Лекция 1
Введение в ТП

Лектор:
Д.Н. Лавров

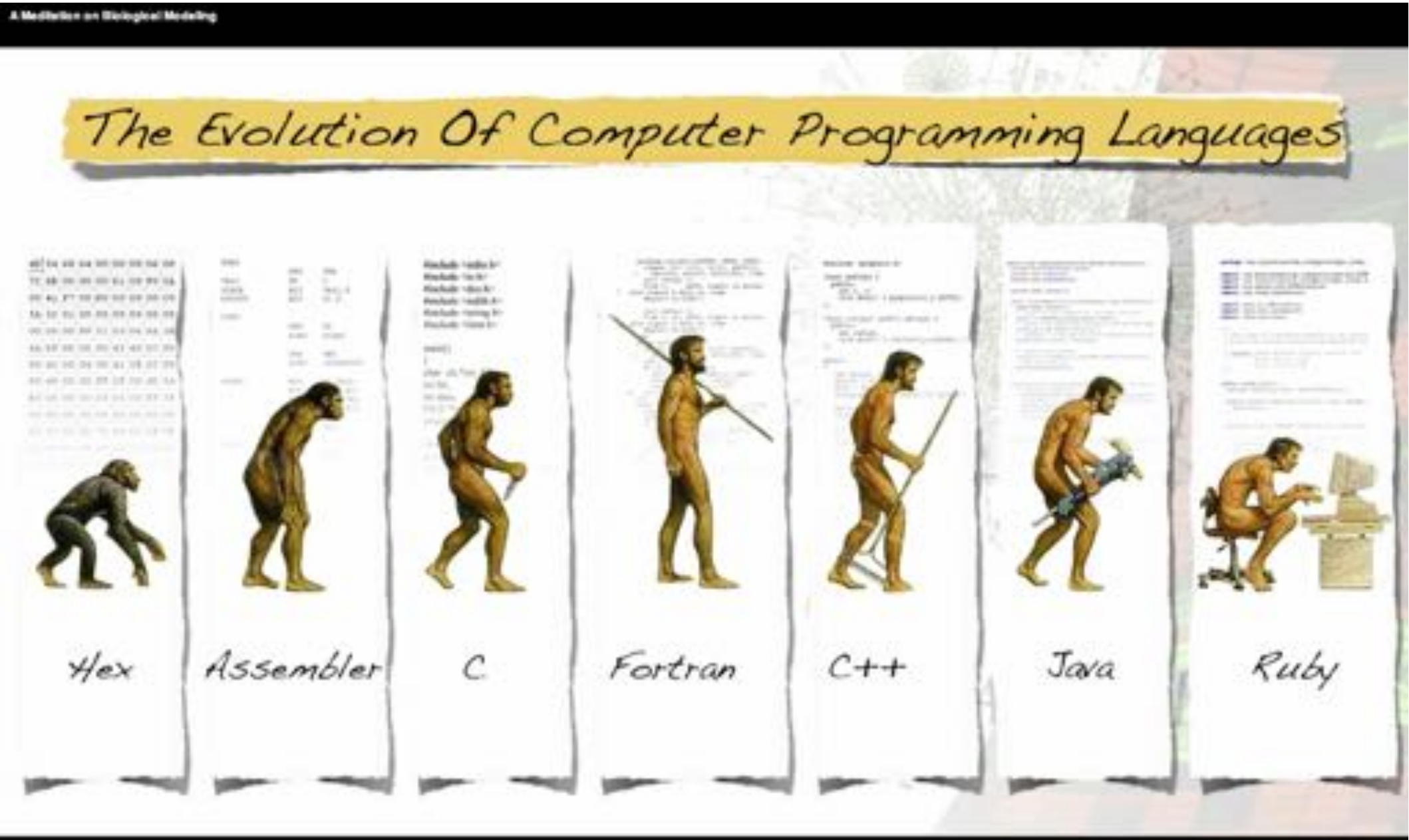
Задачи курса

- Знакомство с основными парадигмами программирования на примере Python 3
- Получение навыков программирования в каждой из парадигм

