Введение в программирование.

Цикл for

Циклы в Python

Одним из преимуществ компьютеров является их способность повторять одни и те же действия большое количество раз.

В Python существует две основные разновидности цикла:

- циклы, повторяющиеся определенное количество раз (**for**);
- циклы, повторяющиеся пока верно некоторое условие (**while**).

Цикл for

Распечатать слово **Привет** один раз:

```
print('Привет')
```

Цикл for

Распечатать слово **Привет** один раз:

```
print('Привет')
```

Распечатать слово **Привет** пять раз:

```
print('Привет')
print('Привет')
print('Привет')
print('Привет')
print('Привет')
```

Цикл for

Распечатать слово **Привет** один раз:

```
print('Привет')
```

Распечатать слово **Привет** пять раз:

```
print('Привет')
print('Привет')
print('Привет')
print('Привет')
print('Привет')
```

Распечатать слово **Привет** десять раз?

Цикл for

Код, который распечатает 10 раз слово **Привет**:

```
for i in range(10):
    print('Привет')
```

Цикл for

Код, который распечатает 10 раз слово **Привет**:

```
for i in range(10):
    print('Привет')
```

Структура цикла **for** в Python выглядит так:

```
for название переменной in range(количество повторений):
    блок кода
```

Двоеточие **(:)** в конце строки сообщает Python, что дальше находится блок команд, называемый **телом цикла**.

Цикл for VS условный оператор if

Структура цикла **for**:

```
for название переменной in range(количество повторений):
    блок кода
```

Структура условного оператора **if**:

```
if условие:
    блок кода
```

Цикл for

С помощью цикла **for**, можно считывать и обрабатывать сколько угодно чисел:

```
for i in range(5):
    num = int(input())
    print('Квадрат вашего числа равен:', num * num)
print('Цикл завершен')
```

Такая программа считывает 5 чисел и выводит их квадраты вместе с поясняющей надписью.

Цикл for

С помощью цикла **for**, можно считывать и обрабатывать сколько угодно чисел:

```
for i in range(5):
    num = int(input())
    print('Квадрат вашего числа равен:', num * num)
print('Цикл завершен')
```

отступ

тело цикла

Четвертая строка не содержит отступа, поэтому не является частью цикла

Цикл for

Предыдущий код, равнозначен коду:

```
num = int(input())
print('Квадрат вашего числа равен:', num*num)
num = int(input())
print('Квадрат вашего числа равен:', num*num)
num = int(input())
print('Квадрат вашего числа равен:', num*num)
num = int(input())
print('Квадрат вашего числа равен:', num*num)
num = int(input())
print('Квадрат вашего числа равен:', num*num)
print('Цикл завершен')
```

1 число

2 число

3 число

4 число

5 число

Цикл for

Что покажет приведенный ниже фрагмент кода?

```
print('A')
print('B')
for i in range(5):
    print('C')
    print('D')
print('E')
```

Цикл for

Что покажет приведенный ниже фрагмент кода?

```
print('A')
print('B')
for i in range(5):
    print('C')
    print('D')
print('E')
```

```
A
B
C
D
C
D
C
D
C
D
C
D
E
```

Тело цикла состоит из двух строк: четвертой и пятой и именно они будут повторяться!

Цикл for

Что покажет приведенный ниже фрагмент кода?

```
print('A')
print('B')
for i in range(5):
    print('C')
for i in range(5):
    print('D')
print('E')
```

Цикл for

Что покажет приведенный ниже фрагмент кода?

```
print('A')
print('B')
for i in range(5):
    print('C')
for i in range(5):
    print('D')
print('E')
```

```
A
B
C
C
C
C
C
D
D
D
D
D
E
```

В программе может быть сколько угодно циклов!

Переменная цикла

Цикл for: переменная цикла

Структура цикла **for** в Python выглядит так:

```
for название переменной in range(количество повторений):
    блок кода
```

Не совсем понятно, для чего нужна и как работает **переменная цикла**.

Цикл for: переменная цикла

Рассмотрим следующий код:

```
for i in range(10):
    print(i)
```

Что покажет приведенный выше фрагмент кода?

