Казанский (Приволжский) Федеральный Университет
Институт Физики

# Лекция №3
# «Строки. Цикл **while**»

И.И. Файрушин

E-mail: *fairushin_ilnaz@mail.ru*

Казань, 2022

# Строки

Строка считывается со стандартного ввода функцией **input()**. Напомним, что для двух строк определена операция сложения (конкатенации), также определена операция умножения строки на число.

Строка состоит из последовательности символов. Узнать количество символов (длину строки) можно при помощи функции **len**.

Любой другой объект в Питоне можно перевести к строке, которая ему соответствует. Для этого нужно вызвать функцию **str()**, передав ей в качестве параметра объект, переводимый в строку.

```
1  s = input()
2  print(len(s))
3  t = input()
4  number = int(t)
5  u = str(number)
6  print(s * 3)
7  print(s + ' ' + u)
8
```

# Строки

Срез (**slice**) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

Есть три формы срезов. Самая простая форма среза: взятие одного символа строки, а именно, **S[i]** — это срез, состоящий из одного символа, который имеет номер **i**. При этом считается, что нумерация начинается с числа **0**. То есть если **S = 'Hello'**, то **S[0] == 'H', S[1] == 'e', S[2] == 'l', S[3] == 'l', S[4] == 'o'.**

Заметим, что в Питоне нет отдельного типа для символов строки. Каждый объект, который получается в результате среза **S[i]** — это тоже строка типа **str**.

Номера символов в строке (а также в других структурах данных: списках, кортежах) называются индексом. Если указать отрицательное значение индекса, то номер будет отсчитываться с конца, начиная с номера **-1**. То есть **S[-1] == 'o', S[-2] == 'l', S[-3] == 'l', S[-4] == 'e', S[-5] == 'H'.**

# Строки

Если же номер символа в срезе строки **S** больше либо равен **len(S)**, или меньше, чем **-len(S)**, то при обращении к этому символу строки произойдет ошибка IndexError: string index out of range.

Срез с двумя параметрами: **S[a:b]** возвращает подстроку из **b - a** символов, начиная с символа с индексом **a**, то есть до символа с индексом **b**, не включая его. Например, **S[1:4] == 'ell'**, то же самое получится если написать **S[-4:-1]**. Можно использовать как положительные, так и отрицательные индексы в одном срезе, например, **S[1:-1]** — это строка без первого и последнего символа (срез начинается с символа с индексом **1** и заканчиватеся индексом **-1**, не включая его). При использовании такой формы среза ошибки IndexError никогда не возникает. Например, срез **S[1:5]** вернет строку **'ello'**, таким же будет результат, если сделать второй индекс очень большим, например, **S[1:100]** (если в строке не более 100 символов).

# Строки

Если опустить второй параметр (но поставить двоеточие), то срез берется до конца строки. Например, чтобы удалить из строки первый символ (его индекс равен **0**), можно взять срез **S[1:]**. Аналогично если опустить первый параметр, то можно взять срез от начала строки. То есть удалить из строки последний символ можно при помощи среза **S[:-1]**. Срез **S[:]** совпадает с самой строкой **S**.

Если задать срез с тремя параметрами **S[a:b:d]**, то третий параметр задает шаг, как в случае с функцией **range**, то есть будут взяты символы с индексами **a**, **a + d**, **a + 2 * d** и т. д. При задании значения третьего параметра, равному **2**, в срез попадет кажый второй символ, а если взять значение среза, равное **-1**, то символы будут идти в обратном порядке. Например, можно перевернуть строку срезом **S[::-1].**

# Строки

```
1  s = 'abcdefg'
2  print(s[1])
3  print(s[-1])
4  print(s[1:3])
5  print(s[1:-1])
6  print(s[:3])
7  print(s[2:])
8  print(s[:-1])
9  print(s[::2])
10 print(s[1::2])
11 print(s[::-1])
12
```

Обратите внимание на то, как похож третий параметр среза на третий параметр функции **range()**:

```
1  s = 'abcdefghijklm'
2  print(s[0:10:2])
3  for i in range(0, 10, 2):
4      print(i, s[i])
5
```

# Строки

Методы **find** и **rfind**

Метод – это функция, применяемая к объекту, в данном случае – к строке. Метод вызывается в виде **Имя_объекта.Имя_метода (параметры).** Например, **S.find("e")** – это применение к строке **S** метода **find** с одним параметром **"e"**.

