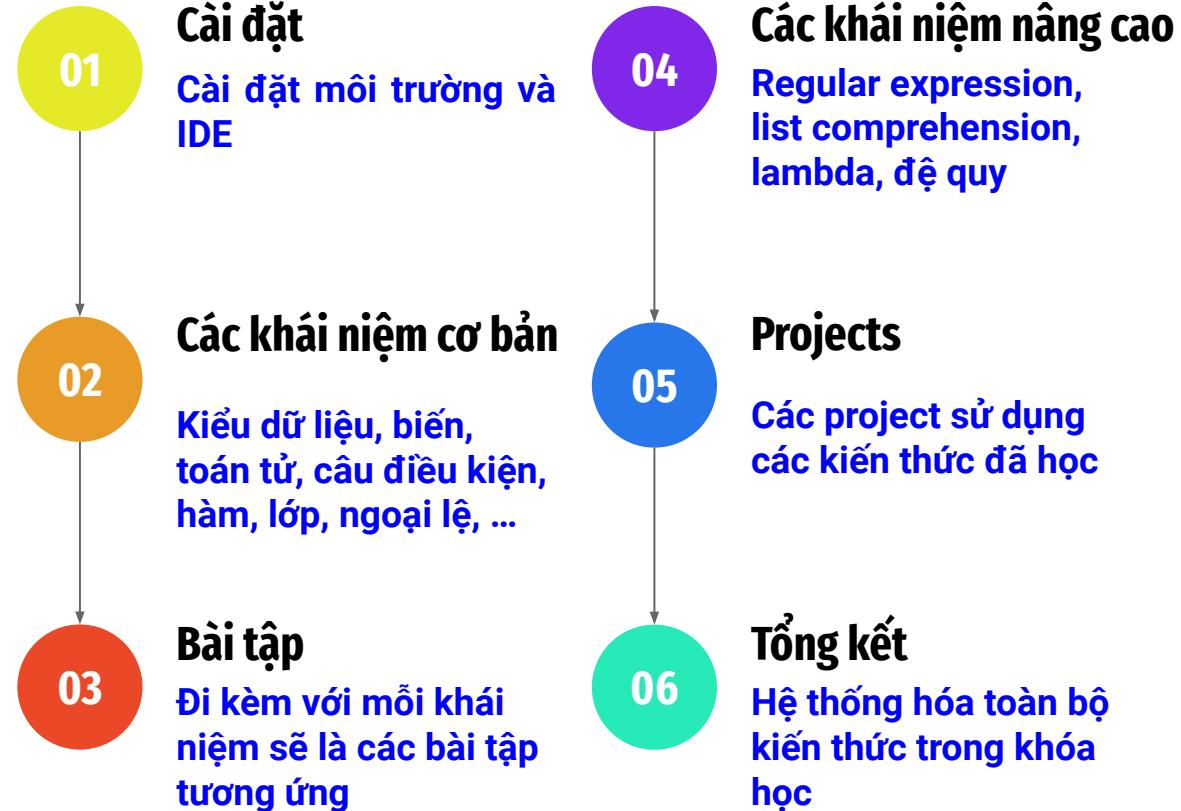
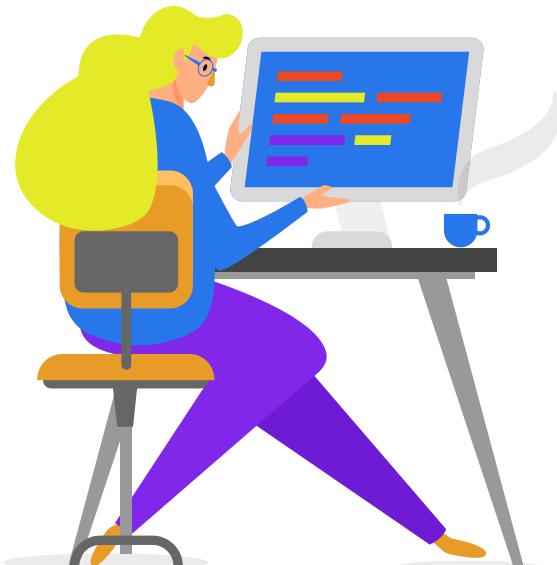


Basic Python Programming

Thang Nguyen
Viet Nguyen

<http://www.viet-it.com>

Nội dung



Cài đặt môi trường và IDE

Cài đặt Python

Bước 1: Vào <https://www.python.org/downloads/>



The screenshot shows the Python.org homepage. A red box highlights the URL bar with the address 'python.org/downloads/'. A red arrow points from the text 'Bước 1:' to this highlighted area. Below the URL bar, the Python logo and the word 'python' are displayed. The top navigation menu includes links for Python, PSF, Docs, PyPI, Jobs, and Community. The main content area features a large yellow button labeled 'Download Python 3.12.1'. This button is also highlighted with a red box, and a red arrow points from the text 'Bước 2:' to it. To the right of the button is a cartoon illustration of two boxes descending from the sky on parachutes.

Bước 2: Ấn vào nút Download

Cài đặt Python



Cài đặt Python

Python 3.12.1 (64-bit) Setup



Setup was successful

New to Python? Start with the [online tutorial](#) and [documentation](#). At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See [what's new](#) in this release, or find more info about [using Python on Windows](#).



Disable path length limit

Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

[Close](#)

Cài đặt Pycharm

A screenshot of a web browser window. The address bar shows the URL <https://www.jetbrains.com/pycharm/download/?section=windows>. The page content includes the PyCharm logo, navigation links like 'Use Cases', 'What's New', 'Features', 'Learn', 'Pricing', and a prominent green 'Download' button. A blue arrow points from the text 'Bước 1: Vào' to the browser's address bar.

Bước 1: Vào

<https://www.jetbrains.com/pycharm/download/?section=windows>

We value the vibrant Python community, and that's why we proudly offer the PyCharm Community Edition for free, as our open-source contribution to support the Python ecosystem.



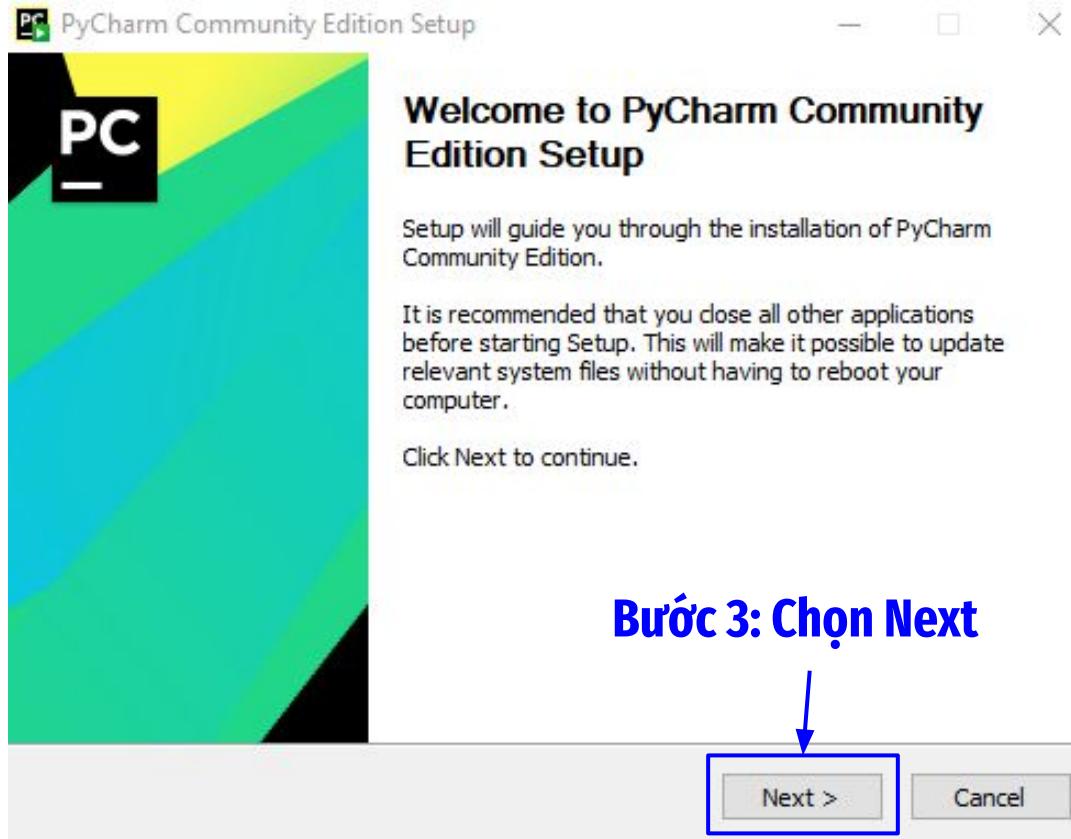
PyCharm Community Edition

The IDE for Pure Python Development



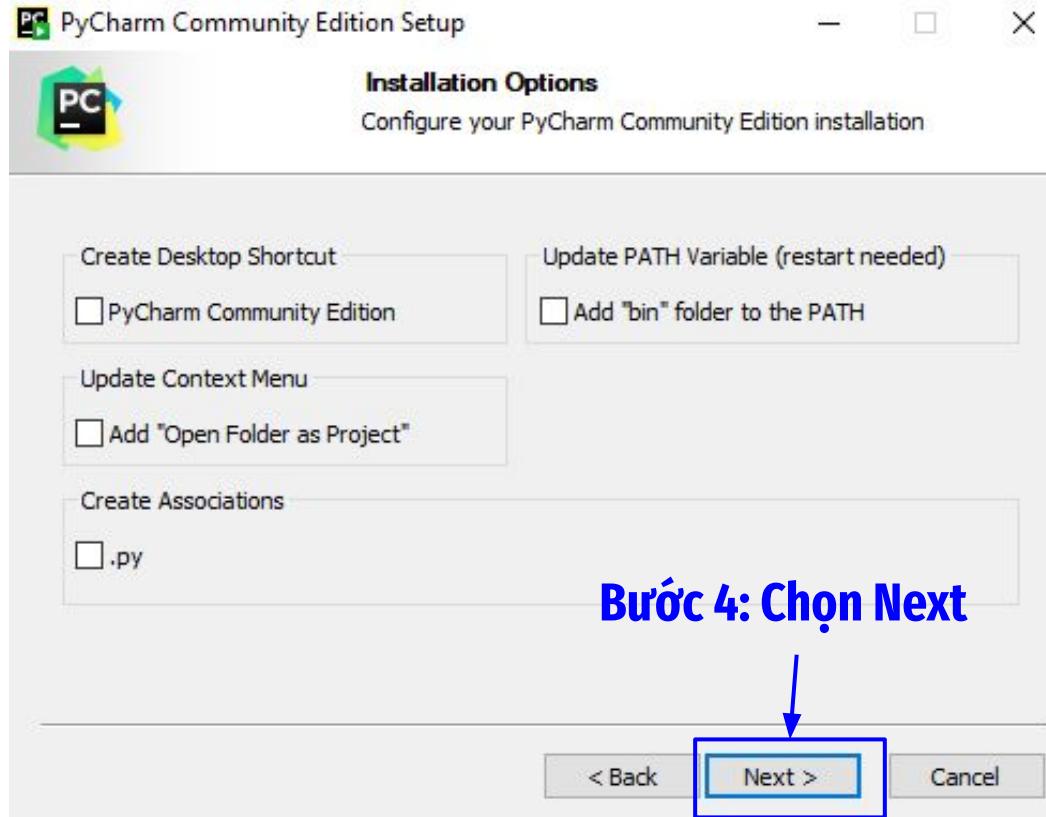
← Bước 2: Ấn vào nút Download

Cài đặt Pycharm

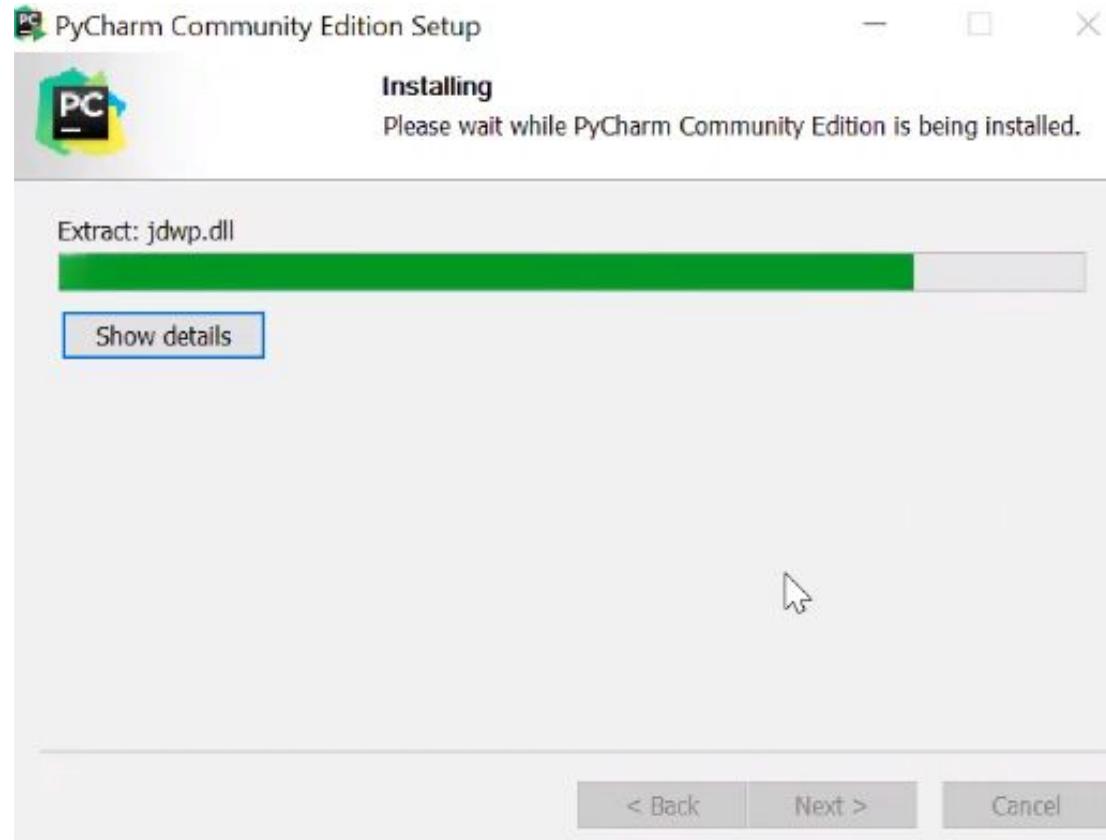


Bước 3: Chọn Next

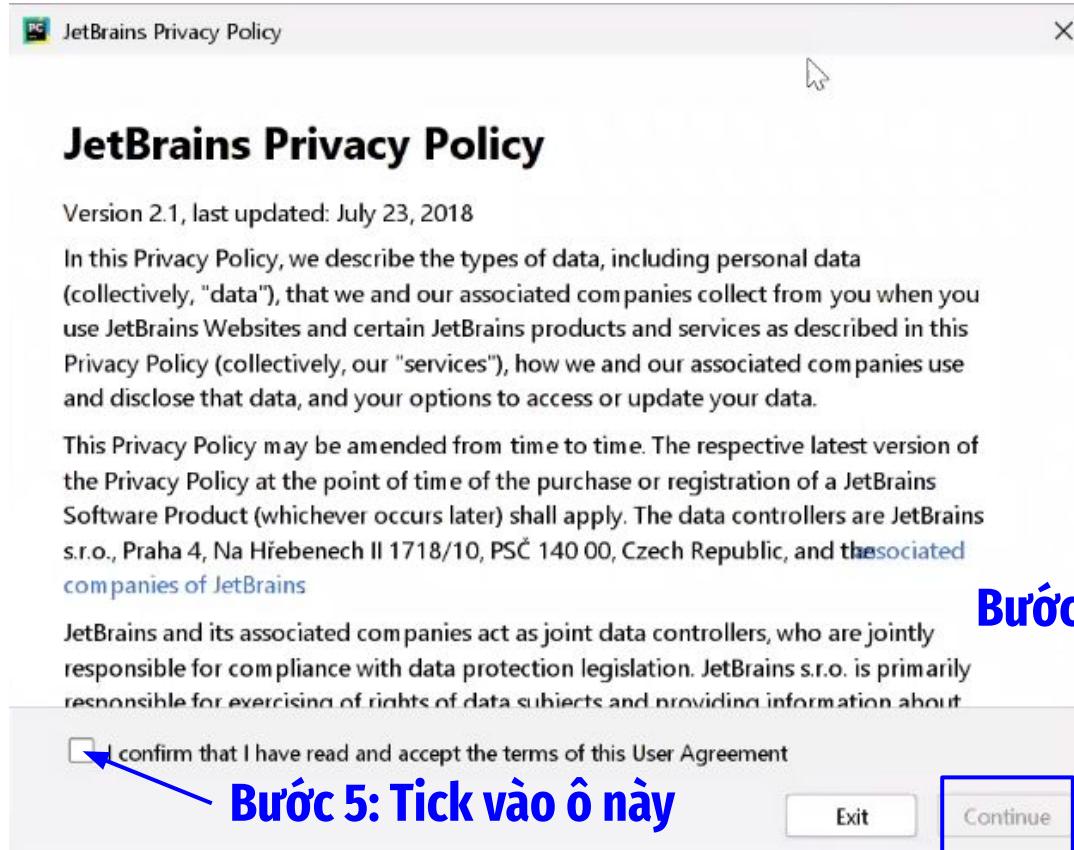
Cài đặt Pycharm



Cài đặt Pycharm



Cài đặt Pycharm



Các khái niệm cơ bản

Các khái niệm cơ bản

01 Kiểu dữ liệu và biến

02 Toán tử

03 Biểu thức điều kiện

04 So sánh

05 Vòng lặp

06 Hàm

Cấu trúc dữ liệu 07

Chuỗi ký tự 08

Lớp 09

Đọc và xuất dữ liệu 10

Xử lý ngoại lệ 11

Import 12



Kiểu dữ liệu và biến

Các kiểu dữ liệu (Data types)

