Введение в Python

Python/Spring-2019

Информация о курсе

- 100500 лекций
- 100499 домашних заданий
- В конце каждой лекции:
    - Ссылка на опорные материалы лекции
    - Ссылка на домашнее задание
    - Ссылку на гуглоформу опроса
- В конце курса:
    - Экзамен
- После курса:
    - Лаборатория
    - Работа
- Информационная поддержка:
    - Tg: https://epa.ms/epam-python-winter-2019

GUIDO VAN ROSSUM

ex-"Великодушный пожизненный диктатор"

- Разрабатывал язык ABC
- alma mater: Амстердамский Университет
- GAE и Mondrian в Google
- Сейчас в Dropbox
- pep 572 [y:=f(x), y**2, y**3]
- …
- Чуть не вылетел из университета
- Python + Amoeba
- https://habr.com/ru/post/315974/

Источники вдохновения

- ABC
- MODULA-3
- C
- SMALLTALK
- FORTRAN
- MIRANDA

Источники вдохновения

- JAVA
- ICON
- И
- ЕЩЁ
- МНОЖЕСТВО
- НЕЯВНЫХ

Дзен Питона

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и, желательно, *только* один — очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем *прямо* сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, *возможно*, хороша.
- Пространства имён — отличная вещь! Давайте будем делать их больше!

Что получилось?

```
1  def magic(dir):
2      acc = []
3      for root, dirs, files in os.walk(dir):
4          acc.extend(os.path.join(root, file) for file in files)
5          return ac
```

Если верить википедии…

- Python (в русском языке распространено название пито́н) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.
- Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — **динамическая типизация**, **автоматическое управление памятью**, **полная интроспекция**, механизм обработки исключений, **поддержка многопоточных вычислений**, **высокоуровневые структуры данных**. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.
- Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Есть реализация интерпретатора для JVM с возможностью компиляции, CLR, LLVM, другие независимые реализации. Проект PyPy использует JIT-компиляцию, которая значительно увеличивает скорость выполнения Python-программ.

Python – динамический интерпретируемый язык

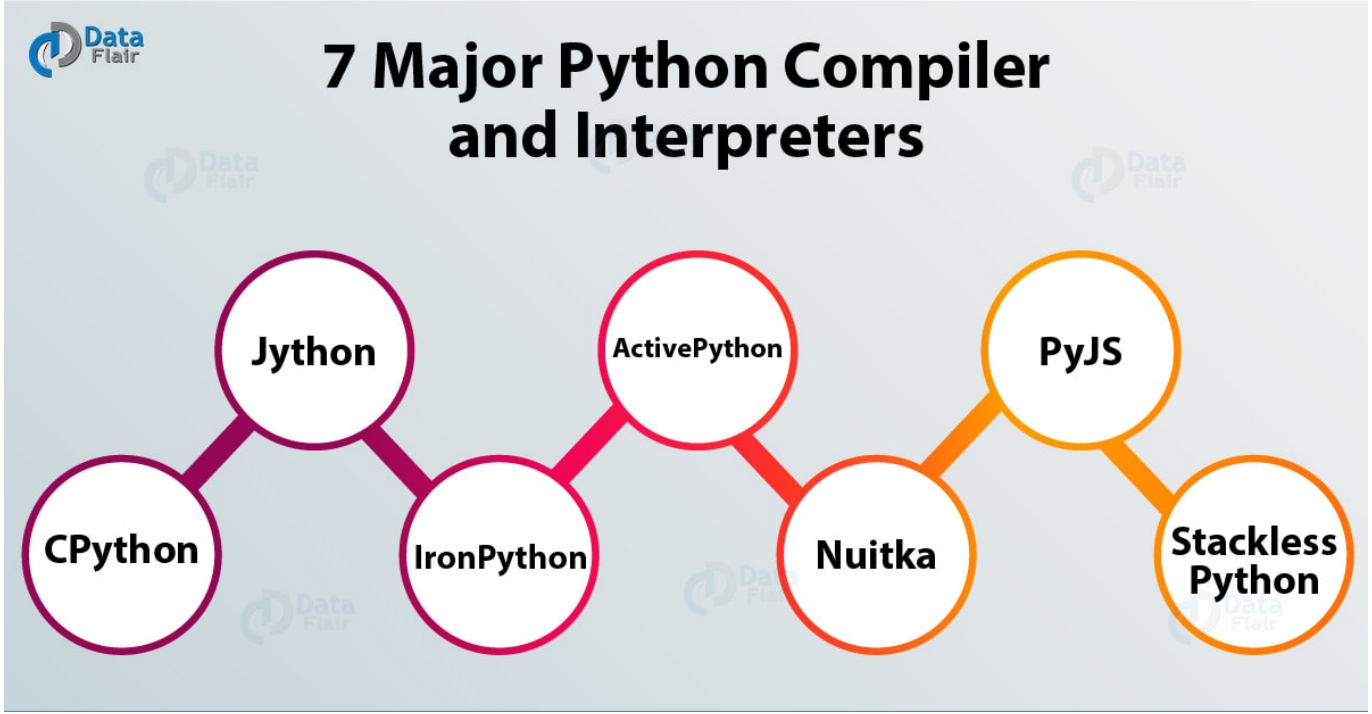
ДИНАМИЧЕСКИЙ ЯЗЫК

```
>>> def add(x,y):
...     return x+y
...
>>> def bar(x):
...     add(x, "1", "2")
...
...
>>> add(1,2)
3
>>> bar(3)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    bar(3)
  File "<input>", line 2, in bar
    add(x, "1", "2")
TypeError: add() takes exactly 2 arguments (3 given)
>>>
```

