

Day 10 – Operators and Number Systems in Python

Understand Python's core operator categories and learn to work with binary, octal, and hexadecimal number systems.

Python Operators

Python provides several types of operators used for performing different types of operations.

1. Arithmetic Operators

Used to perform basic mathematical operations like addition, subtraction, multiplication, etc.

- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division (returns float)
- `%` Modulus (remainder)
- `//` Floor Division (returns integer)
- `**` Exponentiation (power)

```
In [2]: a = 10
b = 3
print('Addition:', a + b)
print('Subtraction:', a - b)
print('Multiplication:', a * b)
print('Division:', a / b)
print('Modulus:', a % b)
print('Floor Division:', a // b)
print('Exponentiation:', a ** b)
```

Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.333333333333335
Modulus: 1
Floor Division: 3
Exponentiation: 1000

2. Comparison (Relational) Operators

Used to compare two values and return a boolean result (True or False).

- `==` Equal
- `!=` Not Equal
- `>` Greater Than
- `<` Less Than
- `>=` Greater Than or Equal To
- `<=` Less Than or Equal To

```
In [3]: print('Equal:', a == b)
print('Not Equal:', a != b)
print('Greater Than:', a > b)
print('Less Than:', a < b)
```

```
print('Greater or Equal:', a >= b)
print('Less or Equal:', a <= b)
```

```
Equal: False
Not Equal: True
Greater Than: True
Less Than: False
Greater or Equal: True
Less or Equal: False
```

3. Logical Operators

Used to combine multiple conditions:

- `and` : True if both operands are true
- `or` : True if at least one operand is true
- `not` : Inverts the truth value

```
In [4]: # Logical operations with comments

x = 5
y = 10

print(x > 2 and y > 5)    # ► True (both conditions are True)
print(x > 6 and y > 5)    # ► False (x > 6 is False)

print(x > 6 or y > 5)    # ► True (one condition is True)
print(x > 6 or y < 5)    # ► False (both conditions are False)

print(not (x > 6))       # ► True (x > 6 is False, not makes it True)
```

```
True
False
True
False
True
```

4. Identity Operators

Check whether two variables point to the same object in memory:

- `is` : True if both refer to the same object
- `is not` : True if they do not refer to the same object

```
In [5]: # Identity operator examples

a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(a is c)      # ► True (c refers to same object as a)
print(a is b)      # ► False (b is a different object with same content)
print(a == b)      # ► True (values are the same)
print(a is not b)  # ► True (different objects)
```

```
True
False
True
True
```

5. Membership Operators

Used to test if a value is in a sequence:

- `in` : True if value exists
- `not in` : True if value does not exist

```
In [6]: # Membership operator examples

fruits = ['apple', 'banana', 'mango']

print('apple' in fruits)      # ► True
print('grape' in fruits)     # ► False
print('grape' not in fruits) # ► True

name = "Mubasshir"
print('a' in name)          # ► True
print('f' not in name)       # ► True
```

True
False
True
True
True

6. Bitwise Operators

Bitwise operators work on bits and perform bit-by-bit operations. They are mainly used in low-level programming, binary calculations, and performance tuning. Used to compare (binary) bits:

6.1. `&` : AND - 1 only if both bits are 1

Rule: $1 \& 1 = 1, 1 \& 0 = 0, 0 \& 1 = 0, 0 \& 0 = 0$

6.2. `|` : OR - 1 if at least one bit is 1

Rule: $1 | 1 = 1, 1 | 0 = 1, 0 | 1 = 1, 0 | 0 = 0$

6.3. `^` : XOR - 1 if bits are different

Rule: $1 ^ 1 = 0, 1 ^ 0 = 1, 0 ^ 1 = 1, 0 ^ 0 = 0$

6.4. `~` : NOT - flips all bits (inverts $1 \rightarrow 0$ and $0 \rightarrow 1$), result is negative in Python

6.5. `<<` : Left Shift - shifts bits left, like multiplying by 2

6.6. `>>` : Right Shift - shifts bits right, like dividing by 2

```
In [8]: a = 5 # 0b0101
b = 3 # 0b0011
print('Bitwise AND:', a & b)
print('Bitwise OR:', a | b)
print('Bitwise XOR:', a ^ b)
print('Bitwise NOT a:', ~a)
print('Left Shift a:', a << 1)
print('Right Shift b:', b >> 1)
```

Bitwise AND: 1
Bitwise OR: 7
Bitwise XOR: 6
Bitwise NOT a: -6
Left Shift a: 10
Right Shift b: 1

Number Systems in Python

Python can handle numbers in various bases like binary, octal, and hexadecimal.

Python supports conversions between different number systems:

- `bin()` : Convert to binary string
- `oct()` : Convert to octal string
- `hex()` : Convert to hexadecimal string
- `int(str, base)` : Convert from string with base to decimal integer

```
In [10]: num = 25
print('Binary:', bin(num))      # 0b11001
print('Octal:', oct(num))       # 0o31
print('Hexadecimal:', hex(num)) # 0x19
```

```
Binary: 0b11001
Octal: 0o31
Hexadecimal: 0x19
```

```
In [12]: # Conversion from Strings with Base
print('Binary to Decimal:', int('11001', 2))
print('Octal to Decimal:', int('31', 8))
print('Hex to Decimal:', int('19', 16))
```

```
Binary to Decimal: 25
Octal to Decimal: 25
Hex to Decimal: 25
```