Федеральное государственное образовательное бюджетное   
учреждение высшего образования

**«Финансовый университет при Правительстве Российской Федерации»**

**(Финансовый университет)**

Колледж информатики и программирования

**ОТЧЕТ   
по практической (самостоятельной) работе**

**Лабораторная работа № 5:** Python

**Студента:** Ерминой С.Д.

**Дисциплина /Профессиональный модуль:** Основы алгоритмизации и программирования

|  |  |  |
| --- | --- | --- |
| **Группа: 2ИСИП-319** |  | **Преподаватель:** |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_/И.В. Сибирев/ |
|  |  | **Дата выполнения:** |
|  |  | 9.04.2021 г. |
|  |  | **Оценка за работу: \_\_\_\_\_\_\_\_\_\_\_\_** |

Москва   
2021

Содержание

[Введение 4](#_Toc68886751)

[Лекция 4](#_Toc68886752)

[Вывод данных в консоль 4](#_Toc68886753)

[Ввод данных с клавиатуры 4](#_Toc68886754)

[Работа с файлами 5](#_Toc68886755)

[Открытие и закрытие файла 5](#_Toc68886756)

[Чтение данных из файла 5](#_Toc68886757)

[Запись данных в файл 6](#_Toc68886758)

[Дополнительные методы для работы с файлами 6](#_Toc68886759)

[Что такое функция в Python? 6](#_Toc68886760)

[Создание функций 7](#_Toc68886761)

[Работа с функциями 7](#_Toc68886762)

[*lambda*-функции 7](#_Toc68886763)

[Типы данных в Python 8](#_Toc68886764)

[Модель данных 8](#_Toc68886765)

[Изменяемые и неизменяемые типы данных 10](#_Toc68886766)

[Условный оператор ветвления *if* 11](#_Toc68886767)

[1. Конструкция *if* 11](#_Toc68886768)

[2. Конструкция *if – else* 12](#_Toc68886769)

[3. Конструкция *if – elif – else* 12](#_Toc68886770)

[Оператор цикла *while* 13](#_Toc68886771)

[Операторы *break* и *continue* 13](#_Toc68886772)

[Оператор цикла *for* 13](#_Toc68886773)

[Что представляют собой строки в Python 14](#_Toc68886774)

[Литералы строк 14](#_Toc68886775)

[Кодировка строк 15](#_Toc68886776)

[Максимальная длина строки в Python 15](#_Toc68886777)

[Перенос строк 15](#_Toc68886778)

[Конкатенация строк 15](#_Toc68886779)

[Сравнение строк 15](#_Toc68886780)

[Пустая строка Python 16](#_Toc68886781)

[Как удалить строку в Python 16](#_Toc68886782)

[Обращение по индексу 17](#_Toc68886783)

[Форматирование строк 17](#_Toc68886784)

[Оператор % 17](#_Toc68886785)

[str.format() 18](#_Toc68886786)

[f-строки (Python 3.6+) 18](#_Toc68886787)

[Стандартная библиотека Template Strings 18](#_Toc68886788)

[Функции для работы со строками 18](#_Toc68886789)

[Методы для работы со строками 19](#_Toc68886790)

[Преобразование из строки в другой тип 20](#_Toc68886791)

[string to int 20](#_Toc68886792)

[string to list 20](#_Toc68886793)

[string to bytes 20](#_Toc68886794)

[string to datetime 20](#_Toc68886795)

[string to float 21](#_Toc68886796)

[string to dict 21](#_Toc68886797)

[string to json 21](#_Toc68886798)

[Best practices 21](#_Toc68886799)

[Как разбить строку на символы 21](#_Toc68886800)

[Как из строки выделить числа 21](#_Toc68886801)

[Как перевернуть строку 22](#_Toc68886802)

[Как удалить последний символ в строке 22](#_Toc68886803)

[Как убрать пробелы из строки 22](#_Toc68886804)

[Список использованной литературы 22](#_Toc68886805)

# Введение

***Что такое Python***

Python – популярный язык программирования. Создан в 1991 году Гвидо ван Россумом.

Сейчас используется для:

* веб-разработки (серверной),
* разработки программного обеспечения,
* Data science,
* парсинга.

***Какие возможности дает Python***

* может использоваться на сервере для создания веб-приложений.
* может использоваться вместе с программным обеспечением для создания рабочих процессов.
* подключается к системам баз данных. С его помощью можно также читать и изменять файлы.
* может обрабатывать большие данные и выполнения сложных математических вычислений.
* используется для быстрого прототипирования или для разработки готового программного обеспечения.

***Почему Python?***

* поддерживается на разных платформах (Windows, Mac, Linux, Raspberry Pi и т. Д.),
* имеет простой синтаксис, похожий на английский язык,
* имеет синтаксис, который позволяет разработчикам писать программы с меньшим количеством строк, в отличии от других языков программирования,
* работает в системе интерпретатора, что означает, что код может быть выполнен сразу же после его написания. Это означает, что прототипирование становится очень быстрым,
* поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное и функциональное.