ИНТЕРПРЕТИРУЕМЫЙ

```
➜ source cat hello.py
message = "Python Epam 2019"
➜ source python -m dis ./hello.py
  1           0 LOAD_CONST               0 ('Python Epam 2019')
              3 STORE_NAME               0 (message)
              6 LOAD_CONST               1 (None)
              9 RETURN_VALUE
➜ source
```

Реализации


Data Flair
7 Major Python Compiler and Interpreters
Jython
CPython
IronPython
ActivePython
Nuitka
PyJS
Stackless Python

Основные идеи

```
>>> import hello
>>> hello
<module 'hello' from 'hello.py'>
>>> dir(hello)
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'message']
>>> hello.__name__
'hello'
>>> hello.__file__
'hello.py'
>>> print(hello.message)
Python Epam 2019
>>>
```

ТИПЫ ДАННЫХ

Специальные

```
>>> None
>>> None == None
True
>>> None is None
True
>>> type(None)
<type 'NoneType'>
>>>
```

Логические

```
>>> True
True
>>> False
False
>>> True == False
False
>>> True is False
False
>>> True is not False
True
>>> True != False
True
>>> type(True), type(False)
(<type 'bool'>, <type 'bool'>)
>>>
```

Числовые

```
>>> 42
42
>>> type(42)
<type 'int'>
>>> .42
0.42
>>> type(.42)
<type 'float'>
>>> 42j
42j
>>> type(42j)
<type 'complex'>
>>> 2019 / 3
673
>>> 2019 / 4
504
>>> 2019 // 4
504
>>> 2019 / 4.0
504.75
>>>
```

Строковые

```
>>> "epam"
'epam'
>>> b"epam"
b'epam'
>>> b"epam".decode("utf-8")
'epam'
>>> text = "epam"
>>> text
'epam'
>>> len(text)
4
>>> text[3]
'm'
>>> "magic " * 5
'magic magic magic magic magic '
>>>
```

Коллекции (последовательности, sequences)

```
>>> []
[]
>>> list()
[]
>>> [1] * 4
[1, 1, 1, 1]
>>> lst = ['e', 'p', 'a', 'm']
>>> lst
['e', 'p', 'a', 'm']
>>> len(lst)
4
>>> lst[3]
'm'
>>> lst[2] = '@'
>>> lst
['e', 'p', '@', 'm']
>>> lst + ['2019']
['e', 'p', '@', 'm', '2019']
>>> lst[0:4]
['e', 'p', '@', 'm']
>>> lst[0:]
['e', 'p', '@', 'm']
>>> lst[:4]
['e', 'p', '@', 'm']
>>> lst[::-1]
['m', '@', 'p', 'e']
>>> lst[::2]
['e', '@']
>>>
```

