

Основы Python3

Python – простой, минималистичный язык и обладает исключительно простым синтаксисом. При написании программы на Python никогда не приходится отвлекаться на такие низкоуровневые детали, как управление памятью, используемой вашей программой, и т.п. Также Python является интерпретируемым языком, то есть не требует компиляции в бинарный код. Программа просто выполняется из исходного текста. Python сам преобразует этот исходный текст в некоторую промежуточную форму, называемую байткодом, а затем переводит его на машинный язык и запускает.

Установка Python

Для установки Python на Linux\Unix системы необходимо в терминале выполнить следующие команды:

```
sudo apt-get update
sudo apt-get install python3.8
```

Для установки Python для Windows необходимо скачать дистрибутив с официального сайта <https://www.python.org/downloads/>.

Интерпретатор Python

Как говорилось, Python является интерпретируемым языком, в следствии чего есть возможность исполнять отдельные команды в интерпретаторе. Для того, чтобы запустить интерпретатор, необходимо открыть терминал и в нем написать команду *python* или *python3*. После чего в том же окне терминала запустится интерпретатор:

```
PS C:\Users\Ksenox> python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Как видно, теперь каждая строка начинается с комбинации символов >>>. Это означает, что интерпретатор ждет ввода команд. Например, давайте найдем произведение двух чисел, для этого введем следующие команды:

```
>>> a = 5
>>> b = 3
>>> a * b
15
```

В данном примере мы создали 2 переменные a и b, а затем присвоили им значения 5 и 3 соответственно. После этого ввели команду a * b и получили произведение двух чисел. Можно заметить, что результат произведения никуда не присваивался, и интерпретатор в данном случае вывел результат выполнения команды.

Редактор для Python

Выполнять большие и сложные программы в среде интерпретатора не эффективно. Поэтому, программу на языке Python, как и большинство программ на других языках, можно написать в файле, а затем ее выполнить. Файлы с исходным кодом на языке Python имеют расширения `.py`. Для того, чтобы выполнить программу из файла необходимо в терминале вызвать интерпретатор `python` и в качестве аргумента указать путь до файла с кодом.

Для написания кода можно использовать любой текстовый редактор, но это неудобно по той причине, что далеко не во всех редакторах есть поддержка подсветки синтаксиса и автодополнения. Поэтому, для разработки программ на языке Python можно использовать следующие средства:

- Visual Studio Code с плагином Python. По сути, эта среда представляет обычный текстовый редактор с подсветкой синтаксиса и автодополнением, а также встроенным терминалом.
- Jupyter Notebook. Интерактивная среда разработки, которая помимо стандартных функций написания кода, предоставляет возможность выполнять отдельные блоки кода.
- PyCharm. Полноценная IDE для написания кода на языке Python и соответственно обладает большим набором функционала присущим IDE для других языков.

Теперь, напишем программу перемножения двух чисел в файле:

```
a = 5
b = 3
a * b
```

И запустим ее на исполнение:

```
PS C:\Univer\NeuroNets\PR1> python example1.py
PS C:\Univer\NeuroNets\PR1> █
```

Как видно, интерпретатор не вывел результат перемножения, так как если результат выполнения в файле не выводится при помощи специальной команды, то интерпретатор ничего не выведет. Для вывода используется функция `print`, в качестве аргументов которой передается выражения, результат которого необходимо вывести. Для того, чтобы считать что-то, используется функция `input`.

Изменим написанную выше программу следующим образом:

```
#Ввод a
a = int(input('Введите a: '))
#Ввод b
b = int(input('Введите b: '))
print(a * b) #Вывод результата a * b
```

В данном коде:

- Комментарии начинаются символом `#`. Они игнорируются интерпретатором.
- Функция `input` может принимать строку в качестве аргумента. Интерпретатор будет выводить данную строку, а затем ожидать ввода.
- Так как функция `input` все введенные значения возвращает в виде строки, то необходимо использовать функцию `int`, чтобы перевести строку в число.

Пример, выполнения программы:

```
PS C:\Univer\NeuroNets\PR1> python example1.py
Введите a: 3
Введите b: 5
15
```

Интерпретатор Python'a завершает работу программы (скрипта), если происходит какая-то ошибка. Например, если введенная строка не является представлением числа и ее нельзя привести к целому числу:

```
PS C:\Univer\NeuroNets\PR1> python example1.py
Введите a: NUMBER
Traceback (most recent call last):
  File "example1.py", line 2, in <module>
    a = int(input('Введите a: '))
ValueError: invalid literal for int() with base 10: 'NUMBER'
```

Также, в интерпретатор Python встроена документация, для того, чтобы ее вызвать, необходимо ввести команду `help()`. В качестве аргумента, можно передать название функции, модуля, объекта или ключевое слово. Например, можно вызвать справку о функции `print`:

```
PS C:\Univer\NeuroNets\PR1> python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Литеральные константы в Python

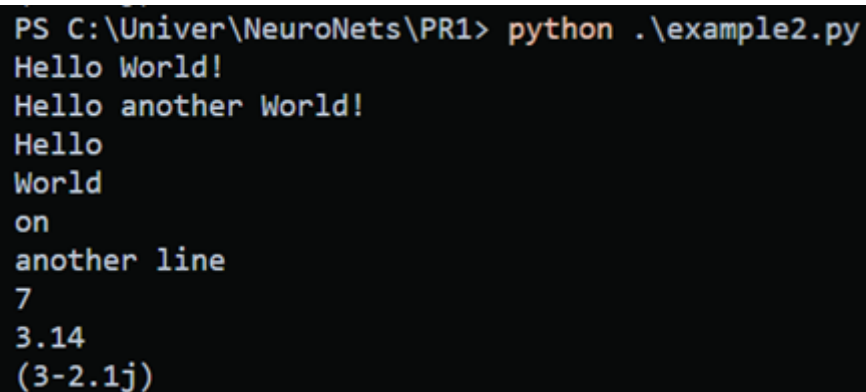
В языке Python существуют следующие литеральные константы:

- Строки
 - Строка может быть записана в одинарных кавычках. Например, 'Hello World!'
 - Строка может быть записана в двойных кавычках. Например, "Hello another World!"
 - Строка может быть написана с переносами в тройных одинарных кавычках
'''Hello
World
on
another
line'''
- Числа
 - Целые числа. Например, 7
 - вещественные числа. Например, 3.14
 - Комплексные числа. 3-2.1j

Пример констант:

```
print('Hello world!')
print("Hello another world!")
print('''Hello
world
on
another line''')
print(7)
print(3.14)
print(3-2.1j)
```

После выполнения программы выше будет выведено следующее:



```
PS C:\Univer\NeuroNets\PR1> python .\example2.py
Hello World!
Hello another World!
Hello
World
on
another line
7
3.14
(3-2.1j)
```

Строки и числами в Python являются иммутабельными, то есть они хранятся в памяти, но их нельзя изменить. То есть в Python можно по индексу обращаться к символу строки, получить его значения, но его нельзя изменять на другой, чтобы изменить строку, в отличие от того как это можно сделать в таких языках как C или C++.

Переменные в Python

В Python переменные просто области памяти компьютера, в которых хранится некоторая информация. В отличие от констант, к такой информации нужно каким-то образом получить доступ, поэтому переменным даются имена.

Первым символом идентификатора должна быть буква или символ нижнего подчеркивания "_". Остальная часть идентификатора может состоять из букв. Имена идентификаторов чувствительны к регистру. Переменные могут хранить значения разных типов, называемых типами данных.

Операторы

Операторы – это некий функционал, производящий какие-либо действия, который может быть представлен в виде символов, как например +, или специальных зарезервированных слов.