Int (số nguyên)

-3, -2, -1, 0, 1, 2

Float (Số thực)

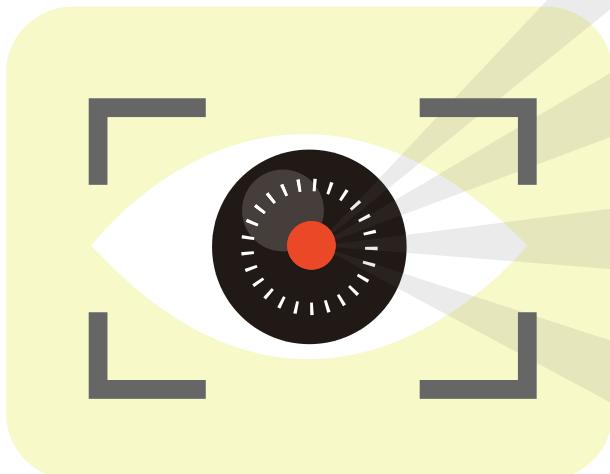
0.3, 1.2, -9.7

String (chuỗi)

“Forever Alone”

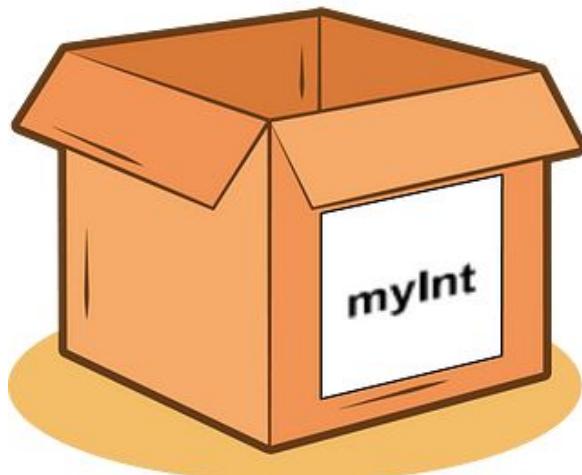
Boolean

True, False



Biến (Variable)

4



2.5



"hello"



Khai báo biến và gán giá trị

```
a = 7                      # Khai báo biến kiểu số nguyên  
b = 6.5                     # Khai báo biến kiểu số thực  
c = 'Today is Monday'      # Khai báo biến kiểu chuỗi (1)  
d = "Ronaldo and Messi"    # Khai báo biến kiểu chuỗi (2)  
e = ""  
  
    I could write the text  
    in many lines  
    without any problem  
"""  
f = True                    # Khai báo biến kiểu boolean  
f = "awesome"               # Gán giá trị mới cho biến f  
print(a)                   # 7  
print(type(f))              # <class 'str'>
```

Dùng hàm **print()** để in 1
biến/giá trị ra màn hình

Dùng hàm **type()** để kiểm tra
kiểu dữ liệu của 1 biến

Các quy tắc khi khai báo biến (1)

- Tên biến dài bao nhiêu cũng được **long_name_variable**
- Tên biến có thể chứa chữ cái, số và dấu gạch dưới (_) **sontung_mtp_123**
- Mono và mono là 2 biến khác nhau
- Tên biến phải bắt đầu bằng chữ cái hoặc dấu gạch dưới:
 - **crying_over_you** Tên biến hợp lệ
 - **crying over you** Tên biến không hợp lệ vì có khoảng trắng trong tên
 - **1990_restaurant** Tên biến không hợp lệ vì bắt đầu bằng chữ số

Các quy tắc khi khai báo biến (2)

Không được đặt tên biến trùng với tên các keywords

and	else	in	return
as	except	is	True
assert	finally	lambda	try
break	false	nonlocal	with
class	for	None	while
continue	from	not	yield
def	global	or	
del	if	pass	
elif	import	raise	

Ghi chú (Comments)

```
...  
a = 7          # Ta có thể ghi chú trên 1 dòng  
  
# Ta cũng có thể ghi chú ở ngay đầu dòng như thế này  
b = 5.5  
  
...  
Ta cũng có thể ghi chú trên nhiều dòng  
ở bên trong 3 cặp nháy đơn như thế này  
...  
  
c = "Learn Python"  
"""  
    Và tất nhiên ta cũng có thể ghi chú  
    trên nhiều dòng bên trong 3 cặp nháy đôi  
"""
```

**Ghi chú (comments) giúp người khác
và chính bản thân các bạn sau này
khi đọc lại code có thể dễ dàng hiểu
đoạn code này dùng để làm gì**

Toán tử

Các toán tử thường dùng

Toán tử số học

Arithmetic operators

01

Toán tử so sánh

Comparison operators

03

Toán tử với bit

Bitwise operators

05

Toán tử gán

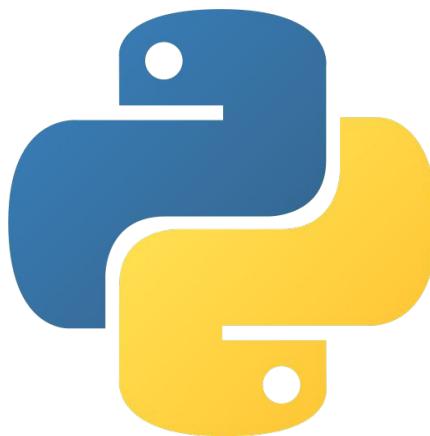
Assignment operators

02

Toán tử logic

Logical operators

04



06

Toán tử thành viên

Membership operators

Toán tử số học (Arithmetic Operators)

Toán tử	Ý nghĩa	Ví dụ	Kết quả
+	Phép cộng	4 + 5	9
-	Phép trừ	6 - 1.5	4.5
*	Phép nhân	4 * 2	8
/	Phép chia	15 / 2	7.5
//	Phép chia lấy nguyên	15 // 2	7
%	Phép chia lấy dư	15 % 2	1
**	Phép lũy thừa	2 ** 3	8

Toán tử gán (Assignment Operators)

Toán tử	Ý nghĩa	Ví dụ	Tương đương với
=	Gán giá trị bên phải dấu bằng cho biến bên trái dấu bằng	$x = 5$	$x = 5$
+=	Phép cộng và gán: Cộng giá trị bên phải dấu bằng vào biến bên trái dấu bằng và lưu kết quả vào biến đó	$x += 5$	$x = x + 5$
-=	Phép trừ và gán	$x -= 5$	$x = x - 5$
*=	Phép nhân và gán	$x *= 5$	$x = x * 5$
/=	Phép chia và gán	$x /= 2$	$x = x / 2$
//=	Phép chia lấy nguyên và gán	$x //= 2$	$x = x // 2$
%=	Phép chia lấy dư và gán	$x %= 5$	$x = x \% 5$
**=	Phép lấy lũy thừa và gán	$x **= 2$	$x = x ** 2$

Toán tử so sánh (Comparison Operators)

Toán tử	Ý nghĩa	Ví dụ	Kết quả
<code>==</code>	So sánh bằng	<code>6 == 6</code>	True
<code>!=</code>	So sánh không bằng	<code>6 != 6</code>	False
<code><</code>	So sánh nhỏ hơn	<code>4 < 5</code>	True
<code><=</code>	So sánh nhỏ hơn hoặc bằng	<code>4 <= 5</code>	True
<code>></code>	So sánh lớn hơn	<code>4 > 5</code>	False
<code>>=</code>	So sánh lớn hơn hoặc bằng	<code>4 >= 5</code>	False

Toán tử logic (Logic Operators)

Toán tử	Ý nghĩa	Ví dụ	Kết quả
and	Và: Nếu điều kiện ở về trái và về phải của toán tử đều là True thì kết quả là True . Tất cả các trường hợp khác kết quả là False	$5 > 4 \text{ and } 5 < 6$	True
		$5 > 4 \text{ and } 5 \geq 6$	False
or	Hoặc: Nếu điều kiện ở về trái và về phải của toán tử đều là False thì kết quả là False . Tất cả các trường hợp khác kết quả là True	$4 \geq 5 \text{ or } 5 \geq 6$	False
		$4 < 5 \text{ or } 5 == 6$	True
not	Phủ định: Đảo ngược trạng thái logic của toán hạng	$\text{not}(5 > 4)$	False
		$\text{not}(5 < 4)$	True

Toán tử thành viên (Membership Operators)

Toán tử	Ý nghĩa	Ví dụ	Kết quả
in	Nằm trong: Trả về True nếu giá trị đang kiểm tra nằm trong danh sách/chuỗi các giá trị	$x = "red"$ $y = ["red", "green", "blue"]$ $x \text{ in } y$	True
not in	Không nằm trong: Trả về True nếu giá trị đang kiểm tra KHÔNG nằm trong danh sách/chuỗi các giá trị	$x = "red"$ $y = ["red", "green", "blue"]$ $x \text{ not in } y$	False

Toán tử với bit (Bitwise Operators)

Toán tử	Ý nghĩa	Ví dụ
&	AND	$x \& y$
	OR	$x y$
^	XOR	$x ^ y$
~	NOT	$\sim x$
<<	Dịch trái	$x << 2$
>>	Dịch phải	$x >> 2$

Thứ tự ưu tiên của toán tử

Thứ tự ưu tiên giảm dần

Toán tử	Mô tả
()	Ngoặc đơn
**	Lũy thừa
*, /, %, //	Phép nhân/chia
+, -	Phép cộng, trừ
<, <=, >, >=	So sánh
==, !=	So sánh
=, +=, -=, *=, /=, %=, //=, **=	Phép gán
is, is not	Toán tử nhận dạng
in, not in	Toán tử thành viên
not, or, and	Toán tử logic

Các hàm cơ bản

- **In 1 giá trị/đối tượng gì đó ra màn hình**
- **Lưu trữ giá trị người dùng nhập vào từ bàn phím**
- **Trả về kiểu dữ liệu của 1 biến**

Code

```
print("Please enter your name: ")
name = input()
print("Name: ", name)
print("Type of input: ", type(name))
```

`print()`

`input()`

`type()`

Kết quả

```
Please enter your name:
Viet
Name: Viet
Type of input: <class 'str'>
```

Biểu thức điều kiện

Giá trị boolean và biểu thức boolean

```
...  
a = 4  
if a > 3:  
    b = True
```

Biểu thức boolean

Giá trị boolean

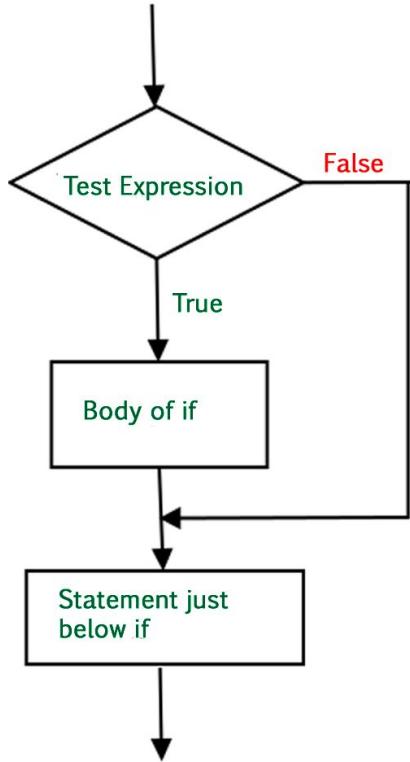
Thụt lề (Indentation)

```
a = 4
if a > 3:
    print("Start of block")
    b = True
    c = 5
    print("End of block")

print("Next step")
```

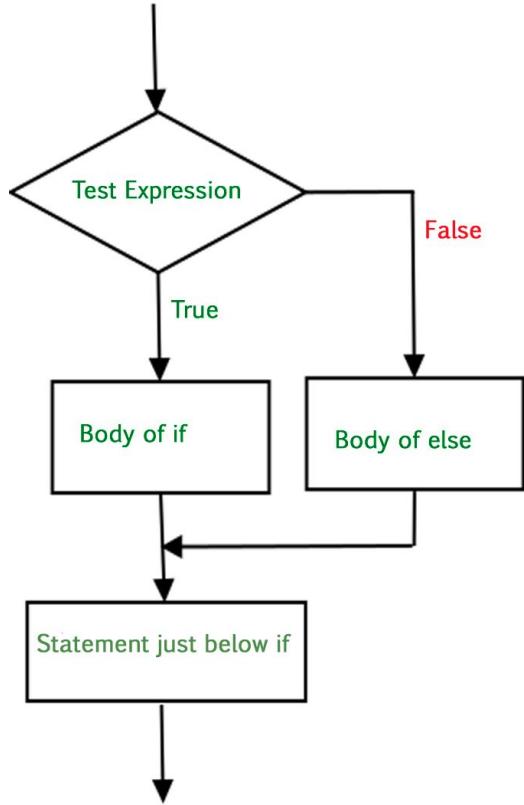
Thụt lề

Biểu thức if



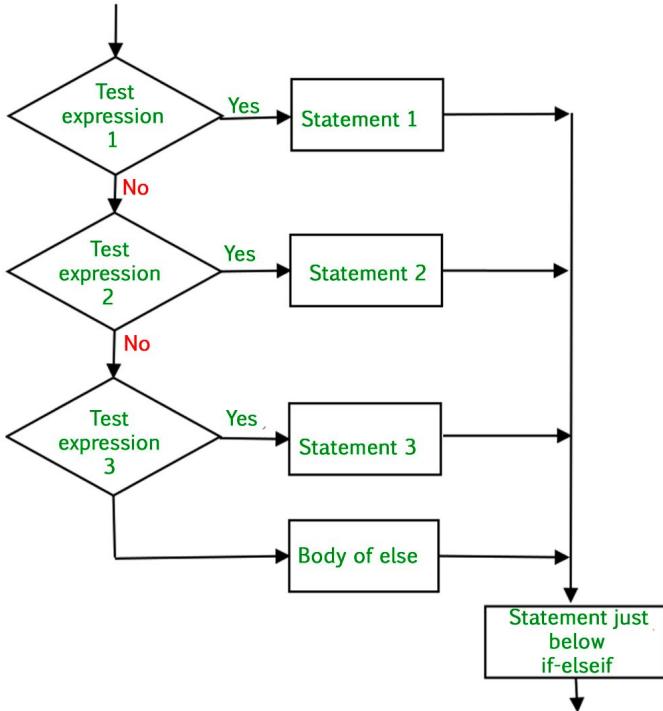
```
...  
if 5 > 3:  
    print("5 is greater than 3")  
print("Done")
```

Biểu thức if-else



```
...  
x = 5  
if x == 2:  
    print("Yes")  
else:  
    print("No")  
  
print("Next step")
```

Biểu thức if-elif-else



```
...  
day = "Monday"  
  
if day == "Monday":  
    print("Today is Monday")  
elif day == "Tuesday":  
    print("Today is Tuesday")  
elif day == "Wednesday":  
    print("Today is Wednesday")  
else:  
    print("Unknown day!")  
  
print("Next step")
```

Câu lệnh pass

```
...  
a = 4  
if a > 3:  
    pass  
    print("Next step")
```

Lỗi

Câu lệnh pass

```
...  
a = 4  
if a > 3:  
    pass  
print("Next step")
```

Biểu thức match-case

```
day = "Monday"

match day:
    case "Monday":
        print("Today is Monday")
    case "Tuesday":
        print("Today is Tuesday")
    case "Wednesday":
        print("Today is Wednesday")
    case "Thursday":
        print("Today is Thursday")
    case "Friday":
        print("Today is Friday")
    case "Saturday":
        print("Today is Saturday")
    case "Sunday":
        print("Today is Sunday")
    case _:
        print("Invalid day")
```

Biểu thức if rút gọn

```
...  
x = 5  
if x % 2 == 0:  
    print("Even number")
```

↓ Rút gọn

```
...  
x = 5  
if x % 2 == 0: print("Even number")
```

Toán tử 3 ngôi (ternary operator)

```
...  
x = 5  
if x % 2 == 0:  
    number = "Even"  
else:  
    number = "Odd"
```

Biểu thức 1 if điều kiện else biểu thức 2

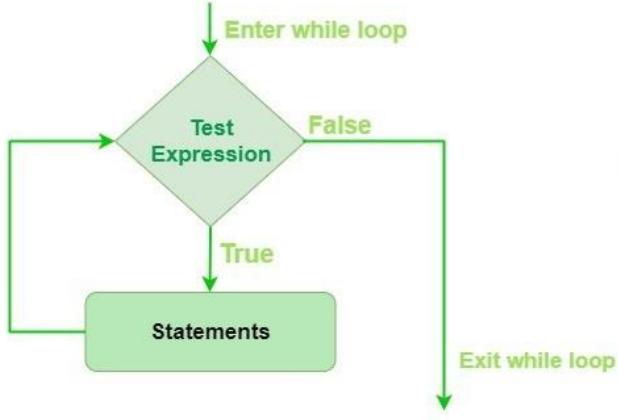
```
...  
x = 5  
number = "Even" if x % 2 == 0 else "Odd"
```

Biểu thức if lồng nhau (Nested if statement)

```
x = 6
if x % 2 == 0:
    if x % 3 == 0:
        print("x is divisible by 6")
    else:
        print("x is divisible by 2 but not by 3")
else:
    if x % 3 == 0:
        print("x is divisible by 3 but not by 2")
    else:
        print("x is not divisible by either 2 or 3")
```

Vòng lặp

Vòng lặp while



```
number = 1
while number < 100:
    print(number)
    number = number * 2    # number *= 2
```

A screenshot of a terminal window showing the execution of the provided Python code. The code initializes a variable `number` to 1 and enters a `while` loop. Inside the loop, it prints the value of `number` and then multiplies it by 2. The output shows the values 1, 2, 4, 8, 16, 32, and 64, each preceded by three gray dots, indicating the loop continues until `number` reaches 100. A red arrow points from the terminal window down to the first output value, "1".

```
1
2
4
8
16
32
64
```

Hàm range()

range(begin, end, step)

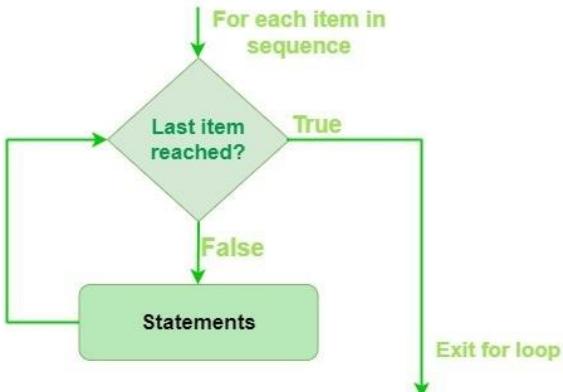
range(1, 10, 2) → 1;3;5;7;9

range(1, 10) → 1;2;3;4;5;6;7;8;9

range(10) → 0;1;2;3;4;5;6;7;8;9

range(10, 1, -2) → 10;8;6;4;2

Vòng lặp for

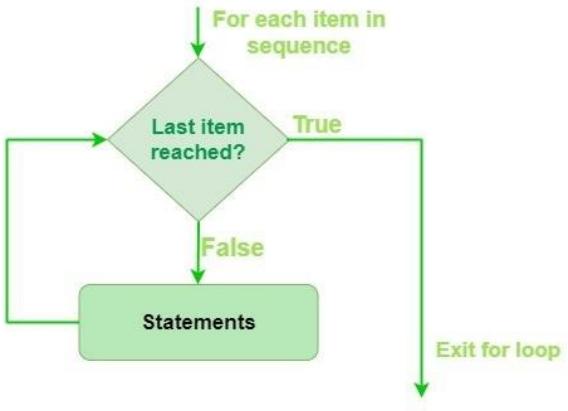


```
for i in range(10):  
    print(i)
```

The output window shows the numbers 0 through 9, each on a new line, indicating the execution of the print statements within the for loop. A red arrow points from the bottom of the code block to the start of the output list.

