Introduction to Python

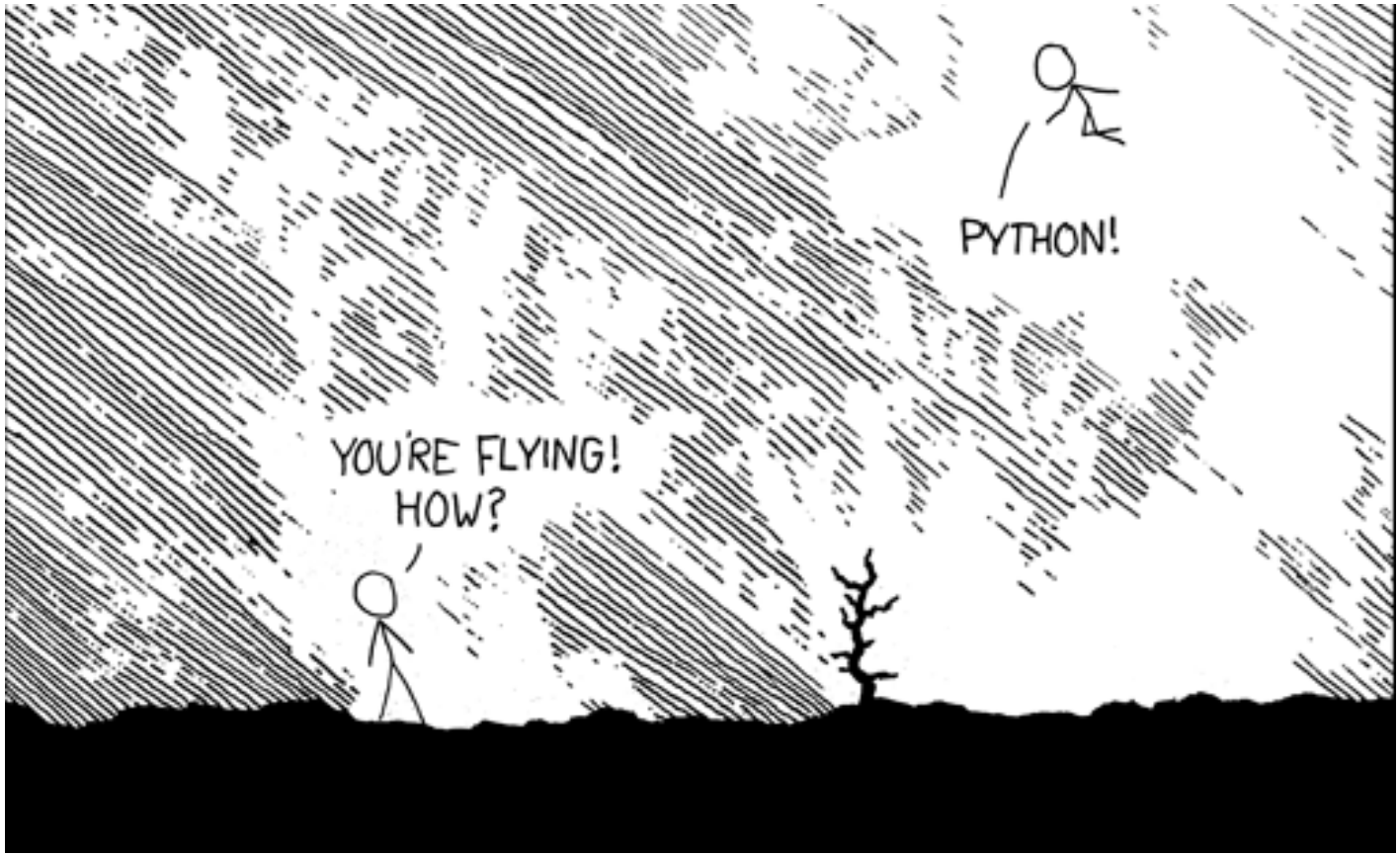
Petr Zemek

Senior Developer at Avast Software
Threat Labs (Viruslab)
petr.zemek@avast.com
https://petrzemek.net

Principles of Programming Languages, BUT FIT, 2017-03-07

Motto

"Python makes you fly."



