Python cho Khoa học dữ liệu

Bài 3: Các Toán Tử Trong Python - Python Operators

Hà Minh Tuấn

Khoa Toán - Tin học
Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM

Ngày 28 tháng 8 năm 2025

Nội dung bài học

Python Operators

Python Operators là các ký hiệu đặc biệt dùng để thực hiện phép toán trên toán hạng (operands).

- **Unary operators**: chỉ cần 1 toán hạng.
- **Binary operators**: cần 2 toán hạng.
- Toán hạng: biến, giá trị hoặc biểu thức.

Các loại toán tử trong Python:

1. Arithmetic Operators
2. Comparison Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

Arithmetic Operators

Arithmetic Operators thực hiện các phép toán cơ bản:

Operator	Name	Example
+	Addition	a + b = 30
-	Subtraction	a - b = -10
*	Multiplication	a * b = 200
/	Division	b / a = 2
%	Modulus	b % a = 0
**	Exponent	a**b = 10**20
//	Floor Div	9//2 = 4

Ví dụ: Arithmetic Operators

```
a = 21
b = 10

print("a+b:", a+b)
print("a-b:", a-b)
print("a*b:", a*b)
print("a/b:", a/b)
print("a%b:", a%b)

print("a**b:", 2**3)
print("a//b:", 10//5)
```

Comparison Operators

So sánh giá trị hai toán hạng (Relational Operators):

Operator	Meaning	Example
==	Equal	a == b
!=	Not Equal	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater or Equal	a >= b
<=	Less or Equal	a <= b

Ví dụ: Comparison Operators

```
a, b = 21, 10
print("a==b?", a==b)
print("a!=b?", a!=b)
print("a<b?", a<b)
print("a>b?", a>b)
```

Assignment Operators

Dùng để gán giá trị cho biến.

Operator	Example
=	a = 10
+=	a += 5 (a = a+5)
-=	a -= 2 (a = a-2)
*=	a *= 3 (a = a*3)
/=	a /= 2 (a = a/2)
%=	a %= 2 (a = a%2)
**=	a **= 3 (a = a**3)
//=	a //= 4 (a= a//4)

Bitwise Operators

Thao tác trên bit nhị phân.

Operator	Meaning	Example
&	AND	a & b
|	OR	a | b
^	XOR	a ^ b
~	NOT	~ a
$\ll$	Zero fill left shift	$a \ll 2$
$\gg$	Signed right shift	$a \gg 2$

Logical Operators

Kết hợp điều kiện logic:

Operator	Meaning	Example
and	AND	a and b
or	OR	a or b
not	NOT	not(a)

Membership & Identity Operators

Membership Operators:

- `in` : True nếu phần tử có trong chuỗi/DS.
- `not in` : True nếu phần tử không có.

Identity Operators:

- `is` : True nếu 2 biến tham chiếu cùng 1 object.
- `is not` : True nếu khác object.

Operator Precedence

Thứ tự ưu tiên toán tử (từ cao đến thấp):

1. `()` : Parentheses
2. `**` : Exponentiation
3. `+x, -x, ~ x` : Unary
4. `*, /, //, %`
5. `+, -`
6. `<<, >>`
7. `&`
8. `^, |`
9. `==, !=, >, <, >=, <=`
10. `=, +=, -=, ...`

Python Arithmetic Operators

- Dùng để thực hiện phép toán: cộng, trừ, nhân, chia, lũy thừa, chia lấy phần nguyên, chia lấy dư.
- Là toán tử nhị phân (cần 2 toán hạng).
- Python hỗ trợ số nguyên, số thực, số phức trong các phép toán.