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Vòng lặp for



```
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(fruit)
```

```
apple
banana
orange
```

while vs for

while

```
...  
number = 1  
while number < 100:  
    print(number)  
    number = number * 2    # number *= 2
```

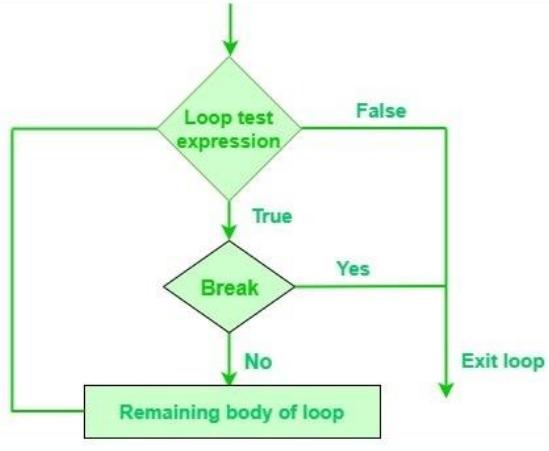
for

```
...  
for i in range(10):  
    print(i)
```

Số lượng bước lặp KHÔNG được
biết trước khi chương trình chạy

Số lượng bước lặp được biết trước
khi chương trình chạy

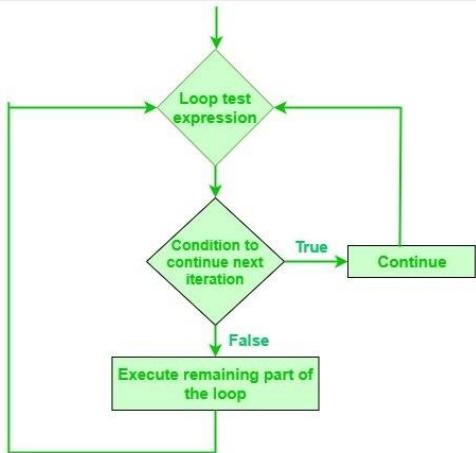
Câu lệnh break



```
...  
for i in [12, 16, 17, 24, 29]:  
    if i % 2 == 1: # If the number is odd  
        break         # ... immediately exit the loop  
    print(i)  
print("done")
```

```
...  
12  
16  
done
```

Câu lệnh continue



```
for char in "Hiphop never dies":  
    if char == "e":  
        continue  
    print(char)
```

A terminal window displays the output of the provided Python code. The word 'Hiphop' is printed vertically, followed by three dots, and then 'never dies' is printed vertically. A red arrow points from the terminal window to the 'print(char)' line in the code block above.

```
H  
i  
p  
h  
o  
p  
n  
v  
r  
e  
v  
e  
r  
n  
e  
r  
d  
i  
s
```

break vs continue

break

```
for i in [12, 16, 17, 24, 29]:  
    if i % 2 == 1: # If the number is odd  
        break      # ... immediately exit the loop  
    print(i)  
print("done")
```



Ngừng vòng lặp NGAY lập tức

continue

```
for char in "Hiphop never dies":  
    if char == "e":  
        continue  
    print(char)
```



Dừng bước lặp hiện tại đang dang dở
để nhảy luôn đến lần lặp tiếp theo

Vòng lặp với else

for-else

```
...  
  
for char in "Hiphop never dies":  
    if char == "e":  
        continue  
    print(char)  
else:  
    print("End")
```

while-else

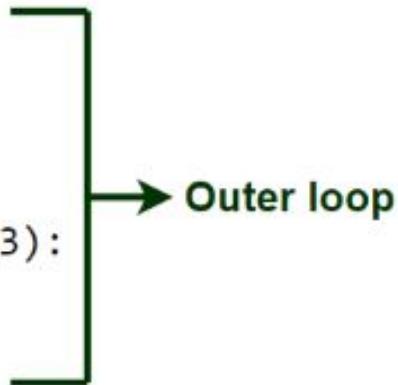
```
...  
  
number = 1  
while number < 100:  
    print(number)  
    number = number * 2  
else:  
    print("End")
```

Được thực hiện nếu vòng lặp không kết thúc vì câu lệnh break

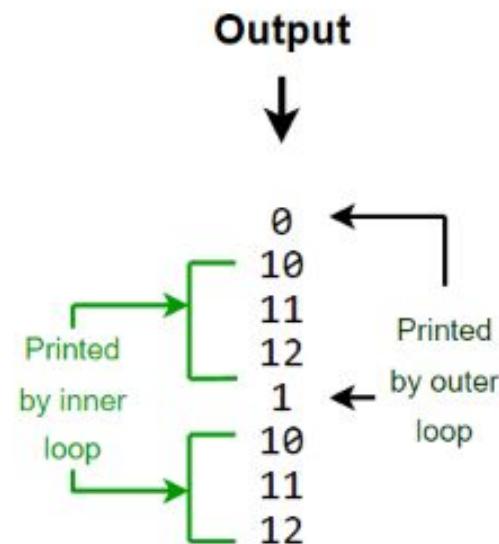
Vòng lặp lồng nhau (Nested loop)

```
for i in range(2):  
    print(i)  
    for j in range(10,13):  
        print(j)
```

Inner loop



Outer loop



Output



0

10

11

12

1

10

11

12

Printed
by inner
loop

Printed
by outer
loop

Hàm

Ví dụ

```
def calculate_sum_of_squares(n):
    result = 0
    for i in range(n + 1):
        result += i ** 2
    return result

if __name__ == '__main__':
    result = calculate_sum_of_squares(3)
    print(result)
    result = calculate_sum_of_squares(8)
    print(result)
```



14

204

Hàm thư viện

Các loại hàm

Hàm người dùng định nghĩa

```
...  
from math import pow  
  
if __name__ == '__main__':  
    result = pow(4, 3)  
    print(result)
```

Khai báo thư viện sử dụng

```
...  
def calculate_power(a, n):  
    return a ** n
```

```
...  
if __name__ == '__main__':  
    result = calculate_power(4., 3)  
    print(result)
```

Tự định nghĩa hàm của bản thân

64.0

Định nghĩa hàm (function definition)

```
def function_name([parameters]):  
    statements  
    [return value]
```

```
...  
def hello():  
    print("Hello, World!")
```

```
...  
def sum_function(a, b):  
    return a + b
```

Gọi hàm

def function_name([parameters]):

statements

[return value]

Tham số (parameters)

```
def hello():
    print("Hello, World!")

if __name__ == '__main__':
    hello() # Hello, World!
```

```
def sum_function(a, b):
    return a + b

if __name__ == '__main__':
    sum_value = sum_function(3, 6)
    print(sum_value) #9
```

Đối số (arguments)

Đối số

Có 5 kiểu đối số trong Python:

- **Đối số vị trí (Positional Arguments)**
- **Đối số từ khóa (Keyword Arguments)**
- **Đối số mặc định (Default Arguments)**
- **Đối số vị trí có độ dài thay đổi (Variable Length Positional Arguments)**
- **Đối số từ khóa có độ dài thay đổi (Variable Length Keyword Arguments)**

Đối số vị trí (Positional arguments)

```
def func(name, job):
    print(name, "is a", job)

func("Bob", "developer")
# Prints Bob is a developer
```

Đối số từ khóa (Keyword arguments)

```
def func(name, job):
    print(name, "is a", job)

func(name="Bob", job="developer")
# Bob is a developer

func(job="developer", name="Bob")
# Bob is a developer
```

Kết hợp đối số vị trí và đối số từ khóa

```
def func(name, job, location):
    print(name, "is a", job, "in", location)

func("Bob", location="London", job="developer")
# Bob is a developer in London
```

Đối số mặc định (Default arguments)

```
...  
  
def func(name, job="developer"):  
    print(name, "is a", job)  
  
  
func("Bob", "manager")  
# Bob is a manager  
  
  
func("Bob")  
# Bob is a developer
```

Giá trị trả về

1 hàm bất kỳ trong Python có
thể trả về:

- 0 giá trị
- 1 giá trị
- Nhiều giá trị

```
def print_sum(a, b):
    print(a + b)

def get_sum(a, b):
    return a + b

def get_math_ops(a, b):
    return a + b, a - b, a * b, a / b

if __name__ == '__main__':
    print_sum(3, 4) #7
    result = get_sum(3, 4)
    print(result) #7
    add, sub, mul, div = get_math_ops(3, 4)
    print(add, sub, mul, div) #7 -1 12 0.75
```

Chuỗi tài liệu (Docstring)

Docstring giúp người khác thuận lợi hơn trong việc sử dụng các hàm do chúng ta viết

```
...  
def get_sum(a, b):  
    """  
        Sum up 2 number and return the result  
    """  
    return a + b
```



Chuỗi tài liệu (Docstring)

1 vài kiểu docstring phổ biến trong Python:

- **Kiểu 3 dấu nháy kép (Triple-Quoted Docstrings)**
- **Kiểu Google (Google Style Docstrings)**
- **Kiểu Numpy (Numpy Style Docstrings)**

Chuỗi tài liệu 3 dấu nháy kép

```
def get_sum(a, b):
    """
    Sum up 2 number and return the result
    """
    return a + b
```

Chuỗi tài liệu kiểu Google

```
def get_sum(a, b):
    """
    This function adds two numbers.

    Args:
        a: The first number.
        b: The second number.

    Returns:
        The sum of the two numbers.
    """
    return a + b
```

Chuỗi tài liệu kiểu Numpy

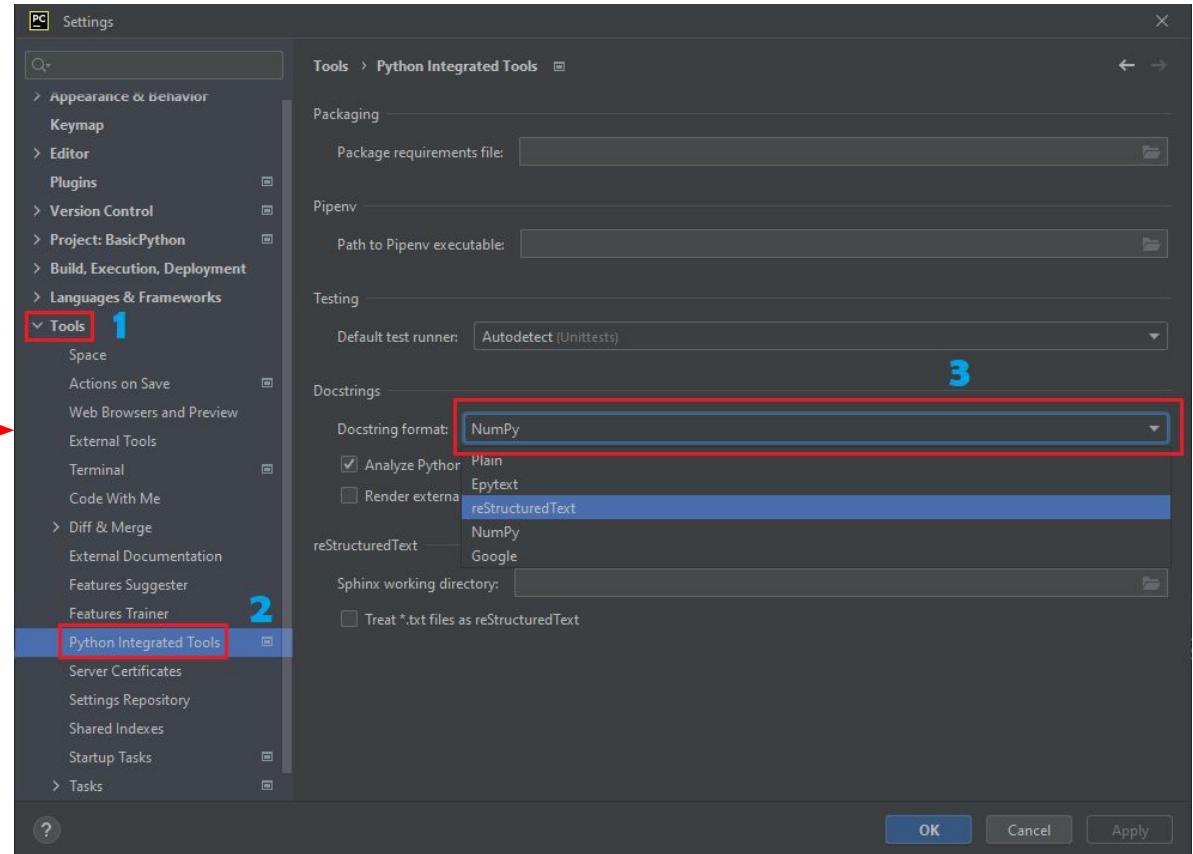
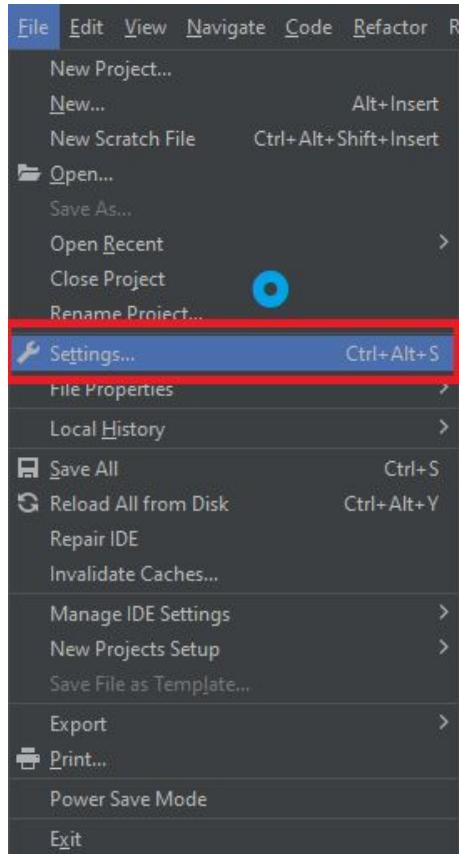
```
def get_sum(a, b):
    """
    This function adds two numbers.

    Parameters
    -----
    a: The first number.
    b: The second number.

    Returns
    -----
    The sum of the two numbers.

    """
    return a + b
```

Chọn kiểu chuỗi tài liệu với Pycharm



Các hàm tích hợp sẵn (builtin functions)

Hàm	Mô tả	Ví dụ	Kết quả
print()	Hiển thị dữ liệu ra màn hình	print("Today")	Today
len()	Trả về độ dài của dữ liệu	len("Today")	5
		len([1, 3, 6, 4, 2, 1])	6
range()	Tạo ra 1 dải giá trị liên tục từ 0	range(5)	0, 1, 2, 3, 4
str()	Chuyển đổi giá trị số thành chuỗi ký tự	str(5.5)	5.5
int()	Chuyển đổi giá trị chuỗi ký tự thành số	int("5.5")	5.5
float()	Chuyển đổi giá trị số nguyên thành số thực	float(5.5)	5
type()	Trả về kiểu của dữ liệu	type("Python")	str

Các hàm tích hợp sẵn (builtin functions)

Hàm	Mô tả	Ví dụ	Kết quả
min()	Trả về số nhỏ nhất trong 1 danh sách	min([1, 3, 6, 4, 2, 1])	1
max()	Trả về số lớn nhất trong 1 danh sách	max([1, 3, 6, 4, 2, 1])	6
sorted()	Sắp xếp danh sách theo thứ tự tăng(giảm) dần	sorted([3, 4, 2, 1])	[1, 2, 3, 4]
		sorted([3, 4, 2, 1], reverse=True)	[4, 3, 2, 1]
input()	Nhận giá trị từ người dùng thông qua bàn phím	input("Enter your name: ")	
sum()	Trả về tổng của các số trong 1 danh sách	sum([1, 3, 6, 4, 2, 1])	17
abs()	Trả về giá trị tuyệt đối của 1 số	abs(-6)	6
round()	Làm tròn 1 số thực đến 2 chữ số phần thập phân	round(3.14159, 2)	3.14

Các giá trị hằng số trong module **math**

```
from math import e, pi, tau, inf, nan
```

Kí hiệu	Mô tả	Giá trị
e	Số euler	~ 2.7182
pi	Số pi	~ 3.1415
tau	Số tau	~ 6.2831
inf	Dương vô cùng	
nan	Đây không phải là 1 số hợp lệ	

Các hàm toán học trong thư module math

```
from math import sqrt, pow, log, log2, log10, exp, degrees, radians
```