***Полезно знать***

* Последней основной версией Python является Python 3, который мы будем использовать в этих уроках. Однако Python 2, хотя и не обновляется никак, кроме обновлений для системы безопасности, все еще довольно популярен.
* В этих уроках код Python будет написан в текстовом редакторе. Можно написать Python в интегрированной среде разработки, такой как Thonny, Pycharm, Netbeans или Eclipse, которые особенно полезны при управлении большими коллекциями файлов Python.

***Синтаксис Python в сравнении с синтаксисом других языков программирования***

* Python был разработан для удобочитаемости и имеет некоторое сходство с английским языком вперемешку с математикой.
* Он использует новые строки для выполнения команды, в отличие от других языков программирования, которые часто используют точки с запятой или круглые скобки.
* Python полагается на отступы, используя пробелы, для определения области; таких как область циклов, функций и классов. Другие языки программирования часто используют фигурные скобки для этой цели.

# Лекция

Вывод данных в консоль

Один из самых распространенных способов вывести данные в *Python* – это напечатать их в консоли. Если вы находитесь на этапе изучения языка, такой способ является основным для того, чтобы быстро просмотреть результат свой работы. Для вывода данных в консоль используется функция *print*.

Рассмотрим основные способы использования данной функции.

>>> print("Hello")

Hello

>>> print("Hello, " + "world!")

Hello, world!

>>> print("Age: " + str(**23**))

Age: 23

По умолчанию, для разделения элементов в функции *print* используется пробел.

>>> print("A", "B", "C")

A B C

Для замены разделителя необходимо использовать параметр *sep* функции *print*.

print("A", "B", "C", sep="#")

A#B#C

В качестве конечного элемента выводимой строки, используется символ перевода строки.

>>> for i in range(**3**):

print("i: " + str(i))

i: 0

i: 1

i: 2

Для его замены используется параметр *end*.

>>> for i in range(**3**):

print("[i: " + str(i) + "]", end=" -- ")

[i: 0] -- [i: 1] -- [i: 2] --

Ввод данных с клавиатуры

Для считывания вводимых с клавиатуры данных используется функция *input()*.

>>> input()

test

'test'

Для сохранения данных в переменной используется следующий синтаксис.

>>> a = input()

hello

>>> print(a)

hello

Если считывается с клавиатуры целое число, то строку, получаемую с помощью функции *input()*, можно передать сразу в функцию *int()*.

>>> val = int(input())

123

>>> print(val)

123

>>> type(val)

<*class* 'int'>

Для вывода строки-приглашения, используйте ее в качестве аргумента функции *input()*.

>>> tv = int(input("input number: "))

input number: 334

>>> print(tv)

334

Преобразование строки в список осуществляется с помощью метода *split()*, по умолчанию, в качестве разделителя, используется пробел.

>>> l = input().split()

1 2 3 4 5 6 7

>>> print(l)

['1', '2', '3', '4', '5', '6', '7']

Разделитель можно заменить, указав его в качестве аргумента метода*split()*.

>>> nl = input().split("-")

1-2-3-4-5-6-7

>>> print(nl)

['1', '2', '3', '4', '5', '6', '7']

Для считывания списка чисел с одновременным приведением их к типу *int* можно воспользоваться вот такой конструкцией.

>>> nums = map(int, input().split())

1 2 3 4 5 6 7

>>> print(list(nums))

[1, 2, 3, 4, 5, 6, 7]

Работа с файлами

Открытие и закрытие файла

Для открытия файла используется функция *open()*, которая возвращает файловый объект. Наиболее часто используемый вид данной функции выглядит так *open(имя\_файла, режим\_доступа)*.

Для указания режима доступа используется следующие символы:

‘*r*’ – открыть файл для чтения;

‘*w*’ – открыть файл для записи;

‘*x*’ – открыть файл с целью создания, если файл существует, то вызов функции *open* завершится с ошибкой;

‘*a*’ – открыть файл для записи, при этом новые данные будут добавлены в конец файла, без удаления существующих;

‘*b*’ – бинарный режим;

‘*t*’ – текстовый режим;

‘*+*’ – открывает файл для обновления.

По умолчанию файл открывается на чтение в текстовом режиме.

У файлового объекта есть следующие атрибуты.

*file.closed* – возвращает *true* если файл закрыт и *false* в противном случае;

*file.mode* – возвращает режим доступа к файлу, при этом файл должен быть открыт;

*file.name* – имя файла.

>>> f = open("test.txt", "r")

>>> print("file.closed: " + str(f.closed))

file.closed: False

>>> print("file.mode: " + f.mode)

file.mode: r

>>> print("file.name: " + f.name)

file.name: test.txt

Для закрытия файла используется метод *close()*.

Чтение данных из файла

Чтение данных из файла осуществляется с помощью методов *read(размер)* и *readline()*.

Метод *read(размер)* считывает из файла определенное количество символов, переданное в качестве аргумента. Если использовать этот метод без аргументов, то будет считан весь файл.

>>> f = open("test.txt", "r")

>>> f.read()

'1 2 3 4 5**\n**Work with file**\n**'

>>> f.close()

В качестве аргумента метода можно передать количество символом, которое нужно считать.

>>> f = open("test.txt", "r")

>>> f.read(**5**)

'1 2 3'

>>> f.close()

Метод *readline()* позволяет считать строку из открытого файла.