Danh sách Arithmetic Operators

Operator	Name	Example
+	Addition	a+b=30
-	Subtraction	a-b=-10
*	Multiplication	a*b=200
/	Division	b/a=2
%	Modulus	b%a=0
**	Exponent	a**b=10**20
//	Floor Division	9//2=4

Addition Operator

- Dùng ký hiệu +, cộng hai toán hạng.
- Nếu có số thực $\rightarrow$ kết quả là float. Nếu có số phức $\rightarrow$ kết quả là complex.

```
a=10
b=20.5
print("a=",a,"b=",b,"a+b=",a+b)

a=10+5j
b=20.5
print("a=",a,"b=",b,"a+b=",a+b)
```

Subtraction Operator

- Dùng ký hiệu -, trừ toán hạng phải khỏi toán hạng trái.

```
a=10
b=20.5
print("a-b=",a-b)
print("b-a=",b-a)

a=10+5j
b=20.5
print("a-b=",a-b)
print("b-a=",b-a)
```

Multiplication Operator

- Dùng ký hiệu *, nhân hai toán hạng.

```
a=10
b=20.5
print("a*b=",a*b)

a=10+5j
b=20.5
print("a*b=",a*b)
```

Division Operator

- Dùng ký hiệu /, luôn trả về `float`.
- Chia cho 0 gây lỗi `ZeroDivisionError`.

```
a=10; b=20
print("a/b=",a/b," b/a=",b/a)

a=7.5+7.5j; b=2.5
print("a/b=",a/b," b/a=",b/a)
```

Modulus Operator (%)

- Trả về phần dư của phép chia.
- Hỗ trợ int và float (không hỗ trợ complex).

```
a=10; b=3
print("10%3 =", a%b)

a=7.7; b=2.5
print("7.7%2.5 =", a%b)
```

Exponent Operator (**)

- Dùng **, lũy thừa: a^b.

```
print("10**2 =", 10**2)
print("10**1.5 =", 10**1.5)

a=1+2j
print("a**4 =", a**4)
```

Floor Division (//)

- Trả về thương, bỏ phần thập phân.
- Với số âm $\rightarrow$ làm tròn xuống âm vô cực.

```
print("9//2 =", 9//2)
print("9//-2 =", 9//-2)
```

Precedence & Associativity

- `**` : Right to Left
- `%`, `*`, `/`, `//` : Left to Right
- `+`, `-` : Left to Right

Arithmetic with Complex Numbers

- Phép cộng/trừ: cộng trừ phần thực & ảo.
- Phép nhân: giống nhân đa thức.
- Phép chia: dùng số liên hợp (conjugate).

```
a=2.5+3.4j
b=-3+1j
print("a+b=", a+b)
print("a-b=", a-b)

a=6+4j; b=3+2j
print("a*b=", a*b)
print("a/b=", a/b)
```

Python Comparison Operators

- Comparison operators (toán tử so sánh) rất quan trọng trong `if`, `else`, `elif` và vòng lặp.
- Các toán tử phổ biến: `<`, `>`, `<=`, `>=`, `==`, `!=`.
- Mỗi biểu thức so sánh luôn trả về `True` hoặc `False`.
- Bảng các toán tử:
  - `<` : Less than
  - `>` : Greater than
  - `<=` : Less than or equal to
  - `>=` : Greater than or equal to
  - `==` : Is equal to
  - `!=` : Is not equal to

Ví dụ cơ bản

```
a=5
b=7
print(a>b)
print(a<b)
```

Kết quả:

```
False
True
```

So sánh hai số nguyên

```
print("Both operands are integer")
a=5
b=7
print("a=",a,"b=",b,"a>b is",a>b)
print("a=",a,"b=",b,"a<b is",a<b)
print("a=",a,"b=",b,"a==b is",a==b)
print("a=",a,"b=",b,"a!=b is",a!=b)
```

Kết quả:

```
Both operands are integer
a= 5 b= 7 a>b is False
a= 5 b= 7 a<b is True
a= 5 b= 7 a==b is False
a= 5 b= 7 a!=b is True
```

So sánh int và float

```
print("comparison of int and float")
a=10
b=10.0
print("a=",a,"b=",b,"a>b is",a>b)
print("a=",a,"b=",b,"a<b is",a<b)
print("a=",a,"b=",b,"a==b is",a==b)
print("a=",a,"b=",b,"a!=b is",a!=b)
```

Kết quả:

```
a= 10 b= 10.0 a>b is False
a= 10 b= 10.0 a<b is False
a= 10 b= 10.0 a==b is True
a= 10 b= 10.0 a!=b is False
```