Hàm	Mô tả	Ví dụ	Kết quả
sqrt()	Trả về căn bậc 2 của 1 số	sqrt(16)	4.0
pow()	Trả về lũy thừa của 1 số	pow(4, 3)	64.0
log()	Trả về loga cơ số tự nhiên (loga nepe) của 1 số	log(3)	1.0986
log2()	Trả về loga cơ số 2 của 1 số	log2(8)	3.0
log10()	Trả về loga cơ số 10 của 1 số	log10(10000)	4.0
exp()	Trả về giá trị mũ với cơ số tự nhiên của 1 số	exp(2)	7.3890

Các hàm trong module **random**

```
from random import randint, random, choice, shuffle
```

Hàm	Mô tả	Ví dụ
randint()	Trả về 1 số tự nhiên ngẫu nhiên nằm giữa 2 số (bao gồm cả 2 đầu)	randint(3, 9)
random()	Trả về 1 số thực ngẫu nhiên nằm giữa 0.0 và 1.0	random()
choice()	Trả về 1 phần tử ngẫu nhiên từ 1 danh sách	letters = ["a", "b", "c"] choice(letters)
shuffle()	Trộn ngẫu nhiên 1 danh sách	shuffle(letters) -> ["c", "a", "b"]

Cấu trúc dữ liệu

Các kiểu cấu trúc dữ liệu

Có 4 kiểu cấu trúc dữ liệu cơ bản trong Python:

- List
- Dictionary
- Tuple
- Set

List

List là 1 chuỗi của các phần tử (items/elements). List có các tính chất:

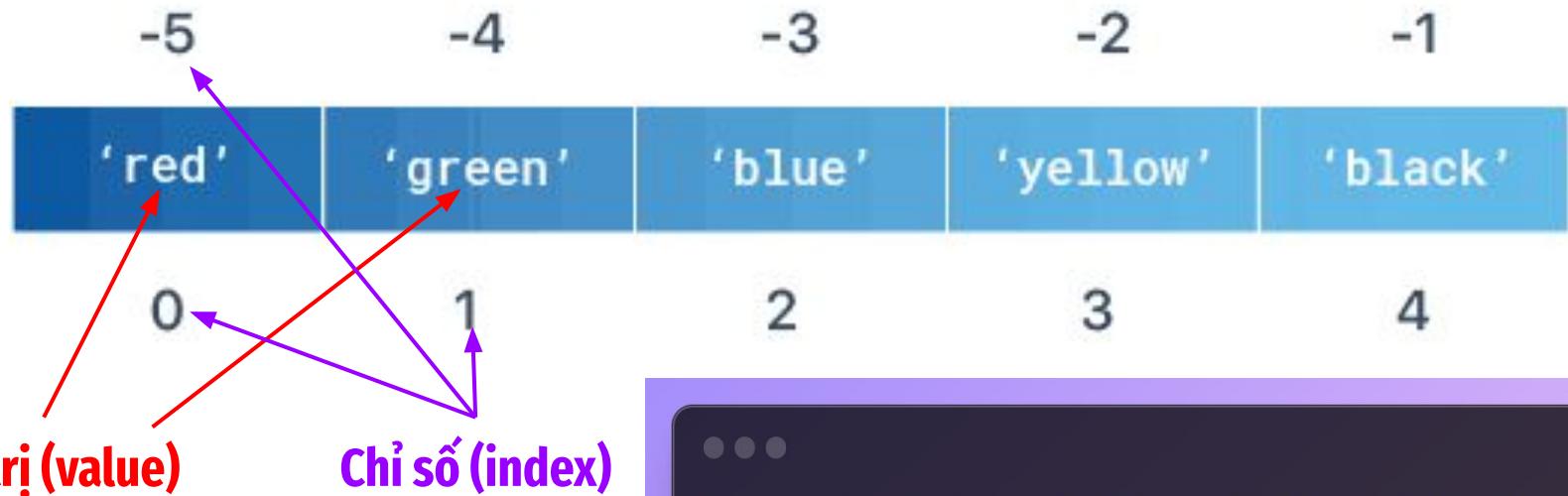
'red'	'green'	'blue'	'yellow'	'black'
-------	---------	--------	----------	---------

- **Có thứ tự:** Thứ tự của các phần tử chính là thứ tự chúng được gán vào list
- **Được truy cập bởi chỉ số (index):** Các phần tử của list có thể được truy cập bởi chỉ số của chúng
- **Chứa bất kỳ đối tượng nào:** Các phần tử của list có thể là bất kỳ đối tượng nào (số, chuỗi, list, các cấu trúc dữ liệu khác, ...)
- **Khả biến (thay đổi được):** Chúng ta có thể thay đổi (thêm, xóa, cập nhật) giá trị các phần tử của list bất kỳ lúc nào

Khai báo list

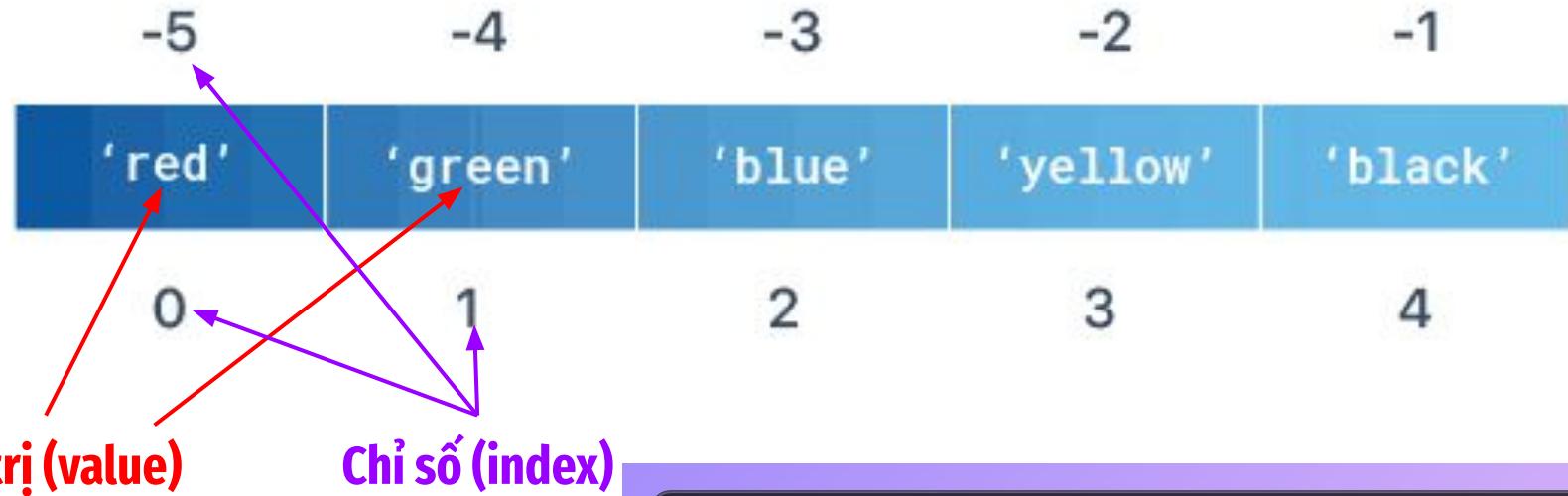
Mục đích	Câu lệnh
Khai báo 1 list rỗng	<code>data = []</code>
Khai báo 1 list có các giá trị nguyên	<code>data = [2, 0, 3, 1, -5, 2]</code>
Khai báo 1 list có các giá trị chuỗi	<code>data = ["red", "green", "blue"]</code>
Khai báo 1 list có các giá trị hỗn hợp	<code>data = [-2, 5.0, "Monday", [1, -2, 3]]</code>
Khai báo 1 list có các giá trị giống nhau	<code>data = [5]*6 (= [5, 5, 5, 5, 5, 5])</code>
Biến đổi 1 chuỗi thành list	<code>data = list("abc") (= ["a", "b", "c"])</code>
List lồng nhau (Nested list)	<code>data = [[1, -2, 3], [2, -1], ["a", -1.5, 2, "b"]]</code>

Truy cập phần tử trong List



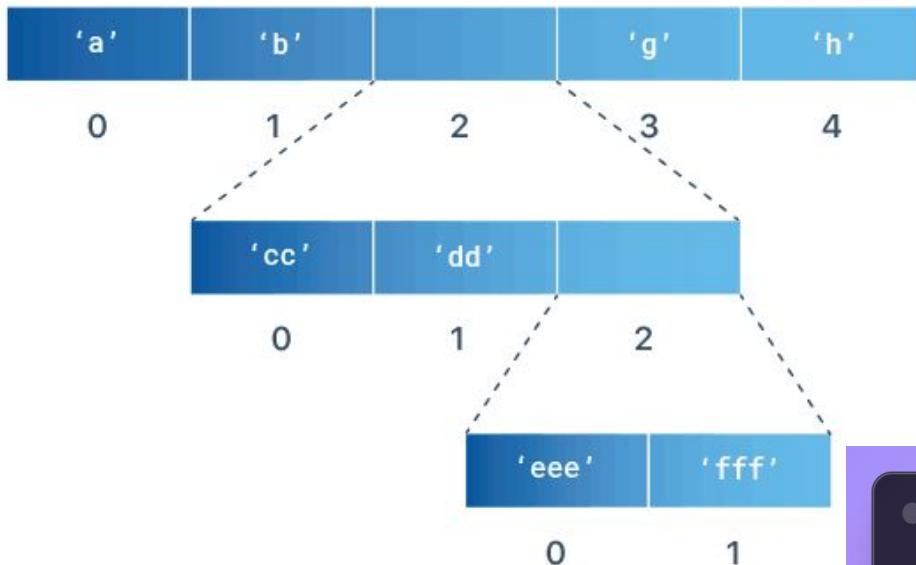
```
...  
data = ["red", "green", "blue", "yellow", "black"]  
  
print(data[3])      # yellow  
print(data[-2])    # yellow  
print(data[5])     # IndexError: list index out of range
```

Thay đổi giá trị của phần tử trong List



```
...  
data = ["red", "green", "blue", "yellow", "black"]  
  
data[3] = "orange"  
print(data) # ['red', 'green', 'blue', 'orange', 'black']
```

Truy cập phần tử trong Nested List



```
...
data = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']

print(data[2][2])      # ['eee', 'fff']
print(data[2][2][0])   # eee
```

Cắt List (Slice List)

`list_name[begin:end:step]`

`data[1] data[2] data[3] data[4]`

```
data = ["red", "green", "blue", "yellow", "black"]
print(data[1:4]) # ['green', 'blue', 'yellow']
```

Cắt List (Slice List)

```
data = [2, 4, 8, 16, 32, 64, 128, 256, 512]

print(data)                  # [2, 4, 8, 16, 32, 64, 128, 256, 512]
print(data[0:3])            # [2, 4, 8]
print(data[4:7])            # [32, 64, 128]
print(data[-6:-2])          # [16, 32, 64, 128]
print(data[:3])              # [2, 4, 8]
print(data[4:])              # [32, 64, 128, 256, 512]
print(data[:])                # [2, 4, 8, 16, 32, 64, 128, 256, 512]
print(data[-50:4])           # [2, 4, 8, 16]
print(data[3:100])           # [16, 32, 64, 128, 256, 512]
print(data[2:-2:2])          # [8, 32, 128]
print(data[::-2])             # [2, 8, 32, 128, 512]
print(data[::-1])             # [512, 256, 128, 64, 32, 16, 8, 4, 2]
```

Thêm phần tử vào List (1)

`list_name.insert(index, item)`

Thêm `item` vào `list_name` tại vị trí `index`

```
...
```

```
data = [2, 4, 6]
data.insert(1, 10)
print(data)    # [2, 10, 4, 6]
```

Chú ý: Hàm `insert()` trả về `None`

Thêm phần tử vào List (2)

list_name.append(item)

Thêm item vào cuối list_name

```
...  
data = [2, 4, 6]  
data.append(10)  
print(data)
```

Chú ý: Hàm append() trả về None

Thêm phần tử vào List (3)

`list_name.extend(iterable)`

Chèn tất cả các phần tử trong `iterable` vào cuối `list_name`

...

```
data = [2, 4, 6]
data.extend([10, 20])
print(data)      # [2, 4, 6, 10, 20]
```

Iterable là đối tượng mà ta có thể
duyệt qua các phần tử của nó

Chú ý: Hàm `extend()` trả về `None`

Thêm phần tử vào List (4)

Sử dụng toán tử +

```
...  
data = [2, 4, 6]  
data = data + [10, 20]  
print(data)      # [2, 4, 6, 10, 20]  
data += [100, 200]  
print(data)      # [2, 4, 6, 10, 20, 100, 200]
```

Xóa phần tử khỏi List (1)

`list_name.pop(index)`

Xóa phần tử tại vị trí `index` khỏi `list_name`

```
...  
data = [2, 4, 6]  
removed_item = data.pop(1)  
print(data)          # [2, 6]  
print(removed_item) # 4
```

Xóa phần tử khỏi List (2)

`del list_name[index]`

Xóa phần tử tại vị trí `index` khỏi `list_name`

```
...  
data = [2, 4, 6]  
del data[2]  
print(data)      # [2, 4]
```

Xóa phần tử khỏi List (3)

`del list_name[begin:end]`

Xóa nhiều phần tử liên tiếp khỏi `list_name` với `slicing`

```
...  
data = [2, 4, 6, 8, 10]  
del data[1:3]  
print(data)      # [2, 8, 10]
```

Xóa phần tử khỏi List (4)

`list_name.remove(item)`

Xóa phần tử `item` đầu tiên khỏi `list_name`

```
...  
data = [2, 4, 6, 7, 6]  
data.remove(6)  
print(data)          # [2, 4, 7, 6]  
data.remove(3)      # ValueError: list.remove(x): x not in list
```

Xóa phần tử khỏi List (5)

`list_name.clear()`

Xóa tất cả phần tử khỏi `list_name`

```
...  
data = [2, 4, 6, 8, 10]  
data.clear()  
print(data)      # []
```

Tính số phần tử của List

`len(list_name)`

...

```
data = [2, 4, 6, 8, 10]
print(len(data))      # 5
```

Kiểm tra sự tồn tại của giá trị trong List

`if item in list_name`

The diagram shows a mobile phone screen displaying Python code. A green arrow points from the text "Kiểm tra sự tồn tại của giá trị trong List" at the top to the word "item" in the code. A red arrow points from the text "Kiểm tra sự tồn tại của giá trị trong List" at the top to the word "list_name" in the code.

```
data = [2, 4, 6, 8, 10]
if 6 in data:
    print("1A")
else:
    print("1B")

if 5 in data:
    print("2A")
else:
    print("2B")

# 1A
# 2B
```

Tìm vị trí của 1 phần tử trong List

`list_name.index(item)`

Tìm vị trí của phần tử `item` đầu tiên trong `list_name`

```
...  
data = ["red", "green", "blue"]  
print(data.index("red"))          # 0  
print(data.index("black"))       # ValueError: 'black' is not in list
```

Duyệt qua các phần tử của List (1)

```
...  
data = ["red", "green", "blue"]  
for color in data:  
    print(color)
```



```
...  
red  
green  
blue
```

Duyệt qua các phần tử của List (2)

```
...  
data = ["red", "green", "blue"]  
for i in range(len(data)):  
    data[i] += " color"  
print(data)
```



```
...  
['red color', 'green color', 'blue color']
```

Đảo ngược thứ tự của List (1)

list_name.reverse()

```
...  
data = ["red", "green", "blue"]  
data.reverse()  
print(data)
```



```
...  
['blue', 'green', 'red']
```

Đảo ngược thứ tự của List (2)

list_name.reverse()

```
...  
data = ["red", "green", "blue"]  
print(data[::-1])
```



```
...  
['blue', 'green', 'red']
```

Sắp xếp các phần tử của List (1)

list_name.sort()

```
...  
data = [3, 1, 4, 8]  
data.sort()  
print(data)
```



```
[1, 3, 4, 8]
```

Sắp xếp các phần tử của List (2)

sorted(list_name)

```
...  
data = [3, 1, 4, 8]  
data = sorted(data)  
print(data)
```



```
[1, 3, 4, 8]
```

Đếm số lần 1 giá trị xuất hiện trong List

`list_name.count(value)`

Đếm số lần giá trị `value` xuất hiện trong `list_name`

...