>>> f = open("test.txt", "r")

>>> f.readline()

'1 2 3 4 5**\n**'

>>> f.close()

Построчное считывание можно организовать с  помощью оператора *for*.

>>> f = open("test.txt", "r")

>>> for line in f:

... print(line)

...

1 2 3 4 5

Work with file

>>> f.close()

Запись данных в файл

Для записи данных файл используется метод *write(строка)*, при успешной записи он вернет количество записанных символов.

>>> f = open("test.txt", "a")

>>> f.write("Test string")

11

>>> f.close()

Дополнительные методы для работы с файлами

Метод *tell()* возвращает текущую позицию “условного курсора” в файле. Например, если вы считали пять символов, то “курсор” будет установлен в позицию 5.

>>> f = open("test.txt", "r")

>>> f.read(**5**)

'1 2 3'

>>> f.tell()

5

>>> f.close()

Метод *seek(позиция)* выставляет позицию в файле.

>>> f = open("test.txt", "r")

>>> f.tell()

0

>>> f.seek(**8**)

8

>>> f.read(**1**)

'5'

>>> f.tell()

9

>>> f.close()

Хорошей практикой при работе с файлами является применение оператора *with*. При его использовании нет необходимости закрывать файл, при завершении работы с ним, эта операция будет выполнена автоматически.

>>> with open("test.txt", "r") as f:

... for line in f:

... print(line)

...

1 2 3 4 5

Work with file

Test string

>>> f.closed

True

Что такое функция в Python?

По своей сути функции в Python практически ничем не отличаются от функций из других языков программирования. Функцией называют именованный фрагмент программного кода, к которому можно обратиться из другого места вашей программы (но есть lambda-функции, у которых нет имени, о них будет рассказано в конце урока). Как правило, функции создаются для работы с данными, которые передаются ей в качестве аргументов, также функция может формировать некоторое возвращаемое значение.

Создание функций

Для создания функции используется ключевое слово *def,* после которого указывается имя и список аргументов в круглых скобках. Тело функции выделяется также как тело условия (или цикла): четырьмя пробелами. Таким образом самая простая функция, которая ничего не делает, будет выглядеть так.

*def* fun():

pass

Возврат значения функцией осуществляется с помощью ключевого слова *return,* после которого указывается возвращаемое значение. Пример функции возвращающей единицу представлен ниже.

>>> *def* fun():

return **1**

>>> fun()

**1**

Работа с функциями

Во многих случаях функции используют для обработки данных. Эти данные могут быть глобальными, либо передаваться в функцию через аргументы. Список аргументов определяется на этапе реализации и указывается в круглых скобках после имени функции. Например операцию сложения двух аргументов можно реализовать вот так.

>>> *def* summa(a, b):

return a + b

>>> summa(**3**, **4**)

**7**

Рассмотрим еще два примера использования функции: вычисление числа Фибоначчи с использованием рекурсии и вычисление факториала с использованием цикла.

Вычисление числа Фибоначчи.

>>> *def* fibb(n):

if n == **0**:

return **0**

elif n == **1**:

return **1**

elif n == **2**:

return **1**

else:

return fibb(n-**1**) + fibb(n-**2**)

>>> print(fibb(**10**))

**55**

Вычисление факториала.

>>> *def* factorial(n):

prod = **1**

for i in range(**1**, n+**1**):

prod \*= i

return prod

>>> print(factorial(**5**))

**120**

Функцию можно присвоить переменной и использовать ее, если необходимо сократить имя. В качестве примера можно привести вариант использования функции вычисления факториала из пакета *math*.

>>> import math

>>> f = math.factorial

>>> print(f(**5**))

**120**

*lambda*-функции

*Lambda*-функция – это безымянная функция с произвольным числом аргументов и вычисляющая одно выражение. Тело такой функции не может содержать более одной инструкции (или выражения). Данную функцию можно использовать в рамках каких-либо конвейерных вычислений (например внутри *filter()*, *map()* и *reduce()*) либо самостоятельно, в тех местах, где требуется произвести какие вычисление, которые удобно “завернуть” в функцию.

>>> (lambda x: x\*\***2**)(**5**)

**25**

*Lambda*-функцию можно присвоить какой-либо переменной и в дальнейшем использовать ее в качестве имени функции.

>>> sqrt = lambda x: x\*\***0.5**

>>> sqrt(**25**)

**5.0**

Списки можно обрабатывать *lambda*-функциями внутри таких функций как *map()*, *filter()*, *reduce()*, о них мы ещё поговорим, а пока рассмотрим пример с *map()*. Функция *map* принимает два аргумента, первый – это функция, которая будет применена к каждому элементу списка, а второй – это список, который нужно обработать.

>>> l = [**1**, **2**, **3**, **4**, **5**, **6**, **7**]

>>> list(map(*lambda* x: x\*\***3**, l))

[**1**, **8**, **27**, **64**, **125**, **216**, **343**]

**Типы данных в Python**

В Python типы данных можно разделить на встроенные в интерпретатор *(built-in*) и не встроенные, которые можно использовать при импортировании соответствующих модулей.