https://xkcd.com/353/

Why Python? Whetting our Appetite

Feb 2017	Feb 2016	Change	Programming Language	Ratings	Change
1	1		Java	16.676%	-4.47%
2	2		C	8.445%	-7.15%
3	3		C++	5.429%	-1.48%
4	4		C#	4.902%	+0.50%
5	5		Python	4.043%	-0.14%
6	6		PHP	3.072%	+0.30%
7	9	^	JavaScript	2.872%	+0.67%
8	7	v	Visual Basic .NET	2.824%	+0.37%
9	10	^	Delphi/Object Pascal	2.479%	+0.32%
10	8	v	Perl	2.171%	-0.08%

http://www.tiobe.com/tiobe-index/

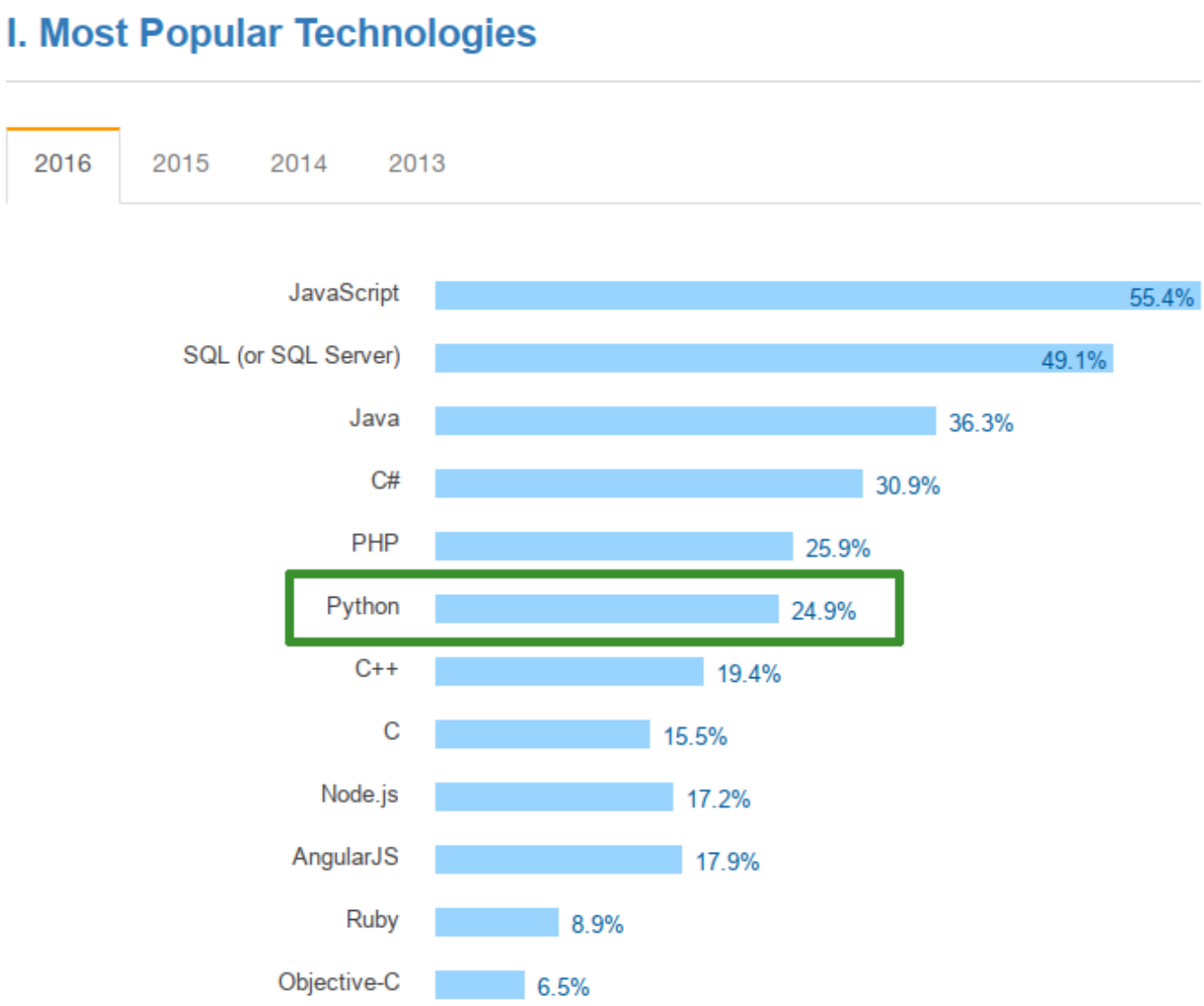
Why Python? Whetting our Appetite

Worldwide, Feb 2017 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.6 %	-1.3 %
2		Python	14.7 %	+2.8 %
3		PHP	9.4 %	-1.2 %
4		C#	8.3 %	-0.3 %
5	↑↑	Javascript	7.7 %	+0.4 %
6		C	7.0 %	-0.2 %
7	↓↓	C++	6.9 %	-0.6 %
8		Objective-C	4.2 %	-0.6 %
9	↑	R	3.4 %	+0.4 %
10	↓	Swift	2.9 %	+0.1 %

http://pypl.github.io/

Why Python? Whetting our Appetite



http://stackoverflow.com/research/developer-survey-2016

What is Python?

