



# Циклические алгоритмы и массивы в Python. Логические (сравнения) операторы в Python.

Алгоритмы и среда программирования.

# На уроке мы узнаем

- Как работать с **циклическими алгоритмами** и **массивами (списками)** в Python.
- Как использовать циклы **for** и **while** для перебора элементов и выполнения повторяющихся задач.
- Познакомимся с операциями над списками, такими как добавление, удаление и поиск элементов, включая использование операторов **in** и **not in**.
- На практике решим задачи на поиск суммы, максимального значения и среднего арифметического в списке.
- Закрепим знания, выполнив задания на работу с циклами и массивами.

# Циклы в Python

- Что такое циклы и зачем они нужны?
- Основные виды циклов в Python:
  - Цикл `for` – используется для перебора элементов последовательности
  - Цикл `while` – выполняется, пока истинно условие

# Пример цикла (вывести числа от 1 до 5)

## FOR

```
1 for i in range(1, 6):  
2     print(i)
```

## WHILE

```
1 n = 1  
2 while n <= 5:  
3     print(n)  
4     n += 1
```

# Массивы (списки) в Python

- Что такое списки (**list**)?
- Основные операции со списками:
  - Создание списка **numbers = [10, 20, 30, 40, 50]**
  - Добавление **append()**, удаление **remove()** элементов
  - Длина **len()**
  - Сортировка **sort()**
  - Перебор элементов с помощью **for num in numbers:**

- Пример создания списка и перебора элементов:

```
1 numbers = [10, 20, 30, 40, 50]
2
3 for num in numbers:
4     print(num)
```

- Добавление элемента в список:

```
1 numbers.append(60)
2 print(numbers)
3 # [10, 20, 30, 40, 50, 60]
```

# Логические (сравнения) операторы в Python

Оператор	Значение	Пример
<code>==</code>	Равно	<code>5 == 5</code> → True
<code>!=</code>	Не равно	<code>5 != 3</code> → True
<code>&gt;</code>	Больше	<code>7 &gt; 3</code> → True
<code>&lt;</code>	Меньше	<code>2 &lt; 5</code> → True
<code>&gt;=</code>	Больше или равно	<code>10 &gt;= 10</code> → True
<code>&lt;=</code>	Меньше или равно	<code>4 &lt;= 6</code> → True

# Примеры использования операторов сравнения

## Применение в циклах

```
1 a = 10
2 b = 20
3
4 if a < b:
5     print("a меньше b")
6
7 if a != b:
8     print("a не равно b")
9
10 if a >= 10:
11     print("a больше или равно 10")
```

```
1 x = 5
2
3 while x > 0:      # Пока x больше 0,
4                 # выполняем цикл
5     print(x)
6     x -= 1      # Уменьшаем x
```

# Разбор простых примеров

## Найти сумму всех чисел в списке

```
1 numbers = [1, 2, 3, 4, 5]
2 sum_numbers = 0
3
4 for num in numbers:
5     sum_numbers += num
6
7 print("Сумма чисел:", sum_numbers)
```



# Разбор простых примеров (2)

## Найти сумму всех чисел в списке

```
1 numbers = [5, 10, 3, 8, 15]
2 max_num = numbers[0] # Предполагаем, что
3                       # первый элемент –
4                       # максимальный
5 for num in numbers:
6     if num > max_num:
7         max_num = num
8
9 print("Максимальное число:", max_num)
```

# Разбор простых примеров (3)

## Найти максимальный элемент в списке

```
1 numbers = [4, 8, 12, 16, 20]
2 sum_numbers = 0
3
4 for num in numbers:
5     sum_numbers += num
6
7 average = sum_numbers / len(numbers)
8 print("Среднее значение:", average)
```

# Практическое задание

1. Вывести все четные числа из списка (`num % 2 == 0`).

Начальные данные:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

2. Посчитать, сколько раз число 3 встречается в списке.

Начальные данные:

```
numbers = [3, 5, 3, 7, 3, 9, 3]
```

# Задача 1: Найти все четные числа в списке (решение)

```
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 for num in numbers:
4     if num % 2 == 0:
5         print(num)
```

## Задача 2: Посчитать количество троек в списке (решение)

```
1 numbers = [3, 5, 3, 7, 3, 9, 3]
2 count = 0
3
4 for num in numbers:
5     if num == 3:
6         count += 1
7
8 print("Число 3 встречается", count, "раз(a) ")
```

# Проверка элемента в массиве: **not in**

- В Python можно проверить, содержится ли элемент в списке, с помощью оператора **in**. А если нужно убедиться, что элемента **нет в списке**, используется **not in**.
- **in** – проверяет, есть ли элемент в списке.
- **not in** – проверяет, отсутствует ли элемент в списке.

## Пример использования **not in**:

```
1 numbers = [1, 2, 3, 4, 5]
2
3 if 3 in numbers:
4     print("Число 3 есть в списке")
5
6 if 10 not in numbers:
7     print("Числа 10 нет в списке")
```

# Подведение итогов

- Обсуждение **различных способов работы с массивами и циклами**
- Разбор **типичных ошибок**
- Вопросы?