Python, the efficient way

...

Sebastián Arango Muñoz
TW: @charangostation
IG: @sebasarango118

About me

Lead Data Scientist @ Experimentality

MSc. (asp) in Software Engineering @ EAFIT

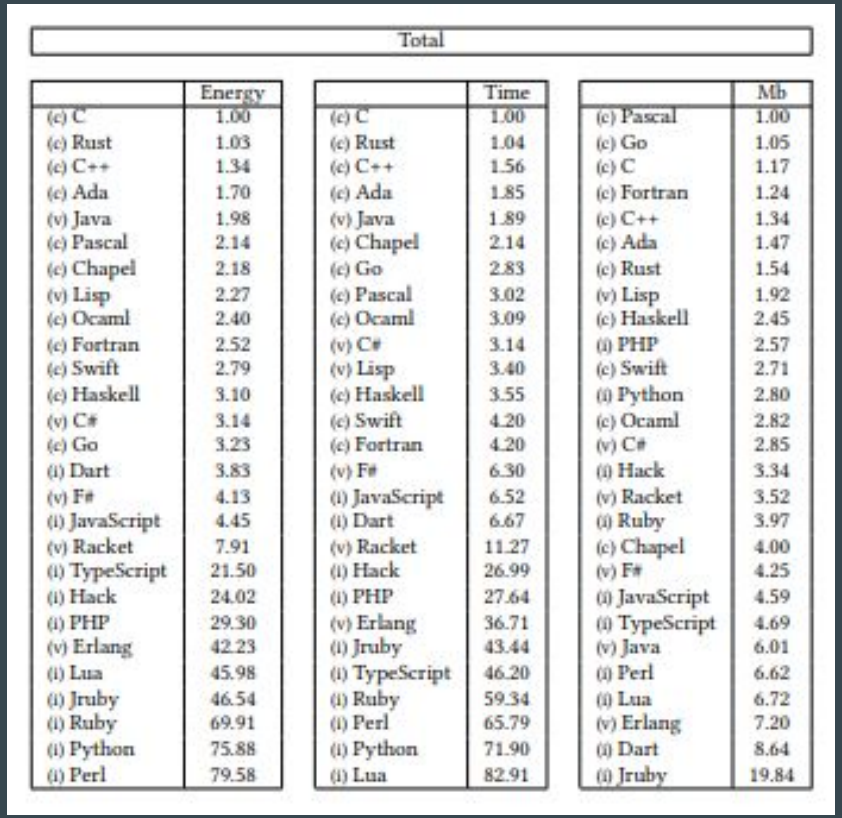
BSc. in Electronics Engineering @ UdeA

About Python

Fundamentals

- **Interpreted** language (over C).
- **Strong** | **dynamic** typing.
- **Self-managed** memory architecture.
- Promotes **rapid** development.
- **Large** community.

But...is it efficient?



Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

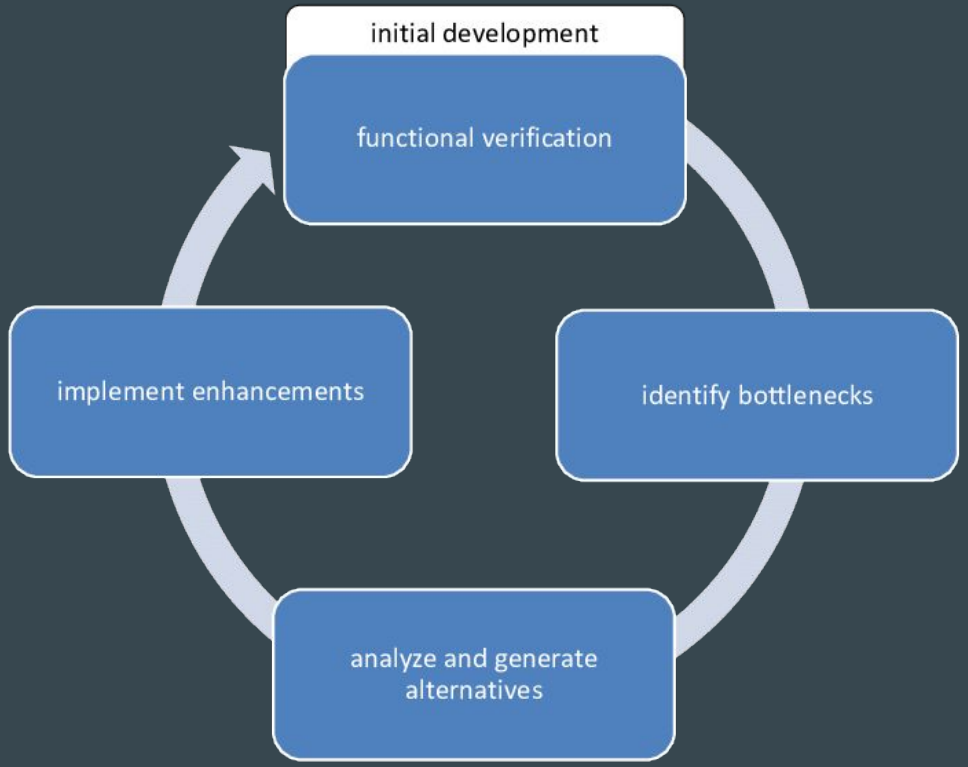
	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

"*Energy Efficiency across Programming Languages*". R, Pereira et al. 2017.