История языков программирования

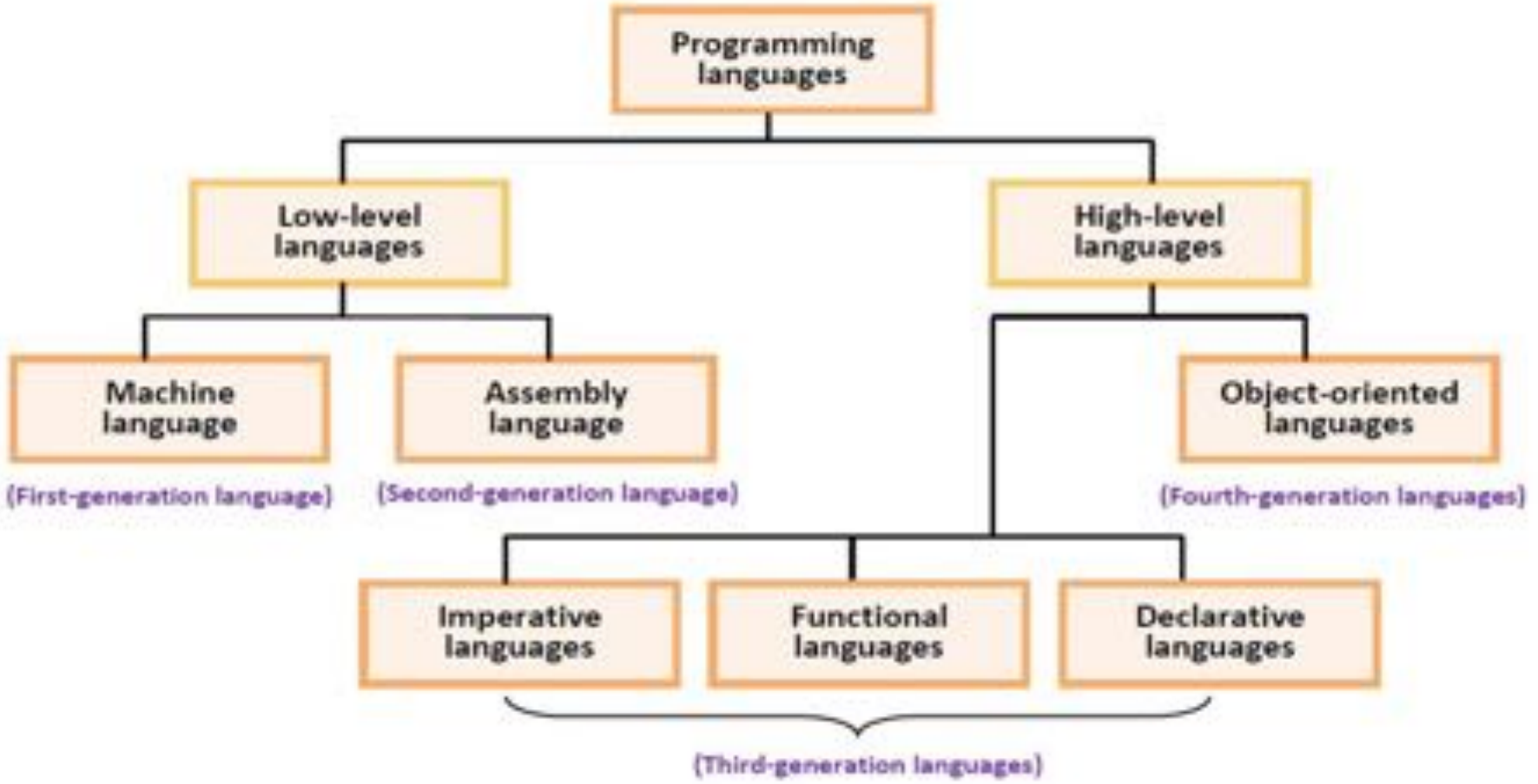
- История развития языков программирования
  - Машинные коды
  - Языки типа Ассемблер
  - Фортрнан (**For**mula **Tran**slation) (середина 50-х)
  - Лисп (функциональное прогр-е, 1958)
  - Кобол (экономические задачи, 1959)
  - Бейсик (1964)
  - Алгол-68 (победитель конкурса) и Паскаль (1968)
  - C (1972)
  - Пролог (логическое прогр-е, 1972)
  - МОДУЛА-2 (1978)
  - Ада (1979-80)
  - C++ (1983)
  - Objective-C (1983)
  - MatLab (1984)
  - Perl (1987)
  - Haskell (1990)
  - Python (1991)
  - Oberon-2 (1991)
  - R (1993)
  - PHP (1995)
  - JavaScript (1995)
  - JScript (1996)
  - Ruby (1995)
  - Java (1995)
  - C# (2000)
  - Scala (2003)
  - Python 3 (2008)
  - Go (2009)

