# Operation

Kittikun Jitpairod

## Introduction to Operations

Operations are the building blocks of computation in Python, allowing us to manipulate data and make decisions in our code.

We'll cover four main types of operations:

1. Arithmetic Operations
2. Comparison Operations
3. Logical Operations
4. Membership Operations
5. Identity Operations

## Arithmetic Operations

Arithmetic operations in Python allow us to perform mathematical calculations. Let's look at the basic arithmetic operators:

```python
# Addition
print("5 + 3 =", 5 + 3)
```

```
5 + 3 = 8
```

```python
# Subtraction
print("5 - 3 =", 5 - 3)
```

```
5 - 3 = 2
```

```python
# Multiplication
print("5 * 3 =", 5 * 3)
```

```
5 * 3 = 15
```

```
# Division (always returns a float)
print("5 / 3 =", 5 / 3)
```

```
5 / 3 = 1.6666666666666667
```

```
# Exponentiation
print("5 ** 3 =", 5 ** 3)
```

```
5 ** 3 = 125
```

```
# Floor Division (returns the largest integer <= the result)
print("5 // 3 =", 5 // 3)
```

```
5 // 3 = 1
```

```
# Modulus (remainder of division)
print("5 % 3 =", 5 % 3)
```

```
5 % 3 = 2
```

It's important to note the difference between / and // in division operations, especially when working with integers.

Python also provides augmented assignment operators that combine an arithmetic operation with assignment:

```
x = 5
x += 3  # Equivalent to x = x + 3
print("x after x += 3:", x)
```

```
x after x += 3: 8
```

```
x -= 2  # Equivalent to x = x - 2
print("x after x -= 2:", x)
```

```
x after x -= 2: 6
```

```python
x *= 4  # Equivalent to x = x * 4
print("x after x *= 4:", x)
```

```
x after x *= 4: 24
```

```python
x /= 2  # Equivalent to x = x / 2
print("x after x /= 2:", x)
```

```
x after x /= 2: 12.0
```

### Comparison Operations

Comparison operations allow us to compare values and return boolean results (True or False). These are crucial for conditional statements and loops.

```python
x, y = 5, 10
# Equal to
print("x == y:", x == y)

# Not equal to
print("x != y:", x != y)
```

```
x == y: False
x != y: True
```

```python
# Greater than
print("x > y:", x > y)

# Less than
print("x < y:", x < y)
```

```
x > y: False
x < y: True
```

```python
# Greater than or equal to
print("x >= y:", x >= y)

# Less than or equal to
print("x <= y:", x <= y)
```

```
x >= y: False
x <= y: True
```

These operators work with numbers, strings, and other data types. For example, string comparisons are done lexicographically.

```python
# String comparison
print("'apple' < 'banana':", 'apple' < 'banana')
print("'apple' == 'Apple':", 'apple' == 'Apple')
```

```
'apple' < 'banana': True
'apple' == 'Apple': False
```

When Python compares strings:

1. Character-by-Character Comparison:
    - Python compares strings character by character, from left to right.
    - It uses the Unicode code point of each character for comparison.
2. ASCII and Unicode Values:
    - For ASCII characters, the comparison is straightforward. For example, a (97) comes before `b` (98).
    - Unicode characters are compared based on their Unicode code points.
3. Case Sensitivity:
    - String comparisons are case-sensitive by default.
    - Uppercase letters have lower ASCII/Unicode values than lowercase letters.
4. String Length:
    - If the characters are the same up to the length of the shorter string, the shorter string is considered “less than” the longer string.
5. Lexicographical Order:
    - This is similar to dictionary order, but based on character codes rather than alphabet position.

In the examples:

- `apple` < banana is `True` because a comes before `b` in lexicographical order.
- `apple` == `Apple` is False because case matters, and uppercase `A` has a different Unicode value than lowercase a.

## Logical Operations

Logical operations allow us to combine boolean expressions. The three main logical operators in Python are and, or, and not.

```
x, y = 5, 10

# and operator
print("x < 10 and y > 5:", x < 10 and y > 5)

# or operator
print("x < 3 or y > 5:", x < 3 or y > 5)

# not operator
print("not x == y:", not x == y)
```

```
x < 10 and y > 5: True
x < 3 or y > 5: True
not x == y: True
```

| X | Y | X and Y | X or Y |
|---|---|---|---|
| T | T | T | T |
| T | F | F | T |
| F | T | F | T |
| F | F | F | F |

## Membership Operations

Membership operations are used to test whether a value or variable is found in a sequence (such as a string, list, tuple, set or dictionary).

Python has two membership operators:

1. in: Returns True if a value is **in** the object
2. not in: Returns True if a value is **not in** the object

```
# List membership
fruits = ['apple','banana','cherry']
print('banana' in fruits)
print('orange' in fruits)
print('orange' not in fruits)
```

```
True
False
True
```

```
# String membership
text = "Hello, World!"
print('H' in text)
print('hello' in text)
print('Python' not in text)
```

```
True
False
True
```

```
# Dictionary membership
# (checks keys, not values)
person = {'name': 'Alice', 'age': 25}
print('name' in person)
print('Alice' in person)
print('height' not in person)
```

```
True
False
True
```

### Identity Operations

Identity operators are used to compare the memory locations of two objects. Python has two identity operators:

1. `is`: Returns True if both variables are the same object
2. `is not`: Returns True if both variables are not the same object

It's important to note that `is` is not the same as `==`. The `==` operator compares the value or equality of two objects, whereas `is` compares their identity.

It's generally recommended to use `==` for value comparisons and reserve `is` for comparing with `None` or when you explicitly want to check if two variables refer to the exact same object.

```
# None comparison
x = None
print(x is None)
print(x == None)
```

```
True
True
```

```python
# Integer identity
a, b = 5, 5
print(a is b)
print(a == b)
print(f'id A: {id(a)}, id B: {id(b)}')
```

```
True
True
id A: 4398687352, id B: 4398687352
```

```python
# Lists (mutable objects: can be changed after creation)
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(f"list1 == list2: {list1 == list2}")
print(f"list1 is list2: {list1 is list2}")
print(f"id(list1): {id(list1)}, id(list2): {id(list2)}")
```

```
list1 == list2: True
list1 is list2: False
id(list1): 4428005120, id(list2): 4427885056
```