К основным встроенным типам относятся:

1. *None* (неопределенное значение переменной)
2. Логические переменные (*Boolean Type*)
3. Числа (*Numeric Type*)
   1. *int* – целое число
   2. *float* – число с плавающей точкой
   3. *complex* – комплексное число
4. Списки (*Sequence Type*)
   1. *list* – список
   2. *tuple* – кортеж
   3. *range* – диапазон
5. Строки (*Text Sequence Type* )
   1. *str*
6. Бинарные списки (*Binary Sequence Types*)
   1. *bytes* – байты
   2. *bytearray* – массивы байт
   3. *memoryview* – специальные объекты для доступа к внутренним данным объекта через protocol buffer
7. Множества (*Set Types*)
   1. *set* – множество
   2. *frozenset* – неизменяемое множество
8. Словари (*Mapping Types*)
   1. *dict* – словарь

**Модель данных**

Рассмотрим как создаются объекты в памяти, их устройство, процесс объявления новых переменных и работу операции присваивания.

Для того, чтобы объявить и сразу инициализировать переменную необходимо написать её имя, потом поставить знак равенства и значение, с которым эта переменная будет создана. Например строка:

b = **5**

объявляет переменную b и присваивает ей значение 5.

Целочисленное значение 5 в рамках языка Python по сути своей является объектом. Объект, в данном случае – это абстракция для представления данных, данные – это числа, списки, строки и т.п. При этом, под данными следует понимать как непосредственно сами объекты, так и отношения между ними (об этом чуть позже). Каждый объект имеет три атрибута – это идентификатор, значение и тип. Идентификатор – это уникальный признак объекта, позволяющий отличать объекты друг от друга, а значение – непосредственно информация, хранящаяся в памяти, которой управляет интерпретатор.

При инициализации переменной, на уровне интерпретатора, происходит следующее:

* создается целочисленный объект 5 (можно представить, что в этот момент создается ячейка и 5 кладется в эту ячейку);
* данный объект имеет некоторый идентификатор, значение: 5, и тип: целое число;
* посредством оператора “=” создается ссылка между переменной b и целочисленным объектом 5 (переменная b ссылается на объект 5).

Имя переменной не должно совпадать с ключевыми словами интерпретатора Python. Список ключевых слов можно найти [здесь](https://pythonworld.ru/osnovy/klyuchevye-slova-modul-keyword.html). Также его можно получить непосредственно в программе, для этого нужно подключить модуль *keyword* и воспользоваться командой *keyword.kwlist.*

>>> import keyword

>>> print("Python keywords: ", keyword.kwlist)

Проверить является или нет идентификатор ключевым словом можно так:

>>> keyword.iskeyword("try")

**True**

>>> keyword.iskeyword("b")

**False**

Для того, чтобы посмотреть на объект с каким идентификатором ссылается данная переменная, можно использовать функцию *id()*.

>>> a = **4**

>>> b = **5**

>>> id(a)

**1829984576**

>>> id(b)

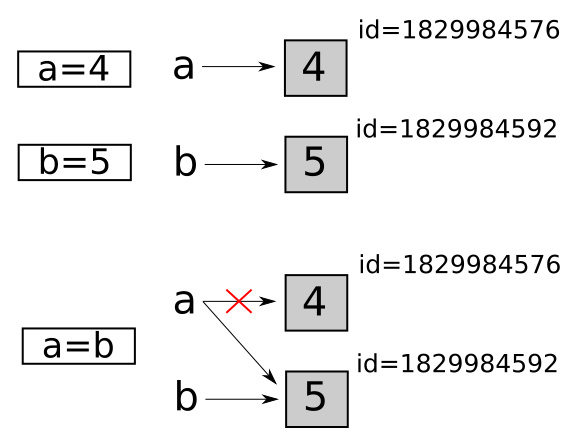
**1829984592**

>>> a = b

>>> id(a)

**1829984592**

Как видно из примера, идентификатор – это некоторое целочисленное значение, посредством которого уникально адресуется объект. Изначально переменная *a* ссылается на объект 4 с идентификатором 1829984576, переменная *b* – на объект с *id* = 1829984592. После выполнения операции присваивания *a = b*, переменная *a* стала ссылаться на тот же объект, что и *b*.



Тип переменной можно определить с помощью функции *type()*. Пример использования приведен ниже.

>>> a = **10**

>>> b = "hello"

>>> c = (**1**, **2**)

>>> type(a)

<*class* 'int'>

>>> type(b)

<*class* 'str'>

>>> type(c)

<*class* 'tuple'>

**Изменяемые и неизменяемые типы данных**

В Python существуют изменяемые и неизменяемые типы.

К **неизменяемым** (*immutable*) типам относятся: целые числа (*int*),  числа с плавающей точкой (*float*), комплексные числа (*complex*), логические переменные (*bool*), кортежи (*tuple*), строки (*str*) и неизменяемые множества (*frozen set*).

К **изменяемым** (*mutable*) типам относятся: списки (*list*), множества (*set*), словари (*dict*).

Как уже было сказано ранее, при создании переменной, вначале создается объект, который имеет уникальный идентификатор, тип и значение, после этого переменная может ссылаться на созданный объект.

Неизменяемость типа данных означает, что созданный объект больше не изменяется. Например, если мы объявим переменную*k* = 15, то будет создан объект со значением 15, типа *int* и идентификатором, который можно узнать с помощью функции *id()*.