So sánh số phức

```
print("comparison of complex numbers")
a=10+1j
b=10-1j
print("a=",a,"b=",b,"a==b is",a==b)
print("a=",a,"b=",b,"a!=b is",a!=b)
```

Kết quả:

```
a= (10+1j) b= (10-1j) a==b is False
a= (10+1j) b= (10-1j) a!=b is True
```

TypeError khi dùng $<$ hoặc $>$ với complex

```
a=10+1j
b=10-1j
print("a<b is",a<b)
```

Kết quả:

```
TypeError: '<' not supported between instances
of 'complex' and 'complex'
```

So sánh Boolean

```
print("comparison of Booleans")
a=True
b=False
print("a=",a,"b=",b,"a<b is",a<b)
print("a=",a,"b=",b,"a>b is",a>b)
print("a=",a,"b=",b,"a==b is",a==b)
print("a=",a,"b=",b,"a!=b is",a!=b)
```

Kết quả:

```
a= True b= False a<b is False
a= True b= False a>b is True
a= True b= False a==b is False
a= True b= False a!=b is True
```

So sánh khác loại sequence

Trong Python, chỉ có thể so sánh các đối tượng chuỗi tương tự. Một đối tượng chuỗi chỉ có thể so sánh với một chuỗi khác. Một danh sách không thể so sánh với một bộ, ngay cả khi cả hai đều có cùng mục.

```
print("comparison of different sequence types")
a=(1,2,3)
b=[1,2,3]
print("a<b is",a<b)
```

Kết quả:

```
TypeError: '<' not supported between instances
of 'tuple' and 'list'
```

So sánh chuỗi

Việc so sánh hai chuỗi/dãy hoạt động giống hệt như cách bạn tra từ trong từ điển. Nó so sánh từng phần tử một từ trái sang phải, và kết quả so sánh của toàn bộ chuỗi/dãy được quyết định ngay tại vị trí có sự khác biệt đầu tiên.

```
print("comparison of strings")
a='BAT'
b='BALL'
print("a<b is",a<b)
print("a>b is",a>b)
print("a==b is",a==b)
print("a!=b is",a!=b)
```

Kết quả:

```
a= BAT b= BALL a<b is False
a= BAT b= BALL a>b is True
a= BAT b= BALL a==b is False
a= BAT b= BALL a!=b is True
```

So sánh tuple

```
print("comparison of tuples")
a=(1,2,4)
b=(1,2,3)
print("a<b is",a<b)
print("a>b is",a>b)
print("a==b is",a==b)
print("a!=b is",a!=b)
```

Kết quả:

```
a= (1, 2, 4) b= (1, 2, 3) a<b is False
a= (1, 2, 4) b= (1, 2, 3) a>b is True
a= (1, 2, 4) b= (1, 2, 3) a==b is False
a= (1, 2, 4) b= (1, 2, 3) a!=b is True
```

So sánh dictionary

Không thể so sánh Lớn hơn/Nhỏ hơn: Python không cho phép dùng các toán tử $<$, $>$, $<=$, $>=$ giữa hai từ điển. Cố gắng làm vậy sẽ gây ra lỗi TypeError.
So sánh Bằng ($==$) dựa trên Nội dung: Hai từ điển được coi là bằng nhau khi và chỉ khi chúng có nội dung y hệt nhau (cùng các cặp key:value), không quan trọng thứ tự của các cặp đó. So sánh bằng Độ dài là SAI: Quan niệm cho rằng từ điển được so sánh bằng độ dài (từ điển ít phần tử hơn thì nhỏ hơn) là sai và đã lỗi thời (chỉ đúng với Python 2).

```
print("comparison of dictionary objects")
a={1:1,2:2}
b={2:2,1:1,3:3}
print("a==b is",a==b)
print("a!=b is",a!=b)
```

Kết quả:

```
a= {1: 1, 2: 2} b= {2: 2, 1: 1, 3: 3} a==b is False
a= {1: 1, 2: 2} b= {2: 2, 1: 1, 3: 3} a!=b is True
```

Python Assignment Operator

- Ký hiệu -"là toán tử gán trong Python, gán giá trị từ vế phải sang biến ở vế trái.
- Không nên nhầm với ký hiệu -"trong Toán học (biểu thị sự bằng nhau).
- Ngoài toán tử gán đơn giản, Python hỗ trợ các toán tử gán kết hợp (augmented assignment).