Some best practices

Development Cycle

1. Get it right.
2. Test it right.
3. Profile it.
4. Optimise.
5. Repeat.



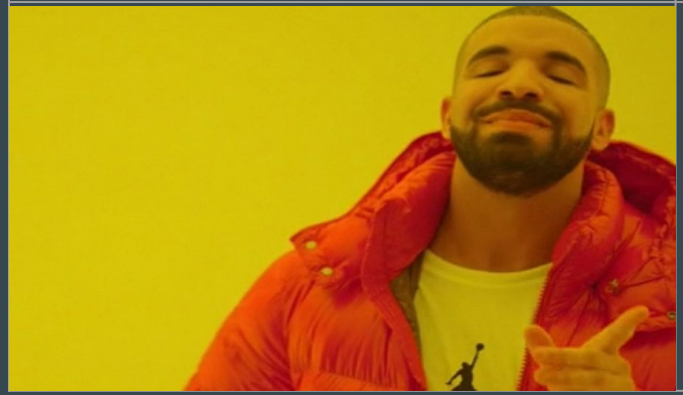
In general

- Only optimize **what needs to be** optimized (your **time** and **effort** are resources, too!).
- Choose the right **data structures**.
- Choose the right **algorithms**.

Strings

Strings

```
s = ""
for _ in iterable:
    s += variable_or_function
```



```
s = "".join(iterable)
```

Strings

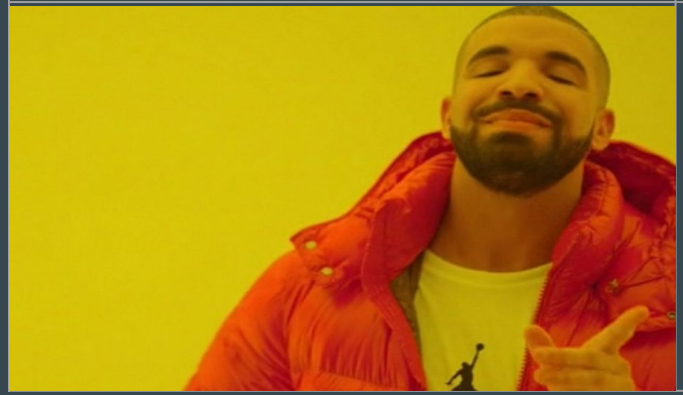
```
out = "<html>" + head + prologue + query + tail + "</html>"
```

```
out = "<html>{}{}{}{}</html>".format(head,prologue,query,tail)
```

Loops

Loops

```
newlist = []
for word in oldlist:
    newlist.append(word.upper())
```



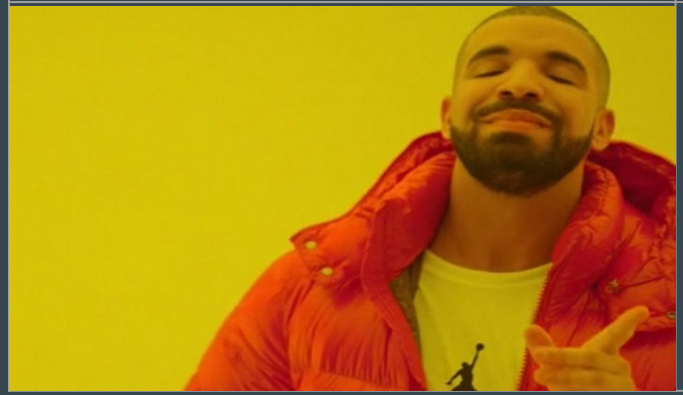
```
newlist = list(map(str.upper, oldlist))
```

```
newlist = [s.upper() for s in oldlist]
```

```
newgen = (s.upper() for s in oldlist)
```

Loops

```
newlist = []
for word in oldlist:
    newlist.append(word.upper())
```



```
upper = str.upper
newlist = []
append = newlist.append
for word in oldlist:
    append(upper(word))
```

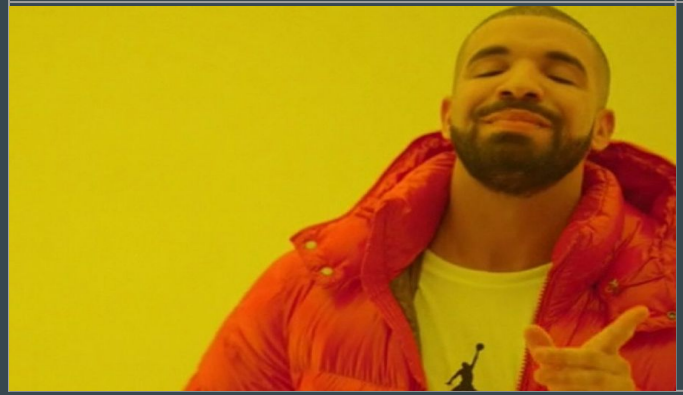
Loops

```
n = 100
while i<n:
  # Some code here
  i += 1
```