- widely used, general-purpose high-level programming language
- design philosophy emphasizes code readability
- multiparadigm (procedural, object oriented)
- compiled to bytecode and interpreted in a virtual machine
- everything is an object
- strongly, dynamically typed
- duck typing
- whitespace is significant
- portable (Windows, Linux, Mac OS)
- many implementations (CPython, PyPy, Jython, IronPython)
- automatic memory management (garbage collector)
- free (both as in "free speech" and "free beer")

A Glimpse at Python History

- invented in the beginning of 1990s by Guido van Rossum

- its name stems from "Monty Python's Flying Circus"
- version history:
  - Python 1.0 (January 1994)
  - Python 2 (October 2000)
    - Python 2.7 (July 2010) – latest 2.x version († 2020)
  - Python 3 (December 2008)
    - Python 3.6 (December 2016) – latest 3.x version

Diving Into Python

- interactive shell

```
$ python
Python 3.6.0 (default, Jan 16 2017, 12:12:55)
>>> print('Hello, world!')
Hello, world!
```

- running from source

```
# In file hello.py:
print('Hello, world!')

$ python hello.py
Hello, world!
```

- combination

```
$ python -i hello.py
Hello, world!
>>>
```

Built-In Primitive Data Types

- NoneType

```
None
```

- bool

```
True, False
```

- int

```
-1024, 0, 178212237348573485387462734645 45
```

- float

```
0.125, 1e200, float('inf'), float('nan')
```

- complex

```
2 + 3j
```

- str

```
'Do you like jalapeño peppers?'
```

- bytes

```
b'\x68\x65\x6c\x6c\x6f'
```

Intermezzo: Encodings

- character set vs encoding
- single-byte vs multi-byte
- Unicode vs UTF-8, UTF-16, UTF-32
- `str` vs `bytes`

https://cs-blog.petrzemek.net/2015-08-09-znakova-sada-vs-kodovani

Built-In Collection Types

- list

```
[1, 2.0, 'hey!', None]
```

- tuple

```
('Cabernet Sauvignon', 1995)
```

- set

```
{1, 2, 3, 4, 5}
```

- dict

```
{
    'John': 2.5,
    'Paul': 1.5,
    'Laura': 1
}
```

Variables and Bindings

- name binding (we attach a name to an object)
- dynamic typing
- no explicit types until Python 3.5 (*type hints*)

```
>>>  x = 1                    # x --> 1
>>>  x = 'hi there'           # x --> 'hi there'

>>>  a = [1, 2]               # a --> [1, 2]
>>>  b = a                    # a --> [1, 2] <-- b
>>>  a.append(3)              # a --> [1, 2, 3] <-- b
>>>  a
[1, 2, 3]
>>>  b
[1, 2, 3]
>>>  b = [4]                  # a --> [1, 2, 3]; b --> [4]
```

Operations

arithmetic	`+ - * / // % **`
comparison	`== != < > <= >=`
bitwise	`<< >> | & ^ ~`
indexing	`[]`
slicing	`[:]`
call	`()`
logical	and or not
assignment	`= += -= *= /= //= %= **=` ...
other	in is

Basic Statements

`=` assignment statements

```
x = 1
x += 41
```

(expr) expression statements

```
print('My name is', name)
```

if conditional execution

```
if x > 10:
    x = 10
elif x < 5:
    x = 5
else:
    print('error')
```

Basic Statements (Continued)

for traversing collections

```
for color in ['red', 'green', 'blue']:
    print(color)
```

while repeated execution

```
while x > 0:
    print(x)
    x -= 1
```

break breaking from a loop

continue continuing with the next cycle of a loop

return returning from a function

pass does nothing

Functions

```
def factorial(n):
    """Returns the factorial of n."""
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

x = factorial(5)  # 120
```

- first-class objects
- can be nested
- default arguments
- keyword arguments
- variable-length arguments

Pass by... What Exactly?

```
def foo(x):
    x = 4

a = 1
foo(a)
print(a)   # ?

def bar(list):
    list.append(4)

b = [1, 2, 3]
bar(b)
print(b)   # ?
```

Arguments are passed by assignment.

