# Object-Oriented Programming

Magic Method / Dunder in Python

**Vo Nguyen Thanh Nhan**

February 2026

# Contents

# Chapter 1

# Magic Methods

## 1.1 Definition

Magic Methods (or Dunder Methods e.g., `__init__()`), are the fundamental of Python's O.O.P. They allow custom objects to integrate with built-in Python features such as operators, iteration, and type conversion. Instead of calling these methods directly, the Python interpreter invokes them in response to specific syntax.

## 1.2 Common Magic Methods

The following example demonstrates the practical implementation of these magic methods:

```
1  class Book:
2      def __init__(self, author, title, pages_num):
3          self.title = title
4          self.author = author
5          self.pages_num = pages_num
6
7      def __str__(self):
8          return f"{self.title} by {self.author}"
9
10     def __eq__(self, other):
11         return (self.title == other.title) and (self.author ==
           ↪  other.author) #dont care about the num of pages
12
13     def __lt__(self, other):
14         return self.pages_num < other.pages_num
```

```
15
16      def __gt__(self, other):
17          return self.pages_num > other.pages_num
18
19      def __add__(self, other):
20          return self.pages_num + other.pages_num
21
22      def __contains__(self, keyword):
23          return keyword in self.title
24
25      def __getitem__(self, key):
26          if key == 'title':
27              return self.title
28
29          elif key == 'author':
30              return self.author
31
32          elif key == 'pages_num':
33              return self.pages_num
34
35          else:
36              return f"{key} was not valid"
```

### 1.2.1 The `__init__(self,...)` method

The `__init__()` method is the most fundamental dunder method in Python. It acts as the initializer for a class instance, which sets the initial state of an object by assigning values to its attributes, and can be automatically invoked when new objects is instantiated (e.g `obj = MyClass(...)`

```
1  def __init__(self, author, title, pages_num):
2      self.title = title
3      self.author = author
4      self.pages_num = pages_num
```

From the above example, we use `__init__(...)` to get initial informations of the object about `author, title, pages_num`

### 1.2.2 The `__str__(self)` method

The `__str__(self)` controls what will be returned when the object is printed. We can modify and adjust to make it readable and descriptive output for end-users.

```
def __str__(self):
    return f"{self.title} by {self.author}"
```

For the above example, we will receive an output of the name of the book with its author when the object is printed

### 1.2.3 Comparison and Arithematic methods

The comparison methods are used when we compare two objects with their attributes, and we could decide which features to be compared. We can use the `__eq__(self, other)` to compare the title and author of two objects, while the `__lt__(self, other)` and `__gt__(self, other)` are used to compare the number of pages Arithmetic methods allow objects to perform mathematical or concatenation operations. The `__add__(self, other)` method, for example, can be used to concatenate string features or calculate the total sum of numerical attributes, such as adding the page counts of two different books in the previous example.

```
def __eq__(self, other):
    return (self.title == other.title) and (self.author ==
    ↪  other.author) #dont care about the num of pages

def __lt__(self, other):
    return self.pages_num < other.pages_num

def __gt__(self, other):
    return self.pages_num > other.pages_num

def __add__(self, other):
    return self.pages_num + other.pages_num
```

### 1.2.4 Collection methods

These methods allow an object to behave like a built-in container (list, dict)

```
def __contains__(self, keyword):
    return keyword in self.title

def __getitem__(self, key):
    if key == 'title':
        return self.title
```

```
7
8       elif key == 'author':
9           return self.author
10
11      elif key == 'pages_num':
12          return self.pages_num
13
14      else:
15          return f"{key} was not valid"
```

**Note:** We can use a different name for `self` but it is conventional and easier for others to read your code.

## 1.3 Testing and Result

Here are the assigned objects and some test cases for each method

```
1  b1 = Book("Kelvin", "A", 150)
2  b2 = Book("John", "B", 200)
3  b3 = Book("Kathy", "C", 300)
4  b4 = Book("John", "B", 201)
5  # __str__()
6  print(b1)
7  # __eq__()
8  print(b1 == b2)
9  print(b2 == b4)
10 #__lt__()
11 print(b1 < b2)
12 print(b3 < b4)
13 # __gt__()
14 print(b1 > b2)
15 print(b3 > b4)
16 # __add__()
17 print(b1 + b2)
18 # __contains__()
19 print("John" in b4)
20 print("John" in b3)
21 # __getitem__()
22 print(b1['title'])
```

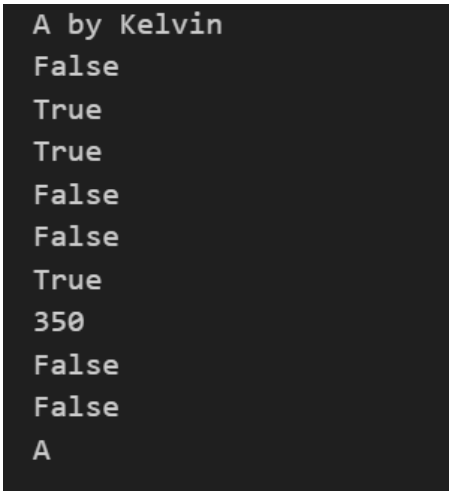


Figure 1.1: Outputs