Ví dụ toán tử gán cơ bản

```
a = 10
b = 5
a = a + b
print(a)
```

Kết quả:

```
15
```

Augmented Assignment Operators

- Các toán tử gán kết hợp (augmented assignment operators):
  - `+=` : cộng và gán
  - `-=` : trừ và gán
  - `*=` : nhân và gán
  - `/=` : chia và gán
  - `%=` : chia lấy dư và gán
  - `**=` : lũy thừa và gán
  - `//=` : chia nguyên và gán

Augmented Addition Operator (+=)

```
a=10; b=5
print("Augmented addition of int and int")
a+=b
print("a=",a, "type(a):", type(a))

a=10; b=5.5
print("Augmented addition of int and float")
a+=b
print("a=",a, "type(a):", type(a))

a=10.50; b=5+6j
print("Augmented addition of float and complex")
a+=b
print("a=",a, "type(a):", type(a))
```

Kế

quả:

```
Augmented addition of int and int
a= 15 type(a): <class 'int'>
Augmented addition of int and float
```

Augmented Subtraction Operator (-=)

```
a=10; b=5
print("Augmented subtraction of int and int")
a-=b
print("a=",a, "type(a):", type(a))

a=10; b=5.5
print("Augmented subtraction of int and float")
a-=b
print("a=",a, "type(a):", type(a))

a=10.50; b=5+6j
print("Augmented subtraction of float and complex")
a-=b
print("a=",a, "type(a):", type(a))
```

Kế

quả:

```
Augmented subtraction of int and int
a= 5 type(a): <class 'int'>
Augmented subtraction of int and float
```

Augmented Multiplication Operator (*=)

```
a=10; b=5
print("Augmented multiplication of int and int")
a*=b
print("a=",a, "type(a):", type(a))

a=10; b=5.5
print("Augmented multiplication of int and float")
a*=b
print("a=",a, "type(a):", type(a))

a=6+4j; b=3+2j
print("Augmented multiplication of complex and complex")
a*=b
print("a=",a, "type(a):", type(a))
```

Kế

quả:

```
Augmented multiplication of int and int
a= 50 type(a): <class 'int'>
Augmented multiplication of int and float
```

Augmented Division Operator (/=)

```
a=10; b=5
print("Augmented division of int and int")
a/=b
print("a=",a, "type(a):", type(a))

a=10; b=5.5
print("Augmented division of int and float")
a/=b
print("a=",a, "type(a):", type(a))

a=6+4j; b=3+2j
print("Augmented division of complex and complex")
a/=b
print("a=",a, "type(a):", type(a))
```

Kế

quả:

```
Augmented division of int and int
a= 2.0 type(a): <class 'float'>
Augmented division of int and float
```

Augmented Modulus Operator (%=)

```
a=10; b=5
print("Augmented modulus operator with int and int")
a%=b
print("a=",a, "type(a):", type(a))

a=10; b=5.5
print("Augmented modulus operator with int and float")
a%=b
print("a=",a, "type(a):", type(a))
```

Kế

quả:

```
Augmented modulus operator with int and int
a= 0 type(a): <class 'int'>
Augmented modulus operator with int and float
a= 4.5 type(a): <class 'float'>
```

Augmented Exponent Operator (**=)

```
a=10; b=5
print("Augmented exponent operator with int and int")
a**=b
print("a=",a, "type(a):", type(a))

a=10; b=5.5
print("Augmented exponent operator with int and float")
a**=b
print("a=",a, "type(a):", type(a))

a=6+4j; b=3+2j
print("Augmented exponent operator with complex and complex")
a**=b
print("a=",a, "type(a):", type(a))
```

Kế

quả:

```
Augmented exponent operator with int and int
a= 100000 type(a): <class 'int'>
Augmented exponent operator with int and float
```

