



Больше информации по Python тут



Наталья Кайда 22 декабря 2022



2



0



0



0



## Самоучитель по Python для начинающих. Часть 10: Условный цикл while

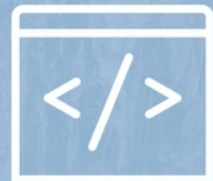
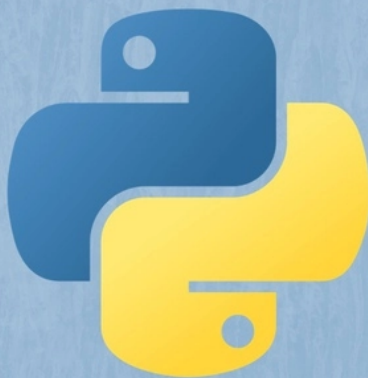
Разберем особенности условного цикла: в каких случаях необходимо использовать while, как управлять работой бесконечного цикла, и когда while лучше заменить циклом for. В конце статьи – практические задания и пример простейшей игры.



Обсудить



0



10

10

[← Часть 9](#)

Наш сайт использует файлы cookie для вашего максимального удобства.  
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

## Цикл while в Python

В прошлой статье мы изучали цикл **for** – он используется в тех случаях, когда **заранее** известно количество итераций, совершаемых в цикле. Число исполнений цикла **for** определяется функцией `range()` или размером коллекции. Если диапазон значений не известен заранее, необходимо использовать другой цикл – **while**: он выполняется, пока не наступит определенное **событие** (или не выполнится необходимое **условие**). По этой причине цикл **while** называют **условным**. Вот пример простейшего цикла **while** – он выполняется, пока пользователь не введет `0`:

```
n = int(input())
while n != 0:
    print(n + 5)
    n = int(input())
```

Цикл **while** также часто называют **бесконечным**, поскольку он может выполняться до тех пор, пока пользователь не остановит его нажатием определенной клавиши. Бесконечные циклы можно создавать **намеренно** – для выполнения фонового скрипта, игры, прикладной программы. Но иногда цикл **while** может стать бесконечным **из-за ошибки**. Например, если в приведенном выше коде не указать ввод новой переменной `n = int(input())` в теле цикла, **while** будет бесконечно выводить одно и то же значение, пока пользователь не остановит выполнение программы нажатием **Ctrl + C**.

## Управление бесконечным циклом while в Питоне

Самый простой способ управления бесконечным циклом – использование оператора `break`. В приведенном ниже коде список `lst` генерируется случайным образом, и до начала цикла его длина неизвестна. Однако выполнение цикла можно оставить, как только список опустеет в результате многократного выполнения операции `pop()`:

```
import random
lst = [i for i in range(random.randint(5, 500))]
while True:
    if not lst:
        break
    print(lst.pop())
```

```
IndexError: pop from empty list
```

Оператор `break` также помогает сократить количество итераций и прекратить выполнение программы, как только нужное решение найдено. Например, таким образом можно найти наименьший делитель числа **n**, отличный от **1**:

```
n = int(input())
i = 2
while True:
    if n % i == 0:
        break
    i += 1
print(i)
```

Помимо `break`, управлять бесконечным циклом можно с помощью **флагов** (сигнальных меток). В приведенном ниже примере программа бесконечно запрашивает у пользователя ввод любого слова, пока пользователь не введет `exit`. Это событие меняет статус цикла на `False`, и работа программы завершается:

```
text = 'Введите любое слово: '
text += '\nИли введите exit для выхода: '

active = True
while active:
    message = input(text)
    if message == 'exit':
        active = False
    else:
        print(message)
```

## Пропуск итераций в цикле `while`

Оператор `continue` можно использовать для пропуска операций, если элементы **не соответствуют** заданным критериям. Этот код работает, пока не будет сформирован список из 5 элементов – при этом в список не включаются числа в диапазоне между 90 и 120, а также число 50:

```

num = int(input())
if num == 50 or 90 <= num <= 120:
    continue
sp.append(num)
print(sp)

```

Если пользователь введет набор цифр 45 50 121 119 95 105 3 4 7 , в список будут добавлены только числа, соответствующие критериям:

```
[45, 121, 3, 4, 7]
```

## Особенности цикла while

**1.** В цикле **while** можно использовать опциональный параметр `else` . В этом примере процедура `pop()` выполняется, пока список не опустеет, после чего выводится сообщение `Список пуст` :

```

import random
lst = [i for i in range(random.randint(5, 500))]
while len(lst) > 1:
    print(lst.pop())
else:
    print('Список пуст')