Цикл for: переменная цикла

Рассмотрим следующий код:

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Когда цикл впервые начинает работу Python устанавливает значение переменной цикла `i = 0`. Каждый раз когда мы повторяем тело цикла Python **увеличивает значение переменной на 1**

Начальное значение `i = 0`, а не 1

Цикл for: переменная цикла

Поскольку переменная цикла **i** увеличивается на 1 каждый раз, то ее можно использовать для отслеживания номера итерации, на которой мы находимся в циклическом процессе.

```
for i in range(10):
    print(i + 1, '-- Привет')
```

```
1 -- Привет
2 -- Привет
3 -- Привет
4 -- Привет
5 -- Привет
6 -- Привет
7 -- Привет
8 -- Привет
9 -- Привет
10 -- Привет
```

Обратите внимание, за счет выражения **i + 1**, мы начинаем вывод с 1, а не с 0.

Имена переменных цикла

Для переменных цикла обычно используют буквы **`i, j, k`**.

```
for i in range(5):
    print(i)
```

```
for num in range(5):
    print(num)
```

Результат выполнения обеих программ:

```
0
1
2
3
4
```

Имена переменных цикла

Переменная цикла **i** не используется в теле цикла:

```
for i in range(5):
    print('Python – это здорово!')
```

Имена переменных цикла

Мы можем использовать символ нижнего подчеркивания _:

```
for _ in range(5):
    print('Python - это здорово!')
```

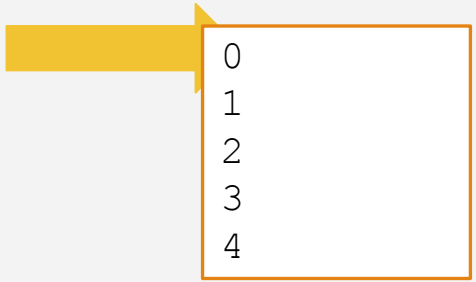
Результат выполнения такой программы:

```
Python - это здорово!
Python - это здорово!
Python - это здорово!
Python - это здорово!
Python - это здорово!
```

Примечания

Следует помнить, что правая граница цикла в Python всегда невключительна:

```
for i in range(5):
    print(i)
```



```
0
1
2
3
4
```

Если требуется распечатать числа от 1 до 5, то:

```
for i in range(5):
    print(i + 1)
```

Функция range

Функция range с одним параметром


```
for i in range(10):
    сказать('Привет', i)
```

Переменная **i** принимает последовательно значения: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Функция **range(n)** генерирует последовательность чисел от **0** до **n-1**, а цикл **for** последовательно перебирает эту последовательность.

Функция range с двумя параметрами


```
for i in range(3, 10):
    print('Привет', i)
```

Переменная **i** принимает последовательно значения: 3, 4, 5, 6, 7, 8, 9.

Функция **range(n, m)** генерирует последовательность чисел от **n** до **m-1** с шагом 1, а цикл **for** последовательно перебирает эту последовательность.

Функция range с тремя параметрами


```
for i in range(3, 10, 2):
    присесть(i раз)
```

Переменная **i** принимает последовательно значения: 3, 5, 7, 9.

Функция **range(n, m, k)** генерирует последовательность чисел от **n** до **m-1** с шагом **k**, а цикл **for** последовательно перебирает эту последовательность.

Отрицательный шаг


```
for i in range(10, 3, -1):
    сказать('Пока', i)
```

Переменная `i` принимает последовательно значения: 10, 9, 8, 7, 6, 5, 4.

В случае отрицательного шага, мы должны гарантировать, что старт генерации (первый параметр) больше чем конец генерации (второй параметр).

Подсчет количества

Подсчет количества

Очень часто нужно, чтобы наши программы **считали** как часто что-либо произошло.

Ключом к использованию **подсчета количества** является использование переменной счетчика.

Подсчет количества

Программа считывает 10 чисел и определяет сколько из них больше 10.

```
counter = 0
for i in range(10):
    num = int(input())
    if num > 10:
        counter = counter + 1
print(counter)
```

начальное значение счетчика

Каждый раз когда мы считываем число большее 10, мы добавляем **1** к нашему текущему значению переменной counter

Подсчет количества

Подсчет количества – очень частый сценарий.
Он состоит из двух шагов:

1. объявление переменной счетчика и инициализация начального значения: `counter = 0`;
2. увеличение переменной счетчика на 1: `counter = counter + 1`.

