2025

The Hidden Potential of Python’s Dunder Methods

Writing smarter, more intuitive code with Python’s magic methods

By: Ijeoma Eti

→ Next

About Me:

- **Software Engineer** with over 5 years of experience.
- Passionate about sharing knowledge through writing.
- Active contributor to the community through open-source projects.

Find me on:

- LinkedIn: https://www.linkedin.com/in/ijeoma-eti
- X (formerly Twitter): https://x.com/EtiIjeoma
- GitHub: http://github.com/Aijeyomah

→

What Are Dunder Methods?

- *Dunder methods, or "magic methods," are double-underscore methods like `__init__`, __add__, and __str__.*
- *These methods are predefined by Python and enable objects to interact with built-in functions and operators seamlessly.*
- *Examples of dunder methods:*

```
  __len__ – Determines the length of an
object.
__add__ – Defines behavior for the + operator.

__getitem__ – Enables index-based access.
```

All dunder methods are part of the __builtins__ module, automatically imported when Python starts.

→

Why Use Dunder Methods

- *Dunder methods make objects feel like built-in types, enhancing usability.*
- *Enable intuitive APIs by integrating custom objects with Python syntax.*
- *Powerful popular libraries like NumPy and SQLAlchemy.*
- *Simplify complex tasks with domain-specific designs.*

Examples from Popular Libraries

Real-World Examples of Dunder Methods

Example 1: NumPy Array Operations

```
import numpy as np

# Create two arrays
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Use the + operator (calls __add__ internally)
result = a + b
print(result)  # Output: [5 7 9]
```

- *NumPy overloads operators like + and * to enable element-wise operations.*

Example 2: SQLAlchemy Query Expressions

```
from sqlalchemy import Column, Integer, String, MetaData, Table

metadata = MetaData()
users = Table(
    'users', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String),
)

query = users.c.name == "John"  # Calls __eq__
print(query)
# Output: users.name = :name_1
```

- *SQLAlchemy uses dunder methods like __eq__ to simplify SQL query construction.*

How Dunder Methods Work

How Are Dunder Methods Defined?

- *Dunder methods are not arbitrary; they are predefined by Python's data model.*
- *You cannot create custom dunder methods with random names like __custom__.*

Example:

```
class MyClass:
    def __custom__(self):
        return "This does nothing special."
```

  - *Python ignores __custom__ unless explicitly called.*
  - *Stick to predefined methods like __add__, __str__, and __eq__ for consistency.*
- *These methods are triggered implicitly by the interpreter based on specific syntax or operations.*

→

Creating a Custom Class

Building Intuitive APIs with Dunder Methods

Step 1: A Basic Class

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Instantiate the class
point = Point(2, 3)
print(point)  # Output: <__main__.Point object at
0x...>
```

- *Default behavior: Python shows the object's memory address.*

→

Step 2: Adding __repr__ for Readability

```
class Point:
    def __repr__(self):
        return f"Point({self.x}, {self.y})"

# Improved representation
print(Point(2, 3)) # Output: Point(2, 3)
```

By Implementing __repr__, objects now display meaningful information.

Adding Custom Behavior

Enhancing Classes with __add__

Problem: Adding two Point objects doesn't work:

```
point1 = Point(2, 3)
point2 = Point(4, 5)
result = point1 + point2
# TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

Solution: Implement __add__ to define how addition should work:

```
def __add__(self, other):
    return Point(self.x + other.x, self.y +
```

Result:

```
point1 = Point(2, 3)
point2 = Point(4, 5)
print(point1 + point2)  # Output: Point(6, 8)
```

This makes the + operator work intuitively for Point objects.

→

Exploring More Dunder Methods

Other Useful Dunder Methods

1. Equality with __eq__

Define how two objects are compared for equality:

```
 def __eq__(self, other):
    return self.x == other.x and self.y ==
other.y
```

Example:

```
Point(2, 3) == Point(2, 3)  # True
Point(2, 3) == Point(4, 5)  # False


Makes objects comparable using ==.
```

→

Exploring More Dunder Methods

2. Scaling with __mul__

Allow objects to be multiplied by a scalar:

```
def __mul__(self, scalar):
    return Point(self.x * scalar, self.y *
scalar)
```

Example:

```
point = Point(2, 3)
print(point * 3)  # Output: Point(6, 9)


Provides a clean way to scale objects.
```

→

Exploring More Dunder Methods

3. Iterating with __iter__

Make objects iterable, enabling looping over their attributes:

```
def __iter__(self):
    return iter((self.x, self.y))
```

Example:

```
point = Point(2, 3)
for coord in point:
    print(coord)
# Output:
# 2
# 3
```

- *Simplifies access to an object's internal data.*

→

Iterating Over Objects

Title: Making Classes Iterable

- *Use __iter__ to allow looping over objects.*

```
class Point:
    def __iter__(self):
        return iter((self.x, self.y))

point = Point(2, 3)
for coord in point:
    print(coord)
# Output:
# 2
# 3
```

→

When Not to Use Dunder Methods

Avoiding Overuse and Misuse

1. *Don’t Overcomplicate:*
    - *Use regular methods for simple tasks.*

```
class Calculator:
    def add(self, a, b):
        return a + b
```

Stick to the Rules:

Don’t invent unsupported dunder methods like __custom__.

Be Intuitive:

Avoid confusing operator overloads.

for simple tasks.

```
def __add__(self, other):
    return self * other  # Misleading
```

Conclusion

- *Key Takeaways:*
  - *Dunder methods integrate custom objects seamlessly with Python’s syntax.*
  - *They simplify APIs, enhance readability, and enable powerful domain-specific designs.*
- *Call to Action:*
  - *Use dunder methods thoughtfully to build intuitive, Pythonic solutions.*

Quote: "Dunder methods are not just tools; they’re a way to speak Python’s native language."

Thank you