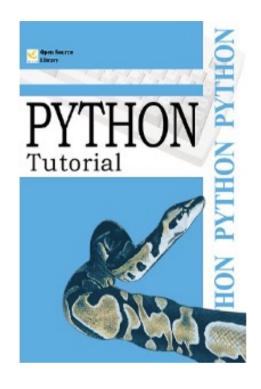# Operators Overloading

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Understanding what is Operator Overloading
- Main Magic Methods for Operator Overloading

Source

- https://docs.python.org/3/reference/
- https://ellibrodepython.com/
- Python Tutorial - Tapa blanda. GuidoVan Rossum (2012)

Outline

- Introduction

- Magic Methods

- Best Practices

Introduction

- Customizing how operators work with user-defined classes
- Allows objects to behave like built-in types
- Makes code more intuitive and readable
- Operator overloading allows operators to have different meanings based on their operands
- Implemented using special methods (magic methods)
- For example:
    - `+` for addition of numbers, concatenation of strings, or merging of lists

Outline

- Introduction
- **Magic Methods**
- Best Practices

Magic Methods

- Arithmetic Operators
  - `__add__(self, other)` for `+`
  - `__sub__(self, other)` for `-`
  - `__mul__(self, other)` for `*`
  - `__truediv__(self, other)` for `/`

- Comparison Operators
  - `__eq__(self, other)` for `==`
  - `__lt__(self, other)` for `<`
  - __gt__(self, other) for >
  - __le__(self, other) for <=

Example: Add two Vector

Example: +

```python
class Vector:
    def __init__(self, x, y):

        self.x = x

        self.y = y

    def sumar(self, other):

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

v1 = Vector(2, 3)
v2 = Vector(4, 5)
v3 = v1.sumar(v2)
```

Example: Overloading `+` Operator

Example: +

```python
class Vector:

    def __init__(self, x, y):

        self.x = x

        self.y = y

    def __add__(self, other):

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

v1 = Vector(2, 3)
v2 = Vector(4, 5)
v3 = v1.__add__(v2)  #Explicit call
```

Example: Overloading `+` Operator

```python
class Vector:

    def __init__(self, x, y):

        self.x = x

        self.y = y

    def __add__(self, other):

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

v1 = Vector(2, 3)
v2 = Vector(4, 5)
v3 = v1 + v2   # Output: <__main__.Vector object at 0x7ee28d312830>
```

Example: str

```python
class Vector:
    def __init__(self, x, y):

        self.x = x

        self.y = y

    def __add__(self, other):

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

    def __str__(self)

        return str("({self.x}, {self.y})") # Formato deseado

        #return "(" + str(self.x) + "," + str(self.y) + ")"
```

Example: Overloading `+` Operator

Example: + & str

```
v1 = Vector(2, 3)

v2 = Vector(4, 5)

v3 = v1 + v2

print(v3)  # Output: (6, 8)
```

Example: Overloading `+` Operator

Example: + with different types

```
v1 = Vector(2, 3)

v2 = Vector(4, 5)

#What happen if we add a number?

v3 = v1 + 4
```

Example: __add__ with different types

Example: add a number

```python
class Vector:

    …

    def __add__(self, other):

        if isinstance(other, Vector):

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

        elif isinstance(other, (int, float)):

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

        else:

            raise TypeError("Unsupported operand type")
```

Example: conmutative `+` Operator

Example: + with different types

```
v1 = Vector(2, 3)

v2 = Vector(4, 5)

#What happen if we add a number?

v5 =  4 + v2 # ERROR: unsupported operand type(s) for +: 'int' and 'Vector'
```

Example: __radd__

Example: add a number in both sides of +

```python
class Vector:

    …

    def __radd__(self, other):

        return self.__add__(other)




v5= v2.__radd__(4)
```

Outline

- Introduction

- Magic Methods

- **Best Practices**

Best Practices

- Follow Python's operator semantics (e.g., + should add, not subtract)

- Implement related operators together (if you implement <, implement >)

- Return *NotImplemented* for unsupported operand types

- Maintain consistency with built-in types

- Document the behavior of overloaded operators

- Consider implementing reverse (conmutative) operations (radd, rsub, etc.)

Practical Exercises

1.   Create a Rectangle class and overload the * operator to multiply areas