Логическите оператори `AND`, `OR` и `NOT` са основни инструменти в булевата алгебра, използвани за създаване на сложни логически изрази от по-прости булеви изрази. Ето как функционират тези оператори в табличен вид:

Таблица на истинност за `AND`

`AND` връща истина (`True`), само ако и двете сравнявани стойности са истина.

A	B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

Таблица на истинност за `OR`

`OR` връща истина, ако поне една от сравняваните стойности е истина.

A	B	A OR B
True	True	True
True	False	True
False	True	True
False	False	False

Таблица на истинност за `NOT`

`NOT` е унарен оператор, който инвертира булевата стойност — превръща `True` в `False` и обратно.

A	NOT A
True	False
False	True

Логическите оператори `AND`, `OR`, и `NOT` могат да бъдат представени с използване на 0 и 1, където 0 означава `False` (ложно), а 1 означава `True` (истина). Ето таблиците на истинност за тези оператори, използвайки 0 и 1

Побитовите операции са операции, които се изпълняват директно върху двоичното представяне на числата. Те работят на ниво битове и са много бързи и ефективни за задачи като настройка на флагове, маскиране на битове, и др. В Python има шест основни побитови оператора:

Таблица на истинност за `AND`

Операторът `AND` връща 1 (истина), само ако и двете сравнявани стойности са 1.

A (0 или 1)	B (0 или 1)	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

Таблица на истинност за `OR`

Операторът `OR` връща 1, ако поне една от сравняваните стойности е 1.

A (0 или 1)	B (0 или 1)	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

Таблица на истинност за `NOT`

`NOT` е унарен оператор, който инвертира стойността — превръща 1 в 0 и обратно.

A (0 или 1)	NOT A
1	0
0	1

1. **AND** (`&`): Побитово И между два операнда. **Резултатът има бит, който е 1, само ако и двата бита на операндите са 1.**

   Пример:

   ```
   a = 5  # (в двоичен вид 101)
   b = 3  # (в двоичен вид 011)
   result = a & b  # (101 & 011 = 001)
   print(result)  # резултатът е 1
   ```

2. **OR** (`|`): Побитово ИЛИ между два операнда. **Битът в резултата е 1, ако поне един от двата бита е 1.**

   Пример:

   ```
   a = 5  # (в двоичен вид 101)
   b = 3  # (в двоичен вид 011)
   result = a | b  # (101 | 011 = 111)
   ```

```
print(result)  # резултатът е 7
```

3. **XOR** (^): Побитово изключващо ИЛИ. **Битът в резултата е 1, ако само един от двата бита е 1, но не и двата.**

   Пример:

   ```
   a = 5  # (в двоичен вид 101)
   b = 3  # (в двоичен вид 011)
   result = a ^ b  # (101 ^ 011 = 110)
   print(result)  # резултатът е 6
   ```

4. **NOT** (~): Побитово НЕ, което обръща всеки бит (1 става 0, а 0 става 1).

   Пример:

   ```
   a = 5  # (в двоичен вид 101)
   result = ~a  # (~101 = -110 в двоично представяне, което е -6)
   print(result)  # резултатът е -6
   ```

   Забележка: В Python отрицателните числа се представят с помощта на допълнителен код, което е причината за този резултат.

5. **Ляво изместване** (<<): Премества всички битове на операнда наляво с даден брой позиции. Всяко преместване наляво е еквивалентно на умножение по 2.

   Пример:

   ```
   a = 5  # (в двоичен вид 101)
   result = a << 1  # (101 << 1 = 1010)
   print(result)  # резултатът е 10
   ```

6. **Дясно изместване** (>>): Премества всички битове на операнда надясно с даден брой позиции. Всяко преместване надясно е еквивалентно на деление на 2 (с изхвърляне на дробната част).

   Пример:

   ```
   a = 5  # (в двоичен вид 101)
   result = a >> 1  # (101 >> 1 = 10)
   print(result)  # резултатът е 2
   ```

Примерна програма с всички побитови операции:

```
a = 5  # 101 в двоичен вид
b = 3  # 011 в двоичен вид

print("a & b =", a & b)  # Побитово AND
print("a | b =", a | b)  # Побитово OR
print("a ^ b =", a ^ b)  # Побитово XOR
print("~a =", ~a)        # Побитово NOT
print("a << 1 =", a << 1)  # Ляво изместване
print("a >> 1 =", a >> 1)  # Дясно изместване
```

Това са основните побитови операции в Python, които са полезни за оптимизация и работа с низко ниво на абстракция.

Логическите операции в Python се използват за комбиниране на булеви стойности или за проверка на условия. Основните логически оператори в Python са:

1. **AND** (`and`): Операторът връща `True`, ако и двата операнда (условия) са `True`. Ако поне едно от тях е `False`, връща `False`.

   Пример:

   ```
   a = True
   b = False
   print(a and b)  # False, защото b е False
   ```

2. **OR** (`or`): Операторът връща `True`, ако поне един от операндите (условията) е `True`. Връща `False` само когато и двата операнда са `False`.

   Пример:

   ```
   a = True
   b = False
   print(a or b)  # True, защото a е True
   ```

3. **NOT** (`not`): Операторът връща обратната булева стойност на операнда. Ако операндът е `True`, връща `False`, и обратното.

   Пример:

   ```
   a = True
   print(not a)  # False, защото a е True
   ```

Логическите оператори и при сравнения

Логическите оператори често се използват за комбиниране на условия. Можем да комбинираме оператори като `and`, `or`, и `not` със сравнения като ==, !=, >, <, >=, <=.