Augmented Floor Division Operator (//=)

```
a=10; b=5
print("Augmented floor division operator with int and int")
a//=b
print("a=",a, "type(a):", type(a))

a=10; b=5.5
print("Augmented floor division operator with int and float")
a//=b
print("a=",a, "type(a):", type(a))
```

Kết quả:

```
Augmented floor division operator with int and int
a= 2 type(a): <class 'int'>
Augmented floor division operator with int and float
a= 1.0 type(a): <class 'float'>
```

Python Logical Operators

Python cung cấp các toán tử logic để tạo biểu thức Boolean phức hợp. Mỗi toán hạng của các toán tử này đều là một biểu thức Boolean.

- Ví dụ:
  - `age > 16 and marks > 80`
  - `percentage < 50 or attendance < 75`
- Các giá trị được coi là `False`: `False`, `None`, số 0, chuỗi rỗng, list rỗng, tuple rỗng, dict rỗng, set rỗng.
- Các giá trị khác được coi là `True`.
- Các toán tử logic trong Python: `and`, `or`, `not`.

Toán tử and

- Trả về `True` khi cả hai toán hạng đều `True`.
- Nếu một trong hai hoặc cả hai là `False` thì kết quả là `False`.

Bảng chân trị của `and`:

a	b	a and b
F	F	F
F	T	F
T	F	F
T	T	T

Toán tử `or`

- Trả về `True` nếu ít nhất một toán hạng là `True`.
- Chỉ trả về `False` khi cả hai toán hạng đều `False`.

Bảng chân trị của `or`:

a	b	a or b
F	F	F
F	T	T
T	F	T
T	T	T

Toán tử `not`

- Đây là toán tử một ngôi.
- Nó đảo ngược giá trị Boolean:
  - `not True` $\rightarrow$ `False`
  - `not False` $\rightarrow$ `True`

Bảng chân trị của `not`:

a	not(a)
F	T
T	F

Cách Python đánh giá toán tử logic

- Biểu thức `x and y`:
  - Đầu tiên đánh giá `x`.
  - Nếu `x` là `False` thì trả về `x`.
  - Nếu `x` là `True`, đánh giá `y` và trả về `y`.
- Biểu thức `x or y`:
  - Đầu tiên đánh giá `x`.
  - Nếu `x` là `True`, trả về `x`.
  - Nếu `x` là `False`, đánh giá `y` và trả về `y`.

Ví dụ 1: Toán tử logic với điều kiện Boolean

```
x = 10
y = 20
print("x > 0 and x < 10:", x > 0 and x < 10)
print("x > 0 and y > 10:", x > 0 and y > 10)
print("x > 10 or y > 10:", x > 10 or y > 10)
print("x%2 == 0 and y%2 == 0:", x%2 == 0 and y%2 == 0)
print("not (x+y>15):", not (x+y>15))
```

```
x > 0 and x < 10: False
x > 0 and y > 10: True
x > 10 or y > 10: True
x%2 == 0 and y%2 == 0: True
not (x+y>15): False
```

Ví dụ 2: Toán tử logic với số Non-Boolean

Chúng ta có thể sử dụng toán hạng không phải boolean với các toán tử logic. Ở đây, chúng ta cần lưu ý rằng bất kỳ số nào khác không và các chuỗi không rỗng đều được đánh giá là True. Do đó, các bảng chân trị của toán tử logic cũng được áp dụng.

Trong ví dụ sau, toán hạng số được sử dụng cho các toán tử logic. Các biến "x", "y"được đánh giá là True, "z"được đánh giá là False.

```
x = 10
y = 20
z = 0
print("x and y:", x and y)
print("x or y:", x or y)
print("z or x:", z or x)
print("y or z:", y or z)
```

```
x and y: 20
x or y: 10
z or x: 10
y or z: 20
```

Ví dụ 3: Toán tử logic với chuỗi và tuple

Biến chuỗi được coi là True và một bộ rỗng được coi là False trong ví dụ sau

```
a = "Hello"
b = tuple()
print("a and b:", a and b)
print("b or a:", b or a)
```

```
a and b: ()
b or a: Hello
```