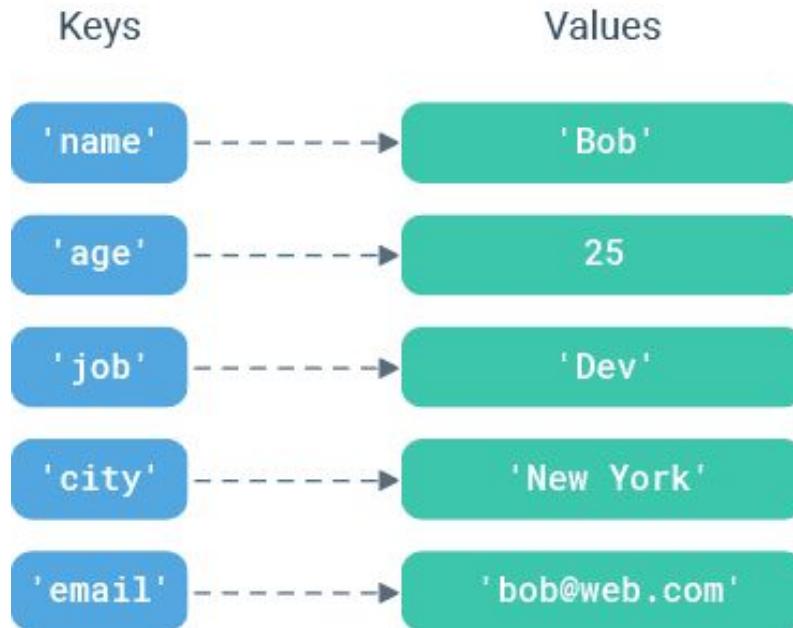
```
data = ["red", "green", "blue", "red"]
print(data.count("red"))          # 2
print(data.count("black"))        # 0
```

Tổng kết các hàm tích hợp sẵn với List

Hàm	Mô tả
Insert()	Thêm 1 phần tử vào 1 vị trí cụ thể trong List
append()	Thêm 1 phần tử vào cuối List
extend()	Thêm tất cả các phần tử trong 1 iterable vào cuối List
pop()	Xóa 1 phần tử ở 1 vị trí cụ thể khỏi List
remove()	Xóa phần tử đầu tiên với giá trị cụ thể khỏi List
clear()	Xóa tất cả các phần tử khỏi List
index()	Trả về vị trí của phần tử đầu tiên với giá trị cụ thể trong List
reverse()	Đảo ngược thứ tự các phần tử trong List
sort()/sorted()	Sắp xếp các phần tử trong List theo thứ tự tăng(giảm) dần
count()	Đếm số lần 1 giá trị xuất hiện trong List

Dictionary

Dictionary là 1 tập hợp các phần tử KHÔNG có thứ tự, được lưu trữ dưới dạng các cặp khóa (key) và giá trị (value)



Khai báo dictionary

Mục đích	Câu lệnh
Khai báo 1 dictionary rỗng	<code>data = {}</code>
Khai báo 1 dictionary với các cặp key-value	<code>data = {"name": "Viet", "age": 32, "job": "AI engineer"}</code>
Khai báo 1 dictionary với 1 list của các tuples	<code>data = dict([("name", "Viet"), ("age", 32), ("job", "AI engineer")])</code>
Khai báo 1 dictionary với 1 tuple của các lists	<code>data = dict((["name", "Viet"], ["age", 32], ["job", "AI engineer"]))</code>
Khai báo 1 dictionary với các đối số là key	<code>data = dict(name="Viet", age=32, job="AI engineer")</code>

Các tính chất của Dictionary

Key phải là duy nhất

- 1 key bất kỳ chỉ được xuất hiện đúng 1 lần
- Nếu 1 key được khai báo nhiều hơn 1 lần, value cuối cùng đi kèm với key đó sẽ được sử dụng

Key phải bất biến

- Key của Dictionary phải bất biến (immutable)
- List là có thể thay đổi (mutable), do đó không thể sử dụng List làm key trong Dictionary

Value không có giới hạn gì

- Value trong Dictionary có thể thuộc bất kỳ kiểu dữ liệu gì
- Value trong Dictionary có thể xuất hiện nhiều lần

Các tính chất của Dictionary

Key phải là duy nhất

- 1 key bất kỳ chỉ được xuất hiện đúng 1 lần
- Nếu 1 key được khai báo nhiều hơn 1 lần, value cuối cùng đi kèm với key đó sẽ được sử dụng

```
...  
data = {"name": "Viet", "age": 32, "name": "Thang"}  
print(data)      # {'name': 'Thang', 'age': 32}
```

Các tính chất của Dictionary

Key phải bất biến

```
data = {  
    "name": "Viet",  
    True: "abc",  
    5: False,  
    (1, 2): 5  
}
```

- Key của Dictionary phải **bất biến** (immutable)
- List là **khả biến** (mutable), do đó **KHÔNG** thể sử dụng List làm key trong Dictionary

```
data = {[1, 2]: "5"}  
# TypeError: unhashable type: 'list'
```

Các tính chất của Dictionary

```
...  
data1 = {"a": [1, 2, 3],  
         "b": {1, 2, 3}}
```

```
data_2 = {"a": [1, 2],  
          "b": [1, 2],  
          "c": [1, 2]}
```

Value không có giới hạn gì

- **Value trong Dictionary** có thể thuộc bất kỳ kiểu dữ liệu gì
- **Value trong Dictionary** có thể xuất hiện nhiều lần

Truy cập phần tử trong Dictionary (1)

`dictionary_name[key_name]`

Trả về phần tử có từ khóa là `key_name` trong `dictionary_name`

```
...  
  
data = {"name": "Viet", "age": 32, "job": "AI engineer"}  
print(data["age"])          # 32  
print(data["location"])    # KeyError: 'location'
```

Truy cập phần tử trong Dictionary (2)

`dictionary_name.get(key_name)`

Trả về phần tử có từ khóa là `key_name` trong `dictionary_name`

```
...  
data = {"name": "Viet", "age": 32, "job": "AI engineer"}  
print(data.get("age"))          # 32  
print(data.get("location"))    # None
```

Thêm hoặc cập nhật phần tử trong Dictionary

Dictionary_name[key_name] = value

...

```
data = {"name": "Viet", "age": 32, "job": "AI engineer"}  
data["gender"] = "male"  
data["age"] = 33  
print(data)      # {'name': 'Viet', 'age': 33, 'job': 'AI engineer', 'gender': 'male'}
```

Xóa phần tử khỏi Dictionary (1)

dictionary_name.pop(key_name)

Xóa phần tử với từ khóa **key_name** khỏi **dictionary_name**

```
...  
  
data = {"name": "Viet", "age": 32, "job": "AI engineer"}  
removed_value = data.pop("age")  
print(data)          # {'name': 'Viet', 'job': 'AI engineer'}  
print(removed_value) # 32
```

Xóa phần tử khỏi Dictionary (2)

del dictionary_name[key_name]

Xóa phần tử với từ khóa **key_name** khỏi **dictionary_name**

```
data = {"name": "Viet", "age": 32, "job": "AI engineer"}  
del data["age"]  
print(data)      # {'name': 'Viet', 'job': 'AI engineer'}
```

Xóa phần tử khỏi Dictionary (3)

dictionary_name.clear()

Xóa tất cả phần tử khỏi **dictionary_name**

```
...  
  
data = {"name": "Viet", "age": 32, "job": "AI engineer"}  
data.clear()  
print(data)      # {}
```

Trả về tất cả các key, value và item trong Dictionary

```
data = {"name": "Viet", "age": 32, "job": "AI engineer"}
```

Hàm	Mô tả	Ví dụ	Kết quả
keys()	Trả về tất cả các keys	data.keys()	dict_keys(['name', 'age', 'job'])
values()	Trả về tất cả các values	data.values()	dict_values(['Viet', 32, 'AI engineer'])
items()	Trả về tất cả các items	data.items()	dict_items([('name', 'Viet'), ('age', 32), ('job', 'AI engineer')])

Tính số phần tử của Dictionary

len(dictionary_name)

...

```
data = {"name": "Viet", "age": 32, "job": "AI engineer"}  
print(len(data))          # 3
```

Tổng kết các hàm tích hợp sẵn với Dictionary

Hàm	Mô tả
get()	Trả về value tương ứng với 1 key cụ thể trong Dictionary
pop()	Xóa 1 phần tử khỏi Dictionary và trả về value tương ứng
clear()	Xóa tất cả các phần tử khỏi Dictionary
keys()	Trả về tất cả các khóa (key) trong Dictionary
values()	Trả về tất cả các giá trị (value) trong Dictionary
items()	Trả về tất cả các phần tử (item) trong Dictionary

Phân biệt đối tượng khả biến (mutable) và bất biến (immutable)

Các kiểu dữ liệu và cấu trúc dữ liệu trong Python



Khả biến (mutable)

- List
- Dictionary
- Set

Bất biến (immutable)

- Number
- Bool
- String
- Tuple

Đối tượng khả biến (mutable)

Khả biến (mutable)

- List
- Dictionary
- Set

```
...  
  
my_list = [1, 2, 3]  
my_list[0] = 5  
my_list.append(4)  
my_list.pop(1)  
print(my_list)      # [5, 3, 4]  
  
my_dict = {"name": "John", "age": 30}  
my_dict["age"] = 31  
print(my_dict)      # {'name': 'John', 'age': 31}  
  
my_set = {1, 2, 3}  
my_set.add(4)  
print(my_set)      # {1, 2, 3, 4}
```

Đối tượng bất biến (**immutable**)(1)

```
...  
  
my_tuple = (2, 3, 5)  
my_tuple[0] = 4      # TypeError: 'tuple' object does not support item assignment  
  
my_string = "Today is a beautiful day!"  
my_string[0] = "S"   # TypeError: 'str' object does not support item assignment
```

Bất biến (**immutable**)

- Number
- Bool
- String
- Tuple

Đối tượng bất biến (**immutable**)(2)

```
my_string = "Today is a beautiful day!"  
print(id(my_string))      # 2192474992720  
my_string += " Let's go swimming"  
print(id(my_string))      # 2192474364144  
  
my_number = 5  
print(id(my_number))      # 2192472932720  
my_number = 6  
print(id(my_number))      # 2192472932752  
  
my_bool = True  
print(id(my_bool))        # 140716373260464  
my_bool = False  
print(id(my_bool))        # 140716373260496
```

Bất biến (**immutable**)

- Number
- Bool
- String
- Tuple

Tuple

Tuple là 1 tập hợp các phần tử Có thứ tự

Tuple có rất nhiều đặc điểm giống với List

Giống

- **Có thứ tự:** Các phần tử được đánh số từ trái sang phải
- **Chỉ số:** Các phần tử có thể được truy cập thông qua chỉ số
- **Phần tử:** Tuple có thể chứa bất kỳ loại dữ liệu gì

Khác

Tuple là bất biến, do đó ta không thể:

- Thêm
- Xóa
- Thay đổi

Các phần tử sau khi Tuple được định nghĩa

Khai báo Tuple

Mục đích	Câu lệnh
Khai báo 1 tuple rỗng	<code>data = ()</code>
Khai báo 1 tuple có các giá trị nguyên	<code>data = (2, 0, 3, 1, -5, 2)</code>
Khai báo 1 tuple có các giá trị chuỗi	<code>data = ("red", "green", "blue")</code>
Khai báo 1 tuple có các giá trị hỗn hợp	<code>data = (-2, 5.0, "Monday", [1, -2, 3])</code>
Khai báo 1 tuple có các giá trị giống nhau	<code>data = (5)*6 (= (5, 5, 5, 5, 5, 5))</code>
Biến đổi 1 list thành tuple	<code>data = tuple([1, 2, 3]) (= (1, 2, 3))</code>
Biến đổi 1 chuỗi thành tuple	<code>data = tuple("abc") (= ("a", "b", "c"))</code>
Tuple lồng nhau (Nested tuple)	<code>data = ((1, -2, 3), (2, -1), ("a", -1.5, 2, "b"))</code>

Tuple packing

```
data = ("red", "green", "blue", "cyan")
```



Cách khởi tạo tuple thông thường

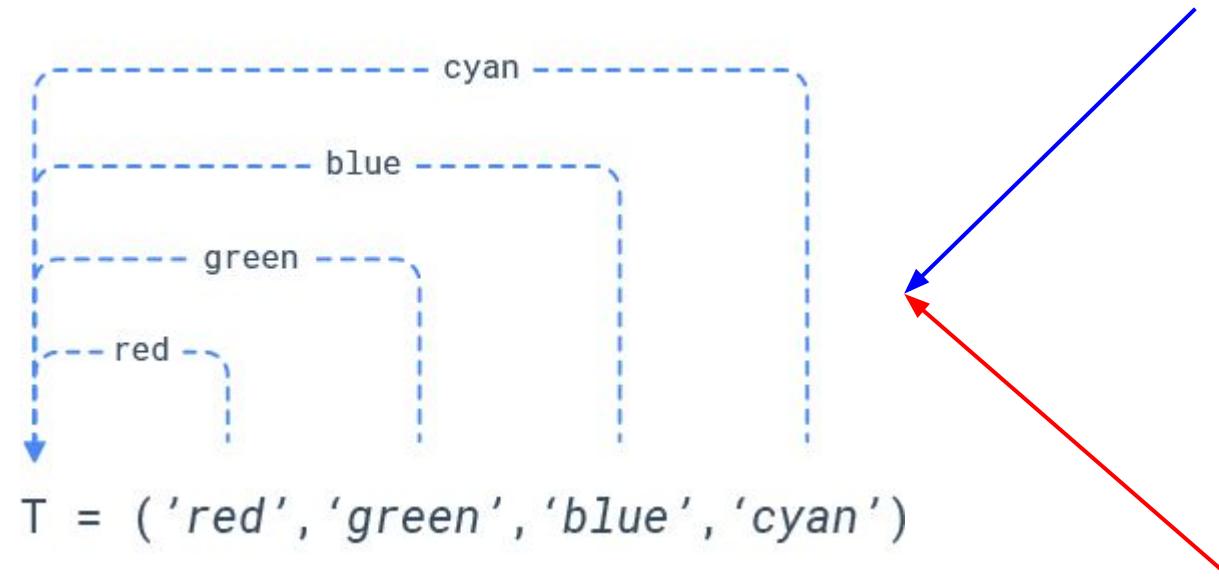
Tuple packing



```
data = "red", "green", "blue", "cyan"
```

Tuple packing

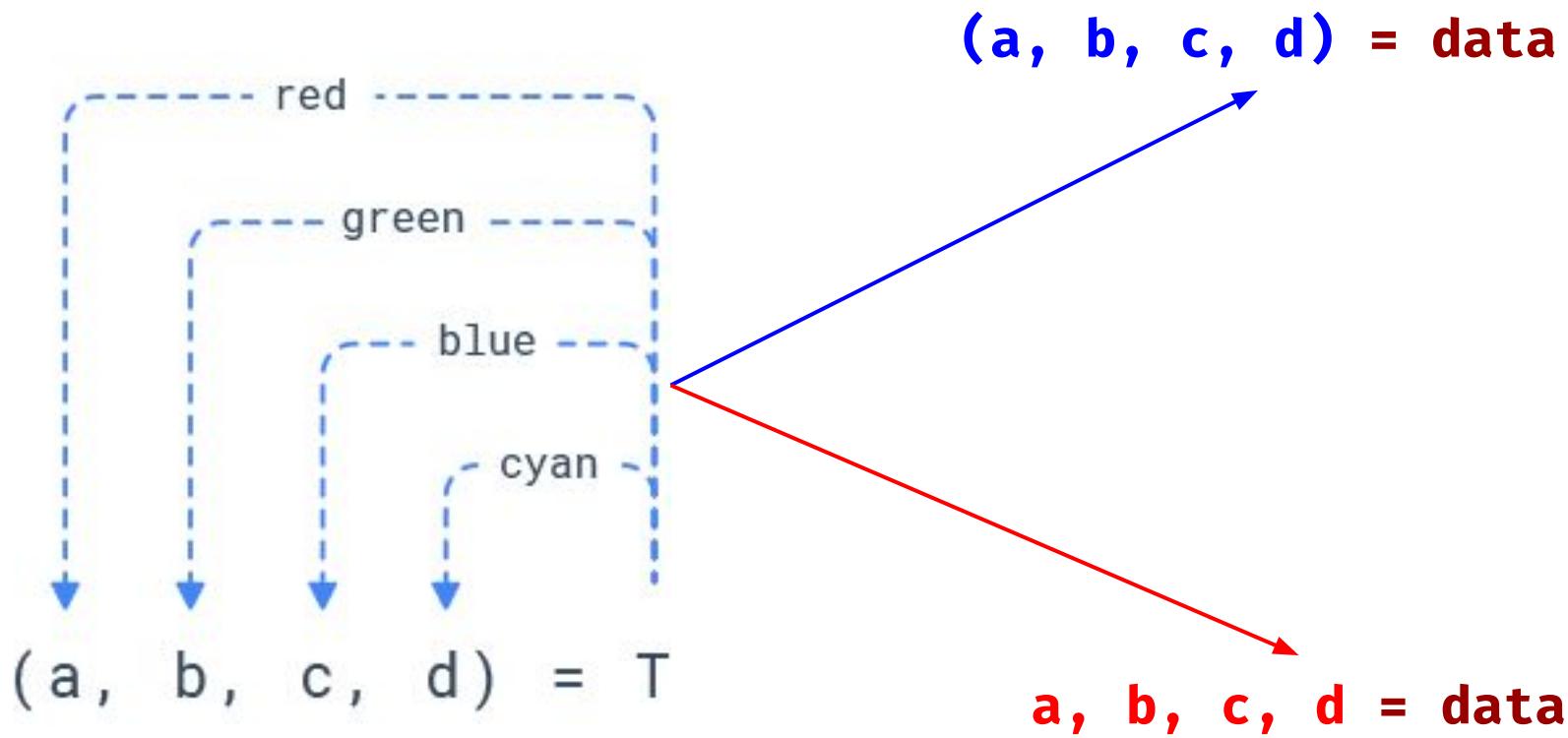
```
data = ("red", "green", "blue", "cyan")
```



```
T = ('red', 'green', 'blue', 'cyan')
```

```
data = "red", "green", "blue", "cyan"
```

Tuple unpacking



Ứng dụng của tuple unpacking (1)

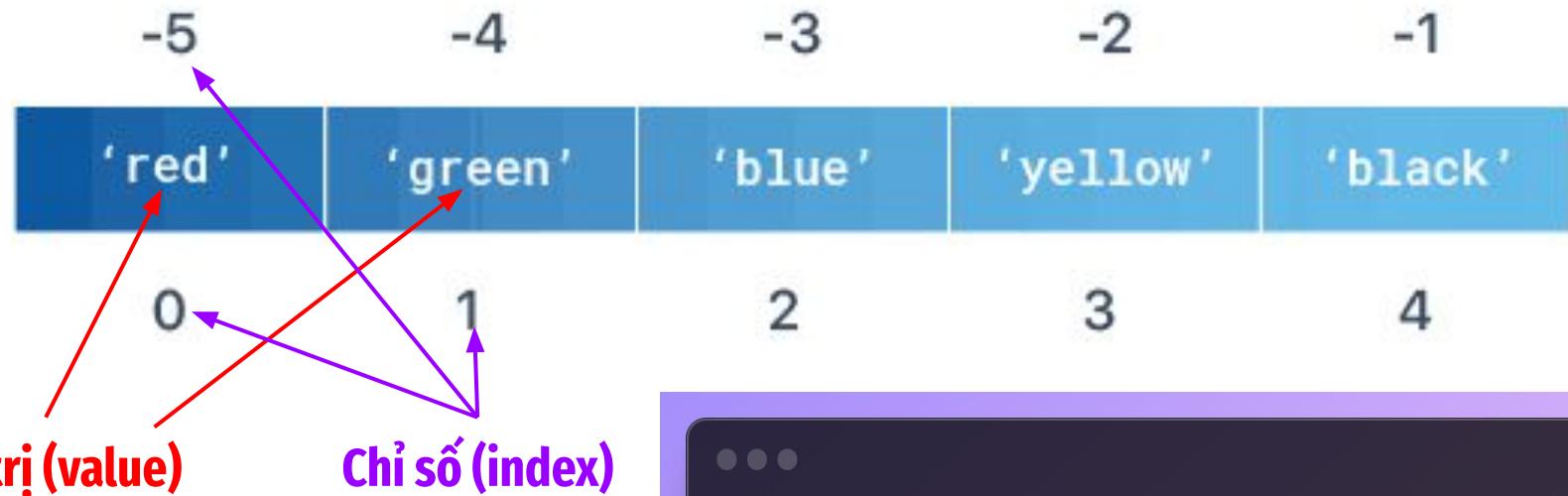
```
• • •  
a = 5  
b = 10  
a, b = b, a  
print(a, b)      # 10 5
```