```

**2.** В цикле **while** можно использовать любое количество условий и условных операторов `and` , `or` , и `not` :

```

n = int(input())
while True:
    if n == 0:
        break
    elif n > 50 or n <= -50:
        break
    elif n % 2 == 0:
        break
    print(n / 5)
    n = int(input())

```

```
i = 2
while(i < 100):
    j = 2
    while j <= i / j:
        if not i % j:
            break
        j = j + 1
    if j > i / j:
        print(f'{i} - простое число')
    i = i + 1
```

4. В качестве вложенных циклов **while** могут включать в себя циклы **for**. Этот код, к примеру, будет бесконечно печатать цифры в диапазоне от 0 до 5:

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

5. Любой цикл **for** можно заменить циклом **while**, но обратное возможно только в том случае, когда количество итераций можно определить до начала цикла. К примеру, эти циклы **while** и **for** равнозначны – оба печатают цифры от 0 до 9:

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

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

А этот цикл **while** заменить циклом **for** невозможно – программа будет бесконечно возводить в квадрат все введенные пользователем числа, пока не получит 0:

```
n = int(input())
while True:
    if n == 0:
```

# Практика

## Задание 1

Напишите программу, которая принимает на вход **целые** числа и вычисляет их сумму, пока пользователь не введет **0**.

**Пример ввода:**

4  
5  
6  
0

**Вывод:**

15

**Решение:**

```
summa = 0
while True:
    n = int(input())
    summa += n
    if n == 0:
        break
print(summa)
```

## Задание 2

Напишите программу, которая получает от пользователя число **n > 100**, и вычисляет (**без** использования методов строк) произведение цифр, из которых **n** состоит.

**Пример ввода:**

335

**Решение:**

```
n = int(input())
prod = 1

while n:
    prod *= n % 10
    n //= 10
print(prod)
```

**Задание 3**

Напишите программу, которая получает на вход два числа **a** и **b**, и находит наименьшее число **c**, которое без остатка делится на **a** и **b**.

**Пример ввода:**

```
7
12
```

**Вывод:****Решение:**

```
a, b = int(input()), int(input())
c = a
while c % b:
    c += a
print(c)
```

**Задание 4**

### Пример ввода:

о  
бойся  
Бармаглота  
сын  
он  
так  
свиреп  
и  
дик  
а  
в гуще  
рымит

### Вывод:

бойся Бармаглота сын так свиреп дик в гуще рымит

### Решение:

```
st = ''
while len(st) < 50:
    word = input()
    if word[0] in 'аиеёоуыэя':
        continue
    st += ' ' + word
print(st)
```

## Задание 5

Напишите программу для конвертации числа из десятичной системы в двоичную **без** использования функции `bin()` .

### Пример ввода:

25



11001

### Решение:

```
n = int(input())
result = ''
while n > 0:
    result = str(n % 2) + result
    n = n // 2
print(result)
```

## Задание 6

Напишите программу, которая получает на вход число и **без** использования строковых методов переставляет цифры в **обратном порядке**.

### Пример ввода:

176435

### Вывод:

534671

### Решение:

```
n = int(input())
rev = 0
while n != 0:
    r = n % 10
    rev = rev * 10 + r
    n = n // 10
print(rev)
```

## Задание 7

Наш сайт использует файлы cookie для вашего максимального удобства.  
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

### Пример ввода:

7

### Вывод:

5040

### Решение:

```
n = int(input())
fact = 1

while n > 1:
    fact *= n
    n -= 1
print(fact)
```

## Задание 8

Напишите программу, которая получает от пользователя число **n** и определяет, является ли оно простым, или у него есть делители, кроме 1 и самого себя.

### Пример ввода:

60

### Вывод:

60 делится на 2  
60 делится на 3  
60 делится на 4  
60 делится на 5  
60 делится на 6  
60 делится на 10  
60 делится на 12

60 делится на 30

Таким образом, 60 не является простым числом

### Решение:

```
n = int(input())
flag = False
i = 2
while i < n:
    if n % i == 0:
        flag = True
        print(f'{n} делится на {i}')
    i += 1
if flag:
    print(f'Таким образом, {n} не является простым числом')
else:
    print(f'{n} - простое число')
```

### Задание 9

Напишите программу, использующую вложенный цикл **while** для вывода треугольника размером **n** х **n** х **n**, состоящего из символов \* .