Scoping

```
...  # 1
def foo():
    ...  # 2
    def bar():
        ...  # 3
        while cond:
            ... # 4
            print(x)
```

- lexical scoping
- LEGB: a concise rule for scope resolution
  1. **L**ocal
  2. **E**nclosing
  3. **G**lobal
  4. **B**uilt-in
- `if`, `for`, etc. do not introduce a new scope
- explicit declarations via `global` and `nonlocal`

Lifetimes

- global variables exist until program exits
- local variables exist until function exits
- explicit deletion via `del`

Namespaces, Modules, and Packages

```
# Example of a custom package:

network/
    __init__.py
    socket.py
    http/
        __init__.py
        request.py
        response.py
        ...
    bittorrent/
        __init__.py
        torrent.py
        bencoding.py
        ...
    ...

from network.http.request import Request
```

Imports

```
# Import a single module.
import time

# Import multiple modules at once.
import os, re, sys

# Import under a different name.
import multiprocessing as mp

# Import a single item from a module.
from threading import Thread

# Import multiple items from a module.
from collections import namedtuple, defaultdict

# Import everything from the given module.
# (Use with caution!)
from email import *
```

Object-Oriented Programming

```
from math import sqrt

class Point:
    """Representation of a point in 2D space."""

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, other):
        return sqrt((other.x - self.x) ** 2 +
                    (other.y - self.y) ** 2)

a = Point(1, 2)
b = Point(3, 4)
print(a.distance(b))  # 2.8284271247461903
```

Object-Oriented Programming (Continued)

- classes are first-class objects
- everything is public
- everything can be overridden
- special methods (`__method__`)
- initializers and finalizers
- each class automatically inherits from `object`
- multiple inheritance, method resolution order (MRO)
- classes are instances of metaclasses
- classes can be extended at runtime
- instance variables vs class variables
- instance methods vs class methods vs static methods

Error Handling and Exceptions

```
try:
    # code
except IOError as ex:
    # handle a specific exception
except:
    # handle all other exceptions
else:
    # no exception was raised
finally:
    # clean-up actions, always executed

# Raising an exception:
raise RuntimeError('not enough space')
```

Exception-Safe Resource Management

```
# Bad:
f = open('file.txt', 'r')
contents = f.read()
f.close()

# Better:
f = open('file.txt', 'r')
try:
    contents = f.read()
finally:
    f.close()

# The best:
with open('file.txt', 'r') as f:
    contents = f.read()
```

https://cs-blog.petrzemek.net/2013-11-17-jeste-jednou-a-lepe-prace-se-souborem-v-pythonu

Intermezzo: Text vs Binary Files

- text vs binary mode

```
with open(file_path, 'r') as f:
    text = f.read()

with open(file_path, 'rb') as f:
    data = f.read()
```

- differences between text and binary modes in Python:
  1. decoding
  2. end-of-line conversions
  3. buffering

https://cs-blog.petrzemek.net/2015-08-26-textove-vs-binarni-soubory

Some Cool Language Features

- string formatting (*f-strings*, Python 3.6)

```
name = 'Joe'
item = 'bike'
print(f'Hey {name}, where is my {item}?')
```

- anonymous functions

```
people.sort(key=lambda person: person.name)
```

- list/set/dict comprehensions

```
list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = [x ** 2 for x in list if x % 2 == 0]
# [4, 16, 36, 64, 100]
```

- conditional expressions

```
cost = 'cheap' if price <= 100 else 'expensive'
```

Some Cool Language Features (Continued)

- `eval()` and `exec()`

```
a = eval('1 + 3')          # a = 4
exec('b = [1, 2, 3]')      # b = [1, 2, 3]
```

- dynamic typing

```
def print_all(col):
    for i in col:
        print(i)

print_all([1, 2, 3])
print_all(('a', 'b', 'c'))
```

- `enumerate()`

```
for i, person in enumerate(people):
    print(i, ':', person)
```

Some Cool Language Features (Continued)

- chained comparisons

```
if 1 < x < 5:
    # ...
```

- digits separator (Python 3.6)

```
1_483_349_803
```

- tuple unpacking

```
head, *middle, tail = [1, 2, 3, 4, 5]
```

Some Cool Language Features (Continued)

- generators

```
def fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

fib = fibonacci()
next(fib)   # 0
next(fib)   # 1
next(fib)   # 1
next(fib)   # 2
next(fib)   # 3
next(fib)   # 5
next(fib)   # 8
```

Weird Language Features

- `for` with `else`

```
for item in some_list:
    if item == 5:
        break
else:
    print("not found")
```

- mutating default arguments

```
def foo(x=[]):
    x.append(4)
    return x

print(foo([1, 2, 3]))  # [1, 2, 3, 4]
print(foo())           # [4]
print(foo())           # [4, 4]
```

- non-ASCII identifiers

```
π = 3.1415
```

What We Have Skipped

- decorators
- properties
- metaclasses
- descriptors
- context managers
- asynchronous I/O
- coroutines
- ...and more...