Эволюция языков



Классификации языков

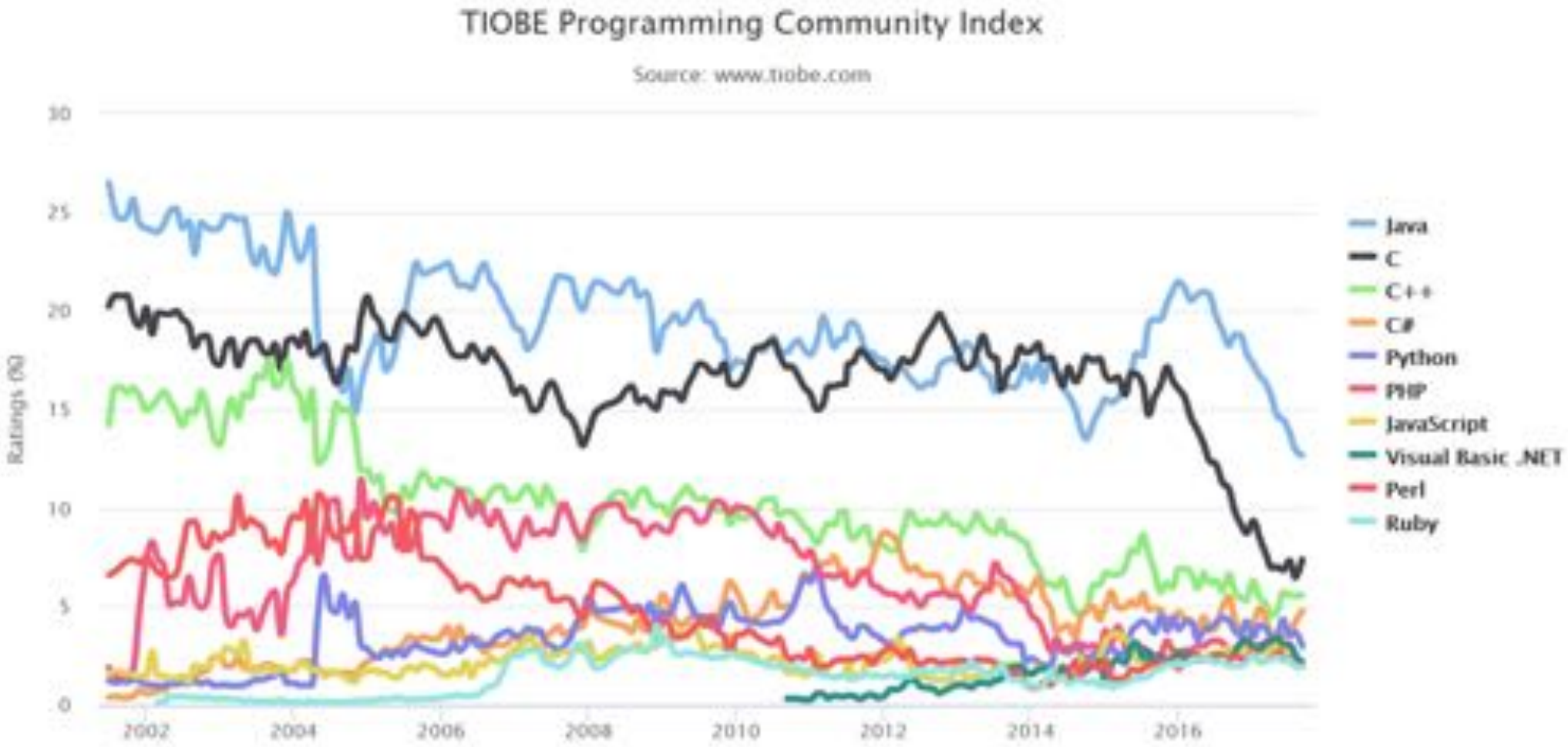
Классификации языков



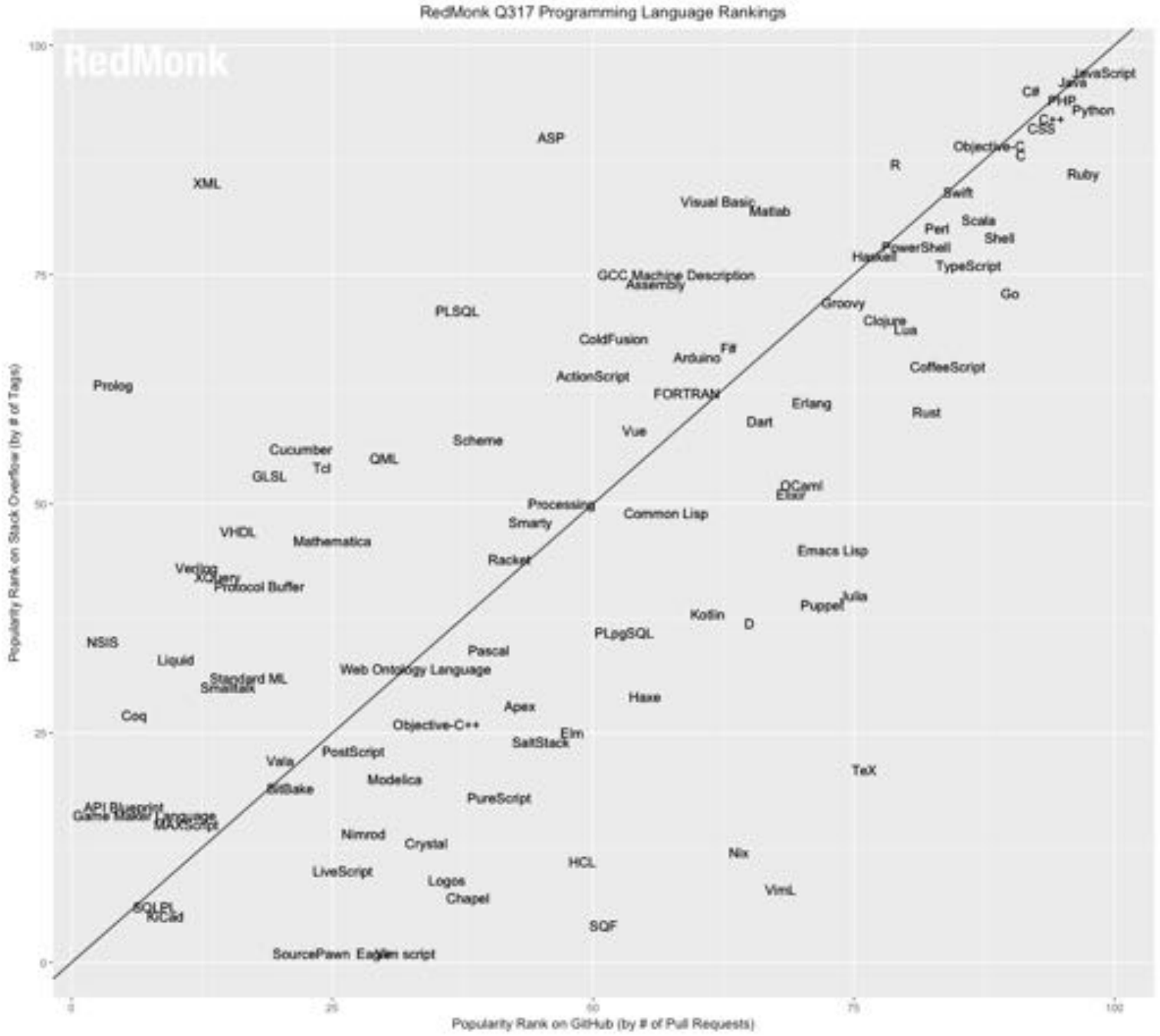
Основные парадигмы программирования

- **Парадигма программирования** — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером
- **Императивное** (процедурное) программирование
  - Программа набор действий
  - Переменные и присваивания
  - Подпрограммы
- **Объектно-ориентированное** программирование
  - Программа набор взаимодействующих объектов, в свою очередь объединенных в классы
- **Функциональное** программирование
  - Программа как композиция математических функций; в «чистом» языке нет переменных

