CBS Department

Special Methods

Soma Soleiman Zadeh

Object-Oriented Programming (CBS 215)

Fall 2025 - 2026

Week 11

December 17 – 18, 2025

Outline

- **Special Methods** in Class
  - **__init__( )**
  - **__new__( )**
  - **__str__( )**
  - **__repr__( )**
- **Special Methods** for Comparison Operators
- **Special Methods** for Arithmetic Operators

Special Methods (Dunder Methods) in Class

- **Special Methods** are predefined methods in Python that have double underscores at the beginning and end of their names.
- Example of Special Methods → **__init__( )**
- **Special methods** allow your classes to interact with built-in Python operations, such as addition, string representation, comparison, and many more.

__init__( ) Method

- **__init__( )** method is called the **class constructor**. This method is invoked when an object of a class is created.
- The goal of **__init__( )** method is to initialize all attributes that you have in your class.

```
class Student():
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

__new__( ) Method

- **__new__( )** is a method for creating a new empty object of a class.
- When the **__init__( )** method is called, Python implicitly calls the **__new__( )** method to return a new empty object of the class.
- The default implementation of **__new__( )** is enough for most use cases. So, there is no need to change its implementation.

__str__( ) Method and __repr__( ) Method

- **__str__( )** and **__repr__( )** are from the most commonly used methods in custom classes.
- Both **__str__( )** and **__repr__( )** methods return string presentations of an object.
- **Why two methods for string representation of objects?**

Difference Between __str__( ) and __repr__( )

- **__str__( )** provides a string presentation of an object, **for end-users**.
  - A **simple description** of the object
- **__repr__( )** provides a string presentation of an object, for **developers or programmers**.
  - A **detailed description** of the object

Example of __str__( ) and __repr__( ) Methods

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

    def __str__(self):
        return f"Point coordinates are x = {self.x} and y = {self.y}."

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

```
p1 = Point(3,4)

print(p1)
print(repr(p1))
```

Special Methods Behind Comparison Operators

Operator	Special Method	Description
<	__lt__(self, other)	self is Less Than other.
<=	__le__(self, other)	self is Less than or Equal to other.
==	__eq__(self, other)	self is Equal to other.
!=	__ne__(self, other)	self is Not Equal to other.
>	__gt__(self, other)	self is Greater Than other.
>=	__ge__(self, other)	self is Greater than or Equal to other.

Example of Equality Comparison: __eq__( ) Method

- The **__eq__( )** method is a special method that allows for the customization of the equality comparison operator == for objects of a class.
- By default, **__eq__( )** method compares the memory addresses of the objects.
  - Two objects of a class are equal if they are actually the same object.

Overriding __eq__( ) Method

- Overriding **__eq__( )** allows for more meaningful comparisons between objects.
- We are going to override the **__eq__( )** method to compare objects based on their content, not their memory addresses.

Overriding __eq__( ) Method

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

    def __str__(self):
        return f"Point coordinates are x = {self.x} and y = {self.y}."

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

    def __eq__(self, other):
        if not isinstance(other,Point):
            return NotImplemented

        return self.x==other.x and self.y==other.y
```

Special Methods Behind Arithmetic Operators

Operator	Special Method	Description
+	__add__(self, other)	Addition
-	__sub__(self, other)	Subtraction
*	__mul__(self, other)	Multiplication
/	__truediv__(self, other)	True Division
//	__floordiv__(self, other)	Floor Division
%	__mod__(self, other)	Modulo Operation
**	__pow__(self, other)	Power Operation (Exponentiation)