>>> k = **15**

>>> id(k)

**1672501744**

>>> type(k)

<*class* 'int'>

Объект с *id* = 1672501744 будет иметь значение 15 и изменить его уже нельзя.

Если тип данных изменяемый, то можно менять значение объекта. Например, создадим список [1, 2], а потом заменим второй элемент на 3.

>>> a = [**1**, **2**]

>>> id(a)

**47997336**

>>> a[**1**] = **3**

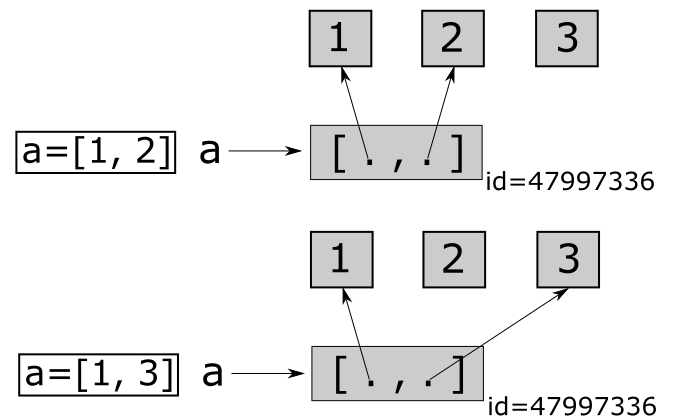
>>> a

[**1**, **3**]

>>> id(a)

**47997336**

Как видно, объект на который ссылается переменная *a*, был изменен. Это можно проиллюстрировать следующим рисунком.



В рассмотренном случае, в качестве данных списка, выступают не объекты, а отношения между объектами. Т.е. в переменной *a*хранятся ссылки на объекты содержащие числа 1 и 3, а не непосредственно сами эти числа.

**Условный оператор ветвления *if***

Оператор ветвления *if*  позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

**1. Конструкция *if***

Синтаксис оператора *if* выглядит так.

if выражение:

инструкция\_**1**

инструкция\_**2**

...

инструкция\_n

После оператора *if*  записывается выражение. Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое *True*. После выражения нужно поставить двоеточие “:”.

**ВАЖНО**: блок кода, который необходимо выполнить, в случае истинности выражения, отделяется четырьмя пробелами слева!

Примеры:

if **1**:

print("hello 1")

Напечатает: *hello 1*

a = **3**

if a == **3**:

   print("hello 2")

Напечатает: *hello 2*

a = **3**

if a > **1**:

print("hello 3")

Напечатает: *hello 3*

lst = [**1**, **2**, **3**]

if lst :

print("hello 4")

Напечатает: *hello 4*

**2. Конструкция *if – else***

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т.е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой. Для этого используется конструкция *if – else*.

if выражение:

инструкция\_1

инструкция\_2

...

инструкция\_n

else:

инструкция\_a

инструкция\_b

...

инструкция\_x

Примеры.

a = **3**

if a > **2**:

print("H")

else:

print("L")

Напечатает: *H*

a = **1**

if a > **2**:

   print("H")

else:

   print("L")

Напечатает: *L*

Условие такого вида можно записать в строчку, в таком случае оно будет представлять собой [тернарное выражение](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%80%D0%BD%D0%B0%D1%80%D0%BD%D0%B0%D1%8F_%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F).

a = **17**

b = **True** if a > **10** else **False**

print(b)

В результате выполнения такого кода будет напечатано: *True*

**3. Конструкция *if – elif – else***

Для реализации выбора из нескольких альтернатив можно использовать конструкцию*if – elif – else*.

if выражение\_1:

инструкции\_(блок\_1)

elif выражение\_2:

инструкции\_(блок\_2)

elif выражение\_3:

инструкции\_(блок\_3)

else:

инструкции\_(блок\_4)

Пример.

a = int(input("введите число:"))

if a < **0**:

print("Neg")

elif a == **0**:

print("Zero")

else:

print("Pos")

Если пользователь введет число меньше нуля, то будет напечатано “*Neg*“, равное нулю – “*Zero*“, большее нуля – “*Pos*“.

**Оператор цикла *while***

Оператор цикла *while* выполняет указанный набор инструкций до тех пор, пока условие цикла истинно. Истинность условия определяется также как и в операторе *if*. Синтаксис оператора *while* выглядит так.

while выражение:

инструкция\_1

инструкция\_2

...

инструкция\_n

Выполняемый набор инструкций называется телом цикла.

Пример.

a = **0**

while a < **7**:

print("A")

a += **1**

Буква “А” будет выведена семь раз в столбик.

Пример бесконечного цикла.

a = **0**

while a == **0**:

print("A")

**Операторы *break* и *continue***

При работе с циклами используются операторы ***break***и ***continue***.

Оператор ***break***предназначен для досрочного прерывания работы цикла *while*.

Пример.

a = **0**

while a >= **0**:

if a == **7**:

break

a += **1**

print("A")

В приведенном выше коде, выход из цикла произойдет при достижении переменной *a*  значения 7. Если бы не было этого условия, то цикл выполнялся бы бесконечно.