Ví dụ 4: Toán tử logic với danh sách (list)

Cuối cùng, hai đối tượng danh sách bên dưới không rỗng. Do đó, x and y trả về đối tượng sau, còn x or y trả về đối tượng trước.

```
x = [1, 2, 3]
y = [10, 20, 30]
print("x and y:", x and y)
print("x or y:", x or y)
```

```
x and y: [10, 20, 30]
x or y: [1, 2, 3]
```

Python Bitwise Operators

- Các toán tử bitwise trong Python được dùng để thao tác trên từng bit của số nguyên.
- Có 6 toán tử bitwise: &, |, ^, ~, <<, >>.
- Ngoại trừ ~, các toán tử này đều là nhị phân (cần 2 toán hạng).
- Mỗi toán hạng được coi như một chuỗi bit (0 hoặc 1).

Bitwise AND Operator (&)

- Giống như toán tử logic `and`, nhưng hoạt động trên từng bit.
- Chỉ trả về 1 khi cả hai bit cùng là 1.

Các trường hợp:

- $0 \,\&\, 0 = 0$
- $1 \,\&\, 0 = 0$
- $0 \,\&\, 1 = 0$
- $1 \,\&\, 1 = 1$

Ví dụ: Bitwise AND

```
a = 60
b = 13
print("a:", a, "b:", b, "a&b:", a & b)
print("a:", bin(a))
print("b:", bin(b))
```

```
a: 60 b: 13 a&b: 12
a: 0b111100
b: 0b1101
```

Bitwise OR Operator (|)

- Toán tử | (pipe) trả về 1 nếu một trong hai bit là 1.

Các trường hợp:

- $0 \mid 0 = 0$
- $0 \mid 1 = 1$
- $1 \mid 0 = 1$
- $1 \mid 1 = 1$

Ví dụ: Bitwise OR

```
a = 60
b = 13
print("a:", a, "b:", b, "a|b:", a | b)
print("a:", bin(a))
print("b:", bin(b))
```

```
a: 60 b: 13 a|b: 61
a: 0b111100
b: 0b1101
```

Bitwise XOR Operator ($\hat{}$)

- XOR (exclusive OR): kết quả là 1 nếu chỉ một trong hai bit bằng 1.

Các trường hợp:

- $0 \hat{} 0 = 0$
- $0 \hat{} 1 = 1$
- $1 \hat{} 0 = 1$
- $1 \hat{} 1 = 0$

Ví dụ: Bitwise XOR

```
a = 60
b = 13
print("a:", a, "b:", b, "a^b:", a ^ b)
```

```
a: 60 b: 13 a^b: 49
```

Bitwise NOT Operator (~)

- Đảo ngược tất cả các bit (1 thành 0, 0 thành 1).
- Python dùng biểu diễn bù 2 (two's complement).
- Ví dụ: `60 = -61`.

Ví dụ: Bitwise NOT

```
a = 60
print("a:", a, "~a:", ~a)
```

```
a: 60 ~a: -61
```

Bitwise Left Shift ($\ll$)

- Dịch trái số bit theo số chỉ định.
- Ví dụ: $x \ll 2$ dịch trái 2 bit.
- Kết quả tương ứng với nhân số đó với 2^n.

Ví dụ: Left Shift

```
a = 60
print("a:", a, "a<<2:", a << 2)
```

```
a: 60 a<<2: 240
```

Bitwise Right Shift ($\gg$)

- Dịch phải số bit theo số chỉ định.
- Ví dụ: $x \gg 2$ dịch phải 2 bit.
- Kết quả tương ứng với chia lấy phần nguyên cho 2^n.