Ứng dụng của tuple unpacking (2)

...

```
a, b, c, d = 4, 5.5, True, "Monday"
```

Truy cập phần tử trong Tuple



```
...  
  
data = ("red", "green", "blue", "yellow", "black")  
  
print(data[3])      # yellow  
print(data[-2])    # yellow  
print(data[5])     # IndexError: tuple index out of range
```

Cắt Tuple (Slice Tuple)

`tuple_name[begin:end:step]`

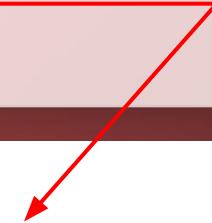
`data[1] data[2] data[3] data[4]`

The diagram illustrates tuple slicing with a purple rounded rectangle containing Python code. At the top, four blue arrows point from the labels `data[1]`, `data[2]`, `data[3]`, and `data[4]` down to the corresponding elements in the tuple definition. A red arrow points from the label `data[4]` to the end of the slice range in the print statement. The code inside the rectangle is:

```
data = ("red", "green", "blue", "yellow", "black")
print(data[1:4])      # ('green', 'blue', 'yellow')
```

Thay đổi giá trị của phần tử trong Tuple (1)

```
...  
data = ("red", "green", "blue", "yellow", "black")  
  
data[0] = "pink"      # TypeError: 'tuple' object does not support item assignment
```



**Tuple là bất biến (immutable), do đó 1 khi tuple đã được
khởi tạo thì sẽ không thể bị thay đổi**

Thay đổi giá trị của phần tử trong Tuple (2)

```
...  
data = ("red", "green", ["light", "blue"], "yellow", "black")  
  
data[2][0] = "dark"  
print(data)      # ('red', 'green', ['dark', 'blue'], 'yellow', 'black')
```



Tính bất biến chỉ được áp dụng vào Tuple chứ không được
áp dụng vào nội dung của nó

Xóa Tuple

`del tuple_name`

```
...
```

```
data = ("red", "green", "blue", "yellow", "black")
```

```
del data
```

Nối 2 Tuple lại với nhau

Sử dụng toán tử +

...

```
data = (2, 4, 6) + (1, 2, 3)
```

```
print(data)      # (2, 4, 6, 1, 2, 3)
```

Tính số phần tử của Tuple

`len(tuple_name)`

...

```
data = (2, 4, 6, 8, 10)
print(len(data))      # 5
```

Kiểm tra sự tồn tại của giá trị trong Tuple

`if item in tuple_name`

```
data = (2, 4, 6, 8, 10)
if 6 in data:
    print("1A")
else:
    print("1B")

if 5 in data:
    print("2A")
else:
    print("2B")
# 1A
# 2B
```

Duyệt qua các phần tử của Tuple

```
...  
data = ("red", "green", "blue")  
for color in data:  
    print(color)
```



```
...  
red  
green  
blue
```

Sắp xếp các phần tử của Tuple

`sorted(tuple_name)`

```
...  
data = (3, 1, 4, 8)  
data = sorted(data)  
print(data)
```



```
[1, 3, 4, 8]
```

Đếm số lần 1 giá trị xuất hiện trong Tuple

`tuple_name.count(value)`

Đếm số lần giá trị `value` xuất hiện trong `tuple_name`

...

```
data = ("red", "green", "blue", "red")
print(data.count("red"))          # 2
print(data.count("black"))       # 0
```

Tổng kết các hàm tích hợp sẵn với Tuple

Hàm	Mô tả
index()	Trả về vị trí của phần tử đầu tiên với giá trị cụ thể trong Tuple
sorted()	Sắp xếp các phần tử trong Tuple theo thứ tự tăng(giảm) dần
count()	Đếm số lần 1 giá trị xuất hiện trong Tuple
min()	Trả về phần tử có giá trị nhỏ nhất trong Tuple
max()	Trả về phần tử có giá trị lớn nhất trong Tuple
sum()	Trả về tổng của tất cả các phần tử trong Tuple

Set

Set là 1 tập hợp các phần tử KHÔNG trùng nhau và KHÔNG có thứ tự

Không có thứ tự

Các phần tử trong Set không được lưu trữ theo 1 thứ tự cụ thể nào

Không trùng lặp

Không tồn tại các giá trị trùng lặp trong Set

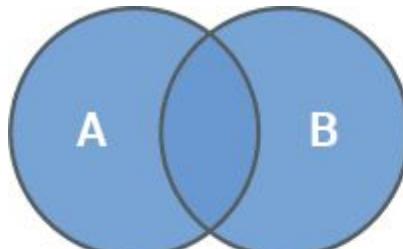
Không có chỉ số

Không thể truy cập vào các phần tử của Set thông qua chỉ số

Khả biến

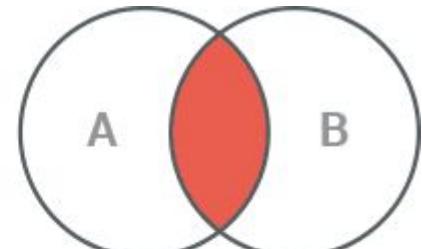
Set sau khi được khởi tạo có thể được thay đổi

01



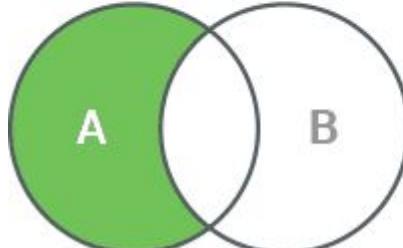
Union

02



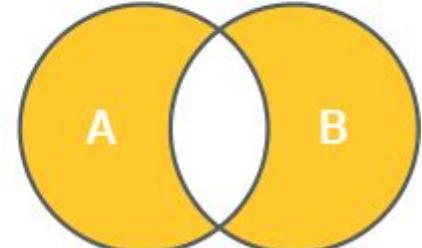
Intersection

03



Difference

04



Symmetric Difference

Khởi tạo Set (1)

...

```
data = {"red", "green", "blue"}
```

```
data = {1, 4, 2, 5, 7}
```

```
data = {"red", 1, True, 5.6}
```

Khởi tạo Set (2)

```
data = {"red", "green", "blue", "red", "blue", "white"}  
print(data)      # {'green', 'red', 'blue', 'white'}
```

Các giá trị trùng nhau sẽ bị loại bỏ

Khởi tạo Set (3)

```
...  
  
data = {"red", ("light", "blue"), "white"}      # OK  
  
data = {"red", ["light", "blue"], "white"}          # TypeError: unhashable type: 'list'
```

Set là khả biến (mutable) nhưng nó **KHÔNG** được phép
chứa các phần tử khả biến

Khởi tạo Set (4)

• • •

```
data = set("Wednesday")
print(data)      # {'e', 'y', 'n', 'W', 'a', 's', 'd'}
```



```
data = set([1, 2, 3, 4, 5])
print(data)      # {1, 2, 3, 4, 5}
```

Thêm phần tử vào Set (1)

`set_name.add(item)`

Thêm item vào set_name

```
...  
data = {2, 4, 6}  
data.add(10)  
print(data)      # {2, 10, 4, 6}  
data.add(4)  
print(data)      # {2, 10, 4, 6}
```

Chú ý: Hàm add() trả về None

Thêm phần tử vào Set (2)

`set_name.update(set1, set2, ...)`

Thêm các phần tử không trùng lặp trong `set1, set2, ...` vào `set_name`

```
...  
data = {2, 4, 6}  
data.update({1, -3, 5})  
print(data)      # {2, 4, 5, 6, -3, -1}  
data.update({1, 3}, {4, 8})  
print(data)      # {1, 2, 3, 4, 5, 6, 8, -3, -1}
```

Chú ý: Hàm `update()` trả về `None`

Xóa phần tử khỏi Set (1)

set_name.remove(item)

Xóa item khỏi set_name

```
data = {2, 4, 6}
data.remove(4)
print(data)      # {2, 6}
data.remove(1)
print(data)      # KeyError: 1
```

Xóa phần tử khỏi Set (2)

set_name.discard(item)

Xóa item khỏi set_name

```
data = {2, 4, 6}
data.discard(4)
print(data)      # {2, 6}
data.discard(1)
print(data)      # {2, 6}
```

Xóa phần tử khỏi Set (3)

`set_name.pop()`

Xóa 1 phần tử ngẫu nhiên ra khỏi `set_name` và trả lại phần tử đó

```
data = {2, 4, 6}
item = data.pop()
print(data)      # {4, 6}
print(item)      # 2
```

Xóa phần tử khỏi Set (4)

`set_name.clear()`

Xóa tất cả phần tử khỏi `set_name`

```
...  
data = {2, 4, 6}  
data.clear()  
print(data)      # set()
```

Tính số phần tử của Set

`len(set_name)`

...

```
data = {2, 4, 6, 8, 10}
print(len(data))      # 5
data = {2, 4, 6, 8, 10, 6, 8}
print(len(data))      # 5
```

Kiểm tra sự tồn tại của giá trị trong Set

if item in set_name

```
data = {2, 4, 6, 8, 10}
if 6 in data:
    print("1A")
else:
    print("1B")

if 5 in data:
    print("2A")
else:
    print("2B")
# 1A
# 2B
```

Duyệt qua các phần tử của Set

```
...  
data = {"red", "green", "blue"}  
for color in data:  
    print(color)
```

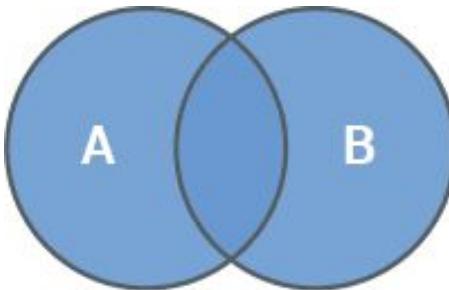


```
...  
  
blue  
green  
red
```

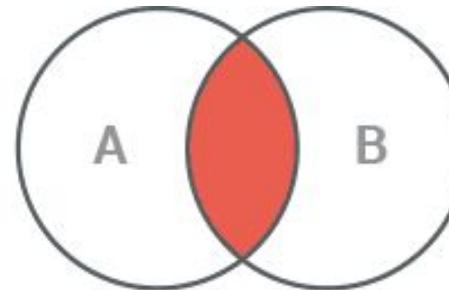
Tổng kết các hàm tích hợp sẵn với Set

Hàm	Mô tả
<code>add()</code>	Thêm 1 phần tử vào Set
<code>update()</code>	Thêm nhiều phần tử vào Set
<code>remove()</code>	Xóa phần tử khỏi Set
<code>discard()</code>	Xóa phần tử khỏi Set
<code>pop()</code>	Xóa phần tử ngẫu nhiên khỏi Set
<code>clear()</code>	Xóa tất cả các phần tử khỏi Set

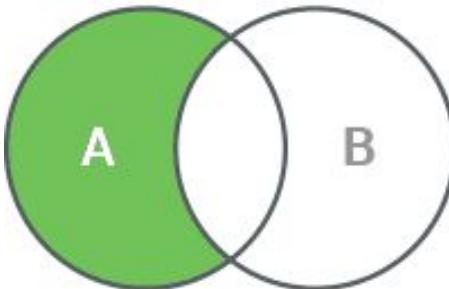
Các phép toán tập hợp



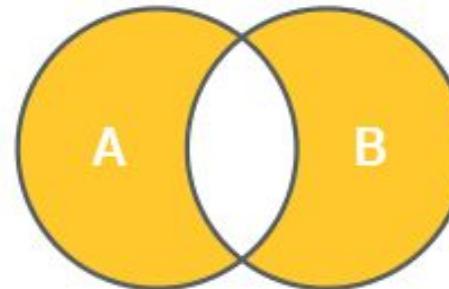
Union



Intersection



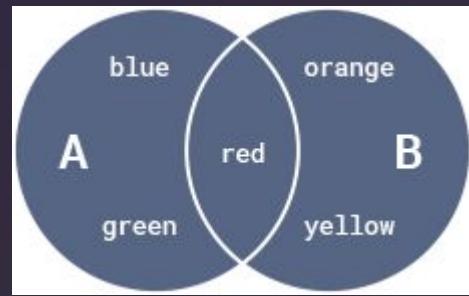
Difference



Symmetric Difference

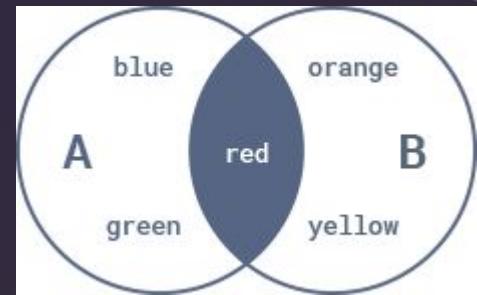
Phép hợp (Union)

```
● ● ●  
  
set1 = {'red', 'green', 'blue'}  
set2 = {'yellow', 'red', 'orange'}  
  
# by operator  
print(set1 | set2)          # {'blue', 'yellow', 'red', 'green', 'orange'}  
  
# by method  
print(set1.union(set2))     # {'blue', 'yellow', 'red', 'green', 'orange'}
```



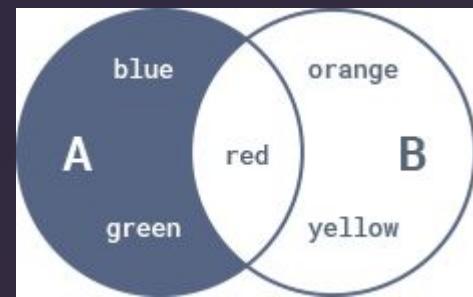
Phép giao (Intersection)

```
...  
  
set1 = {'red', 'green', 'blue'}  
set2 = {'yellow', 'red', 'orange'}  
  
# by operator  
print(set1 & set2)                      # {'red'}  
  
# by method  
print(set1.intersection(set2))           # {'red'}
```



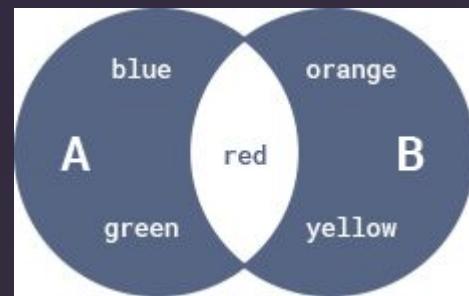
Phép hiệu (Difference)

```
● ● ●  
  
set1 = {'red', 'green', 'blue'}  
set2 = {'yellow', 'red', 'orange'}  
  
# by operator  
print(set1 - set2) # {'green', 'blue'}  
  
# by method  
print(set1.difference(set2)) # {'green', 'blue'}
```



Phép hiệu đối xứng (Symmetric difference)

```
...  
  
set1 = {'red', 'green', 'blue'}  
set2 = {'yellow', 'red', 'orange'}  
  
# by operator  
print(set1 ^ set2)                      # {'blue', 'orange', 'yellow', 'green'}  
  
# by method  
print(set1.symmetric_difference(set2))    # {'blue', 'orange', 'yellow', 'green'}
```



Chuỗi ký tự

String và cách khai báo

String là 1 tập hợp các kí tự (characters) nằm trong 1 nháy đơn, 1 nháy đôi, 3 nháy đơn hoặc 3 nháy đôi

```
...  
  
str1 = 'This is a string.'  
str2 = "This is also a string."  
str3 = '''  
      Sometimes you want to write  
      a string in multiple lines.  
      Here is one example.  
...  
  
str4 = """  
      If you read until the end  
      of this string, you will realize  
      that you waste 5 seconds of your life.  
      This is again another string  
      written in multiple lines.  
"""
```

Biến đổi kiểu dữ liệu khác sang String

str(any_object)

```
...  
  
int_value = 1991  
string_value = str(int_value)  
print(string_value) _____  
print(type(string_value))  
  
float_value = 10.2  
string_value = str(float_value)  
print(string_value) _____  
print(type(string_value))  
  
list_data = [1991, 10.2, "Monday", (2, 3), {"flower": "rose"}]  
string_value = str(list_data)  
print(string_value) _____  
print(type(string_value))
```

1991
<class 'str'>

10.2
<class 'str'>

[1991, 10.2, 'Monday', (2, 3), {'flower': 'rose'}]
<class 'str'>

Truy cập ký tự trong String



```
string = "ABCDEFGHI"  
print(string[1])      # B  
print(string[-8])    # B  
  
print(string[3])      # D  
print(string[-6])    # D
```

Cắt String (Slice String)

string_name[begin:end:step]



Start of the slice

end of the slice

```
string = "ABCDEFGHI"  
print(string[2:7]) # CDEFG
```

Đảo ngược String

string_name[:: -1]

...

```
string = "ABCDEFGHI"  
print(string[::-1]) # IHGFEDCBA
```

Thay đổi (1 phần) String

Nếu chúng ta thử làm theo cách tương tự như với List

```
...  
string = "ABCDEFGHI"  
string[0] = "P"  
# TypeError: 'str' object does not support item assignment
```



