- **created by**: Sudip Ghimire
- **URL**: https://www.sudipghimire.com.np
- **GitHub**: https://github.com/ghimiresdp


go to course contents

# Chapter 2.5. Basic Operations

**Table of Contents**



## arithmetic operations

Arithmetic operations includes the basic arithmetic operations to different data types. The following are the basic arithmetic operations

### Addition

**Syntax**: `a + b`

```
a1 = 5 + 6  # integer and integer
a2 = 5.5 + 2.3  # float and float
a3 = 4.7 + 2  # integer and float
a4 = "Hello " + "world"  # string and string
a5 = (5 + 4j) + (6 + 5j)  # complex and complex
```

## Subtraction

**Syntax**: `a - b`

```
b1 = 5 - 6  # integer and integer
b2 = 5.5 - 2.3  # float and float
b3 = 4.7 - 2  # integer and float
# b4 = "Hello " - "world"  # string subtraction not supported
b5 = (5 + 4j) - (6 + 5j)  # complex and complex
```

## Multiplication

**Syntax**: `a * b`

```
c1 = 5 * 6  # integer and integer
c2 = 5.5 * 2.3  # float and float
c3 = 4.7 * 2  # integer and float
c4 = "Hello " * 5  # string and int
c5 = (5 + 4j) * (6 + 5j)  # complex and complex
```

## Division

**Syntax**: `a / b`

```
d1 = 5 / 6  # integer and integer
d2 = 5.5 / 2.3  # float and float
d3 = 4.7 / 2  # integer and float
d4 = (5 + 4j) / (6 + 5j)  # complex and complex
```

## Modulus

**Syntax**: `a % b`

```
e1 = 55 % 4  # integer and integer
e2 = 5.5 % 2.3  # float and float
e3 = 4.7 % 2  # integer and float
```

### Exponentiation

**Syntax**: a ** b`

```
f1 = 5 ** 4  # integer and integer
f2 = 5.5 ** 2.3  # float and float
f3 = 4.7 ** 2  # integer and float
```

### Integer Division

**Syntax**: a // b`

```
print(5 // 2)
g1 = 45 // 2
g2 = 45.8 // 5.1  # 8.0   # integer equivalent of float
```

## Relational Operations

Relational operations or comparison operations compare 2 values and returns either `True` or `False`. When comparing `less than` or `greater than` in strings or sequences, it compares the ASCII value using lexicographical ordering.

### equals (==)

```
print(5 == 6)  # False
print(4 + 1 == 6 - 1)  # True
print('John' == 'John')  # True
```

### not equals (!=)

```
print(5 != 6)  # True
print(4 + 1 != 6 - 1)  # False
print('John' != 'John')  # False
```

### less than (<)

```
print(4 < 5)  # True
print(4 < 4)  # False
print(5 < 4)  # False
```

```
print('Jane' < 'John')  # True
```

less than or equals (`<=`)

```
print(4 <= 5)  # True
print(4 <= 4)  # True
print(5 <= 4)  # False
print([1, 2, 3] <= [1, 3, 2])  # True
```

greater than (`>`)

```
print(4 > 5)  # False
print(4 > 4)  # False
print(5 > 4)  # True
print('Jane' > 'John')  # False
print([1, 2, 3] > [1, 3, 2])  # False
```

greater than or equals (`>=`)

```
print(4 >= 5)  # False
print(4 >= 4)  # True
print(5 >= 4)  # True
print('A' >= 'B')  # False
```

## logical operations

Logical operations are performed on boolean values. The following are logical operations available in python.

### `and` operation

- **Syntax**: `[value_1] and [value_2]`
- returns true when both `value_1` and `value_2` are `True`.

```
print(True and True)  # True
print(True and False)  # False
print(False and True)  # False
print(False and False)  # False
```

An another example

```
is_married = True
has_children = False
is_life_complete = is_married and has_children

print(is_life_complete)  # False
```

### `or` operation

- **Syntax**: `[value_1] or [value_2]`
- returns true when any or all of `value_1` and `value_2` are `True`.

```
print(True or True)  # True
print(True or False)  # True
print(False or True)  # True
print(False or False)  # False
```

an another example:

```
is_father_rich = True
has_high_income = False
is_rich = is_father_rich or has_high_income

print(is_rich)  # True
```

### `not` operation

- **Syntax**: `not [value]`
- returns the negative of the `value`.

```
print(not True)  # False
print(not False)  # True

is_work = False
print(not is_work)  # True
```

### Compound logical operations

We can combine multiple logical operations to form a compound logical operation. We use compound logical operation to perform advanced operations. We generally use brackets to combine multiple logical operations.

some example are shown below:

```

```

```
print(True or False or True)  # True
print(True and False or False and True)  # False

# in some conditions, use of brackets alter the expected result
print(not True or not False)  # True
print(not (True or not False))  # False
print((not True) or (not False))  # True
```

another example:

```
# if it is cloudy today and if it rained yesterday, then it rains today
# in other conditions it doesn't rain
cloudy = False
rained = True

rains = cloudy and rained

if rains:
    print("it rains today")
else:
    print("It does not rain today")

# it does not rain today
```

## identity operations

Identity operation compares whether 2 objects are same objects or not. Remember, They are not used to compare for equality. There are 2 basic identity operations

1. `is`
2. `is not`

```
print(type('abc') is str)       # True
```

```
list1 = ['abc', 'def']
list2 = list1           # same object is referenced here
list3 = list1.copy()    # shallow copy is made here

print(list1 is list2)   # True
print(list1 is list3)   # False

print(type(list1) is not int)  # True
```

**Note**: Sometimes, we use identity operations in wrong places. for example:

```
print((1 + 4) is (6 - 1))
```

The above statement gives the equality, but still shows warning suggesing `==` instead of `is` operator.

## membership operations

Membership Operation checks if an element is present in the specified object or collection or not.

Basic Membership Operators:

- `in`
- `not in`

Example 1:

```
sentence = 'A quick brown fox jumps over the lazy dog.'
print('fox' in sentence)    # True
print('monkey' in sentence) # False

print('fox' not in sentence)    # False
print('monkey' not in sentence) # True
```

Example 2:

```
x = [1, 2, 3, 4, 5]
print(5 in x)   # True
print(10 in x)  # False

print(2 not in x)   # False
print(20 not in x)  # True
```

## bitwise operations

## assignment operations

Assignment Operations assign values from right side operands to left side operand/operands. Following are assignment operations used in python:

1. `=`
    - example: `x = 5`
2. `+=`
    - example: `x += 5`
    - equivalent code: `x = x + 5`

3. `-=`
    - example: `x -= 5`
    - equivalent code: `x = x - 5`

**Other assignment operations:**

1. `*=`
2. `/=`
3. `%=`
4. `//=`
5. `**=`
6. `&=`
7. `|=`
8. `^=`
9. `>>=`
10. `<<=`