Ví dụ: Right Shift

```
a = 60
print("a:", a, "a>>2:", a >> 2)
```

```
a: 60 a>>2: 15
```

Sự Khác Biệt: Toán Tử Thông Thường vs. Toán Tử Bitwise

- **Toán tử thông thường** (`+`, `-`, `*`, `>`, `and`...):
  - Làm việc trực tiếp trên **giá trị toán học** của dữ liệu.
  - Thực hiện các phép tính, so sánh và logic quen thuộc trong toán học và lập trình hằng ngày.
- **Toán tử bitwise** (`&`, `|`, `^`, `<<`, `>>`...):
  - Thao tác trên **từng bit nhị phân** (0 hoặc 1) của số nguyên.
  - Được dùng cho các tác vụ chuyên biệt, cấp thấp như:
    - ★ Tối ưu hóa hiệu năng.
    - ★ Xử lý cờ trạng thái (*flags*).
    - ★ Lập trình hệ thống, nhúng.

Python Membership Operators

- Toán tử **Membership** trong Python dùng để kiểm tra một phần tử có nằm trong một **container** (string, list, tuple, set, dict) hay không.
- Có 2 toán tử:
  - `in` : trả về `True` nếu phần tử thuộc container.
  - `not in` : trả về `True` nếu phần tử không thuộc container.
- Cả hai đều trả về giá trị `Boolean`.

Ví dụ: Membership với Chuỗi

```
var = "TutorialsPoint"
a = "P"
b = "tor"
c = "in"
d = "To"
print(a, "in", var, ":", a in var)
print(b, "in", var, ":", b in var)
print(c, "in", var, ":", c in var)
print(d, "in", var, ":", d in var)
```

```
P in TutorialsPoint : True
tor in TutorialsPoint : True
in in TutorialsPoint : True
To in TutorialsPoint : False
```

Ví dụ: Membership với Chuỗi (not in)

```
var = "TutorialsPoint"
a = "P"
b = "tor"
c = "in"
d = "To"
print(a, "not in", var, ":", a not in var)
print(b, "not in", var, ":", b not in var)
print(c, "not in", var, ":", c not in var)
print(d, "not in", var, ":", d not in var)
```

```
P not in TutorialsPoint : False
tor not in TutorialsPoint : False
in not in TutorialsPoint : False
To not in TutorialsPoint : True
```

Membership với List

```
var = [10,20,30,40]
a = 20
b = 10
c = a-b
d = a/2
print(a, "in", var, ":", a in var)
print(b, "not in", var, ":", b not in var)
print(c, "in", var, ":", c in var)
print(d, "not in", var, ":", d not in var)
```

```
20 in [10, 20, 30, 40] : True
10 not in [10, 20, 30, 40] : False
10 in [10, 20, 30, 40] : True
10.0 not in [10, 20, 30, 40] : False
```

Membership với Tuple

```
var = (10,20,30,40)
a = 10
b = 20
print((a,b), "in", var, ":", (a,b) in var)

var = ((10,20),30,40)
print((a,b), "in", var, ":", (a,b) in var)
```

```
(10, 20) in (10, 20, 30, 40) : False
(10, 20) in ((10, 20), 30, 40) : True
```

Membership với Set

```
var = {10,20,30,40}
a = 10
b = 20
print(b, "in", var, ":", b in var)

var = {(10,20),30,40}
print((a,b), "in", var, ":", (a,b) in var)
```

```
20 in {40, 10, 20, 30} : True
(10, 20) in {40, 30, (10, 20)} : True
```

Membership với Dictionary

```
var = {1:10, 2:20, 3:30}
a = 2
b = 20
print(a, "in", var, ":", a in var)
print(b, "in", var, ":", b in var)
```

```
2 in {1: 10, 2: 20, 3: 30} : True
20 in {1: 10, 2: 20, 3: 30} : False
```

Python Identity Operators

- Toán tử **Identity** so sánh các đối tượng để kiểm tra xem chúng có cùng vùng nhớ (memory) hay cùng kiểu dữ liệu không.
- Python cung cấp 2 toán tử Identity:
  - `is` : True nếu cả hai đối tượng cùng trỏ tới cùng một vùng nhớ.
  - `is not` : True nếu hai đối tượng không cùng vùng nhớ.
- Hàm `id()` dùng để lấy địa chỉ vùng nhớ của một đối tượng.