Оператор ***continue***запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

Пример.

a = -**1**

while a < **10**:

a += **1**

if a >= **7**:

continue

print("A")

При запуске данного кода символ “А” будет напечатан 7 раз, несмотря на то, что всего будет выполнено 11 проходов цикла.

**Оператор цикла *for***

Оператор *for* выполняет указанный набор инструкций заданное количество раз, которое определяется количеством элементов в наборе.

Пример.

for i in range(**5**):

print("Hello")

В результате *“Hello”* будет выведено пять раз.

Внутри тела цикла можно использовать операторы ***break***и ***continue***, принцип работы их точно такой же как и в операторе *while*.

Если у вас есть заданный список, и вы хотите выполнить над каждым элементом определенную операцию (возвести в квадрат и напечатать получившееся число), то с помощью *for*  такая задача решается так.

lst = [**1**, **3**, **5**, **7**, **9**]

for i in lst:

print(i \*\* **2**)

Также можно пройти по всем буквам в строке.

word\_str = "Hello, world!"

for l in word\_str:

print(l)

Строка *“Hello, world!”* будет напечатана в столбик.

Что представляют собой строки в Python

С точки зрения Питона, строка – это упорядоченная последовательность символов, которая предназначена для хранения информации в виде простого текста.

Поэтому тип данных string используется в случае, когда что-то нужно представить в текстовой форме.

Литералы строк

Литерал – способ создания объектов, в случае строк Питон предлагает несколько основных вариантов:

**>>>** 'string' *# одинарные кавычки*

'string'

**>>>** "string" *# двойные кавычки*

'string'

**>>>** """string"""

'string'

**>>>** '''string'''

'string'

Если внутри строки необходимо расположить двойные кавычки, и сама строка была создана с помощью двойных кавычек, можно сделать следующее:

**>>>** 'book "war and peace"' *# разный тип кавычек*

'book "war and peace"'

**>>>** "book 'war and peace'" *# разный тип кавычек*

"book 'war and peace'"

**>>>** "book \"war and peace\"" *# экранирование кавычек одного типа*

'book "war and peace"'

**>>>** 'book \'war and peace\'' *# экранирование кавычек одного типа*

"book 'war and peace'"

Какие кавычки использовать – решать вам, соглашение PEP 8 не дает рекомендаций по использованию кавычек. Просто выберите один тип кавычек и придерживайтесь его. Однако если в стоке используются те же кавычки, что и в литерале строки, используйте разные типы кавычек – обратная косая черта в строке ухудшает читаемость кода.

Кодировка строк

В третьей версии языка программирования Python все строки представляют собой последовательность Unicode-символов.

В Python 3 кодировка по умолчанию исходного кода – UTF-8. Во второй версии по умолчанию использовалась ASCII. Если необходимо использовать другую кодировку, можно разместить специальное объявление на первой строке файла, к примеру:

*# -\*- coding: windows-1252 -\*-*

Максимальная длина строки в Python

Максимальная длина строки зависит от платформы. Обычно это:

* 2\*\*31 - 1 – для 32-битной платформы;
* 2\*\*63 - 1 – для 64-битной платформы;

Константа maxsize , определенная в модуле sys :

**>>> import** sys

**>>>** sys.maxsize

2147483647

Перенос строк

Перенос строки осуществляется с помощью символа \n :

**>>>** text = "one\ntwo\nthree"

**>>>** print(text)

one

two

three

Конкатенация строк

Одна из самых распространенных операций со строками – их объединение (конкатенация). Для этого используется знак + , в результате к концу первой строки будет дописана вторая:

**>>>** s1 = "Hello" + " world"

**>>>** s2 = " world"

**>>>** s1+s2

'Hello world'

При необходимости объединения строки с числом его предварительно нужно привести тоже к строке, используя функцию str() :

**>>>** name = "John"

**>>>** age = 30

**>>>** "Name: " + name + ", age: " + str(age)

'Name: John, age: 30'

Сравнение строк

При сравнении нескольких строк рассматриваются отдельные символы и их регистр:

* цифра условно меньше, чем любая буква из алфавита;
* алфавитная буква в верхнем регистре меньше, чем буква в нижнем регистре;
* чем раньше буква в алфавите, тем она меньше;

При этом сравниваются по очереди первые символы, затем – 2-е и так далее.

**>>>** s1 = "1a"

**>>>** s2 = "aa"

**>>>** s3 = "Aa"

**>>>** s4 = "ba"

**>>>** "1a" > "aa" *# сравнение цифры с буквой*

False

**>>>** "aa" > "Aa" *# сравнение регистров*

True

**>>>** "aa" > "ba" *# сравнение букв по алфавитному порядку*

False

**>>>** "aa" < "az" *# первые буквы одинаковые, сравниваются следующие две*

True

Далеко не всегда желательной является зависимость от регистра, в таком случае можно привести обе строки к одному и тому же регистру. Для этого используются функции lower() – для приведения к нижнему и upper() – к верхнему:

**>>>** s1 = "Intel"

**>>>** s2 = "intel"

**>>>** s1 == s2

False

**>>>** s1.lower() == s2.lower()

True

Пустая строка Python

Объявить пустую строку можно следующими способами:

**>>>** ""

''

**>>>** ''

''

**>>>** str()

''