Список основных операторов в Python:

Оператор	Название	Объяснение
+	Сложение	Суммирует два объекта. В случае строк – конкатенация строк
-	Вычитание	Дает разность двух чисел. Если первый операнд отсутствует, то он считается равным 0
*	Умножение	Произведение двух объектов. В случае строк – повторение строки заданное кол-во раз
**	Степень	Возведение x в степень y ($x^{**}y$)
/	Деление	Возвращает частное от деления x на y (x / y)
//	Целочисленное деление	Возвращает неполное частное от деления
%	Деление по модулю	Возвращает остаток от деления
<<	Сдвиг влево	Побитовый сдвиг влево на заданное кол-во бит
>>	Сдвиг вправо	Побитовый сдвиг вправо на заданное кол-во бит
&	Побитовое И	
	Побитовое ИЛИ	
^	Побитовое ИСКЛЮЧТЕЛЬНО ИЛИ	
~	Побитовое НЕ	
<	Меньше	
>	Больше	
<=	Меньше или равно	
>=	Больше или равно	
==	Равно	
!=	Не равно	
not	Логическое НЕ	
and	Логическое И	
or	Логическое ИЛИ	

Структуры данных

Структуры данных – это, по сути, и есть структуры, которые могут хранить некоторые данные вместе. Другими словами, они используются для хранения связанных данных.

В Python существуют четыре встроенных структуры данных:

- Список – это структура данных, которая содержит упорядоченный набор элементов, т.е. хранит последовательность элементов. Список элементов должен быть заключён в квадратные скобки, чтобы Python понял, что это список. Как только список создан, можно добавлять, удалять или искать элементы в нём. Поскольку элементы можно добавлять и удалять, мы говорим, что список – это мутабельный тип данных, т.е. его можно модифицировать.
- Кортеж – служит для хранения нескольких объектов вместе. Их можно рассматривать как аналог списков, но без такой обширной функциональности, которую предоставляет список. Одна из важнейших особенностей кортежей заключается в том, что они иммутабельны, так же, как и строки. Т.е. модифицировать кортежи невозможно. Кортежи обозначаются указанием элементов, разделённых запятыми; по желанию их можно ещё заключить в круглые скобки. Кортежи обычно используются в тех случаях, когда оператор или пользовательская функция должны наверняка знать, что набор значений, т.е. кортеж значений, не изменится.
- Словарь – это некий аналог адресной книги, в которой можно найти адрес или контактную информацию о человеке, зная лишь его имя; т.е. некоторые ключи (имена) связаны со значениями (информацией). Заметьте, что ключ должен быть уникальным – вы ведь не сможете получить корректную информацию, если у вас записаны два человека с полностью одинаковыми именами. Обратите также внимание на то, что в словарях в качестве ключей могут использоваться только неизменяемые объекты (как строки). Пары ключ-значение указываются в словаре следующим образом: “ d = {key1 : value1, key2 : value2 } ”. Обратите внимание, что ключ и значение разделяются двоеточием, а пары друг от друга отделяются запятыми, а затем всё это заключается в фигурные скобки.
- Множества – это неупорядоченные наборы простых объектов. Они необходимы тогда, когда присутствие объекта в наборе важнее порядка или того, сколько раз данный объект там встречается. Используя множества, можно осуществлять проверку принадлежности, определять, является ли данное множество подмножеством другого множества, находить пересечения множеств и так далее. Для создания множества используется функция `set` в качестве аргумента для которой можно передать список, кортеж или словарь. В случае словаря, будет создано множество из ключей.

Пример создания различных структур данных:

```
l = [1, 2.4, [1, 'a']]
print(l)
c = 1, 2.4, (1, 'a')
print(c)
d = {'1': 1, 2: 2.4, "Three": [1, 'a']}
print(d)
s1 = set(l[0:2])
print(s1)
s2 = set(c)
print(s2)
s3 = set(d)
print(s3)
```

Списки, кортежи и строки являются примерами последовательностей. Основные возможности—это проверка принадлежности (т.е. выражения “ in ”и “ not in ”) и оператор индексирования, позволяющий получить на прямую некоторый элемент последовательности:

```
l = [1, 2.4, [1, 'a']]
print(2.4 in l)
print(3 in l)
print([1, 'b'] not in l)
```