Variable Scope

Variable Scope

```
upper = str.upper
newlist = []
append = newlist.append
for word in oldlist:
    append(upper(word))
```

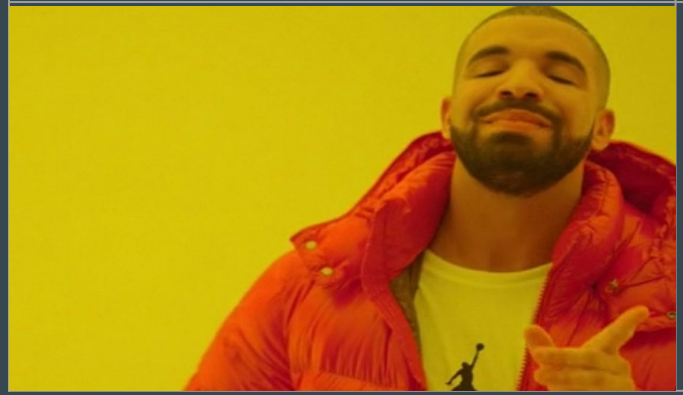


```
def func():
    upper = str.upper
    newlist = []
    append = newlist.append
    for word in oldlist:
        append(upper(word))
    return newlist
```

Initialization

Initialization

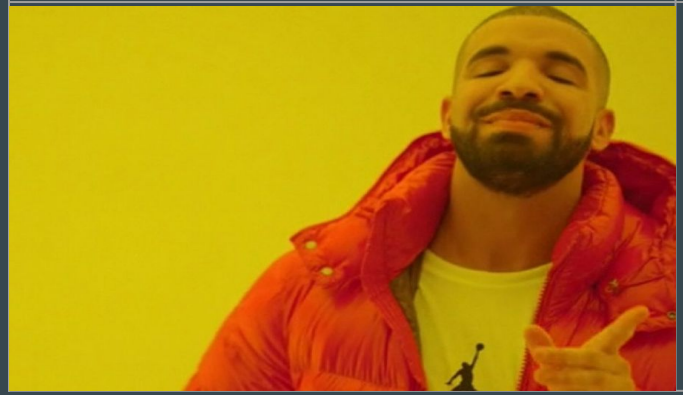
```
# Dict for storing frecuency of words
words_dict = {}
for word in words:
    if word not in words_dict:
        words_dict[word] = 0
    words_dict[word] += 1
```



```
# Dict for storing frecuency of words
words_dict = {}
for word in words:
    try:
        words_dict[word] += 1
    except KeyError:
        words_dict[word] = 1
```

Initialization

```
# Dict for storing frecuency of words
words_dict = {}
for word in words:
    if word not in words_dict:
        words_dict[word] = 0
    words_dict[word] += 1
```



```
# Dict for storing frecuency of words
words_dict = {}
get = words_dict.get
for word in words:
    words_dict[word] = get(word, 0) + 1
```

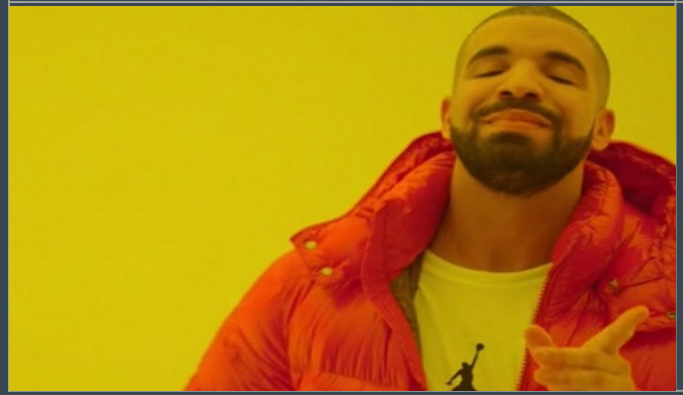
Imports

Imports

```
#Function to create a transposed DataFrame

def df(data):
  import pandas as pd
  return pd.DataFrame(data).T

for _ in range(n):
  df(data)
```



```
#Function to create a transposed DataFrame

import pandas as pd
def df(data):
  return pd.DataFrame(data).T

for _ in range(n):
  d = df(data)
```

Vectors

Vectors

```
import numpy as np
import time

t1 = time.time()
print(sum([i for i in range(100000)])/100000)
print((time.time()-t1))
```

```
49999.5
0.01303243637084961
```

```
import numpy as np
import time

t2 = time.time()
print(np.linspace(0, 100000, 99999).sum()/100000)
print((time.time()-t2))
```

```
49999.5
0.0028629302978515625
```

Library Design

Library Design

- **Cython**
- **CPython**
- Learn to detect whether

  Python is the **right choice**.

Demo

Questions?

Thank You!