Как удалить строку в Python

Строки, как и некоторые другие типы данных в языке Python, являются неизменяемыми объектами. При задании нового значения строке просто создается новая, с заданным значением. Для удаления строки можно воспользоваться методом replace() , заменив ее на пустую строку:

**>>>** s = "test"

**>>>** s.replace("test", "")

''

Или перезаписать переменную пустой строкой:

**>>>** s = "test"

**>>>** s = ""

**>>>** s

''

Обращение по индексу

Для выбора определенного символа из строки можно воспользоваться обращением по индексу, записав его в квадратных скобках:

**>>>** s = "abcdef"

**>>>** s[0]

'a'

**>>>** s[2]

'c'

Индекс начинается с 0

В Python предусмотрена возможность получить доступ и по отрицательному индексу. В таком случае отсчет будет вестись от конца строки:

**>>>** s = "abcdef"

**>>>** s[-1]

'f'

Форматирование строк

Часто возникает ситуация, когда необходимо создать строку, подставив в нее определенные значения, полученные во время выполнения программы. Подстановка данных в таком случае выполняется при помощи форматирования строк, сделать это можно несколькими способами.

Оператор %

Строки в Python обладают встроенной операцией, к которой можно получить доступ оператором %, что дает возможность очень просто делать форматирование. Самый простой пример – когда для подстановки нужен только один аргумент, значением будет он сам:

**>>>** name = "Alex"

**>>>** 'Hello, %s' % name

'Hello, Alex'

Если же для подстановки используется несколько аргументов, то значением будет кортеж со строками:

**>>>** '%d %s, %d %s' % (6, 'bananas', 10, 'lemons')

'6 bananas, 10 lemons'

Как видно из предыдущего примера, зависимо от типа данных для подстановки и того, что требуется получить в итоге, пишется разный формат. Наиболее часто используются:

1. '%d', '%i', '%u – десятичное число;
2. '%c' – символ, точнее строка из одного символа или число – код символа;
3. '%r' – строка (литерал Python);
4. '%s' – строка.

Такой способ форматирования строк называет "старым" стилем, который в Python 3 был заменен на более удобные способы.

str.format()

В Python 3 появился более новый метод форматирования строк, который вскоре перенесли и в Python 2.7. Такой способ избавляет программиста от специального синтаксиса %-оператора. Делается все путем вызова .format() для строковой переменной. С помощью специального символа – фигурных скобок – указывается место для подстановки значения, каждая пара скобок указывает отдельное место для подстановки, значения могут быть разного типа:

**>>>** print('{}'.format(100))

100

**>>>** '{0}, {1}, {2}'.format('one', 'two', 'three')

'one, two, three'

**>>>** '{2}, {1}, {0}'.format('one', 'two', 'three')

'three, two, one'

💭 В Python 3 форматирование строк с использованием "нового стиля" является более предпочтительным по сравнению с использованием %-стиля, так как предоставляет более широкие возможности, не усложняя простые варианты использования.

f-строки (Python 3.6+)

В Python версии 3.6 появился новый метод форматирования строк – "f-строки", с его помощью можно использовать встроенные выражения внутри строк:

**>>>** name = 'Alex'

**>>>** f'Hello, {name}!'

'Hello, Alex!'

Такой способ форматирования очень мощный, так как дает возможность встраивать выражения:

**>>>** a = 5

**>>>** b = 10

**>>>** f'Five plus ten is {a + b} and not {2 \* (a + b)}.'

'Five plus ten is 15 and not 30.'

Таким образом, форматирование с помощью f-строк напоминает использование метода format(), но более гибкое, быстрое и читабельное.

Стандартная библиотека Template Strings

Еще один способ форматирования строк, который появился еще с выходом Python версии 2.4, но так и не стал популярным – использование библиотеки Template Strings. Есть поддержка передачи значения по имени, используется $-синтаксис как в языке PHP:

**>>> from** string **import** Template

**>>>** name = "Alex"

**>>>** age = 30

**>>>** s = Template('My name is $name. I’m $age.')

**>>>** print(s.substitute(name=name, age=age))

My name **is** Alex. I'm 30

Функции для работы со строками

Для работы со строками в Питоне предусмотрены специальные функции. Рассмотрим их:

Преобразование числового или другого типа к строке:

* str(n) – преобразование числового или другого типа к строке;
* len(s) – длина строки;
* chr(s) – получение символа по его коду ASCII;
* ord(s) – получение кода ASCII по символу;

Методы для работы со строками

Кроме функций, для работы со строками есть немало методов:

* find(s, start, end) – возвращает индекс первого вхождения подстроки в s или -1 при отсутствии. Поиск идет в границах от start до end ;
* rfind(s, start, end) – аналогично, но возвращает индекс последнего вхождения;
* replace(s, new) – меняет последовательность символов s на новую подстроку new ;
* split(x) – разбивает строку на подстроки при помощи выбранного разделителя x;
* join(x) – соединяет строки в одну при помощи выбранного разделителя x;
* strip(s) – убирает пробелы с обеих сторон;
* lstrip(s), rstrip(s) – убирает пробелы только слева или справа;
* lower() – перевод всех символов в нижний регистр;
* upper() – перевод всех символов в верхний регистр;
* capitalize() – перевод первой буквы в верхний регистр, остальных – в нижний.