Все три типа последовательностей, упоминавшиеся выше (списки, кортежи и строки), также предоставляют операцию получения вырезки, которая позволяет получить вырезку последовательности, т.е. её фрагмент:

```
shoplist = ['яблоки', 'манго', 'морковь', 'бананы']
name = 'swaroop'
# Операция индексирования
print('Элемент 0 -', shoplist[0])
print('Элемент 1 -', shoplist[1])
print('Элемент 2 -', shoplist[2])
print('Элемент 3 -', shoplist[3])
print('Элемент -1 -', shoplist[-1])
print('Элемент -2 -', shoplist[-2])
print('Символ 0 -', name[0])
# Вырезка из списка
print('Элементы с 1 по 3:', shoplist[1:3])
print('Элементы с 2 до конца:', shoplist[2:])
print('Элементы с 1 по -1:', shoplist[1:-1])
print('Элементы от начала до конца:', shoplist[:])
# Вырезка из строки
print('Символы с 1 по 3:', name[1:3])
print('Символы с 2 до конца:', name[2:])
print('Символы с 1 до -1:', name[1:-1])
print('Символы от начала до конца:', name[:])
```

Синтаксис Python

В Python есть разделение на логические и физические строки.

Физическая строка – это то, что вы видите, когда набираете программу. Логическая строка – это то, что Python видит как единое предложение. Python неявно предполагает, что каждой физической строке соответствует логическая строка. Примером логической строки может служить предложение `print('Привет, Мир!')` – если оно на одной строке (как вы видите это в редакторе), то эта строка также соответствует физической строке.

Чтобы записать более одной логической строки на одной физической строке, необходимо явно указать это при помощи точки с запятой (;), которая отмечает конец логической строки/предложения. Пример разных способов записи одного и того же фрагмента кода:

```
i = 5
print(i)
#То же самое, что
i = 5;
print(i);
#может быть записано как
i = 5; print(i);
#или как
i = 5; print(i)
```

Настоятельно рекомендуется придерживаться написания одной логической строки в одной физической строке. В таких случаях, нет необходимости писать точку с запятой, а также это упрощает чтение и понимание кода.

В Python пробелы важны. Точнее, пробелы в начале строки важны. Это называется отступами. Передние отступы (пробелы и табуляции) в начале логической строки используются для определения уровня отступа логической строки, который, в свою очередь, используется для группировки предложений. Это означает, что предложения, идущие вместе, должны иметь одинаковый отступ. Каждый такой набор предложений называется блоком. Далее будет рассмотрено, для чего нужны блоки.

Оператор if

Оператор if используется для проверки условий: если условие верно, выполняется блок выражений (называемый «if-блок»), иначе выполняется другой блок выражений (называемый «else-блок»). Блок «else» является необязательным. Пример оператора if:

```
number = 23
guess = int(input('Введите целое число : '))
if guess == number:
    print('Поздравляю, вы угадали,') # Здесь начинается новый блок
    print('(хотя и не выиграли никакого приза!') # Здесь заканчивается новый
блок
elif guess < number:
    print('Нет, загаданное число немного больше этого.') # Ещё один блок
# Внутри блока вы можете выполнять всё, что угодно ...
else:
    print('Нет, загаданное число немного меньше этого.')
# чтобы попасть сюда, guess должно быть больше, чем number
print('Завершено')
# Это последнее выражение выполняется всегда после выполнения оператора if
```

В Python нет оператора switch. Однако, при помощи конструкции if..elif..else можно достичь того же самого.

Оператор while

Оператор while позволяет многократно выполнять блок команд до тех пор, пока выполняется некоторое условие. Это один из так называемых операторов цикла. Он также может иметь необязательный пункт else . Пример использования оператора while:

```
number = 23
running = True
while running:
```



```

guess = int(input('Введите целое число : '))
if guess == number:
    print('Поздравляю, вы угадали.')
    running = False # это останавливает цикл while
elif guess < number:
    print('Нет, загаданное число немного больше этого')
else:
    print('Нет, загаданное число немного меньше этого.')
else:
    print('Цикл while закончен.')
# Здесь можете выполнить всё что вам ещё нужно
print('Завершение.')

