Introduction to Python

Petr Zemek

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

Principles of Programming Languages, BUT FIT, 2021-03-08

About Me and Avast

Petr Zemek

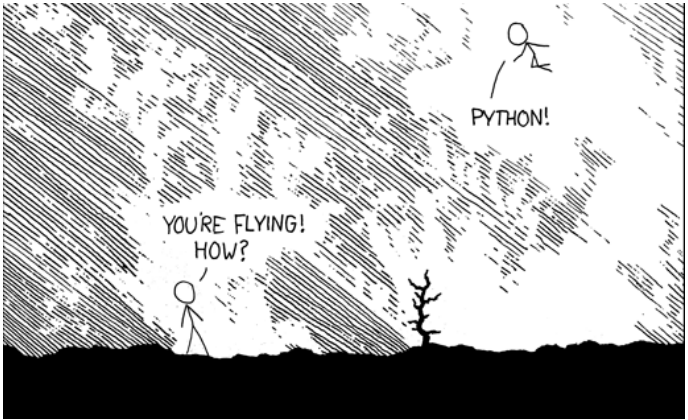
- Lead Software Engineer at Avast (2016/10 – *)
- Ph.D. in theoretical computer science from BUT FIT
- 10 years of professional experience with developing software
- https://petrzemek.net
- Czech and English blogs, talks, screencasts, open-source projects, . . .

Avast

- An international cybersecurity company protecting 400M+ people worldwide
- Safeguarding digital data, identity, and privacy
- Cooperating with universities
- https://www.avast.com/

Motto

"Python makes you fly."



https://xkcd.com/353/

Why Python? Whetting our Appetite

Worldwide, Feb 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	30.06 %	+0.3 %
2		Java	16.88 %	-1.7 %
3		JavaScript	8.43 %	+0.4 %
4		C#	6.69 %	-0.6 %
5	↑	C/C++	6.5 %	+0.5 %
6	↓	PHP	6.19 %	-0.1 %
7		R	3.82 %	+0.0 %
8		Objective-C	3.66 %	+1.2 %
9		Swift	2.05 %	-0.3 %
10		TypeScript	1.87 %	+0.0 %

http://pypl.github.io/

Why Python? Whetting our Appetite

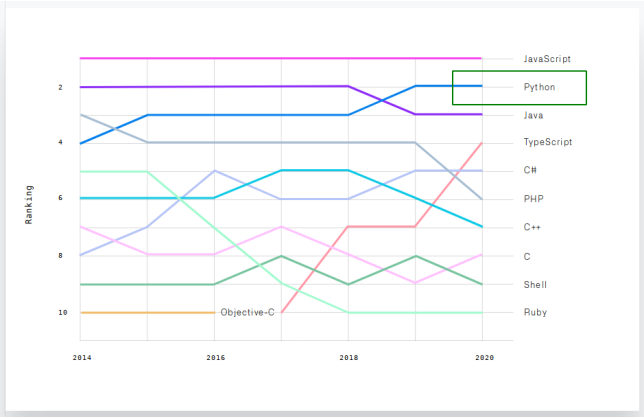
Feb 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	^	C	16.34%	-0.43%
2	1	v	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.52%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.48%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	^	Assembly language	1.65%	+0.54%

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

Why Python? Whetting our Appetite

https://insights.stackoverflow.com/survey/2020

Why Python? Whetting our Appetite



https://octoverse.github.com/

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 the History of Python

- 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 3.9 (October 2020) – latest version

https://cs-blog.petrzemek.net/2020-10-09-co-je-noveho-v-pythonu-3-9

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'
  ```

Character Sets and Encodings

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

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

```
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 iteration of a loop

assert — assertions

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

Scoping

```
...  # A
def foo():
    ...  # B
    def bar():
        ...  # C
        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`, `while` do not introduce a new scope
- explicit declarations via `global` and `nonlocal`

Lifetimes

- global variables exist until the program ends
- local variables exist until the function call ends
- 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

```
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

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*

Python's object model: https://youtu.be/QnDku649JFI

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:
    # cleanup 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

Writing Python Code In a Pythonic Way

- language idioms
- "Pythonic" vs "Unpythonic"

  Pythonic (*comparative* **more Pythonic**, *superlative* **most Pythonic**)

  1. (*programming jargon*) Using the idioms of the Python programming language.

- example:

```
# Unpythonic
i = 0
while i < len(items):
    print(items[i])
    i += 1

# Pythonic
for item in items:
    print(item)
```

- The Zen of Python (`import this`)

Selected Language Features (Part I/III)

- 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'
```

Selected Language Features (Part II/III)

- 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)
```

Selected Language Features (Part III/III)

- 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 collection:
    if item == 5:
        break
else:  # ?!
    print("not found")
```

- mutable 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
```

A Brief Overview 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`)
- ...

Not Enough? Check Out PyPI!

https://pypi.org/

```
$ pip install <package_name>
```

- official package repository for Python
- over 200 000 packages at your disposal
- you can create and publish your own packages
- you can create your own private repository

What We Have Skipped

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

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 (+ PyPI)
+ 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 (GIL: 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 (2011)
http://www.diveintopython3.net/

Learning Python, 5th Edition (2013)
http://shop.oreilly.com/product/0636920028154.do

Fluent Python (2015) – 2nd edition to be released in 2021
http://shop.oreilly.com/product/0636920032519.do

Oblasti témat bakalářských prací vypisovaných Avastem

Pro školní rok 2021/2022:

- Sběr dat z honeypotů a jejich využití pro threat intelligence
- Reverzní inženýrství a analýza malware
- (A možná další témata. Sledujte informační systém FIT.)

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