Метод **find** находит в данной строке (к которой применяется метод) данную подстроку (которая передается в качестве параметра). Функция возвращает индекс первого вхождения искомой подстроки. Если же подстрока не найдена, то метод возвращает значение **-1**.

```
1  S = 'Hello'
2  print(S.find('e'))
3  # вернёт 1
4  print(S.find('ll'))
5  # вернёт 2
6  print(S.find('L'))
7  # вернёт -1
8
```

# Строки

Аналогично, метод **rfind** возвращает индекс последнего вхождения данной строки (“поиск справа”).

```
1  S = 'Hello'
2  print(S.find('l'))
3  # вернёт 2
4  print(S.rfind('l'))
5  # вернёт 3
6
```

Если вызвать метод **find** с тремя параметрами **S.find(T, a, b)**, то поиск будет осуществляться в срезе **S[a:b]**. Если указать только два параметра **S.find(T, a)**, то поиск будет осуществляться в срезе **S[a:]**, то есть начиная с символа с индексом a и до конца строки. Метод **S.find(T, a, b)** возвращает индекс в строке **S**, а не индекс относительно среза.

# Строки

Метод **replace**

Метод **replace** заменяет все вхождения одной строки на другую. Формат: **S.replace(old, new)** – заменить в строке **S** все вхождения подстроки **old** на подстроку **new**. Пример:

```
1  print('Hello'.replace('l', 'L'))
2  # вернёт 'HeLLo'
3
```

Если методу **replace** задать еще один параметр: **S.replace(old, new, count)**, то заменены будут не все вхождения, а только не больше, чем первые **count** из них.

```
1  print('Abrakadabra'.replace('a', 'A', 2))
2  # вернёт 'AbrAkAdabra'
3
```

# Строки

Метод **count**

Подсчитывает количество вхождений одной строки в другую строку. Простейшая форма вызова **S.count(T)** возвращает число вхождений строки **T** внутри строки **S**. При этом подсчитываются только непересекающиеся вхождения, например:

```
1  print('Abracadabra'.count('a'))
2  # вернёт 4
3  print(('a' * 10).count('aa'))
4  # вернёт 5
5
```

При указании трех параметров **S.count(T, a, b)**, будет выполнен подсчет числа вхождений строки **T** в срезе **S[a:b]**.

# Задач

## Задача «Количество слов»

### Условие

Дана строка, состоящая из слов, разделенных пробелами. Определите, сколько в ней слов. Используйте для решения задачи метод `count`.

## Задача «Две половинки»

### Условие

Дана строка. Разрежьте ее на две равные части (если длина строки — четная, а если длина строки нечетная, то длина первой части должна быть на один символ больше). Переставьте эти две части местами, результат запишите в новую строку и выведите на экран.

При решении этой задачи не стоит пользоваться инструкцией `if`.

## Задача «Второе вхождение»

### Условие

Дана строка. Найдите в этой строке **второе** вхождение буквы `f`, и выведите индекс этого вхождения. Если буква `f` в данной строке встречается только один раз, выведите число `-1`, а если не встречается ни разу, выведите число `-2`.

# Задачи

## Задача «Обращение фрагмента»

Условие

Дана строка, в которой буква `h` встречается как минимум два раза. Разверните последовательность символов, заключенную между первым и последним появлением буквы `h`, в противоположном порядке.

## Задача «Замена внутри фрагмента»

Условие

Дана строка. Замените в этой строке все появления буквы `h` на букву `H`, кроме первого и последнего вхождения.

## Задача «Удалить каждый третий символ»

Условие

Дана строка. Удалите из нее все символы, чьи индексы делятся на 3.

# Цикл while

Цикл **while** (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл **while** используется, когда невозможно определить точное значение количества проходов исполнения цикла. Синтаксис цикла **while** в простейшем случае выглядит так:

```
1 while условие:
2     блок инструкций
3
```

При выполнении цикла **while** сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла **while**. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

# Цикл while

Например, следующий фрагмент программы напечатает на экран квадраты всех целых чисел от **1** до **10**. Видно, что цикл **while** может заменять цикл **for** ... **in range(...):**

```
1  i = 1
2  while i <= 10:
3      print(i ** 2)
4      i += 1
5
```

В этом примере переменная **i** внутри цикла изменяется от **1** до **10**. Такая переменная, значение которой меняется с каждым новым проходом цикла, называется счетчиком. Заметим, что после выполнения этого фрагмента значение переменной **i** будет равно **11**, поскольку именно при **i == 11** условие **i <= 10** впервые перестанет выполняться.

# Цикл while

Вот еще один пример использования цикла **while** для определения количества цифр натурального числа **n**:

```
1  n = int(input())
2  length = 0
3  while n > 0:
4      n //= 10  # это эквивалентно n = n // 10
5      length += 1
6  print(length)
7
```

В этом цикле мы отбрасываем по одной цифре числа, начиная с конца, что эквивалентно целочисленному делению на **10** (**n //= 10**), при этом считаем в переменной **length**, сколько раз это было сделано.

В языке Питон есть и другой способ решения этой задачи: **length = len(str(i))**.