Пример за проверка на няколко условия:

```
x = 5
y = 10
z = 15

# Проверка дали x е по-малко от y и y е по-малко от z
if x < y and y < z:
    print("И двете условия са изпълнени")

# Проверка дали поне едно от условията е вярно
if x > y or y < z:
    print("Поне едно от условията е изпълнено")
```

Истинност на обекти в Python

В Python не само булевите стойности `True` и `False` могат да се използват в логически операции, но и други обекти могат да се третират като `True` или `False`. Ето няколко примера:

- **True**: Нечислени стойности като непразни низове (`"Hello"`), непразни списъци и кортежи, и всички числа, различни от нула.
- **False**: Празни низове (`""`), празни списъци (`[]`), кортежи, и числото `0`.

Пример:

```
a = ""
b = "Hello"

print(bool(a))  # False, защото низът е празен
print(bool(b))  # True, защото низът не е празен
```

Пример за използване на логически операции в реален код:

```
age = 20
has_permission = True

# Проверка дали човек е пълнолетен и има разрешение
if age >= 18 and has_permission:
    print("Може да участва")
else:
    print("Не може да участва")
```

Тези оператори са много полезни за управление на потока в програми и при проверка на сложни условия.

Аритметичните операции в Python се използват за извършване на основни математически изчисления. Те включват събиране, изваждане, умножение, деление и други операции. Ето списък на основните аритметични оператори, които можем да използваме в Python:

Основни аритметични операции:

1. **Събиране** (+): Добавя две числа.

   Пример:

   ```
   a = 5
   b = 3
   result = a + b
   print(result)  # 8
   ```

2. **Изваждане** (-): Изважда едно число от друго.

   Пример:

```
a = 5
b = 3
result = a - b
print(result)  # 2
```

3. **Умножение** (`*`): Умножава две числа.

   Пример:

   ```
   a = 5
   b = 3
   result = a * b
   print(result)  # 15
   ```

4. **Деление** (`/`): Деление на две числа. Винаги връща число с плаваща запетая (float), дори ако делението е точно.

   Пример:

   ```
   a = 5
   b = 2
   result = a / b
   print(result)  # 2.5
   ```

5. **Целочислено деление** (`//`): Деление на две числа, като резултатът е закръглен към най-близкото цяло число надолу (целочислено деление).

   Пример:

   ```
   a = 5
   b = 2
   result = a // b
   print(result)  # 2
   ```

6. **Остатък от деление (Модуло)** (`%`): Връща остатъка при делението на две числа.

   Пример:

   ```
   a = 5
   b = 2
   result = a % b
   print(result)  # 1
   ```

7. **Степенуване** (`**`): Повдига числото на определена степен (експоненция).

   Пример:

   ```
   a = 5
   b = 3
   result = a ** b  # 5 на трета степен
   print(result)  # 125
   ```

Операции с плаваща запетая (float):

Python автоматично работи с дробни числа, когато има нужда. Операции с плаваща запетая връщат числа тип `float`.

Пример:

```
a = 5.5
b = 2
result = a * b
print(result)  # 11.0
```

Смесване на типове:

Python автоматично преобразува типовете, ако е необходимо. Например, ако събираме цяло число (int) и число с плаваща запетая (float), резултатът ще е `float`.

Пример:

```
a = 5
b = 2.5
result = a + b
print(result)  # 7.5
```

Оператори за кратки изрази (Compound Assignment Operators):

Python поддържа съкратени версии на аритметичните операции за по-удобно записване.

1. **`+=`**: Събира и присвоява стойността.

   ```
   a = 5
   a += 3  # Същото като a = a + 3
   print(a)  # 8
   ```

2. **`-=`**: Изважда и присвоява стойността.

   ```
   a = 5
   a -= 3  # Същото като a = a - 3
   print(a)  # 2
   ```

3. **`*=`**: Умножава и присвоява стойността.

   ```
   a = 5
   a *= 3  # Същото като a = a * 3
   print(a)  # 15
   ```

4. **`/=`**: Деление и присвояване (винаги връща float).

   ```
   a = 5
   a /= 2  # Същото като a = a / 2
   print(a)  # 2.5
   ```

5. **`//=`**: Целочислено деление и присвояване.

   ```
   a = 5
   ```

```
a //= 2  # Същото като a = a // 2
print(a)  # 2
```

6. **%=**: Остатък от деление и присвояване.

```
a = 5
a %= 2  # Същото като a = a % 2
print(a)  # 1
```

7. ****=**: Степенуване и присвояване.

```
a = 5
a **= 2  # Същото като a = a ** 2
print(a)  # 25
```

Пример за използване на аритметични операции:

```
a = 10
b = 3

sum_result = a + b      # Събиране
diff_result = a - b     # Изваждане
product_result = a * b  # Умножение
div_result = a / b      # Деление
int_div_result = a // b # Целочислено деление
mod_result = a % b      # Остатък
power_result = a ** b   # Степенуване

print("Събиране:", sum_result)
print("Изваждане:", diff_result)
print("Умножение:", product_result)
print("Деление:", div_result)
print("Целочислено деление:", int_div_result)
print("Остатък:", mod_result)
print("Степенуване:", power_result)
```

Аритметичните операции са основата на всички изчисления в програмирането и са често използвани за различни задачи.