```

Оператор for

Оператор `for..in` также является оператором цикла, который осуществляет итерацию по последовательности объектов, т.е. проходит через каждый элемент в последовательности. Пример оператора `for`:

```

l = [1,2,3,4,5,6,7,8,9]

for i in range(0,9):
    if l[i] % 2 == 1:
        print(l[i] ** 2)

```

В программе выше выводятся квадраты нечетных чисел от 1 до 9. Данное решение является неоптимальным, так как сначала генерируется список индексов от 0 до 8, а затем в блоке цикла происходит обращение к элементу списка по индексу. Но цикл `for` позволяет итерироваться по последовательностям сразу получая элементам:

```

l = [1,2,3,4,5,6,7,8,9]

for i in l:
    if i % 2 == 1:
        print(i ** 2)

```

В данном случае, мы сразу указываем, что переменная *i* является элементом списка и уже нет необходимо обращаться по индексу.

Также как и для оператора `while`, оператор `for` может содержать `else`-блок. И как в языках C/C++ циклы могут содержать операторы *break* (остановка цикла) и *continue* (переход к следующей итерации).

Функции

Функции определяются при помощи зарезервированного слова `def`. После этого слова указывается имя функции, за которым следует пара скобок, в которых можно указать имена некоторых переменных, и заключительное двоеточие в конце строки. Далее следует блок команд, составляющих функцию.

Параметры указываются в скобках при объявлении функции и разделяются запятыми. Пример простой функции:

```
def printMax(a, b):
    if a > b:
        print(a, 'максимально')
    elif a == b:
        print(a, 'равно', b)
    else:
        print(b, 'максимально')

printMax(3, 4) # прямая передача значений

x = 5
y = 7
printMax(x, y) # передача переменных в качестве аргументов
```

Зачастую часть параметров функций могут быть необязательными, и для них будут использоваться некоторые заданные значения по умолчанию, если пользователь не укажет собственных. Этого можно достичь с помощью значений аргументов по умолчанию. Их можно указать, добавив к имени параметра в определении функции оператор присваивания (=) с последующим значением. Значение по умолчанию должно быть константой.

Значениями по умолчанию могут быть снабжены только параметры, находящиеся в конце списка параметров. Таким образом, в списке параметров функции параметр со значением по умолчанию не может предшествовать параметру без значения по умолчанию:

```
def say(message, times = 1):
    print(message * times)

say('Привет')
say('Мир', 5)
```

Если имеется некоторая функция с большим числом параметров, и при её вызове требуется указать только некоторые из них, значения этих параметров могут задаваться по их имени – это называется ключевые параметры. В этом случае для передачи аргументов функции используется имя (ключ) вместо позиции (как было до сих пор):

```
def func(a, b=5, c=10):
    print('a равно', a, ', b равно', b, ', a с равно', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Иногда бывает нужно определить функцию, способную принимать любое число параметров. Этого можно достичь при помощи звёздочек. Когда мы объявляем параметр со звёздочкой (например, *param), все позиционные аргументы начиная с этой позиции и до конца будут собраны в кортеж под именем param. Аналогично, когда мы объявляем параметры с двумя звёздочками (**param), все ключевые аргументы начиная с этой позиции и до конца будут собраны в словарь под именем param:

```
def total(initial=5, *numbers, **keywords):
    count = initial
    for number in numbers:
        count += number
    for key in keywords:
        count += keywords[key]
    return count

print(total(10, 1, 2, 3, vegetables=50, fruits=100))
```

Если некоторые ключевые параметры должны быть доступны только по ключу, а не как позиционные аргументы, их можно объявить после параметра со звёздочкой. Если вам нужны аргументы, передаваемые только по ключу, но не нужен параметр со звёздочкой, то можно просто указать одну звёздочку без указания.

Для возвращения результата из функции используется оператор `return`. Также, обратим внимание, что при объявлении функции не указывается возвращаемый тип, так как язык Python является языком с динамической типизацией.

Установка и подключение библиотек(модулей) (pip3)

Для того, чтобы подключать модули (стандартные или сторонние), используется команда `import` с указанием имени модуля:

```
import math

a = math.sin(5)
b = math.cos(5)
```

Как видно, для того, чтобы обращаться к идентификаторам из модуля, необходимо указывать имя модуля и через точку указать имя необходимо идентификатора.

Иногда, имена модулей могут быть очень длинными и, чтобы всё время не прописывать длинное имя модуля, можно указать псевдоним. Пример того как задавать псевдонимы:

```
import math as m

a = m.sin(5)
b = m.cos(5)
```

Для того, чтобы устанавливать сторонние модули, используется утилита `pip3`, которая поставляется вместе с интерпретатором `python`. Например, для того чтобы установить библиотеку `Keras`, в терминале необходимо ввести команду:

```
pip3 install keras
```

После чего, начнется скачивание и установка модуля. После завершения, установленную библиотеку можно подключать командой `import`. При установке модулей также происходит всех необходимых дополнительных модулей. Например, при установке модуля `Keras` также будет скачан модуль `NumPy`.

Для просмотра всех установленных библиотек используется команда:

Библиотека Keras

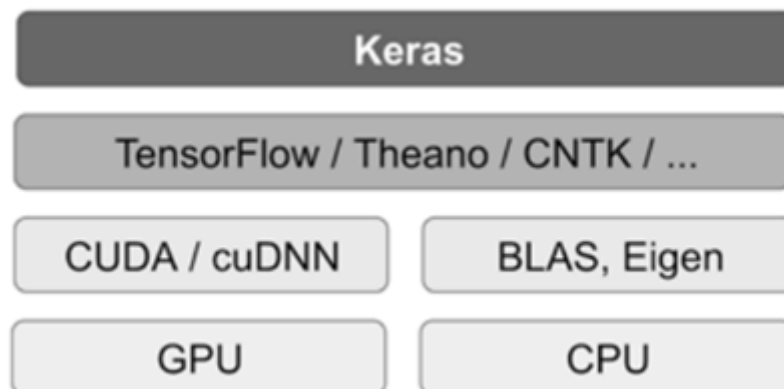
Keras — это фреймворк поддержки глубокого обучения для Python, обеспечивающий удобный способ создания и обучения практически любых моделей глубокого обучения.

Официальный сайт фреймворка Keras: keras.io

Keras обладает следующими ключевыми характеристиками:

- позволяет выполнять один и тот же код на CPU или GPU
- имеет дружелюбный API, упрощающий разработку прототипов моделей глубокого обучения
- включает в себя встроенную поддержку сверточных сетей (для распознавания образов), рекуррентных сетей (для обработки последовательностей) и всевозможных их комбинаций
- включает в себя поддержку произвольных сетевых архитектур

Keras — это библиотека уровня модели, предоставляющая высокоуровневые строительные блоки для конструирования моделей глубокого обучения. Она не реализует низкоуровневые операции, такие как манипуляции с тензорами и дифференцирование, — для этого используется специализированная и оптимизированная библиотека поддержки тензоров. При этом Keras не полагается на какую-то одну библиотеку поддержки тензоров, а использует модульный подход:



К фреймворку Keras можно подключить несколько разных низкоуровневых библиотек.

TensorFlow, CNTK и Theano — это одни из ведущих платформ глубокого обучения в настоящее время.

Любой код, использующий Keras, можно запускать с любой из этих библиотек без необходимости менять что-либо в коде: вы можете легко переключаться между ними в процессе разработки, что часто оказывается полезно, например, если одна из библиотек показывает более высокую производительность при решении данной конкретной задачи.

Вот как выглядит типичный процесс использования Keras:

1. Определяются обучающие данные: входные и целевые тензоры.
2. Определяются слои сети (модель), отображающие входные данные в целевые.
3. Настраивается процесс обучения выбором функции потерь, оптимизатора и некоторых параметров для мониторинга.
4. Выполняются итерации по обучающим данным вызовом метода `fit()` модели.