

# Искусственные нейронные сети: основы практического применения

Лекция 0

Крощенко А.А.

Брестский государственный технический университет

30.05.2017

# Язык программирования Python























Python – **высокоуровневый** язык программирования, в последнее время все чаще используемый для решения задач **машинного обучения** в силу своей **простоты, расширяемости и переносимости**. Кроме этого Python является **мультипарадигменным**. Это означает, что он поддерживает различные технологии программирования, позволяя писать программы в объектно-ориентированном, классическом процедурном и даже функциональном стиле.

Python является преимущественно **скриптовым языком**, а это означает, что программы, написанные на Python выполняются последовательно строка за строкой специальной программой, называемой **интерпретатором**.

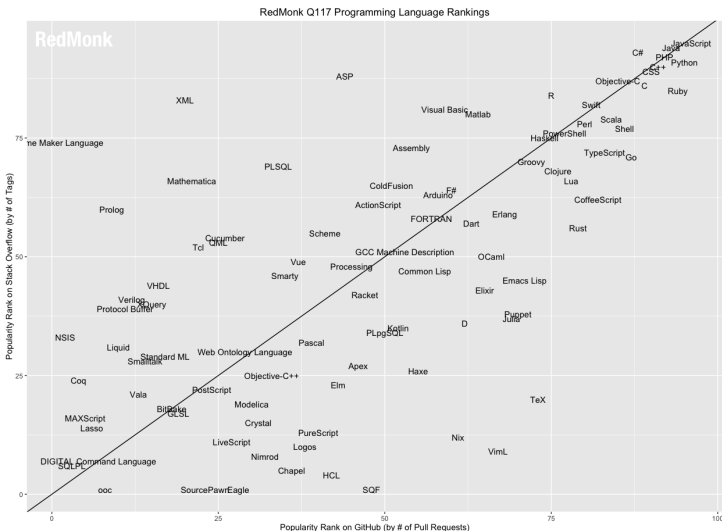
Ряд зарубежных ресурсов ставит язык Python в топ самых популярных языков программирования современности. Например, TechWorm (<https://www.techworm.net/>) – 5 место, Forbes (<https://www.forbes.com>) – 1 место, CodeJow – 6 место (<http://codejow.com>).

Некоторые рекомендуют Python как **первый** язык программирования (Freelancer – <https://www.freelancer.com>)

# IEEE Spectrum: The Top Programming Languages 2016

Language Rank	Types	Spectrum Ranking
1. C	  	100.0
2. Java	  	98.1
3. Python	 	97.9
4. C++	  	95.8
5. R		87.7
6. C#	  	86.4
7. PHP		82.4
8. JavaScript	 	81.9
9. Ruby	 	74.0
10. Go	 	71.5

# The RedMonk Programming Language Rankings: January 2017



# Сферы применения

- Научные вычисления (вычислительная математика, статистика, машинное обучение и др.)
- Веб-разработка (фреймворки, поддержка разнообразных протоколов)
- Образование
- Desktopные решения
- Тестирование и управление разработкой ПО
- Инженерия и робототехника

# Где используется Python?

- Значительная часть Youtube реализована с использованием этого языка
- Python используется как рабочий язык при разработке роботизированных решений на платформе Raspberry Pi
- Используется Intel, Cisco, Hewlett-Packard, IBM и др. компаниями для тестирования аппаратного обеспечения
- Nasa, Fermilab, Los Alamos используют для научных вычислений
- На Python написаны такие известные программы как BitTorrent, Blender, DropBox, протоколы многопользовательских игр (World Of Tanks).
- Google Inc. использует этот язык в своей поисковой системе
- Python используется людьми различных профессий, не связанных с программированием для решения своих задач.

# Характеристики языка

- Python - язык с нестрогой (динамической) типизацией. Это означает, что одна и та же переменная в разные моменты времени может хранить значения разных типов
- Интерпретируемый (программные конструкции выполняются последовательно, транслируясь в промежуточную форму – байт-код)
- Объектно-ориентированный
- Расширяемый
- Портiruемый (переносимый)
- Простой
- Бесплатный

# Почему Python используется для машинного обучения (критичные характеристики)?

- ❶ **Простота** – язык Python намного проще и компактнее языка Си или Си++. Программы на питоне отличаются понятностью и отсутствием ряда привычных для Си-подобных языков элементов (например, в Python нет открывающихся и закрывающихся фигурных скобок, выделяющих блоки кода). Здесь многое завязано на правильном форматировании. Роль операторных блоков теперь полностью перенесена на отступы, позволяющие определять, где находится тело функции, цикла и т.д. Так язык помогает выработать чувство стиля.
- ❷ **Расширяемость**. Питон бы не смог конкурировать с другими высокоуровневыми языками, если бы не большое количество готовых решений, написанных для этого языка. Пакеты расширяют возможности языка, делая его пригодным для высокопроизводительных вычислений.
- ❸ **Переносимость**. Для Питона существуют свои версии интерпретатора для каждой операционной системы, что позволяет написанный однажды скрипт на одной системе перенести на другую и получить тот же результат.