Ví dụ: Toán tử is

```
a = [1, 2, 3, 4, 5]
b = [1, 2, 3, 4, 5]
c = a

print(a is c)
print(a is b)

print("id(a) :", id(a))
print("id(b) :", id(b))
print("id(c) :", id(c))
```

```
True
False
id(a) :  140114091859456
id(b) :  140114091906944
id(c) :  140114091859456
```

Ví dụ: Toán tử is not

```
a = [1, 2, 3, 4, 5]
b = [1, 2, 3, 4, 5]
c = a

print(a is not c)
print(a is not b)

print("id(a) :", id(a))
print("id(b) :", id(b))
print("id(c) :", id(c))
```

```
False
True
id(a) :  140559927442176
id(b) :  140559925598080
id(c) :  140559927442176
```

Ví dụ Identity với Chuỗi

```
a = "TutorialsPoint"
b = a
print("id(a), id(b):", id(a), id(b))
print("a is b:", a is b)
print("b is not a:", b is not a)
```

```
id(a), id(b): 2739311598832 2739311598832
a is b: True
b is not a: False
```

Ví dụ Identity với List

```
a = [1,2,3]
b = [1,2,3]
print("id(a), id(b):", id(a), id(b))
print("a is b:", a is b)
print("b is not a:", b is not a)

print(id(a[0]), id(a[1]), id(a[2]))
print(id(b[0]), id(b[1]), id(b[2]))
```

```
id(a), id(b): 1552612704640 1552567805568
a is b: False
b is not a: True
140734682034984 140734682035016 140734682035048
140734682034984 140734682035016 140734682035048
```

Giải thích

- Dù hai list `a` và `b` chứa cùng các giá trị, vùng nhớ của chúng khác nhau.
- Các toán tử `is` và `is not` so sánh **vùng nhớ**, không phải giá trị.
- Trong ví dụ trên, các phần tử bên trong cùng trỏ tới cùng vùng nhớ các số nguyên, nên id của phần tử giống nhau.

Python Operator Precedence

- Khi một biểu thức có nhiều toán tử, **operator precedence** xác định thứ tự thực thi.
- Python tuân theo quy tắc BODMAS với các toán tử số học:
  - Dấu ngoặc trước, nhân/chia sau, cộng/trừ cuối cùng.
- **Associativity**: Nếu các toán tử cùng mức precedence, Python thực hiện từ trái sang phải.

Ví dụ: Precedence và Associativity

```
a = 2 + 3 * 5
print("a =", a)   # 3*5=15, 15+2=17

b = 10 / 5 * 4
print("b =", b)   # 10/5=2, 2*4=8
```

```
a = 17
b = 8.0
```

Bảng Operator Precedence trong Python

STT	Operator & Description
1	(), [], {} - Parentheses and braces
2	[index], [index:index] - Subscription, slicing
3	await x - Await expression
4	** - Exponentiation
5	+x, -x, ~x - Positive, negative, bitwise NOT
6	*, @, /, //, % - Multiplication, matrix multiply, division, floor division, remainder
7	+, - - Addition and subtraction

Bảng Operator Precedence trong Python

STT	Operator & Description
8	«, » - Left & Right shifts
9	& - Bitwise AND
10	$\hat{}$ - Bitwise XOR
11	| - Bitwise OR
12	in, not in, is, is not, <, <=, >, >=, !=, == - Comparisons
13	not x - Boolean NOT
14	and - Boolean AND
15	or - Boolean OR
16	if else - Conditional expression
17	lambda - Lambda expression

Ví dụ: Python Operator Precedence

```
a = 20
b = 10
c = 15
d = 5
e = 0
e = (a + b) * c / d        #( 30 * 15 ) / 5
print("Value of (a + b) * c / d is ", e)
e = ((a + b) * c) / d      # (30 * 15 ) / 5
print("Value of ((a + b) * c) / d is ", e)
e = (a + b) * (c / d);     # (30) * (15/5)
print("Value of (a + b) * (c / d) is ", e)
e = a + (b * c) / d;       #  20 + (150/5)
print("Value of a + (b * c) / d is ", e)
```

```
Value of (a + b) * c / d is  90.0
Value of ((a + b) * c) / d is  90.0
Value of (a + b) * (c / d) is  90.0
Value of a + (b * c) / d is  50.0
```