Рейтинг TIOBE 2017



Рейтинг RedMonk



1 JavaScript

2 Java

3 Python

4 PHP

5 C#

6 C++

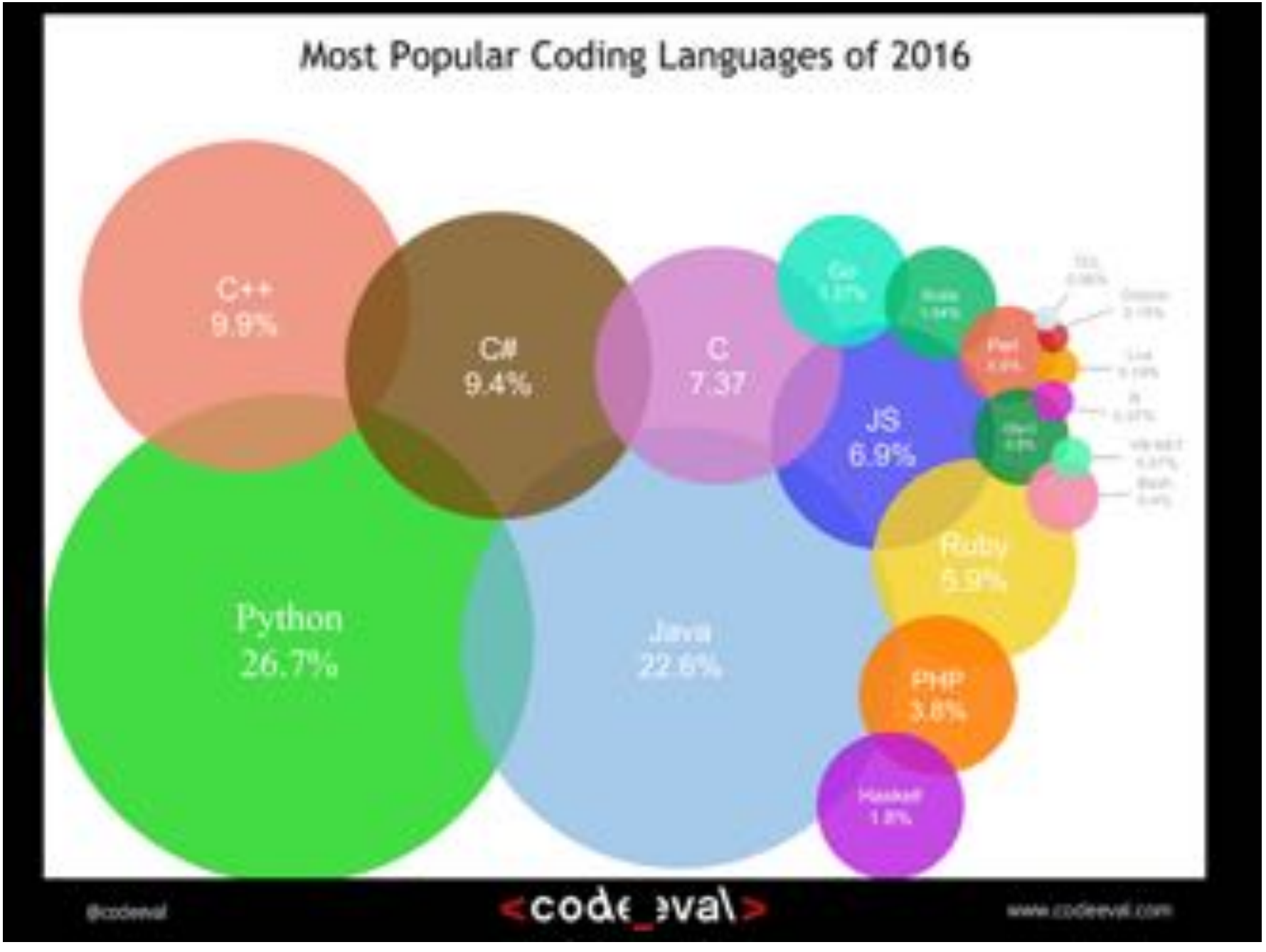
7 CSS

8 Ruby

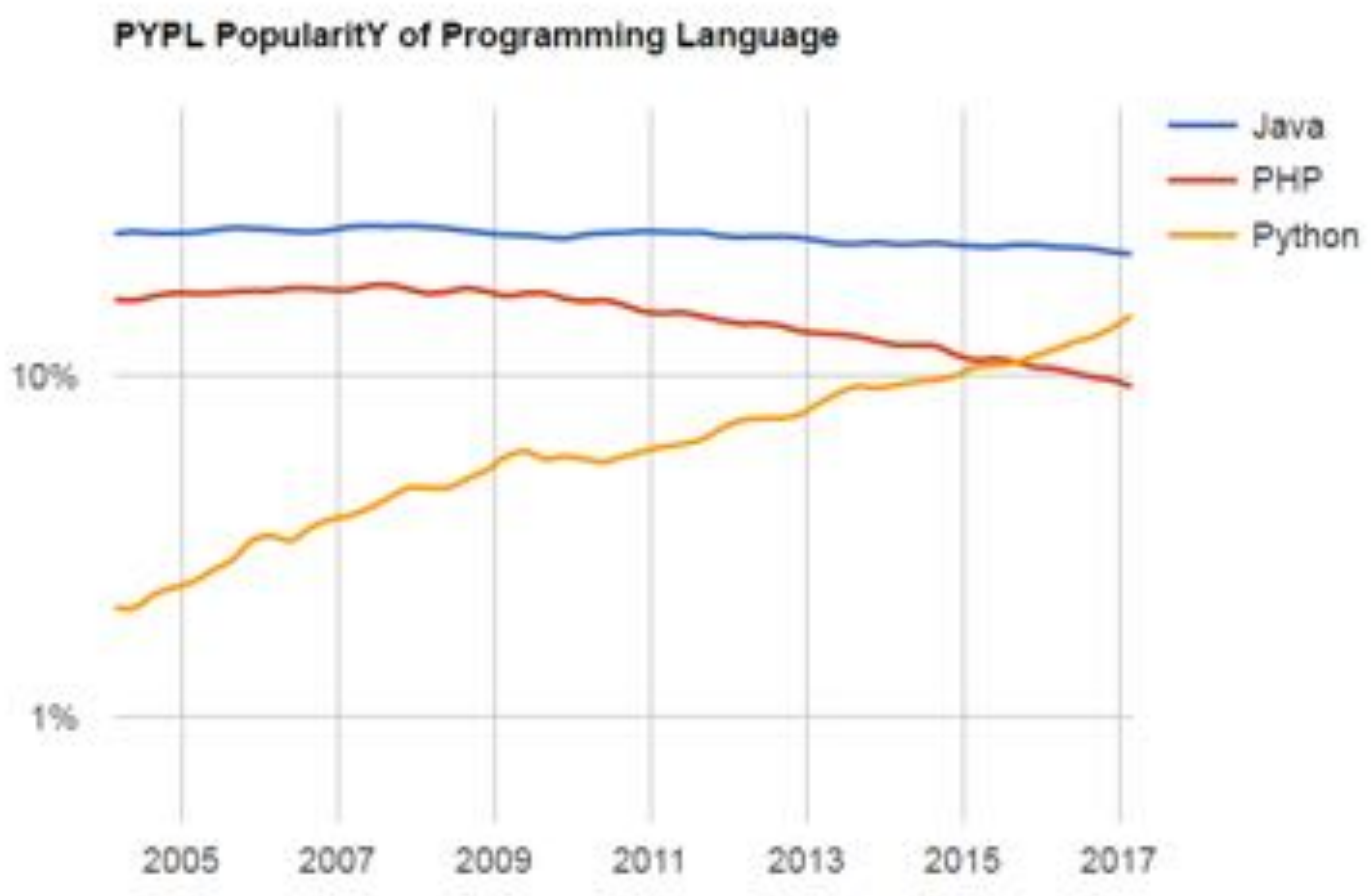
9 C

10 Objective-C

Рейтинг CodeEval



Рейтинг PYPL GitHub 2017



- Java
- Python
- PHP
- C#
- JavaScript
- C
- C++
- Objective-C
- R
- Swift

Рейтинг IEEE Spectrum 2017

Внутренние силы развития

- Борьба с возрастающей сложностью программных систем (проектов)
- Реализация концептуальных идей
- Адаптация под предметную область

Характеристики Python

- **Достоинства**
  - Динамическая строгая типизация
  - Многопарадигменный
  - Легкий вход
  - Встроенная реализация многих шаблонов проектирования
  - В Python 3 нет ограничения на целые
  - Встроенные списки, кортежи и словари
  - Интроспекция
  - Метапрограммирование
  - Большое количество библиотек под всевозможные нужды
  - Быстрая разработка
- **Недостатки**
  - Медленный (возможно частично исправить)
  - Многопоточность
  - Существование параллельно двух несовместимых веток Python 2 и Python 3
  - Запрет на модификацию встроенных классов
  - GIL (Глобальная блокировка интерпретатора) в части реализаций (н-р CPython), проблемы со скоростью выполнения потоков выполнения из-за синхронизации потоков.

Быстрый старт с Python 3

- Атомарные типы
- Арифметика
- Ссылочные (структурные) типы
- Основные алгоритмические конструкции
- Обработка исключений
- Подпрограммы и модули

Первая программа

- **Java**

```
public class HelloWorld {
  public static void main(String[] args)
  {