Коллекции (последовательности, sequences)

```
>>> ()
()
>>> tuple()
()
>>> lst = ('e', 'p', 'a', 'm')
>>> lst
('e', 'p', 'a', 'm')
>>> lst[2] = '@'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    lst[2] = '@'
TypeError: 'tuple' object does not support item assignment
>>>
```

Коллекции (последовательности, sequences)

```
>>> set()
set()
>>> a = {'e', 'p', 'a', 'm'}
>>> a
{'p', 'e', 'm', 'a'}
>>> 'e' in a
True
>>> '2019' in a
False
>>> a.add('2019')
>>> a
{'e', '2019', 'a', 'p', 'm'}
>>> '2019' in a
True
>>> a.discard('2019')
>>> a
{'e', 'a', 'p', 'm'}
```

Коллекции (последовательности, sequences)

```
>>> a = {1,2,3}
>>> b = {2,4}
>>> a or b
{1, 2, 3}
>>> a and b
{2, 4}
>>> a & b
{2}
>>> a | b
{1, 2, 3, 4}
>>> a.intersection(b)
{2}
>>> a.union(b)
{1, 2, 3, 4}
>>>
```

Словари (хэш-таблицы, mappings)

```
>>> {} == dict()
True
>>> course = {"name": "epam", "year": 2019}
>>> course
{'name': 'epam', 'year': 2019}
>>> course["name"]
'epam'
>>> course["year"]
2019
>>> "year" in course
True
>>> course.keys()
dict_keys(['name', 'year'])
>>> course.values()
dict_values(['epam', 2019])
>>>
```

УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ

if

```
>>> text = "epam"
>>> if text[2] == 'a':
...     print('yes')
... elif text[1] == 'e':
...     print('no')
... else:
...     print('i dont know')
...
...
i dont know
>>> "yes" if text[0] == 'e' else "no"
'yes'
>>>
```

Циклы (while и for)

```
>>> i = 0
>>> while i < 10:
...     print(i)
...     i += 1
...
...
0
1
2
3
4
5
6
7
8
9
```

```
>>> for i in range(0, 10):
...     print(i)
...
...
0
1
2
3
4
5
6
7
8
9
>>>
```

Циклы (while и for)

```
>>> while False:
...     print('nonono:)')
... else:
...     print('explain this')
...
...
explain this
>>>
```

МОДУЛИ И ПАКЕТЫ

Модули и пакеты

```
>>> import hello
>>> hello
<module 'hello' from 'hello.py'>
>>> dir(hello)
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'message']
>>> hello.__name__
'hello'
>>> hello.__file__
'hello.py'
>>> print(hello.message)
Python Epam 2019
>>>
```

IDE AND TEXT EDITORS

Pycharm, VSCode, VIM

		Cross Platform	Commercial/ Free	Auto Code Completion	Integrated Python Debugging	Bracket Matching	Line Numbering	Code Folding	Code Templates	Unit Testing	Integrated DB Support	Rapid Application Development	Notes
2	Atom	Y	F		Y	Y	Y	Y	Y				
3	BlackAdder	Y	C					Y					
4	BlueFish	L											
5	ConTEXT	W	C										
6	DABO	Y											
7	Dr.Python		F										
8	DreamPie		F	Y									
9	E-Texteditor	W											
10	Emacs	Y	F	Y	Y	Y	Y	Y	Y	Y			
11	Editra	Y	F	Y		Y	Y	Y					
12	Eric Ide	Y	F	Y	Y		Y	Y		Y			
13	Geany	Y	F	Y*		Y	Y	Y					*very limited
14	Gedit	Y	F	Y[1]		Y	Y		Y[2]				[1]with plugin; [2]sort of
15	Idle	Y	F	Y	Y	Y							
16	JEdit	Y	F			Y	Y	Y					
17	KDevelop	Y	F			Y	Y	Y					
18	Komodo	Y	CF	Y	Y	Y	Y	Y	Y	Y	Y		
19	NetBeans*	Y	F	Y	Y	Y	Y	Y	Y	Y		Y	*pre-v7.0
20	Ninja-IDE	LW	F	Y	Y**	Y	Y	Y	Y	Y*			*No, but plugins for ruby, **through plugins
21	Notepad++	W	F	Y		Y	Y	Y	Y*				*with plugin
22	Pfaide	W	C	Y		Y	Y	Y	Y				
23	PIDA	LW	F	Y		Y	Y	Y					VIM based
24	PTVS	W	F	Y	Y	Y	Y	Y				Y	*WPF bsed
25	PyCharm	Y	CF	Y	Y	Y	Y	Y		Y			*JavaScript
26	PyDev(Eclipse)	Y	F	Y	Y	Y	Y	Y	Y	Y			
27	Pyscripter	W	F	Y	Y		Y		Y	Y			
28	PythonWin	W	F	Y	Y	Y		Y					
29	SciTE	Y	F[1]			Y	Y	Y	Y				[1]Mac version is commercial
30	ScriptDev	W	C	Y	Y	Y	Y	Y	Y				
31	SPE		F	Y									
32	Spyder	Y	F	Y	Y	Y	Y						
33	Sublime Text	Y	CF	Y		Y	Y	Y	Y	Y*			*extensible w/Python, PythonTestRunner
34	TextMate	M	F			Y	Y	Y	Y				
35	UltiPad	Y	F	Y	Y	Y			Y	Y			
36	Vim	Y	F	Y	Y	Y	Y	Y	Y	Y			
37	WingIde	Y	C	Y	Y	Y	Y	Y	Y	Y			*support for C
38	Zeus	W	C			Y	Y	Y	Y				