Подсчет количества

Часто при написании программ требуется использовать несколько счетчиков.

```
counter1 = 0
counter2 = 0
for i in range(10):
    num = int(input())
    if num > 10:
        counter1 = counter1 + 1
    if num == 0:
        counter2 = counter2 + 1
print(counter1, counter2)
```

Что выводит данная программа?

Подсчет количества

Часто при написании программ требуется использовать несколько счетчиков.

```
counter1 = 0
counter2 = 0
for i in range(10):
    num = int(input())
    if num > 10:
        counter1 = counter1 + 1
    if num == 0:
        counter2 = counter2 + 1
print(counter1, counter2)
```

количество чисел больших 10

количество нулевых чисел

Что выводит данная программа?

Подсчет суммы и произведения

Подсчет суммы

Не менее частой задачей наравне с подсчетом количества, является **подсчет суммы**.

Ключом к использованию подсчета суммы является использование **переменной сумматора**

Подсчет суммы

Программа считывает 10 чисел и определяет **сумму** тех из них, которые больше 10.

```
total = 0
for i in range(10):
    num = int(input())
    if num > 10:
        total = total + num
print(total)
```

начальное значение сумматора

Каждый раз когда мы считываем число большее 10, мы добавляем **его** к нашему текущему значению переменной **`total`**

Подсчет суммы

Подсчет суммы – очень частый сценарий.
Он состоит из двух шагов:

1. Объявление переменной сумматора и инициализация начального значения: `total = 0`;
2. увеличение переменной сумматора на нужное число: `total = total + num`.

Подсчет суммы

Что делает следующий программный код?

```
total = 0
for i in range(1, 101):
    total = total + i
print(total)
```

Подсчет суммы

Что делает следующий программный код?

```
total = 0
for i in range(1, 101):
    total = total + i
print(total)
```

```
5050
```

сумма чисел от 1 до 100

Подсчет суммы

Что делает следующий программный код?

```
total = 0
for i in range(10):
    num = int(input())
    total = total + num
average = total / 10
print(average)
```

Подсчет суммы

Что делает следующий программный код?

```
total = 0
for i in range(10):
    num = int(input())
    total = total + num
average = total / 10
print(average)
```

считывает 10 целых чисел и выводит их среднее значение

Подсчет произведения

При подсчете произведения, начальное значение переменной **мультипликатора** мы устанавливаем в 1, в отличии от сумматора, где оно равно 0.

```
mult = 1
for i in range(10):
    num = int(input())
    if num > 10:
        mult = mult * num
print(mult)
```

начальное значение мультипликатора

Каждый раз когда мы считываем число большее 10, мы умножаем текущее значение переменной **`mult`** на него

Поиск максимума и минимума

Поиск максимума и минимума

Поиск наибольшего (наименьшего) значения в некоторой последовательности чисел, также частая задача в программировании.

Ключом к поиску наибольшего (наименьшего) значения является использование переменной, в которой хранится **текущее** наибольшее (наименьшее) значение.

Поиск максимума

Программа считывает 10 положительных чисел и находит среди них наибольшее значение.

```
largest = -1
for i in range(10):
    num = int(input())
    if num > largest:
        largest = num
print(largest)
```

начальное значение принимаем за -1

каждый раз когда мы считываем число большее **largest**, мы переприсваиваем значение переменной **largest**.

Поиск максимума

В качестве начального значения переменной принимается первый элемент последовательности:

```
largest = int(input())
for i in range(9):
    num = int(input())
    if num > largest:
        largest = num
print(largest)
```

в качестве начального значения принимаем первый элемент

каждый раз когда мы считываем число большее **largest**, мы переприсваиваем значение переменной **largest**.

Поиск минимума

Для нахождения наименьшего значения последовательности следует поменять знак неравенства (>) на противоположный (<).

```
smallest = int(input())
for i in range(9):
    num = int(input())
    if num < smallest:
        smallest = num
print(smallest)
```

в качестве начального значения принимаем первый элемент

каждый раз когда мы считываем число меньшее **smallest**, мы переприсваиваем значение переменной **smallest**.