# Простейшая программа на Python

```
print "Hello, world!"
```

# Синтаксические конструкции языка

- Оператор присваивания: `<имя_переменной> = <значение>`

```
a = 12
```

- Операторы ветвления: `if else` или `if elif`

```
if age > 18:  
    print 'adult'  
else:  
    print 'baby'  
if age > 30:  
    print 'adulthood'  
elif age < 16:  
    print 'childhood'  
else:  
    print 'adolescence'
```

```
A = Y if X else Z
```

# Синтаксические конструкции языка

- Операторы цикла: **while** или **for**

```
while <test1>:  
    <statements1>  
    if <test2>: break  
    if <test3>: continue  
else:  
    <statements2>
```

**break** – выход из цикла, **continue** – продолжить цикл с новой итерации, блок кода после ключевого слова **else** выполняется по завершении цикла, если не был выполнен выход через **break**.

```
while a < 10:  
    a = a + 1
```

```
while True:  
    if exitTest():  
        break
```

# Синтаксические конструкции языка

Цикл **for** имеет семантику цикла **foreach**, т.е. осуществляет последовательный перебор элементов заранее определенной коллекции.

```
for <target> in <object>:  
    <statements>  
else:  
    <statements>
```

```
for i in range(0, 15):  
    print i
```

# Функции и их назначение

Элементарными структурными единицами программы на Python являются **функции**. Они позволяют создавать универсальные блоки кода, которые могут использоваться многократно. Функция имеет простой синтаксис:

```
def <имя_функции>(<параметр1>, <параметр2>, ...):  
    <тело_функции>  
    return <возвращаемое_значение>
```

Пример:

```
def threshold_func(x):  
    if x > 0:  
        return 1  
    else:  
        return -1
```

В Python можно создавать функции-заглушки, которые ничего не делают:

```
def threshold_func(x):  
    pass
```

# Пример полезной функции

```
import httpplib2

def load_something(url, path_to_save, count_images):
    h = httpplib2.Http('.cache')
    if not url[-1] == '/':
        url = url + '/'
    if not path_to_save[-1] == '/':
        path_to_save = path_to_save + '/'
    for i in range(1, count_images):
        response, content = h.request(url + str(i) + '.jpg')
        out = open(path_to_save + str(i) + ".jpg", "wb")
        out.write(content)
        out.close()
    print "Image " + str(i) + " is loaded..."
```

# Основные типы данных

- **Число** (1, 1.2, 3+4j)
- **Строка** – структура данных, представляющая собой последовательность символов. Строка – **неизменяемая** структура данных, т.е. ее элементы не могут быть изменены. Строки можно конкатенировать.

```
T = "Hello"  
print len(T) # 5  
print T[-1] # o  
print T[0] # H
```

- **Кортеж** – структура данных, представляющая собой последовательность неизменяемых элементов. Литерал кортежа заключается в круглые скобки.

```
T = (1, 2, 3, 4, 5)  
print len(T) # 5  
print T[1] # 2  
T = (1, 2, 3)  
E = (4, 5)  
print T + E # (1, 2, 3, 4, 5)
```

Кортежи можно конкатенировать и они допускают возможность вложения. Могут хранить данные разных типов.

# Основные типы данных

- Список – структура данных, представляющая собой последовательность изменяемых элементов. Литерал списка заключается в квадратные скобки.

```
T = [1, 2, 5, [6, 7]]
print len(T)      # 4
print T[0]        # 1
print T[0:2]      # [1, 2]
```

Списки можно конкатенировать и они также допускают возможность вложения как и кортежи. Могут хранить данные разных типов.

- Словарь – это структура, доступ к элементам которой осуществляется по ключам.

```
D = {'max_epochs': 10, 'min_error': 1e-5, 'minibatch': 100}
print D['max_epochs']+1
```

- Файл

Все типы Python имеют методы, специфичные для своего типа. Например, метод `append` для списка.



# Файловый ввод и вывод

<файловый дескриптор> = open(<путь\_к\_файлу>, <режим\_доступа>)  
<режим\_доступа>: "w", "r", "r+", "a"

## Чтение:

```
f = open("file.txt", 'r')
for _str in f:
    print _str
f.close()
```

```
with open("file.txt", 'r') as f:
    for _str in f:
        print _str
```

## Запись:

```
f = open("file.txt", 'w')
f.write("String\n")
f.write("Another string")
f.close()
```

# Классы

**Классы** – основа парадигмы **ООП**. Представляют собой шаблоны для создания объектов. Объекты могут представлять как существующие предметы реального мира, так и абстрактные понятия.