     System.out.println("Hello, World");
  }
}
```

- **Python**

```
print("Hello, World")
```

- **C++**

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main() {
 cout<<"Hello, world!"<<endl;
 return 0;
}
```

Простые типы данных

- int — целый тип,
- float — числовой тип с плавающей точкой,
- complex — комплексное число и
- str — строковый тип.

- Пример

```
>>> 5+7
12
>>> ’abc’+’cda’
’abccda’
>>> a=(1+1j)+(3+4j)
>>> a
(4+5j)
>>> a.real
4
>>> a.imag
5
```

- Строки могут быть представлены в четырёх видах:

```
>>> ’string’
’string’
>>> "string"
’string’
>>> ’’’string’’’
’string’
>>> """string"""
’string’
```

- Сравнения на равенство строк и целых чисел осуществляется с помощью оператора ==

Арифметика

- Отметим некоторые отличительные особенности отличающие Python от C/C++ и Java. Использование операций + , - , * аналогичны другим языкам программирования. Деление отличается, результатом деления с целочисленными аргументами будет число с плавающей точкой. Если нужно вычисление целого частного то используют // и, если остаток от деления, то %.
- +=, *= и т.п.
- Имеется оператор возведения в степень **

Строки

- Строки и числа являются неизменяемыми. Пример.

  >>> s[1]=’a’

  Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

  TypeError: ’str’ object does not support item

  assignment

Представление целых

- Ограничения на диапазон целых не накладывается и определяется только лишь наличием свободной оперативной памяти.

- Пример.

```
>>> 2**1024
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
```

Слайсинг