Sẽ xuất hiện lỗi

Cách thay thế việc thay đổi String

Tạo 1 String mới dựa trên String cũ kết hợp với Slice

```
string = "ABCDEFGHI"  
new_string = "P" + string[1:]  
print(new_string)      # PBCDEFGHI
```

Kết hợp các String (1)

Sử dụng toán tử + để kết hợp các String

```
...  
string = "Hello," + " World!"  
print(string)  
string += " I start learning Python today"  
print(string)
```

Kết hợp các String (2)

Các String cạnh nhau sẽ được tự động nối lại với nhau

...

```
string = "Hello, "" World! ''Python'  
print(string)    # Hello, World! Python
```

Nhân bản String

1 String có thể được nhân bản và kết nối với nhau thông qua toán tử *

```
...  
string = "Hello "  
string = string * 5  
print(string)    # Hello Hello Hello Hello Hello
```

Tính độ dài của String

len(string_name)

• • •

```
string = "hello!"  
print(len(string))    # 6
```

Thay thế Text bên trong String

`string_name.replace(old_text, new_text)`

Thay thế `old_text` bằng `new_text` trong `string_name`

```
...
```

```
string = "long, longer, longest"
string = string.replace("long", "high")
print(string)    # high, higher, highest
```

Chuyển đổi in hoa/in thường trong String

```
string = "Ho Chi Minh city"
```

Hàm	Mô tả	Kết quả
lower()	Viết thường tất cả các ký tự	ho chi minh city
upper()	Viết hoa tất cả các ký tự	HO CHI MINH CITY
capitalize()	Ký tự đầu tiên viết hoa, còn lại viết thường	Ho chi minh city
title()	Ký tự đầu của mỗi từ viết hoa, còn lại viết thường	Ho Chi Minh City

Kiểm tra Substring bên trong String

substring_name in string_name

Trả về giá trị True nếu **substring_name** nằm trong **string_name**

```
string = "Today is a beautiful day"  
substring1 = "beautiful"  
substring2 = "monday"  
print(substring1 in string) # True  
print(substring2 in string) # False
```

Các ký tự đặc biệt (Escape sequence)

Ký tự	Ý nghĩa
\'	Nháy đơn
\\"	Nháy kép
\	Gạch chéo ngược
\n	Xuống dòng
\t	Dấu tab

Các ký tự đặc biệt (Escape sequence)

```
...  
print('Let's learn Python together!')  
  
# SyntaxError: unterminated string literal (detected at line 1)
```

...

```
print('Let\'s learn Python together!')  
print("This is\na Python class")
```

...

```
Let's learn Python together!  
This is  
a Python class
```

Tìm vị trí của Substring bên trong String

string_name.find(substring_name)

Trả về vị trí của ký tự đầu tiên của **substring_name** đầu tiên nằm trong **string_name**

```
...  
  
string = "Today is a beautiful day"  
substring1 = "beautiful"  
substring2 = "monday"  
print(string.find(substring1)) # 11  
print(string.find(substring2)) # -1
```

Duyệt qua các phần tử của String

```
...  
...
```

```
string = "Today is Sunday"  
for char in string:  
    print(char)
```

```
...
```

```
T  
o  
d  
a  
y  
  
i  
s  
  
S  
u  
n  
d  
a  
y
```



Loại bỏ khoảng trắng dư thừa khỏi String

string_name.strip()

Loại bỏ toàn bộ các khoảng trắng ở đầu và cuối của string_name

```
...  
string = "Today is Sunday"  
print("Before:", string, ". Length:", len(string))  
string = string.strip()  
print("After:", string, ". Length:", len(string))
```

...
Before: Today is Sunday . Length: 28
After: Today is Sunday . Length: 15

Định dạng String (1)

String sẽ được định dạng

“.....{}.....”.format(value)

value sẽ được đặt vào {}

```
...  
day = "Monday"  
string = "Today is {}".format(day)  
print(string)    # Today is Monday
```

Định dạng String (2)

Thứ tự mặc định cho các giá trị được định dạng

```
string = "{} is {} years old. He is from {}".format("Viet", 25, "Vietnam")
```

Viet is 25 years old. He is from Vietnam

Chỉ ra thứ tự của các giá trị được định dạng

```
string = "{1} is {2} years old. He is from {0}".format("Vietnam", "Viet", 25)
```

Tách String

string_name.split(delimiter)

Tách **string_name** bởi **delimiter** và trả về List của các **substrings**

```
string = "blue,green,red,yellow"
result = string.split(",")
print(result)  # ['blue', 'green', 'red', 'yellow']

string = "Today is a beatiful day"
result = string.split()
print(result)  # ['Today', 'is', 'a', 'beatiful', 'day']
```

Nối String

delimiter.join(iterable)

Nối các phần tử của iterable bằng delimiter và trả về 1 String

```
strings = ['blue', 'green', 'red', 'yellow']
result = ", ".join(strings)
print(result) # blue, green, red, yellow

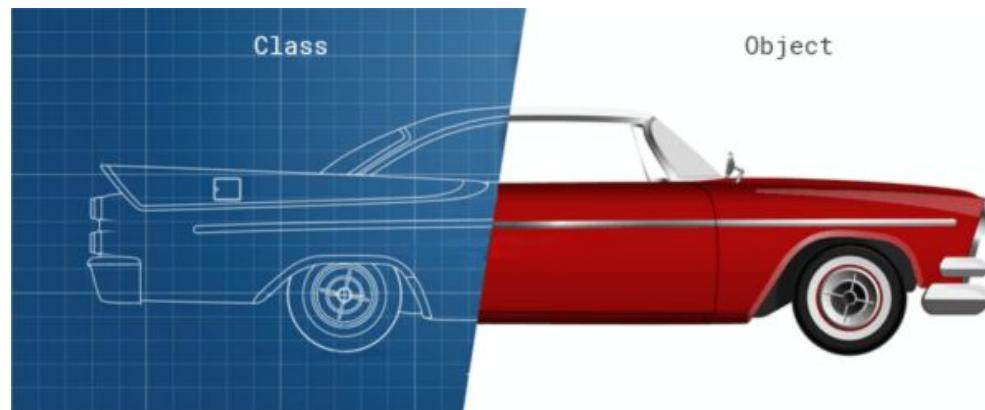
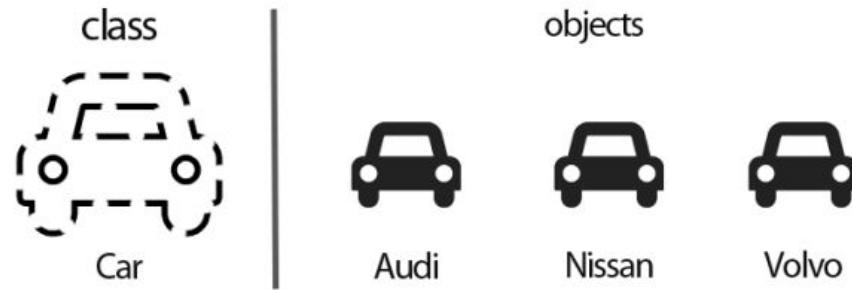
strings = ['Today', 'is', 'a', 'beatiful', 'day']
result = " ".join(strings)
print(result) # Today is a beatiful day
```

Tổng kết các hàm tích hợp sẵn với String

Hàm	Mô tả
<code>replace(old_text, new_text)</code>	Thay thế <code>old_text</code> bằng <code>new_text</code> trong String
<code>lower()</code>	Viết thường tất cả các ký tự trong String
<code>upper()</code>	Viết hoa tất cả các ký tự trong String
<code>capitalize()</code>	Viết hoa duy nhất ký tự đầu của String
<code>title()</code>	Viết hoa ký tự đầu của mỗi từ trong String
<code>find(substr)</code>	Tìm vị trí của Substring trong String
<code>strip()</code>	Loại bỏ toàn bộ các khoảng trắng ở đầu và cuối trong String
<code>split(delimiter)</code>	Tách String bởi <code>delimiter</code>
<code>join(strings)</code>	Nối các Strings lại với nhau

Lập trình hướng đối tượng

Lớp (Class) và đối tượng (Object)



Lớp

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Định nghĩa Lớp

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Sử dụng từ khóa **class**, theo sau bởi tên class đó để định nghĩa 1 lớp mới

Nếu tên lớp có nhiều từ:

- Viết các từ này liền nhau
- Chữ cái đầu của mỗi từ viết hoa

Ví dụ:

- **MyClass**
- **FootballDataset**

Định nghĩa hàm khởi tạo

Tham số đầu tiên luôn là self

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Phương thức khởi tạo `__init__()`

Hàm `__init__()` là 1 hàm đặc biệt
Hàm này sẽ khởi tạo 1 đối tượng cụ thể của lớp

Hàm `__init__()` sẽ được gọi tự động
mỗi khi một đối tượng của lớp được tạo ra

Hàm `__init__()` dùng để làm các công việc cần thiết khi 1 đối tượng mới của lớp được tạo ra

Tham số self

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

self luôn là tham số đầu tiên của tất cả các phương thức của lớp

Thuộc tính (Attribute)

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Các thuộc tính của lớp

Thuộc tính là những thứ thể hiện các tính chất mà mỗi thể hiện của Lớp có

Nhìn chung, thuộc tính là những thứ được dùng để phân biệt 1 đối tượng này với 1 đối tượng khác của Lớp
Các thuộc tính trong Lớp được định nghĩa bởi các biến

Thuộc tính (Attribute)

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Có 2 loại thuộc tính

Thuộc tính thể hiện (Instance attribute)

Đây là thuộc tính đặc trưng riêng của mỗi đối tượng

Bất kỳ thay đổi nào đến các thuộc tính thể hiện của 1 đối tượng sẽ không ảnh hưởng đến các đối tượng khác

Thuộc tính (Attribute)

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Có 2 loại thuộc tính

Thuộc tính lớp (Class attribute)

Đây là thuộc tính chung cho tất cả mọi đối tượng của Lớp

Bất kỳ 1 thay đổi nào đến thuộc tính này sẽ có hiệu lực đối với tất cả các đối tượng của lớp

Tham số **self** và các thuộc tính

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Nhờ việc **self** là tham số đầu tiên
của tất cả các hàm của Lớp...

... mà ta có thể truy cập hoặc thay
đổi giá trị các thuộc tính

Phương thức (Method)

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Các phương thức của Lớp

Các phương thức được dùng để xác định các hành vi, chức năng mà các đối tượng của lớp có

Hàm (Function) vs Phương thức (Method)

Hàm

- Nằm bên ngoài và không liên quan tới Class
- Có thể được gọi 1 cách độc lập, không phụ thuộc và Lớp hay đối tượng cụ thể nào

Phương thức

- Nằm bên trong Lớp
- Là 1 thành phần của đối tượng
- Chỉ có thể được gọi thông qua Lớp hoặc đối tượng

Khởi tạo đối tượng

```
...  
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color  
  
my_car = Car(color="Black", style="Sedan")  
your_car = Car("White", "Hatchback")  
her_car = Car("Red", style="SUV")
```

Tạo đối tượng bằng cách gọi tên của Lớp và truyền vào các đối số (tương tự cách gọi hàm)

Không cần truyền đối số **self**

Sử dụng phương thức của Lớp

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color  
  
  
my_car = Car(color="Black", style="Sedan")  
print(my_car.color)      # Black  
my_car.change_color("Orange")  
print(my_car.color)      # Orange
```

Áp dụng công thức sau để sử dụng phương thức
instance_name.method(arguments)

Truy cập thuộc tính của đối tượng

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color  
  
my_car = Car(color="Black", style="Sedan")  
print(my_car.color)      # Black  
my_car.change_color("Orange")  
print(my_car.color)      # Orange
```

Áp dụng công thức sau để truy cập thuộc tính
instance_name.attribute

Thay đổi thuộc tính của đối tượng

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color  
  
my_car = Car(color="Black", style="Sedan")  
print(my_car.color)      # Black  
my_car.color = "Orange"  
print(my_car.color)      # Orange
```

Áp dụng công thức sau để thay đổi giá trị của thuộc tính

instance_name.attribute = value

Xóa thuộc tính của đối tượng

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color  
  
  
my_car = Car(color="Black", style="Sedan")  
del my_car.color  
print(my_car.color)      # AttributeError: 'Car' object has no attribute 'color'
```

Xóa 1 thuộc tính của đối tượng

del instance_name.attribute

Sau khi xóa thì đối tượng không
còn thuộc tính này nữa

Xóa đối tượng

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color  
  
my_car = Car(color="Black", style="Sedan")  
del my_car  
print(my_car)      # NameError: name 'my_car' is not defined
```

Xóa 1 đối tượng

del instance_name

Sau khi xóa thì đối tượng không
còn tồn tại nữa

Các đặc điểm cơ bản của Lập trình hướng đối tượng

01

Inheritance

Tính kế thừa

02

Encapsulation

Tính đóng gói

03

Polymorphism

Tính đa hình

04

Abstraction

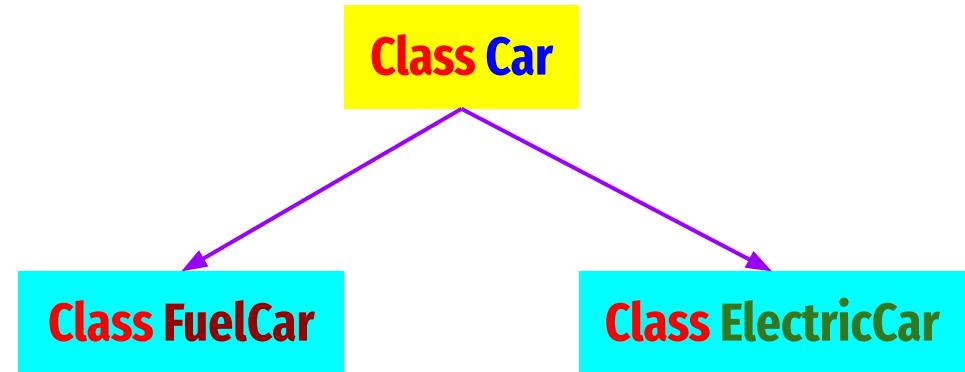
Tính trừu tượng

Tính kế thừa (Inheritance)



Tính kế thừa (Inheritance)

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```



Class ChildClass(ParentClass):

```
class FuelCar(Car):  
    pass
```

Viết đè phương thức (Override a method)

Parent class

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Child class

```
class FuelCar(Car):  
    num_wheels = 4  
  
    def __init__(self, color, style, fuel_type):  
        self.color = color  
        self.style = style  
        self.fuel_type = fuel_type  
        self.speed = 0
```

Hàm super()

Parent class

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

Child class

```
class FuelCar(Car):  
    num_wheels = 4  
  
    def __init__(self, color, style, fuel_type):  
        super().__init__(color, style)  
        self.fuel_type = fuel_type
```

call

Thêm thuộc tính và phương thức cho Lớp Con

Parent class

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self.color = color  
        self.style = style  
        self.speed = 0  
  
    def change_speed(self, speed):  
        self.speed = speed  
  
    def change_color(self, color):  
        self.color = color
```

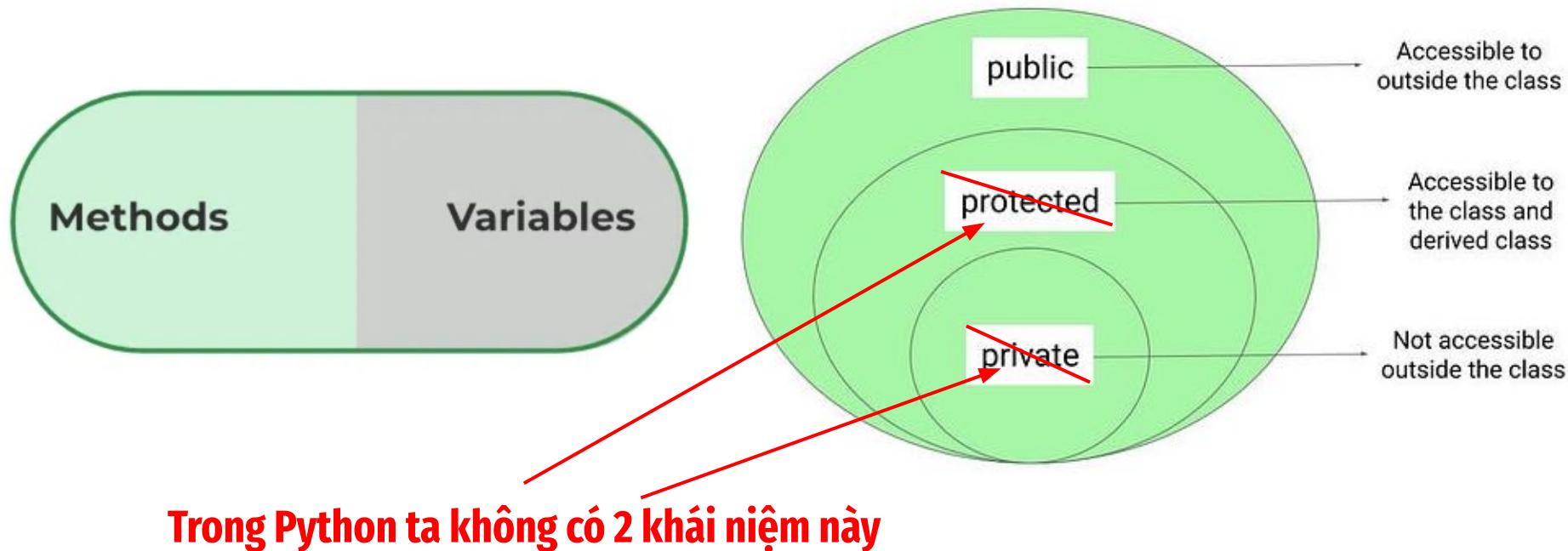
Child class

```
class FuelCar(Car):  
    num_wheels = 4  
  
    def __init__(self, color, style, fuel_type):  
        super().__init__(color, style)  
        self.fuel_type = fuel_type  
  
    def change_fuel_type(self, fuel_type):  
        self.fuel_type = fuel_type
```

Thuộc tính mới

Phương thức mới

Tính đóng gói (Encapsulation)