Классы являются **типами**, определяемыми пользователями. Они могут **инкапсулировать (содержать)** в себе данные (поля класса), представляемые переменными любых типов, а также **функции**, которые обрабатывают эти данные.

Определение простейшего класса выглядит следующим образом:

```
class <имя_класса>:  
    def <имя_функции1>(self, <список_параметров>):  
        <тело_функции>  
    ...
```

К данным объекта класса можно получить доступ с помощью ссылки **self**. Особая роль среди методов класса отведена **конструктору**. Этот метод имеет фиксированное описание и предназначен для конструирования объектов класса. Хорошим тоном является инициализация полей класса только внутри конструктора.

# Классы

```
class Person:
    def __init__(self, name, lastName, age):
        self.name = name
        self.lastName = lastName
        self.age = age

    def printInfoAbout(self):
        print lastName + " " + name + ": " + str(age) + " years old"
```

```
person = Person("Vasya", "Pupkin", 20)
person.printInfoAbout() # Pupkin Vasya: 20 years old
```

```
class Person:
    @staticmethod
    def printInfoAbout(person):
        print person.lastName + " " + person.name + ": " + str(person.
            age) + " years old"
```

```
person = Person("Vasya", "Pupkin", 20)
Person.printInfoAbout(person) # Pupkin Vasya: 20 years old
```

# Классы: статические методы

Методы могут быть статическими. Такие методы вызываются от имени всего класса, а не только конкретного объекта. Обозначаются директивой **@staticmethod**.

```
class Person:
    ...
    @staticmethod
    def printInfoAbout(person):
        print person.lastName + " " + person.name + ": " + str(person.
            age) + " years old"

person = Person("Vasya", "Pupkin", 20)
Person.printInfoAbout(person) # Pupkin Vasya: 20 years old
```

# Использование классов

В рамках нейронных сетей классы могут представлять сущности, характерные для этой области науки. Например, класс Нейрон, НейронныйСлой, ФункцияАктивации и т.д.

```
class HebbNeuron:
    def __init__(self):
        self.w1 = 0
        self.w2 = 0
        self.T = 0

    def train(self, samples, targets):
        for sample, target in it.izip(samples, targets):
            self.w1 += sample[0] * target
            self.w2 += sample[1] * target
            self.T += target
```

# ОО-программирование. Три принципа ООП

- **Инкапсуляция** – сокрытие данных и методов внутри класса
- **Наследование** – возможность расширения функциональности класса, придания ему особых свойств
- **Полиморфизм** – объекты каждого класса, сходящего в иерархию, несмотря на родственные отношения, ведут себя по-разному. Например, все транспортные средства перемещаются, но каждое делает это по-своему.

```
class Function:
    def apply(self, x):
        return NotImplemented

    def applyD(self, x):
        return NotImplemented

class Logistic(Function):
    def apply(self, x):
        return 1.0 / (1 + np.exp(-x))

    def applyD(self, x):
        return x * (1 - x)
```

# Модули Python

Модули Python служат для упорядочения исходного кода в целях его **повторного использования**. Модулем является **любой файл с программой на языке Python**.

Например, функции обучения нейронной сети одного определенного типа можно поместить в такой модуль и использовать их по мере необходимости. Кроме того, существует много уже созданных модулей, используемых при решении различных задач (в том числе и машинного обучения).

Подключение модулей к программе выполняется следующей командой:

```
import <имя_модуля>
```

При этом, если необходимо импортировать определенный класс, используется инструкция

```
import <имя_модуля>.<имя_класса>
```

```
import math
```

```
print math.e # 2.718281828459045
```

Можно использовать имя-псевдоним:

```
import backpropagation as bp
```

# Модули

Можно подключить лишь определенные атрибуты:

```
from math import sin
```

Или все:

```
from math import *
```

В этом случае префикс **math** уже не требуется:

```
from math import *
```

```
print sin(1.25) # 0.948984619356
```

Можно использовать модуль в качестве самостоятельной программы. Для этого в текст включается дополнительная строка:

```
if __name__ == "__main__":  
    mainFunction()
```



# Пакеты, используемые в курсе

Пакеты – объединение нескольких модулей

- **NumPy** – пакет, в котором реализованы функции для работы с многомерными массивами
- **matplotlib** – пакет, предоставляющий продвинутые возможности для построения графиков
- **scikit-learn** – пакет, реализующий некоторые алгоритмы машинного обучения
- **tensorflow** – пакет, представляющий фреймворк для автоматизации работы с различными архитектурами нейронных сетей. Средства пакета позволяют точно описать модель, а после этого запустить процесс обучения и тестирования в рамках т.н. сессии.
- **caffe** – пакет, также предоставляющий фреймворк для работы с нейросетями. Отличается от tensorflow другой организацией работы. В частности все настройки нейросети задаются в отдельном файле, данные хранятся в специализированных файлах с расширением hdf5.