```
>>> a=’strin’
>>> a[0:3]
’str’
>>> a[3:]
’in’
>>> a[-1]
’n’
>>> len(a)
5
```

элементы	s	t	r	i	n
индексы	0 (-5)	1 (-4)	2 (-3)	3 (-2)	4 (-1)

Последовательность	a	b	c	d	e	f	g	**Результат слайсинга**
Индексы	0 (-7)	1 (-6)	2 (-5)	3 (-4)	4 (-3)	5 (-2)	6 (-1)	
[:] (→)	+	+	+	+	+	+	+	abcdefg
[::-1] (←)	+	+	+	+	+	+	+	gfedcba
[::2] (→)	+		+		+		+	aceg
[1::2] (→)		+		+		+		bdf
[:1]	+							a
[-1:]							+	g
[3:4]				+				d
[-3:] (→)					+	+	+	efg
[-3:1:-1] (←)			+	+	+			edc
[2:5] (→)			+	+	+			cde

Кортежи

- Кортежи — это неизменяемые последовательности. Создаются с помощью круглых скобок.
- **Пример**.

  >>> "Январь", "Февраль", "Март"

  (’Январь’, ’Февраль’, ’Март’)

  >>> ("Январь", "Февраль")

  (’Январь’, ’Февраль’)

  >>> ("Январь",)

  (’Январь’,)

  >>> ()
- ()

Списки

- Список – это изменяемая коллекция элементов
- Списки создаются аналогично с использование квадратных скобок:
- >>> [1, 3, 5]
- [1, 3, 5]
- >>> a=[1, 3, 5]
- >>> a[1]
- 3
- >>> a[1:3]
- [3, 5]

```
>>> a=[1, 3, 5]
>>> len(a)
3
>>> b=(4, "web", 6, 8)
>>> len(b)
4
```

Методы списка

- Основные методы

append() — добавляет элемент в список;

insert() — вставляет элемент в список;

sort() — сортирует список;

remove() — удаляет элемент из списка.

Работает и слайсинг!

- Пример

```
>>> a = [1, 3, 5, 1]
>>> a.append(8)
>>> a
[1, 3, 5, 1, 8]
>>> a.remove(1)
>>> a
[3, 5, 1, 8]
>>> a.remove(1)
>>> a
[3, 5, 8]
>>> a.insert(1,13)
>>> a
[3, 13, 5, 8]
>>>a.sort()
>>>a
[3, 5, 8, 13]
```

Словари

- Словари в Python – неупорядоченные коллекции произвольных объектов с доступом по ключу. Другие названия ассоциативные массивами или хеш-таблицы.

```
>>> d = {'dict': 1, 'dictionary': 2}
>>> d
{'dict': 1, 'dictionary': 2}
```

Пример работы со словарём

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d[1]
2
>>> d[4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d['1']
Traceback (most recent call last):
  File "", line 1, in
    d['1']
KeyError: '1'
```

Операторы проверки идентичности

Пример

```
>>> a = ["String", 3, None]
>>> b = ["String", 3, None]
>>> a is b
False
>>> a == b
True
>>> b = a
>>> a is b
True
>>> a == b
True
```

Продолжение примера:

```
>>> a
['String', 3, None]
>>> a[0]=6
>>> b
[6, 3, None]
```

```
>>> a = "Something"
>>> b = None
>>> a is not None, b is None
(True, True)
```

Операторы сравнения и принадлежности

```
>>> a>b, a>=b, a!=b, a<=b<=5
   (False, True, True, True)
>>> a>b, a<=b, a!=b, a<=5<=b
   (False, False, True, False)

>>> a = ["String", 3, None]
>>> 3 in a
   True
```

Логические операторы

```
>>> not ((5>7) or (2<4))
   False


>>> five = 5
>>> two = 2
>>> zero = 0
>>> five and two
    2
>>> two and five
   5
>>> five and zero
   0
```

Ветвления

```
if логическое_выражение_1:
    блок_операторов_1
elif логическое_выражение_2:
    блок_операторов_2
...
elif логическое_выражение_N:
    блок_операторов_N
else:
    блок_операторов_else
```

Пример. Вычисление знака числа.

```
>>> x=-3
>>> if x>0:
...     sign=1
...  elif x==0:
...     sign=0
...  else:
...     sign=-1
>>> sign
-1
```

Цикл while

while *лог_выр*:

    блок_операторов

```
>>> i=1
>>> p=1
>>> while i<=10:
        p=p*i
        print(i,"\t",p)
        i=i+1
```

- 1 1
- 2 2
- 3 6
- 4 24
- 5 120
- 6 720
- 7 5040
- 8 40320
- 9 362880
- 10 3628800

Цикл for

for *переменная* **in** *коллекция*:
*блок*_операторов

- **Пример**

>>> list=[2,5,4,2,4]

>>> for x in list:

... print(x)

2

5

4

2

4

- range(n) — представляет диапазон целых чисел от 0 до n-1 с шагом 1.
- range(a,b) — представляет диапазон целых чисел от a до b-1 с шагом 1.
- range(a,b,step) — представляет диапазон целых чисел от a до b-1 с шагом step. Шаг может принимать и отрицательные значения.
- **Пример**. Вычисление таблицы факториалов.