👉 Примеры использования:

**>>>** text = "Wikipedia is a Python library that makes it easy to access and parse data from Wikipedia"

**>>>** text.find("Wikipedia")

0

**>>>** text.rfind("Wikipedia")

79

**>>>** text.replace("from Wikipedia", "from https://www.wikipedia.org/")

'Wikipedia is a Python library that makes it easy to access and parse data from https://www.wikipedia.org/'

**>>>** text.split(" ")

['Wikipedia', 'is', 'a', 'Python', 'library', 'that', 'makes', 'it', 'easy', 'to', 'access', 'and', 'parse', 'data', 'from', 'Wikipedia']

split\_text = text.split(" ")

**>>>** "\_".join(split\_text)

'Wikipedia\_is\_a\_Python\_library\_that\_makes\_it\_easy\_to\_access\_and\_parse\_data\_from\_Wikipedia'

**>>>** text = " test "

**>>>** text.strip()

'test'

**>>>** text.lstrip()

'test '

**>>>** text.rstrip()

' test'

**>>>** text = "Python is a product of the Python Software Foundation"

**>>>** text.lower()

'python is a product of the python software foundation'

**>>>** text.upper()

'PYTHON IS A PRODUCT OF THE PYTHON SOFTWARE FOUNDATION'

**>>>** text = "python is a product of the python software foundation"

**>>>** text.capitalize()

'Python is a product of the python software foundation'

Преобразование из строки в другой тип

В Питоне строки можно преобразовывать в другие типы данных:

string to int

Функция int() преобразовывает целое число в десятичной системе, заданное как строка, в тип int:

**>>>** int("10")

10

При необходимости можно указывать систему счисления:

**>>>** int("0x12F", base=16)

303

string to list

Самый простой способ преобразования строки в список строк – метод split():

**>>>** 'one two three four'.split()

['one', 'two', 'three', 'four']

При необходимости можно указывать разделитель:

**>>>** 'one, two, three, four'.split(',')

['one', ' two', ' three', ' four']

string to bytes

Преобразование строкового типа в байтовый выполняется функцией encode() с указанием кодировки:

**>>>** 'Байты'.encode('utf-8')

b'\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'

string to datetime

Строка в дату преобразовывается функцией strptime() из стандартного модуля datetime:

**>>> from** datetime **import** datetime

**>>>** print(datetime.strptime('Jan 1 2020 1:33PM', '%b %d %Y %I:%M%p'))

2020-01-01 13:33:00

string to float

Для преобразования строки в число с плавающей точкой используется стандартная функция float:

**>>>** float('1.5')

1.5

string to dict

Создание словаря из строки возможно, если внутри нее данные в формате json. Для этого можно воспользоваться модулем json :

**>>> import** json

**>>>** print(json.loads('{"Russia": "Moscow", "France": "Paris"}'))

{'Russia': 'Moscow', 'France': 'Paris'}

string to json

Конвертация объектов Python в объект json выполняется функцией dumps():

**>>> import** json

**>>>** json.dumps("hello")

'"hello"'

Best practices

Как разбить строку на символы

Разбиение строки на отдельные символы выполняется несколькими способами:

**>>>** text = "django"

*# вариант 1*

**>>>** list(text)

['d', 'j', 'a', 'n', 'g', 'o']

*# вариант 2*

**>>>** [c **for** c **in** "text"]

['t', 'e', 'x', 't']

*# вариант 3*

**>>> for** c **in** text:

print(c)

d

j

a

n

g

o

Как из строки выделить числа

Для извлечения чисел из строки можно воспользоваться методами строк:

**>>>** str = "h3110 23 cat 444.4 rabbit 11 2 dog"

**>>>** [int(s) **for** s **in** str.split() **if** s.isdigit()]

[23, 11, 2]

Данный пример извлекает только целые положительные числа. Более универсальный вариант – регулярные выражения:

**>>>** str = "h3110 23 cat 444.4 rabbit 11 2 dog"

**>>> import** re

**>>>** re.findall(r'\d+', str)

['3110', '23', '444', '4', '11', '2']

Как перевернуть строку

Существует несколько способов перевернуть строку, начиная от классического – запустить цикл в обратной последовательности, выбирая каждый символ с конца и вставляя его в новую строку, и заканчивая срезами – вариант только для Питона.

С помощью среза – самый быстрый способ:

**>>>** 'test'[::-1]

'tset'

Использование reversed() и str.join() :

**>>>** ''.join(reversed('test'))

'tset'

Как удалить последний символ в строке

Как и в предыдущем случае – чтобы убрать последний символ наиболее простым и быстрым вариантом будет использование среза:

**>>>** "Some text1"[:-1]

'Some text'

Как убрать пробелы из строки

В случае удаления пробелов со строки может быть два варианта:

1. Обрезать строку так, чтобы удалить с нее первый и последний пробел, такой вариант может пригодиться, когда пользователь случайно поставит пробел в конце введенного текста:

**>>>** " Some text ".strip()

'Some text'

2. Удалить со строки все пробелы:

**>>>** " So me t e x t ".replace(' ', '')

'Sometext'

# Список использованной литературы

<https://devpractice.ru/python-lesson-1-install/>