# Numpy: многомерные массивы в Python

**Numpy** – специализированный пакет, в котором реализована структура данных массив (поддерживается разное количество измерений) и операции над ним.

- Позволяет создавать массивы из внутренних структур данных языка Python (списки, кортежи).
- Содержит большое количество готовых функций для работы с массивами (в частности, умножение, нахождение собственных значений, вычисление нормы, вычисления определителя, обратной матрицы, вычисление матричных функций...).
- Позволяет легко находить все основные статистики (максимум, минимум, среднее, стандартное отклонение,...)
- Поддерживается почти всеми современными фреймворками
- Позволяет сохранять и загружать массивы данных

```
array = np.array([1, 2, 3, 4, 5])  
weighted_sums = np.dot(data, self.weights) + self.biases
```

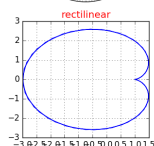
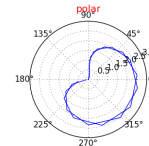
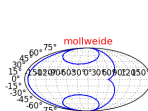
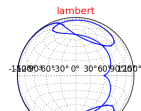
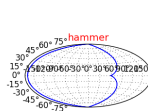
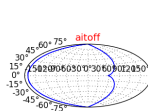
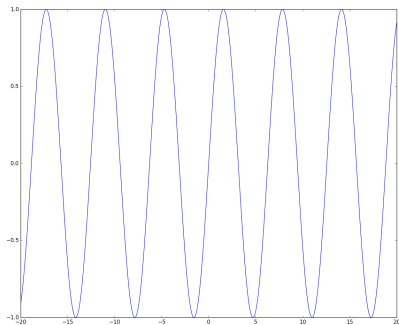
# Gnumpy: многомерные массивы с использованием GPU

Модуль Gnumpy позволяет использовать GPU для операций с многомерными массивами. Это позволяет существенно ускорить выполнение многих ресурсоемких операций (например, умножения матриц). Достоинством предложенного модуля является то, что он реализует интерфейс для работы с массивами, аналогичный используемому в numpy.

Фрагмент обучения RMB с использованием функций модуля gnumpy ([http://www.cs.toronto.edu/~tijmen/gnumpy\\_example.py](http://www.cs.toronto.edu/~tijmen/gnumpy_example.py)):

```
def test_gnumpy(num_epochs):
    import gnumpy as gpu
    dat = gpu.garray(load('mnist_cudaTest').T/255.)
    w_vh = 0.1 * gpu.randn(num_vis, num_hid)
    w_v = gpu.zeros(num_vis)
    w_h = -4. * gpu.ones(num_hid)
    wu_vh = gpu.zeros((num_vis, num_hid))
    wu_v = gpu.zeros(num_vis)
    wu_h = gpu.zeros(num_hid)
    for epoch in range(num_epochs):
        for batch in range(num_batches):
            v1 = dat[batch*batch_size : (batch + 1)*batch_size]
            h1 = (gpu.dot(v1, w_vh) + w_h).logistic()
            ...
```

# Matplotlib: построение графиков любой сложности



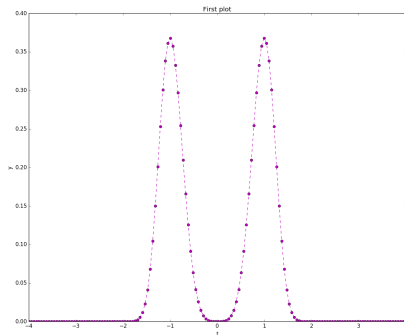
# Пример построения

```
from numpy import *  
import matplotlib.pyplot as  
    plt
```

```
t = linspace(-4, 4, 150)  
y = t**4*exp(-t**4)
```

```
plt.plot(t, y, 'm--o')  
plt.xlabel('t')  
plt.ylabel('y')  
plt.title('First plot')  
plt.legend()
```

```
plt.show()
```



# Scikit-learn: машинное обучение в Python

Scikit-learn позволяет решать множество задач машинного обучения:

- Регрессия
- Классификация
- Предобработка данных
- Кластеризация
- Понижение размерности...

Кроме этого, содержит встроенные средства для загрузки некоторых выборок, на которых можно проводить собственные исследования.

**Официальный сайт:** <http://scikit-learn.org>

# Дополнительная литература для изучения Python

- ❶ **М. Лутц** Изучаем Python
- ❷ **М. Саммерфилд** Программирование на Python 3. Подробное руководство
- ❸ **М. Лутц** Программирование на Python (в 2-х томах)