VCS (GIT)

GIT

- Распределённая система управления версиями
- Авторы – Линус Торвальдс и ~~студенты~~ высококлассные программисты

git

Create a Repository

From scratch -- Create a new local repository
```
$ git init [project name]
```

Download from an existing repository
```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed
```
$ git status
```

Show the changes to files not yet staged
```
$ git diff
```

Show the changes to staged files
```
$ git diff --cached
```

Show all staged and unstaged file changes
```
$ git diff HEAD
```

Show the changes between two commit ids
```
$ git diff commit1 commit2
```

List the change dates and authors for a file
```
$ git blame [file]
```

Show the file changes for a commit id and/or file
```
$ git show [commit]:[file]
```

Show full change history
```
$ git log
```

Show change history for file/directory including diffs
```
$ git log -p [file/directory]
```

Working with Branches

List all local branches
```
$ git branch
```

List all branches, local and remote
```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory
```
$ git checkout my_branch
```

Create a new branch called new_branch
```
$ git branch new_branch
```

Delete the branch called my_branch
```
$ git branch -d my_branch
```

Merge branch_a into branch_b
```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit
```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit
```
$ git add [file]
```

Stage all changed files, ready for commit
```
$ git add .
```

Commit all staged files to versioned history
```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history
```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes
```
$ git reset [file]
```

Revert everything to the last commit
```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)
```
$ git fetch
```

Fetch the latest changes from origin and merge
```
$ git pull
```

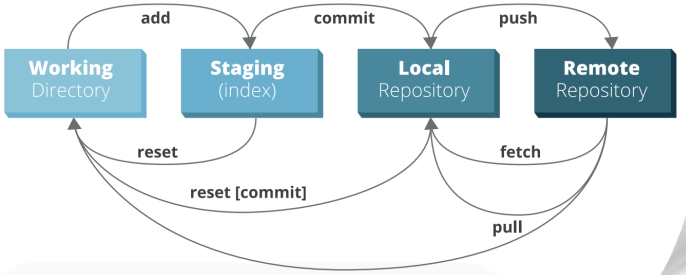
Fetch the latest changes from origin and rebase
```
$ git pull --rebase
```

Push local changes to the origin
```
$ git push
```

Finally!

When in doubt, use git help
```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.



BROUGHT TO YOU BY JRebel

Ссылки

- Документация: https://docs.python.org/
- Сборка материалов: https://vk.com/wall-12446354_1668
- Книги
    - Эндрю Тан(н?)енбаум «Архитектура компьютера»
    - Эндрю Тан(н?)енбаум «Компьютерные сети»
    - Эндрю Тан(н?)енбаум «Современные операционные системы»
    - Олиферы «Компьютерные сети»
    - Дональд Кнут «Искусство программирования»
    - Роберт Седжвик «Алгоритмы на C++»
    - Томас Кормен «Алгоритмы. Построение и анализ»
- ДЗ: https://github.com/budurli/EpamPython2019
- Обратная связь: https://forms.gle/uXuv7a73CYu7QYwx9