### Пример ввода:

6

### Вывод:

```
*
**
***
****
*****
*****
```

### Решение:

Наш сайт использует файлы cookie для вашего максимального удобства.  
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

```
n = int(input())
i, j = 0, 0
while i < n:
    while j <= i:
        print('*', end='')
        j += 1
    j = 0
    i += 1
    print('')
```

## Задание 10

Напишите программу для запоминания английских названий месяцев:

1. Русские названия месяцев выводятся в случайном порядке с помощью метода `random.shuffle()`.
2. Пользователь получает три попытки для написания правильного названия на английском.
3. После трех неверных попыток программа переходит к другому слову.

### Пример ввода:

Месяц март по-английски называется: march  
Месяц январь по-английски называется: January  
Месяц август по-английски называется: august  
Месяц май по-английски называется: may  
Месяц апрель по-английски называется: aprile  
Неверно! Осталось попыток: 2  
Месяц апрель по-английски называется: aprill  
Неверно! Осталось попыток: 1  
Месяц апрель по-английски называется: april  
Неверно! Осталось попыток: 0  
Попытки исчерпаны!  
Месяц июль по-английски называется: july  
Месяц сентябрь по-английски называется: september  
Месяц июнь по-английски называется: june  
Месяц октябрь по-английски называется: october

### Вывод:

Наш сайт использует файлы cookie для вашего максимального удобства.  
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

Конец игры

Количество правильных ответов: 11

Число ошибок: 3

## Решение:

```
import random
correct, wrong, attempts = 0, 0, 3
months = {'январь': 'January', 'февраль': 'February', 'март': 'March',
          'апрель': 'April', 'май': 'May', 'июнь': 'June',
          'июль': 'July', 'август': 'August', 'сентябрь': 'September',
          'октябрь': 'October', 'ноябрь': 'November', 'декабрь': 'December'}
rand_keys = list(months.keys())
random.shuffle(rand_keys)
for key in rand_keys:
    counter = 0
    while counter < attempts:
        spelling = input(f'Месяц {key} по-английски называется: ')
        if spelling.title() == months[key]:
            correct += 1
            break
```

## Подведем итоги

Цикл **while** используют в случаях, когда число итераций невозможно оценить заранее. Во всех остальных случаях лучше применять цикл **for**. Чтобы цикл **while** случайно не превратился в бесконечный, стоит продумать все события и условия, которые должны приводить к своевременному прерыванию программы.

В следующей статье приступим к изучению функций.

\*\*\*

## Содержание самоучителя

1. Особенности, сферы применения, установка, онлайн IDE
2. Все, что нужно для изучения Python с нуля – книги, сайты, каналы и курсы
3. Типы данных: преобразование и базовые операции

- 6. Методы работы со словарями и генераторами словарей
- 7. Методы работы с кортежами
- 8. Методы работы со множествами
- 9. Особенности цикла for
- 10. Условный цикл while
- 11. Функции с позиционными и именованными аргументами

♥ 2    💬 Обсудить    📌 0    🔥 0    💧 0    💩 0

Python



## МЕРОПРИЯТИЯ

### Миграция на отечественный почтовый сервер TEGU

📅 02 февраля    [Онлайн](#)    [Бесплатно](#)

### Цифровое импортозамещение

📅 24 января    [Онлайн](#)    [Бесплатно](#)

### Delivery Meetup SPB #2

📅 26 января    [Онлайн](#)    [Бесплатно](#)

+ Показать еще

## Комментарии

Оставьте свой комментарий (можно использовать markdown)

Отправить

Согласен

## Java Разработчик

Ижевск, до 120000 RUB

## Android разработчик

от 250000 RUB до 300000 RUB

## Технический директор

Москва, по итогам собеседования

+ Показать еще

Опубликовать вакансию

ЛУЧШИЕ СТАТЬИ ПО ТЕМЕ

## ООП на Python: концепции, принципы и примеры реализации

Программирование на Python допускает различные методологии, но в его основе лежит объектный подход, поэтому работать в стиле ООП на Python очень просто.

## 3 самых важных сферы применения Python: возможности языка

Существует множество областей применения Python, но в некоторых он особенно хорош. Разбираемся, что же можно делать на этом ЯП.

## Программирование на Python: от новичка до профессионала

Пошаговая инструкция для всех, кто хочет изучить программирование на Python (или программирование вообще), но не знает, куда сделать первый шаг.

О проекте

Реклама

Пользовательское соглашение

Публичная оферта

Политика конфиденциальности

Наш сайт использует файлы cookie для вашего максимального удобства.  
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

Push уведомления



Темная тема



FB

IG

© 2023, Proglib. При копировании материала ссылка на источник обязательна.

Наш сайт использует файлы cookie для вашего максимального удобства.  
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен