Introduction to Python

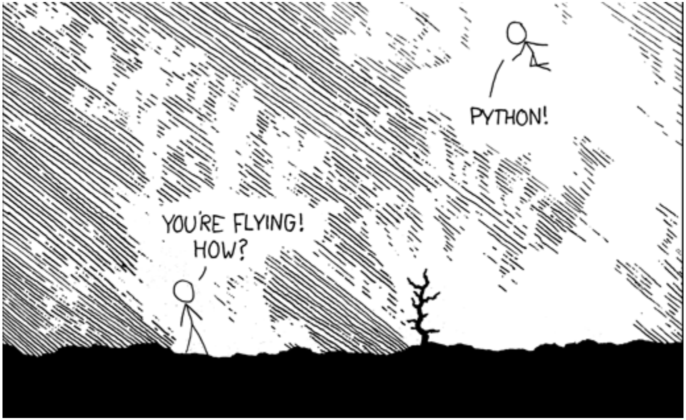
Petr Zemek

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

Principles of Programming Languages, BUT FIT, 2020-03-02

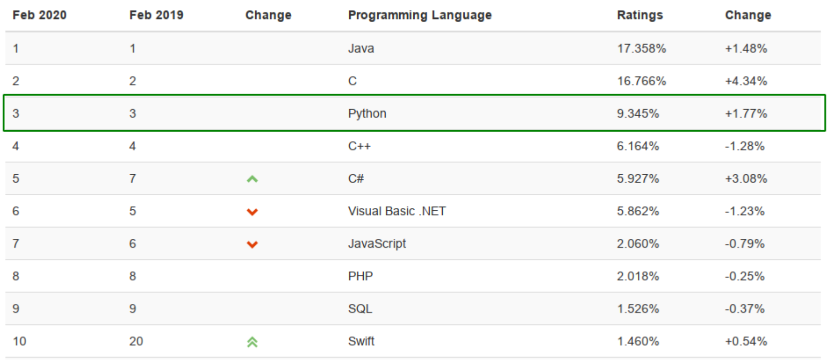
Motto

"Python makes you fly."



https://xkcd.com/353/

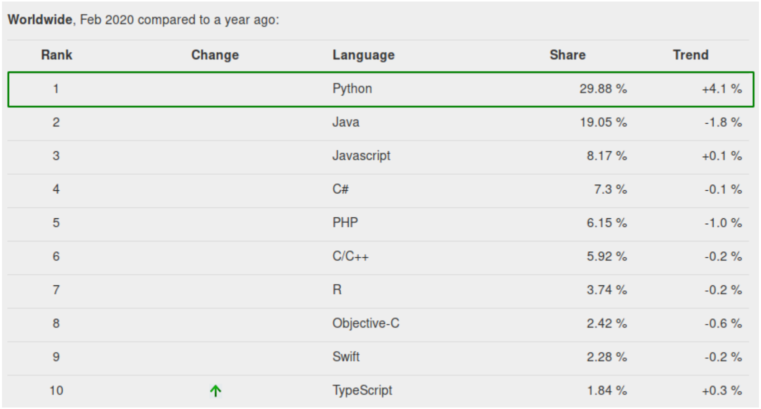
Why Python? Whetting our Appetite



Feb 2020	Feb 2019	Change	Programming Language	Ratings	Change
1	1		Java	17.358%	+1.48%
2	2		C	16.766%	+4.34%
3	3		Python	9.345%	+1.77%
4	4		C++	6.164%	-1.28%
5	7	▲	C#	5.927%	+3.08%
6	5	▼	Visual Basic .NET	5.862%	-1.23%
7	6	▼	JavaScript	2.060%	-0.79%
8	8		PHP	2.018%	-0.25%
9	9		SQL	1.526%	-0.37%
10	20	▲▲	Swift	1.460%	+0.54%

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

Why Python? Whetting our Appetite

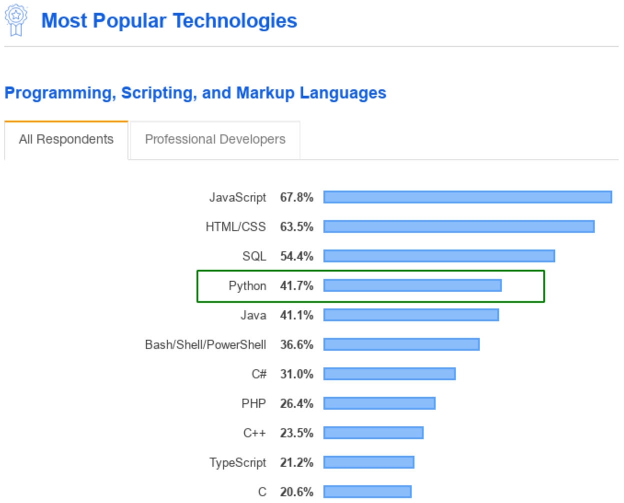


Worldwide, Feb 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.88 %	+4.1 %
2		Java	19.05 %	-1.8 %
3		Javascript	8.17 %	+0.1 %
4		C#	7.3 %	-0.1 %
5		PHP	6.15 %	-1.0 %
6		C/C++	5.92 %	-0.2 %
7		R	3.74 %	-0.2 %
8		Objective-C	2.42 %	-0.6 %
9		Swift	2.28 %	-0.2 %
10	↑	TypeScript	1.84 %	+0.3 %

http://pypl.github.io/

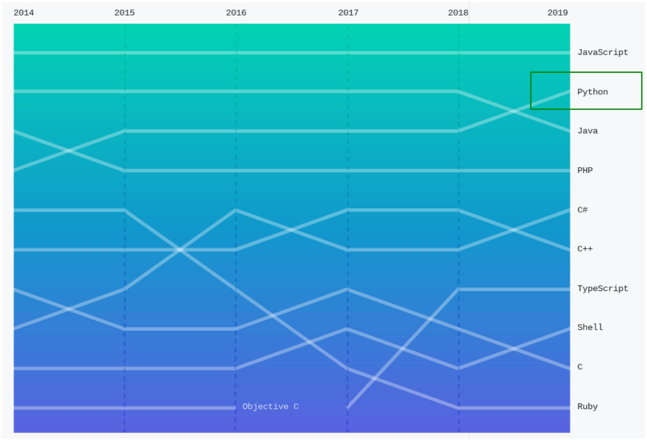
Why Python? Whetting our Appetite



JavaScript	67.8%
HTML/CSS	63.5%
SQL	54.4%
Python	41.7%
Java	41.1%
Bash/Shell/PowerShell	36.6%
C#	31.0%
PHP	26.4%
C++	23.5%
TypeScript	21.2%
C	20.6%

https://insights.stackoverflow.com/survey/2019

Why Python? Whetting our Appetite



https://octoverse.github.com/#top-languages

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 typed
- dynamically typed
- duck typing
- whitespace is significant
- portable (Windows, macOS, Linux, FreeBSD)
- 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 the '90s by Guido van Rossum

- its name stems from "Monty Python's Flying Circus"
- version history:
    - Python (1.0 in 1994)
    - Python 2 (2.0 in 2000, † 2020-01-01)
    - Python 3 (3.0 in 2008)
        - Python 3.8 (October 2019) – latest version

Diving Into Python

- interactive shell

```
$ python
Python 3.8.1 (default, Jan 22 2020, 06:38:00)
>>> 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, 17821223734857348538746273464545
  ```

- 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 declarations 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
while	repeated execution
break	breaking from a loop
continue	continuing with the next iteration of a loop
assert	assertions
return	returning from a function
pass	does nothing

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

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

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

Scoping

```
...  # A
def foo():
    ...  # B
    def bar():
        ...  # C
        while cond:
            ... # D
            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 the program exits
- local variables exist until the function exits
- explicit deletion via `del`

Namespaces, Modules, and Packages

```
# An 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 a module 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

```python
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 (Basics)

- instance creation and initialization
- methods versus functions
- classes are first-class objects
- everything is public
- everything can be overridden
- each class automatically inherits from `object`
- multiple inheritance, method resolution order (MRO)
- calling base-class methods
- instance variables vs class variables
- instance methods vs class methods vs static methods
- properties

Object-Oriented Programming (Advanced)

- instance creation in detail (`__new__()`, `__init__()`)
- instance memory layout (`__dict__`, `__slots__`)
- "internal" (_) and pseudo-private (__) attributes
- special methods (`__$method__()`), operator overloading
- cooperative multiple inheritance, mixins, `super()`
- instance finalization (`__del__()`)
- hooking into attribute lookup (`__getattr[ibute]__()`)
- protocols, duck typing
- interfaces, abstract base classes (`abc`)
- classes can be created and extended during runtime
- classes are instances of *metaclasses*

Error Handling and Exceptions

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

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

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 the 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]
```

- “the walrus operator” (Python 3.8)

```
# Loop over fixed length blocks.
while (block := f.read(256)) != '':
    process(block)
```

Some Cool Language Features (Continued)

- generators

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

for fib in fibonacci():
    print(fib)
    if fib > 100:
        break
```

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

- metaclasses
- descriptors
- decorators
- context managers
- threading
- multiprocessing
- asynchronous I/O
- coroutines
- annotations (including type hints)
- ... and more ...

A Brief Tour of the Standard Library

- text processing (`re`, `json`, `xml`, `csv`, `base64`)
- data types (`datetime`, `collections`, `dataclasses`)
- concurrency (`threading`, `multiprocessing`, `asyncio`)
- math (`math`, `decimal`, `fractions`, `statistics`)
- operating system and filesystem (`os`, `shutil`, `tempfile`)
- IPC and networking (`signal`, `mmap`, `selectors`, `socket`)
- Internet protocols (`urllib`, `email`, `smtplib`, `ipaddress`)
- compression (`zipfile`, `tarfile`, `gzip`)
- cryptography (`hashlib`, `hmac`, `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)

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

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

- *Extrakce informací ze spustitelných souborů, Pokročilá detekce podobnosti binárního kódu*

  C++ (https://retdec.com/)

- *Rozšiřování jazyka YARA pro popis vzorů*

  C (https://virustotal.github.io/yara/)

- *Rozšiřování systému pro shlukovou analýzu souborů*

  Python, C++

- *Automatizace nasazení a sběru dat z honeypotů*

  Python či jiný

Kontaktní osoba: Lukáš Zobal (izobal@fit.vutbr.cz)