Расширенные операторы присваивания

Расширенные операторы присваивания

Часто программы имеют инструкции присваивания, в которых переменная на левой стороне от оператора = также появляется на правой от него стороне.

```
counter = counter + 1
total = total + number
balance = balance - withdrawal
```

Расширенные операторы присваивания

Часто программы имеют инструкции присваивания, в которых переменная на левой стороне от оператора = также появляется на правой от него стороне.

```
counter = counter + 1
total = total + number
balance = balance - withdrawal
```

```
counter += 1
total += number
balance -= withdrawal
```

расширенные операторы присваивания

Расширенные операторы присваивания

Оператор	Пример использования	Эквивалент
`+=`	`x += 5`	`x = x + 5`
`-=`	`x -= 2`	`x = x - 2`
`*=`	`x *= 10`	`x = x * 10`
`/=`	`x /= 4`	`x = x / 4`
`//=`	`x //= 4`	`x = x // 4`
`%=`	`x %= 4`	`x = x % 4`

Цикл while

Цикл while

Структура цикла **`while`** в Python выглядит так:

```
while условие:
    блок кода
```

Двоеточие **(:)** в конце строки сообщает Python, что дальше находится блок команд, называемый **телом** цикла.

Цикл while

Код, который распечатает 10 раз слово **Привет**:

```
i = 0
while i < 10:
    print('Привет')
    i += 1
```

Цикл while

Код, который распечатает 10 раз слово **Привет**:

```
i = 0
while i < 10:
    print('Привет')
    i += 1
```

Такой код можно легко заменить циклом **for**, поскольку мы заранее знаем сколько раз нужно выполнить тело цикла.