Thuộc tính **protected** và **private**

**Không truy cập được thuộc tính private vì
Python đã làm lệch tên của thuộc tính đi**

```
class Car:
    num_wheels = 4

    def __init__(self, color, style):
        self._color = color
        self.__style = style
        self.speed = 0

    def print_color(self):
        print("This is a(n) {} car".format(self._color))

    def print_style(self):
        print("This is a(n) {} car".format(self.__style))

car = Car("Black", "Sedan")
print(car._color) # Black
print(car.__style) # AttributeError: 'Car' object has no attribute '__style'
print(car._Car__style) # Sedan
```

Không nên truy cập thuộc tính protected như thế này

Thuộc tính **protected** và **private**

```
class Car:  
    num_wheels = 4  
  
    def __init__(self, color, style):  
        self._color = color  
        self.__style = style  
        self.speed = 0  
  
    def print_color(self):  
        print("This is a(n) {} car".format(self._color))  
  
    def print_style(self):  
        print("This is a(n) {} car".format(self.__style))  
  
car = Car("Black", "Sedan")  
car.print_color()          # This is a(n) Black car  
car.print_style()          # This is a(n) Sedan car
```

Truy cập thuộc tính thông qua phương thức của lớp

Tính đa hình (Polymorphism)



Ví dụ về tính đa hình

```
print(len("Vietnam"))          # 7  
print(len([1, 2, 3, 4]))       # 4  
print(len({"Name": "Viet", "Job": "engineer"})) # 2
```

Cùng 1 hàm **len()** được dùng cho nhiều kiểu dữ liệu khác nhau

Tính đa hình (Polymorphism)



```
class Duck:  
    def speak(self):  
        print("Quack")  
  
class Dog:  
    def speak(self):  
        print("Woof")  
  
class Cat:  
    def speak(self):  
        print("Meow")  
  
def animal_speak(animal):  
    animal.speak()  
  
duck = Duck()  
dog = Dog()  
cat = Cat()  
  
animal_speak(duck)  
animal_speak(dog)  
animal_speak(cat)
```

Đọc và xuất dữ liệu

Các loại file cơ bản



Mở file .txt

file object ← **f = open(file, mode)**

đường dẫn tương đối/tuyệt đối đến file

file được mở ra để làm gì

mode	Ý nghĩa
“r”	Mở file chỉ để đọc (Đây là mode mặc định)
“w”	Mở file chỉ để ghi (Ghi đè nội dung cũ nếu có)
“a”	Mở file để gắn thêm nội dung mới vào cuối file
“r+”	Mở file vừa để đọc vừa để ghi (ghi đè nội dung cũ nếu có)
“x”	Tạo file mới

Ví dụ

```
...  
  
# Open a file for reading  
f = open('data.txt')  
  
# Open a file for writing  
f = open('data.txt', 'w')  
  
# Open a file for reading and writing  
f = open('data.txt', 'r+')
```

Đọc file (1)



f.read()

Đọc toàn bộ nội dung file

```
f = open("data.txt")
print(f.read())
# First line of the file.
# Second line of the file.
# Third line of the file
```

Đọc file (2)

data.txt

First line of the file.
Second line of the file.
Third line of the file.

f.readline()

Đọc 1 dòng

```
f = open("data.txt")  
  
print(f.readline())  
# First line of the file.
```

Đọc file (3)

```
... data.txt  
First line of the file.  
Second line of the file.  
Third line of the file.
```

f.readlines()

Đọc nhiều dòng

```
...  
  
f = open("data.txt")  
  
print(f.readlines())  
  
# ['First line of the file.\n', 'Second line of the file.\n', 'Third line of the file.']}
```

Đọc file (4)

```
... data.txt  
First line of the file.  
Second line of the file.  
Third line of the file.
```

list(f)

Đọc nhiều dòng

```
f = open("data.txt")  
  
print(list(f))  
  
# ['First line of the file.\n', 'Second line of the file.\n', 'Third line of the file.']}
```

Ghi file (ghi đè 1)

```
... data.txt  
First line of the file.  
Second line of the file.  
Third line of the file.
```

```
... data.txt  
Overwrite existing data.
```

```
... data.txt  
f = open("data.txt", "w")  
f.write("Overwrite existing data.")
```

f.write(text)
Ghi đè toàn bộ nội dung vào file

Ghi file (ghi đè 2)

```
First line of the file.  
Second line of the file.  
Third line of the file.
```

```
New Line 1  
New Line 2  
New Line 3
```

```
f = open("data.txt", "w")
```

```
lines = ['New line 1\n', 'New line 2\n', 'New line 3']  
f.writelines(lines)
```

f.writelines(lines)

Ghi đè nhiều dòng vào file

Ghi file (ghi thêm)

```
... data.txt  
First line of the file.  
Second line of the file.  
Third line of the file.
```

```
... data.txt  
First line of the file.  
Second line of the file.  
Third line of the file. Append this text.
```

```
f = open("data.txt", "a")
```

```
f.write(" Append this text.")
```

f.write(text)

Ghi thêm nội dung mới vào cuối file

Đóng file (thủ công)

```
f = open("data.txt")
```

f.close()

```
f.close()
```

Đóng file sau khi đã xong việc

Đóng file (tự động)

```
...  
with open("data.txt") as f:  
    print(f.read())    # or do something else here  
  
print("At this line, file was already closed")
```

Xử lý ngoại lệ

Lỗi (**Error**) trong Python

Lỗi cú pháp (Syntax error)

- Xảy ra khi cú pháp được quy định của Python không được tuân thủ
- Chương trình sẽ không thể chạy khi có lỗi này

```
...  
if 6 > 5  
    print("True")  
  
# SyntaxError: expected ':'
```

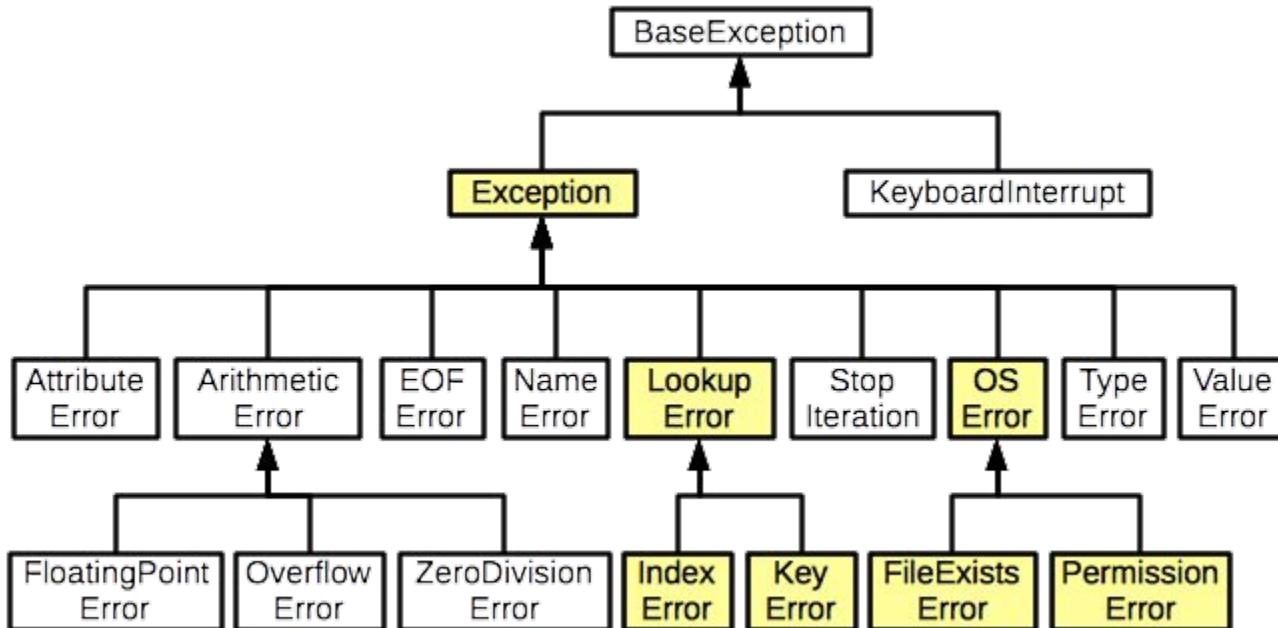
Lỗi logic (Logical error)

Lỗi xảy ra sau khi các cú pháp đều đã thỏa mãn quy định của Python. Ví dụ:

- Phép chia cho số 0
- Thực hiện phép cộng giữa số và chuỗi ký tự
- Import 1 thư viện không tồn tại
- Gọi đến 1 biến chưa được định nghĩa

Lỗi logic còn được gọi là ngoại lệ (**Exception**)

Sơ đồ phân cấp các ngoại lệ



Một vài ngoại lệ phổ biến

Exception (Ngoại lệ)	Ý nghĩa
IndexError	Xảy ra khi chỉ số của List (Tuple) nằm ngoài khoảng cho phép
AttributeError	Xảy ra khi truy cập đến một thuộc tính không tồn tại của Lớp
ImportError	Xảy ra khi import một thư viện/module không tồn tại
KeyError	Xảy ra khi Key (từ khóa) không tồn tại
NameError	Xảy ra khi truy cập đến 1 biến không tồn tại
TypeError	Xảy ra khi hàm hoặc phép toán được áp dụng cho đối tượng có kiểu dữ liệu không phù hợp

IndexError



```
data = [0, 1, 2]
print(data[3])
# IndexError: list index out of range
```

AttributeError



```
class MyClass():
    def __init__(self):
        self.first = 1

obj = MyClass()
print(obj.second)
# AttributeError: 'MyClass' object has no attribute 'second'
```

ImportError



```
from math import abcxyz
# ImportError: cannot import name 'abcxyz' from 'math' (unknown location)
```

ModuleNotFoundError



```
import viethandsome
# ModuleNotFoundError: No module named 'viethandsome'
```

KeyError



```
data = {"a": 1, "b": 2}  
print(data["c"])  
# KeyError: 'c'
```

NameError



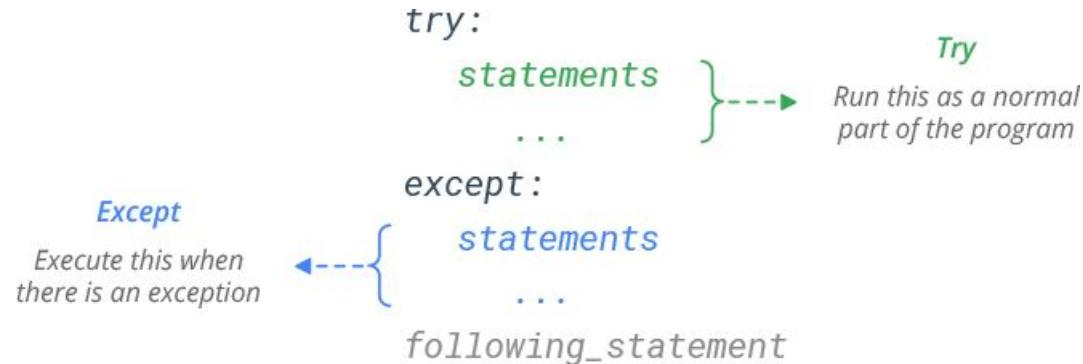
```
print(data)  
# NameError: name 'data' is not defined
```

TypeError



```
print(5 + "5")
# TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Khối lệnh try và except



Khối lệnh try

A screenshot of a code editor window titled "Lỗi xuất hiện tại đây" (Error occurs here). The code inside the window is:

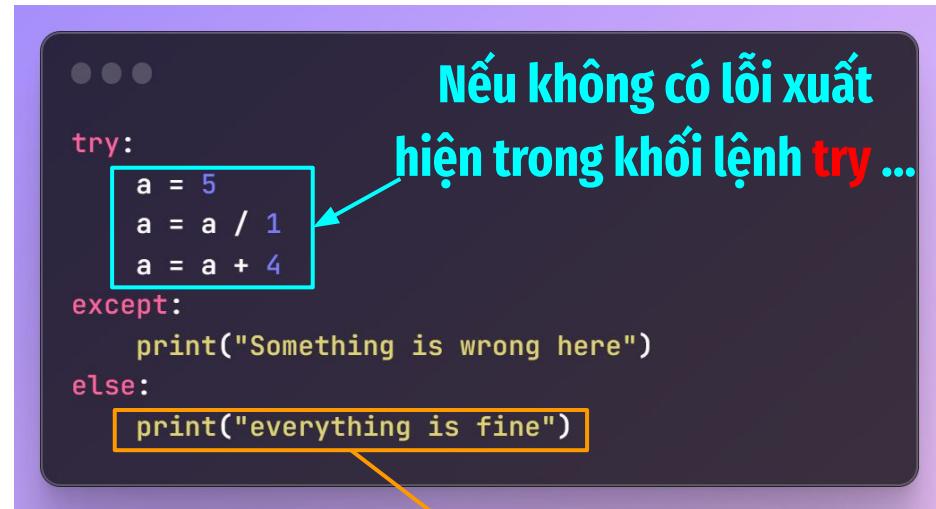
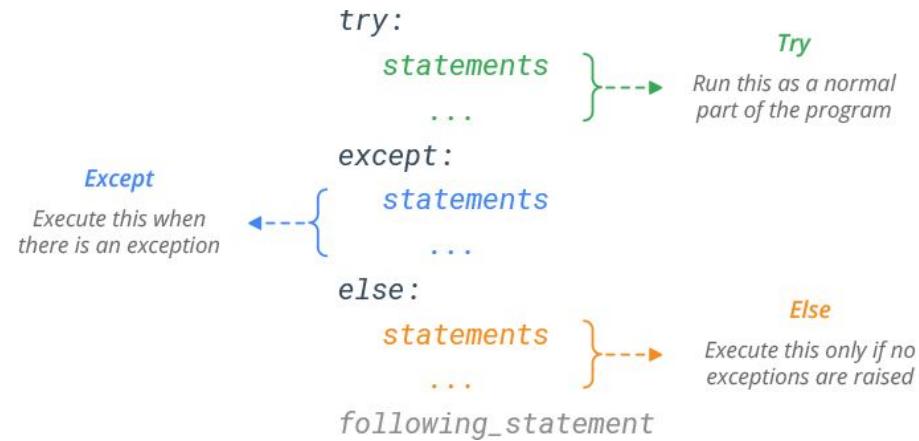
```
try:  
    a = 5  
    a = a / 0  
    a = a + 4  
except:  
    print("Something is wrong here")
```

Annotations explain the execution flow:

- A cyan arrow points from the text "Khối lệnh try" to the **try:** keyword.
- A yellow arrow points from the text "Lỗi xuất hiện tại đây" to the line **a = a / 0**, which is highlighted in red.
- A green arrow points from the text "Dòng này sẽ không được thực hiện" to the line **a = a + 4**, which is highlighted in purple.
- An orange arrow points from the text "Khối lệnh except sẽ được thực thi" to the **except:** block.

Khối lệnh except
sẽ được thực thi

Mệnh đề else



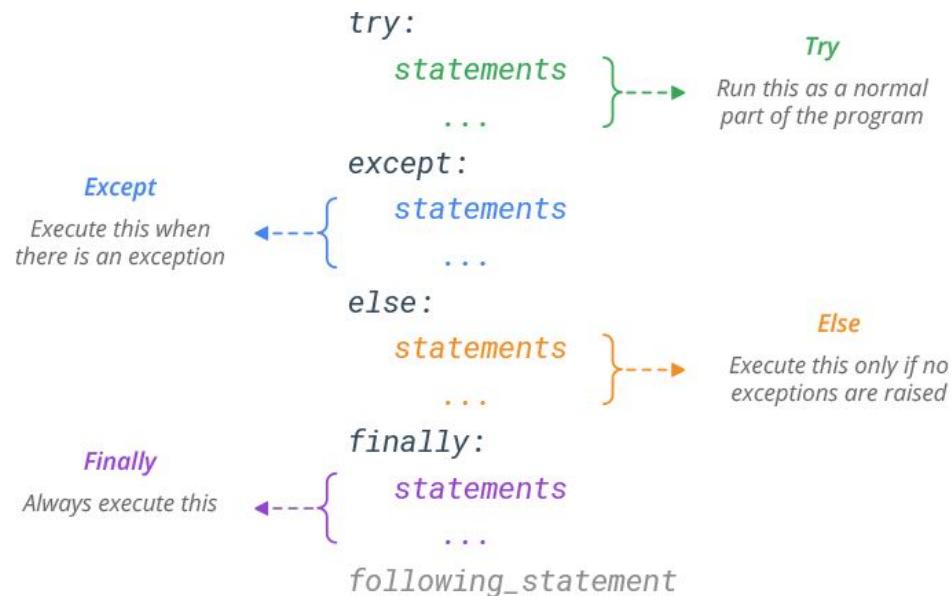
A screenshot of a Python terminal window showing the following code:

```
try:
    a = 5
    a = a / 1
    a = a + 4
except:
    print("Something is wrong here")
else:
    print("everything is fine")
```

The code consists of a `try:` block containing three assignment statements, an `except:` block with a print statement, and an `else:` block with another print statement. The first two assignments in the `try:` block are highlighted with a cyan box, and the print statement in the `else:` block is highlighted with an orange box. Two arrows point from the text "Nếu không có lỗi xuất hiện trong khối lệnh try ..." to the highlighted areas: one from the cyan box and one from the orange box.

Nếu không có lỗi xuất
hiện trong khối lệnh try ...
Thì khối lệnh else sẽ được
thực thi

Mệnh đề finally



A screenshot of a terminal window showing the execution of a Python script. The script contains the following code:

```
try:
    a = 5
    a = a / 0
    a = a + 4
except:
    print("Something is wrong here")
else:
    print("everything is fine")
finally:
    print("this message will always be printed")
```

An orange arrow points from the text "Khối lệnh finally sẽ luôn được thực thi, bất kể có lỗi trong khối lệnh try xảy ra hay không" at the bottom of the slide to the `finally:` block in the terminal output.

**Khối lệnh finally sẽ luôn được thực thi, bất kể có lỗi
trong khối lệnh try xảy ra hay không**

Numpy

