, float)):
            return other + self.distancefromSun
        return NotImplemented

earth: Planet = Planet("Earth", 150)

print(mars_distance_from_Sun := 78 + earth)
228
```

The method handles adding the planet's distance to a number.

Calls earth.__radd__(78)

Magic Methods

7. `__mul__(self, other)`

Defines behavior for the **multiplication operator** (*).

```python
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __mul__(self, other) -> int | float:
        if isinstance(other, (int, float)):
            return f"New distance from Sun is {self.distancefromSun * other} million km."
        return NotImplemented

earth: Planet = Planet("Earth", 150)

print(double_distance_earth := earth * 2)
New distance from Sun is 300 million km.
```

Calls `earth.__mul__(2)`

Magic Methods

8. `__len__(self)`

Allows to get the **length** of an **object** using the **len() function**.

```python
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __len__(self) -> int:
        return len(self.name)

earth: Planet = Planet("Earth", 150)

print(name_length := len(earth)) Calls earth.__len__()
5
```

Magic Methods

9. `__contains__(self, item)`

Defines behavior for the **in operator**.

```
class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __contains__(self, item) -> bool:
        return item in self.name

earth: Planet = Planet("Earth", 150)

print("art" in earth.name)
True
```

The method checks if the item is a substring of the name attribute.

Calls earth.__contains__("art")

Magic Methods

10. `__setattr__(self, name, value)`

Manages **attribute setting**, **enabling validation** or **custom behavior**.

```
import pretty_errors

class Planet:

    def __init__(self, name: str, distancefromSun: int) -> None:
        self.name = name
        self.distancefromSun = distancefromSun

    def __setattr__(self, name, value):
        if name == "distancefromSun" and value < 0:
            raise ValueError("Distance from the Sun cannot be negative")
        super().__setattr__(name, value)

earth: Planet = Planet("Earth", -150)

raise ValueError("Distance from the Sun cannot be negative")
ValueError:
Distance from the Sun cannot be negative
```

Summary

10 Common Magic Methods	
`__init__(self, …)`	Initializes a new instance of the class (it's the constructor)
`__str__(self)`	Returns a user-friendly string representation of the class
`__eq__(self, other)`	Defines how two objects of the class are compared for equality (==)
`__lt__(self, other)`	Defines behavior for the less-than operator (<)
`__add__(self, other)`	Defines behavior for the addtion operator (+)
`__radd__(self, other)`	Defines behavior for the right-hand addition (+)
`__mul__(self, other)`	Defines behavior for the multiplication operator (*)
`__len__(self)`	Allows to get the length of an object using the len() function
`__contains__(self, item)`	Defines behavior for the in operator
`__setattr__(self, name, value)`	Manages attribute setting, enabling validation or custom behavior

Like

Comment

Share

Jaume Boguñá

Aerospace Engineer | Data Scientist