>>> p=1

>>> for i in range(7):

... p=p*(i+1)

... print(i+1,'\t',p)

1 1

2 2

3 6

4 24

5 120

6 720

7 5040

Обработка исключений

```
try:
    защищённый_блок_операторов
except исключение_1 as переменная_1:
    блок_обработки_исключения_1
...
except исключение_N as переменная_N:
    блок_обработки_исключения_N
else:
    блок_обработки_исключения_N_plus_1
finally:
    блок_обработки_исключения_N_plus_2
```

Обработка исключений

- **Пример**:

```
f = open('file.txt')
ints = []
try:
    for line in f:
        ints.append(int(line))
except ValueError:
    print('Это не число.')
except Exception:
    print('Неизвестная ошибка')
else:
    print('OK')
finally:
    f.close()
```

Ввод / вывод

```
print("Введите цело число, затем нажмите ENTER;  для завершения ввода ещё раз ENTER")
total = 0
count = 0
while True:
  line = input("Введите целое: ")
  if line:
    try:
       number = int(line)
    except ValueError as err:
       print(err)
       continue
    total += number
    count += 1
  else: break
  if count:
    print("count =", count, "total =", total, "mean =", total / count)
```

Подпрограммы

```
def имя_функции(аргументы):
    блок_операторов
    return возвращаемое значение
```

*Если явно не возвращать значение функции с помощью оператора **return**, то значение функции после выполнения будет **None**.*

Задача. Решение квадратного уравнения в R

Решение

```
import math # импортируем библиотеку
def solveSquareEquation(a, b, c):
    if a!=0:
        d=b**2-4*a*c # вычисляем дискриминант
        if d>0:
            d=math.sqrt(d)
            return ((-b-d)/(2*a), (-b+d)/(2*a)) # два корня
        elif d==0:
            return (-b/(2*a), -b/(2*a)) # корень кратности 2
        else:
            return "Действительных корней нет"
        elif a==b==c:
            return "Корень любое число"
        elif b!=0: # вырождается в линейное уравнение
            return (-c/b, ) # один корень
        else:  return "Корней нет"
```

Продолжение примера

```
# начинается основная программа
try:
  a=int(input("a="))
  b=int(input("b="))
  c=int(input("c="))
  print(solveSquareEquation(a,b,c))
except ValueError as err:
  print("Ошибка ввода\n",err)
```

Подключение модулей

```
>>> import math as m
>>> m.sqrt(4)
2.0
>>> from math import e, ceil as c
>>> e
2.718281828459045
>>> c(4.6)
5
```

```
>>> from math import (sin, cos, tan, atan)

>>> from sys import *
>>> version
'3.3.2 (v3.3.2:d047928ae3f6, May 16 2013,
00:03:43)
[MSC v.1600 32 bit (Intel)]'
>>> version_info
sys.version_info(major=3, minor=3, micro=2,
releaselevel='final', serial=0)
```

Модули и пакеты

- ***Модуль*** в языке Python – это файл с расширением .py. Обычно модуль содержит программный код на языке Python, но модули могут быть написаны на других языках программирования (чаще всего на языке C)
- ***Пакет*** — это просто папка, содержащая несколько модулей и файл с именем **__init__.py**
- Пакеты обычно содержат модули по сходной тематике, сконцентрированные на каких-то общих однотипных задачах.

Пример

```
Graphics/
    __init__.py
    Bmp.py
    Jpeg.py
    Png.py
    Tiff.py
    Xpm.py
```

```
import Graphics.Bmp
image = Graphics.Bmp.load("bashful.bmp")
import Graphics.Jpeg as Jpeg
image = Jpeg.load("doc.jpeg")
from Graphics import Png
image = Png.load("dopey.png")
```

Управление загрузкой

- Удобно загружать все модули пакета одной инструкцией. Для этого в __init__.py пакета нужно записать инструкцию, которая укажет какие модули должны загружаться.

```
__all__ = ["Bmp", "Jpeg", "Png", "Tiff", "Xpm"]
```

- Теперь после

```
from Graphics import *
```

  мы получим доступ ко всем элементам описанным в переменной __all__ . Кроме этого, мы можем поместить в __init__.py любой программный код, который будет исполнятся при подключении пакета.

- Это всё может применяться и к модулям, то есть если выполнить

```
from module import *
```

- то будут импортированы все элементы (функции, переменные и другие элементы, определяемые модулем, за исключением тех, чьи имена начинаются с символа подчёркивания). Если нужно точно указать, что должно быть импортировано при использовании команды from module import * , то мы можем определить список __all__ непосредственно в модуле.

Среды разработки

- Командная строка и **Блокнот** – для аскетов
- **Spyder** – для ленивых
- **Jupyter Notebook** – для научных расчетов, короче, для ботаников
- **NetBeans** + **Jython** – для любителей Java и JVM
- **PyCharm** – для продвинутых программеров

Управление пакетами

- **PIP** из коробки
- **Anaconda** – рекомендую

СПАСИБО ЗА ВНИМАНИЕ

- Вопросы
- Анонс следующей лекции