A Brief Tour of the Standard Library

- text processing (`re`, `json`, `xml`, `csv`, `base64`)
- data types (`datetime`, `collections`, `queue`)
- concurrency (`threading`, `multiprocessing`, `asyncio`)
- math (`math`, `decimal`, `fractions`, `statistics`)
- operating system and filesystem (`os`, `shutil`, `tempfile`)
- IPC and networking (`signal`, `mmap`, `select`, `socket`)
- Internet protocols (`urllib`, `email`, `smtplib`, `ipaddress`)
- compression (`zipfile`, `tarfile`, `gzip`)
- cryptography (`hashlib`, `secrets`)
- functional-like programming (`itertools`, `functools`)
- development (`unittest`, `doctest`, `venv`)
- debugging and profiling (`pdb`, `timeit`, `dis`)
- other (`logging`, `argparse`, `ctypes`)
- ...

Some Other Interesting Libraries and Projects

- `pip` (installation of Python packages)
- `requests` (HTTP for humans)
- `sphinx` (documentation)
- `sqlalchemy` (database toolkit)
- `numpy`, `scipy` (scientific computing)
- `django`, `flask` (web frameworks)
- `coverage` (code coverage)
- `ply` (Python Lex and Yacc)
- `matplotlib` (2D plotting)
- `pygal` (charting)
- `pygame` (games)
- `pyqt` (GUI)
- `retdec-python` (decompilation)

Advantages of Python

+ clean and simple syntax
+ easy to learn
+ productivity (high-level constructs)
+ powerful built-in types
+ elegant and flexible module system
+ excellent standard library
+ reflection
+ multiparadigm (procedural, object oriented)
+ generic programming (duck typing)
+ widely used

Disadvantages of Python

- not very fast on computationally intensive operations
- not for memory-intensive tasks
- limited parallelism with threads (Global Interpreter Lock)
- limited notion of constness
- portable, but some parts are OS-specific
- Python 2 vs 3 (incompatibilities)

Varying Opinions

+/- everything is public
+/- unsystematic documentation
+/- whitespace is significant
+/- standardization
+/- supports "monkey patching"
+/- not suitable for writing low-level code
+/- dynamic typing

https://cs-blog.petrzemek.net/2014-10-26-co-se-mi-nelibi-na-pythonu

Demos

- counting lines and words in a file
- working with text (regular expressions)
- working with JSON
- working with XML

Summary

- imperative language
- multiparadigm (procedural, object oriented)
- strongly typed
- dynamically typed
- interpreted (translated to internal representation)
- modularity is directly supported (packages, modules)

Where to Look for Further Information?

Python Programming Language – Official Website

https://www.python.org/

Python 3 Documentation

https://docs.python.org/3/

Official Python 3 Tutorial

https://docs.python.org/3/tutorial/

Dive into Python 3

http://www.diveintopython3.net/

Learning Python, 5th Edition (2013)

http://shop.oreilly.com/product/0636920028154.do

Fluent Python (2015)

http://shop.oreilly.com/product/0636920032519.do

Témata bakalářských prací v Avastu

- *Analýza souborů ve formátu PE*
  (https://retdec.com/fileinfo/)
  Přidávání nových analýz, např. informace z hlaviček, Go, Visual Basic, Delphi, instalátory (C++).

- *Analýzy a optimalizace ve zpětném překladači*
  (https://retdec.com/decompilation/)
  Návrh a tvorba nových analýz a optimalizací, např. pro kód napsaný v C++, Delphi, Objective-C (C++).

- *Kategorizace souborů podle podobnosti*
  Shluková analýza skriptů, dokumentů, instalátorů atd. (Python, C++).

- *Honeypot jako nástroj boje proti malware*
  Vytvoření automatizovaného honeypotu, který bude detekovat nové hrozby (pravděpodobně Python).

Kontaktní osoba: Jakub Křoustek (jakub.kroustek@avast.com)