Computer Science Advanced

BÀI 2. SPECIAL METHODS VÀ KẾ THỪA

1. Special Methods

Trong Python, ***special method*** là các phương thức đặc biệt, dùng để định nghĩa *cách tương tác của lớp và đối tượng* trong các hoạt động thực thi của chương trình như *khởi tạo đối tượng*, *tính toán* và *áp dụng hàm*.

Mỗi phương thức có một tên riêng theo quy ước của Python, với định dạng chung là **`__<name>__`**.

Ta có thể xem các special method của một class bằng hàm **`dir()`**.

Ví dụ: Từ kết quả bên, ta có thể thấy `__add__` là một *special method* của lớp `int`.

Phương thức này định nghĩa cách tương tác của một đối tượng `int` với toán tử `+`.

Giả sử ta khởi tạo

```
>>> num = 10
```

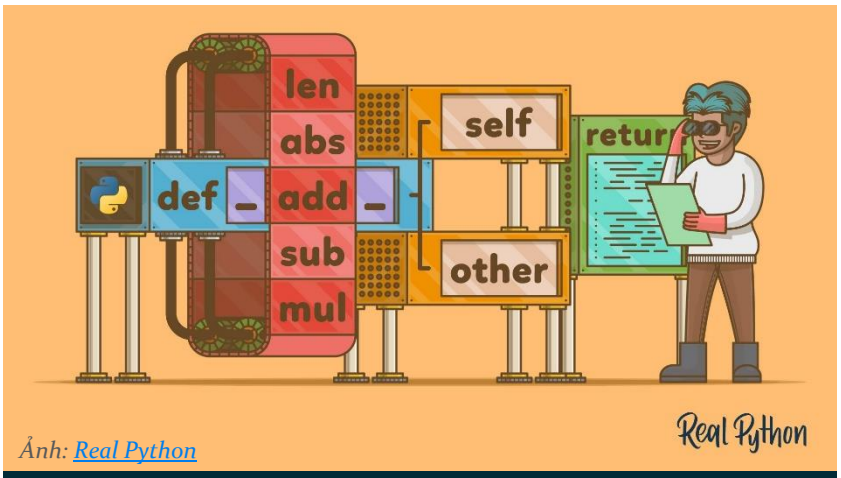
Khi đó, phép toán

```
>>> num + 5
```

sẽ gọi đến lệnh

```
>>> num.__add__(5)
```

để nhận kết quả trả về.



Ảnh: Real Python

```
>>> dir(int)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__',
'__class__', '__delattr__', '__dir__', '__divmod__', ...]
```

SPECIAL METHOD CHO LỚP TỰ ĐỊNH NGHĨA

Khi định nghĩa một class, ta cũng có thể định nghĩa các *special method* này.

Ví dụ: Ta định nghĩa lớp **`BetterString`** với chức năng tự động thêm khoảng trắng khi cộng chuỗi.

```
class BetterString:

  def __init__(self, content):
    self.content = content

  def __str__(self):
    return self.content

  def __add__(self, value):
    return self.content + ' ' + value
```

- Phương thức **`__init__`** nhận vào một *string* để khởi tạo đối tượng
- Phương thức **`__str__`** trả về chuỗi khi đối tượng tương tác với string và trong câu lệnh `print()`
- Phương thức **`__add__`** định nghĩa cách đối tượng xử lý với toán tử **+**

Sử dụng:

Code	Output

`name = BetterString('MindX')`
`print(name)`
`print(name + 'School')`
	`MindX`
	`MindX School`

MỘT SỐ SPECIAL METHOD THÔNG DỤNG

Chức năng	Phương thức	Hoạt động
Khởi tạo đối tượng	**`__init__`**`(self, ...)`	`x = ClassX()` ➞ `x.__init__()`
Chuyển thành string	**`__str__`**`(self)`	`print(x)` ➞ `print(x.__str__())`

Chức năng	Phương thức	Hoạt động
Phép cộng	`__add__(self, other)`	`x + y` → `x.__add__(y)`
Phép trừ	`__sub__(self, other)`	`x - y` → `x.__sub__(y)`
So sánh bằng	`__eq__(self, other)`	`x == y` → `x.__eq__(y)`
So sánh lớn hơn	`__gt__(self, other)`	`x > y` → `x.__gt__(y)`
So sánh bé hơn	`__lt__(self, other)`	`x < y` → `x.__lt__(y)`
So sánh lớn hơn hoặc bằng	`__ge__(self, other)`	`x >= y` → `x.__ge__(y)`
So sánh bé hơn hoặc bằng	`__le__(self, other)`	`x <= y` → `x.__le__(y)`

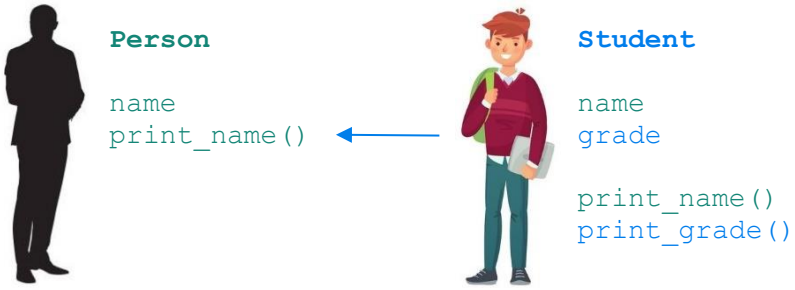
2. Kế Thừa

Kế thừa trong lập trình hướng đối tượng là phương pháp dùng để tạo một lớp mới *có đầy đủ chức năng của một lớp đã có*, từ đó phát triển thêm những tính năng mới.

Lớp đã có được gọi là *lớp cha*. Lớp kế thừa từ lớp cha gọi là *lớp con*.

Ví dụ: Ta tạo **Person** là lớp cha và **Student** là lớp con.

- **Person** có thuộc tính `name` và phương thức `print_name()`, do đó **Student** cũng có thuộc tính và phương thức trên.
- Ngoài ra, **Student** còn có thêm thuộc tính `grade` và phương thức `print_grade()`.
- Ta sử dụng lệnh `super().__init__()` trong constructor của **Student** để thực hiện các khởi tạo được định nghĩa ở **Person**.




```python
class Person:           # parent class

  def __init__(self, name):
    self.name = name

  def print_name(self):
    print('My name is', self.name)


class Student(Person):  # child class

  def __init__(self, name, grade):
    super().__init__(name)
    self.grade = grade

  def print_grade(self):
    print('I am in grade', self.grade)
```

Ghi chú:

- Với cách tổ chức này, ta nói mọi đối tượng thuộc lớp **Student** cũng là một đối tượng **Person**. Tuy nhiên, không phải mọi đối tượng **Person** đều là **Student**.
- Lớp **Person** và **Student** đều có thể tiếp tục được kế thừa.

Sử dụng:

Code	Output
`john = Person("John")` `john.print_name()`	`My name is John`

`jim = Student("Jim", 10)`
`jim.print_name()`
`jim.print_grade()`
	`My name is Jim`
	`I am in grade 10`

OVERRIDE

Khi kế thừa, ta có thể ghi đè lên phương thức của lớp cha để thực hiện các thay đổi riêng cho lớp con. Kĩ thuật này gọi là ***override***.

Ví dụ: Trong lớp **Student**, ta có thể override phương thức `print_name()` của lớp **Person** bằng cách định nghĩa một phương thức cùng tên.

```python
class Student(Person):
  ...
  def print_name(self):
    print('I am', self.name)
```