# Цикл while

После тела цикла можно написать слово **else:** и после него блок операций, который будет выполнен один раз после окончания цикла, когда проверяемое условие станет неверно:

```
1  i = 1
2  while i <= 10:
3      print(i)
4      i += 1
5  else:
6      print('Цикл окончен, i =', i)
7
```

Казалось бы, никакого смысла в этом нет, ведь эту же инструкцию можно просто написать после окончания цикла. Смысл появляется только вместе с инструкцией **break**. Если во время выполнения Питон встречает инструкцию **break** внутри цикла, то он сразу же прекращает выполнение этого цикла и выходит из него. При этом ветка **else** исполняться не будет. Разумеется, инструкцию **break** осмыленно вызывать только внутри инструкции **if**, то есть она должна выполняться только при выполнении какого-то особенного условия.

# Цикл while

Приведем пример программы, которая считывает числа до тех пор, пока не встретит отрицательное число. При появлении отрицательного числа программа завершается. В первом варианте последовательность чисел завершается числом **0** (при считывании которого надо остановиться).

```
a = int(input())
while a != 0:
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
    a = int(input())
else:
    print('Ни одного отрицательного числа не встретилось')

```

# Цикл while

Во втором варианте программы сначала на вход подается количество элементов последовательности, а затем и сами элементы. В таком случае удобно воспользоваться циклом **for**. Цикл **for** также может иметь ветку **else** и содержать инструкции **break** внутри себя.

```
1  n = int(input())
2  for i in range(n):
3      a = int(input())
4      if a < 0:
5          print('Встретилось отрицательное число', a)
6          break
7  else:
8      print('Ни одного отрицательного числа не встретилось')
9
```

Другая инструкция управления циклом — **continue** (продолжение цикла). Если эта инструкция встречается где-то посередине цикла, то пропускаются все оставшиеся инструкции до конца цикла, и исполнение цикла продолжается со следующей итерации.

# Цикл while

Увлечение инструкциями **break** и **continue** не поощряется, если можно обойтись без их использования. Вот типичный пример плохого использования инструкции **break** (данный код считает количество знаков в числе).

```
n = int(input())
length = 0
while True:
    length += 1
    n //= 10
    if n == 0:
        break
print('Длина числа равна', length)

```

Гораздо лучше переписать этот цикл так:

```
n = int(input())
length = 0
while n != 0:
    length += 1
    n //= 10
print('Длина числа равна', length)

```

или

```
n = int(input())
print('Длина числа равна', len(str(n)))

```

# Множественное присваивание

В Питоне можно за одну инструкцию присваивания изменять значение сразу нескольких переменных. Делается это так:

```
1  a, b = 0, 1
2
```

или

```
1  a = 0
2  b = 1
3
```

Отличие двух способов состоит в том, что множественное присваивание в первом способе меняет значение двух переменных одновременно.

Если слева от знака «=» в множественном присваивании должны стоять через запятую имена переменных, то справа могут стоять произвольные выражения, разделённые запятыми. Главное, чтобы слева и справа от знака присваивания было одинаковое число элементов.

# Множественное присваивание

Множественное присваивание удобно использовать, когда нужно обменять значения двух переменных. В обычных языках программирования без использования специальных функций это делается так:

```
1  a = 1
2  b = 2
3  tmp = a
4  a = b
5  b = tmp
6  print(a, b)
7  # 2 1
8
```

В Питоне то же действие записывается в одну строчку:

```
1  a = 1
2  b = 2
3  a, b = b, a
4  print(a, b)
5  # 2 1
6
```

# Задачи

## Задача «Список квадратов»

### Условие

По данному целому числу N распечатайте все квадраты натуральных чисел, не превосходящие N, в порядке возрастания.

## Задача «Степень двойки»

### Условие

По данному натуральному числу N найдите наибольшую целую степень двойки, не превосходящую N. Выведите показатель степени и саму степень.

Операцией возведения в степень пользоваться нельзя!

## Задача «Длина последовательности»

### Условие

Программа получает на вход последовательность целых неотрицательных чисел, каждое число записано в отдельной строке. Последовательность завершается числом 0, при считывании которого программа должна закончить свою работу и вывести количество членов последовательности (не считая завершающего числа 0). Числа, следующие за числом 0, считывать не нужно.

# Задачи

## Задача «Числа Фибоначчи»

### Условие

Последовательность Фибоначчи определяется так:

$$\varphi_0 = 0,\ \varphi_1 = 1,\ \varphi_n = \varphi_{n-1} + \varphi_{n-2}.$$

По данному числу $n$ определите $n$-е число Фибоначчи $\varphi_n$.

Эту задачу можно решать и циклом `for` .

## Задача «Максимальное число идущих подряд равных элементов»

### Условие

Дана последовательность натуральных чисел, завершающаяся числом 0. Определите, какое наибольшее число подряд идущих элементов этой последовательности равны друг другу.

# Спасибо за внимание!