Операторы и выражения

Простые арифметические операции

- Сложение — (a + b)
- Вычитание — (a − b)
- Умножение — (a * b)
- Деление — (a / b)
- Возведение в степень — (a ** b)

В большинстве случаев арифметические операции между разными типами данных завершаются ошибкой. Например, `'Привет' + 2` вызовет ошибку, так как Python не поймёт, что вы хотите сделать.

Приоритет операций

Приоритет операций		
Сначала	Возведение в степень	`5 ** 6`
Потом	Деление, умножение	`3 / 2`, `6 * 7`
В самом конце	Сложение, вычитание	`3 - 2`, `5 + 6`

Изменить приоритет так, как нужно, помогут скобки:

```
2 + 3 * 4 = 2 + 12 = 14
(2 + 3) * 4 = 5 * 4 = 20
```

Функция int()

Функция `int()` помогает привести объект к числу типа int. Это часто используется для преобразования строки в число:

- Пример числа-строки (str) — `'2'`
- Пример числа int — `2`

Обратите внимание: функция `input()` позволяет вводить числа, но любой ввод пользователя эта функция преобразует в строку. Поэтому, если вы введёте число через `input()`, его нужно будет сначала преобразовать в тип int перед выполнением арифметических операций.

```python
x_string = input('Введите число: ')  # Получаем ввод пользователя
x_integer = int(x_string)  # Преобразуем ввод пользователя в число
result = x_integer - 2  # Далее используем это число в расчётах
```

Если вы попытаетесь преобразовать строку, содержащую **буквы и цифры,** к типу int, то получите ошибку:

```python
int('123аб')  # Это вызовет ошибку, так как Python будет пытаться привести ‘а’ и ‘б’ к числам
```

Деление нацело и остаток от деления

1. Деление нацело (целочисленное деление) позволяет получить целую часть от деления одного числа на другое:

- `9 / 2 = 4.5` — обычное деление
- `9 // 2 = 4` — целочисленное деление

2. Операция получения остатка от деления `%` позволяет определить, что остаётся после целочисленного деления одного числа на другое. Эта операция возвращает ту часть числа, которую не удалось разделить нацело.

Пример: `9 % 2 = 1`

Это означает, что при делении 9 на 2:

- целая часть результата деления: `9 / 2 = 4.5`
- целая часть без остатка: `4`
- что осталось: `1` (поскольку `9 = 2 * 4 + 1`)

3. Правило: остаток от деления всегда меньше делителя. Остаток не может быть больше или равен числу, на которое происходит деление.

Пример: `14 % 3 = 2`

Это означает, что при делении 14 на 3:

- результат деления: `14 / 3 ≈ 4.666`
- целая часть без остатка: `4`
- что осталось: `2` (поскольку `14 = 3 * 4 + 2`)

Здесь `2` — это остаток от деления, и это число меньше делителя `3`.

Ещё пример: `15 % 3 = 0`

При делении 15 на 3:

- целая часть результата деления: `15 / 3 = 5`
- целая часть без остатка: `5`
- что осталось: `0` (поскольку `15 = 3 * 5`)

Здесь `0` — это остаток от деления, так как 15 полностью делится на 3 без остатка.

Операция получения остатка от деления важна для определения, какое число остаётся после целочисленного деления. Это особенно полезно при работе с циклами, проверке чётности/нечётности чисел и во многих других задачах, с которыми вы познакомитесь далее.

Сокращённые операторы

Когда нужно увеличить значение переменной, можно использовать сокращённую запись:

```python
a = 1
a = a + 2  # Это можно сократить до:
a += 2
```

Это работает и с другими операторами. Но сокращённые операторы нельзя использовать их с переменными, которые ещё не были созданы:

```python
a = 0
b += 2 # Приведёт к ошибке, так как 'b' ещё не была создана
```

Эти базовые правила помогут вам лучше понять, как работают арифметические операции и операторы в Python.