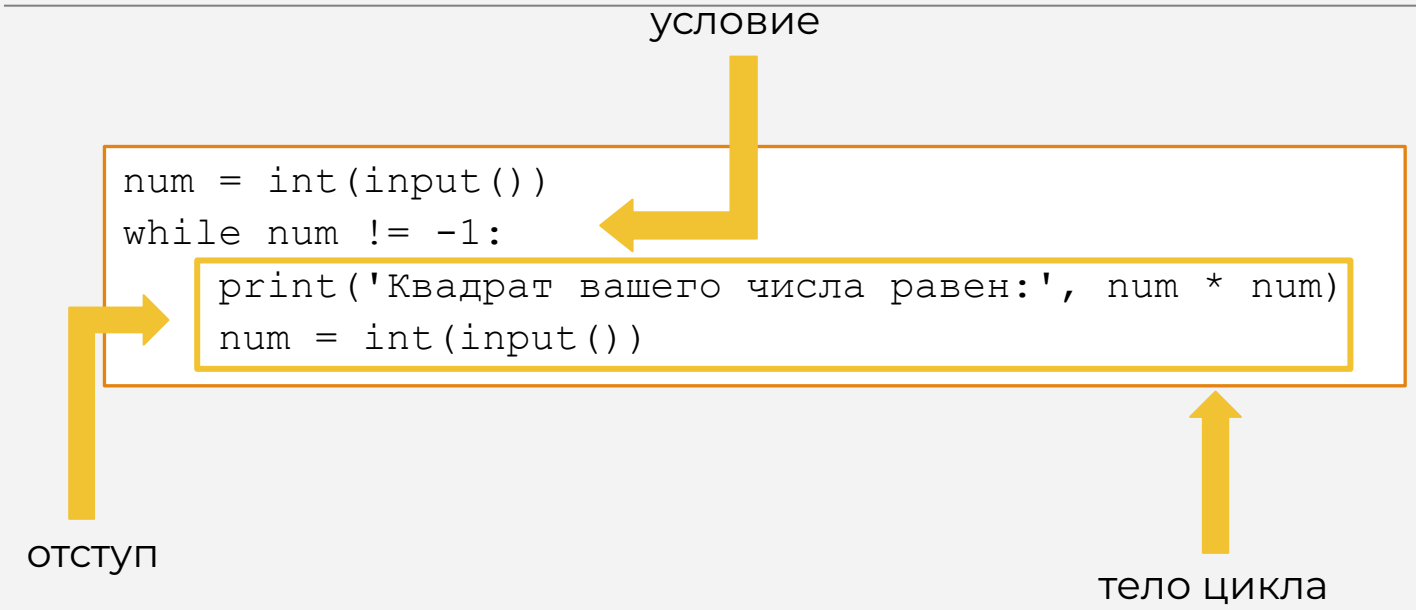
Цикл while

С помощью цикла **while** можно считывать и обрабатывать сколько угодно чисел:

```
num = int(input())
while num != -1:
    print('Квадрат вашего числа равен:', num * num)
    num = int(input())
```

Такая программа считывает числа пока не будет введено число -1 и выводит их квадраты вместе с поясняющей надписью.

Цикл while


условие
num = int(input())
while num != -1:
print('Квадрат вашего числа равен:', num * num)
num = int(input())
отступ
тело цикла

Цикл while


```
num = int(input())
while num != -1:
    print('Квадрат вашего числа равен:', num * num)
    num = int(input())
```

Важным являются два момента:

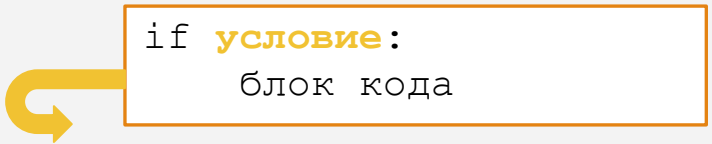
- правильная инициализация переменной **num**;
- изменение переменной **num** внутри цикла **while**.

Цикл while VS условный оператор if

Структура цикла **while**:

```
while условие:
    блок кода
```

Структура условного оператора **if**:



```
if условие:
    блок кода
```

Цикл while VS цикл for

Мы всегда можем заменить цикл **for** с помощью цикла **while**

```
for i in range(101):
    print(i)
```

```
i = 0
while i < 101:
    print(i)
    i += 1
```

```
for i in range(0, 100, 3):
    print(i)
```

```
i = 0
while i < 100:
    print(i)
    i += 3
```

Не всегда удается заменить цикл **while** с помощью цикла **for**

Считывание данных до стоп значения

При решении задач на цикл **while**, мы считываем данные, до тех пор пока пользователь не введет некоторое значение, которое называют **стоп значением**:

```
text = input()
total = 0
while text != 'stop':
    num = int(text)
    total += num
    text = input()
print('Сумма чисел равна', total)
```

Такая программа считывает числа и находит их сумму, до тех пор пока пользователь не введёт слово «stop».

Бесконечный цикл

Цикл **`while`** должен содержать возможность завершиться.

Если цикл не имеет возможности завершиться, то он называется **бесконечным циклом**.

```
i = 0
total = 0
while i < 10:
    total += i
```

переменная **`i`** не меняется в теле цикла и условие **`i < 10`** истинно всегда

Бесконечный цикл продолжает повторяться до тех пор, пока программа не будет прервана

Примечания

Цикл **`while`** получил свое название из-за характера своей работы: он выполняет некую задачу до тех пор, **пока** условие является истинным

Цикл **`while`** называют **циклом с предусловием**, поскольку выполнению тела цикла предшествует проверка условия

Цикл **`while`** может не выполниться ни одного раза:

```
i = -1
while i > 0:
    print('Hello world!')
```

Обработка цифр числа

Обработка цифр числа

Пусть дано натуральное число `n`. Тогда:

- результатом операции `n % 10` – является последняя цифра числа;
- результатом операции `n // 10` – является число с удаленной последней цифрой.

Обработка цифр числа

Программа, считывает число и обрабатывает его цифры:

```
n = int(input())
while n != 0:  # пока в числе есть цифры
    last_digit = n % 10  # получить последнюю цифру
    # код обработки последней цифры
    n = n // 10  # удалить последнюю цифру из числа
```

В качестве кода обработки может быть все, что угодно:

- вывод цифр,
- нахождение суммы, произведения,
- нахождение наибольшей или наименьшей цифры и т.д.