How *Thinking* in Python Made Me a Better Software Engineer

EuroPython 2019

Johnny Dude

bit.ly/ThinkPy

Hi, I'm Johnny Dude

Software Engineer at TogaNetworks

Using Python at work since 2005

I use Python for ***prototyping***

Responsible for c++ ***production*** code

This is my first EuroPython talk

bit.ly/ThinkPy

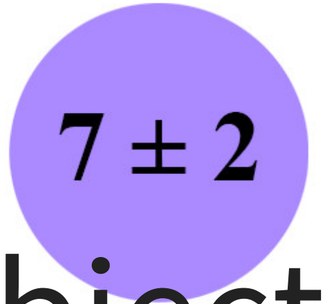
Outline

1. Psychological concepts
2. Relation to the development process
3. Experiment

Psychological Concepts

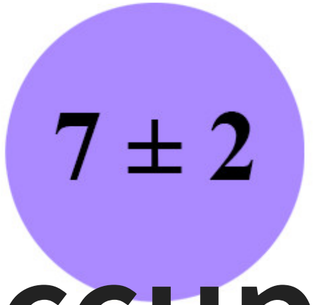
Trying **not** to think about something, makes thinking about it **more** likely

** Ironic Process Theory*



The number of objects an average human can hold in working memory is **7 ± 2**

** The Magical Number Seven, Plus or Minus Two*



Anything that **occupies** your working memory **reduces** your ability to think

** Thinking Fast and Slow*

Capital of France

Priming is a technique whereby exposure to one stimulus influences a response to a subsequent stimulus, without conscious guidance or intention

* *Thinking Fast and Slow*
* *The Priming Effect*

You **cannot** prevent it

Task switching **reduces** your productivity time

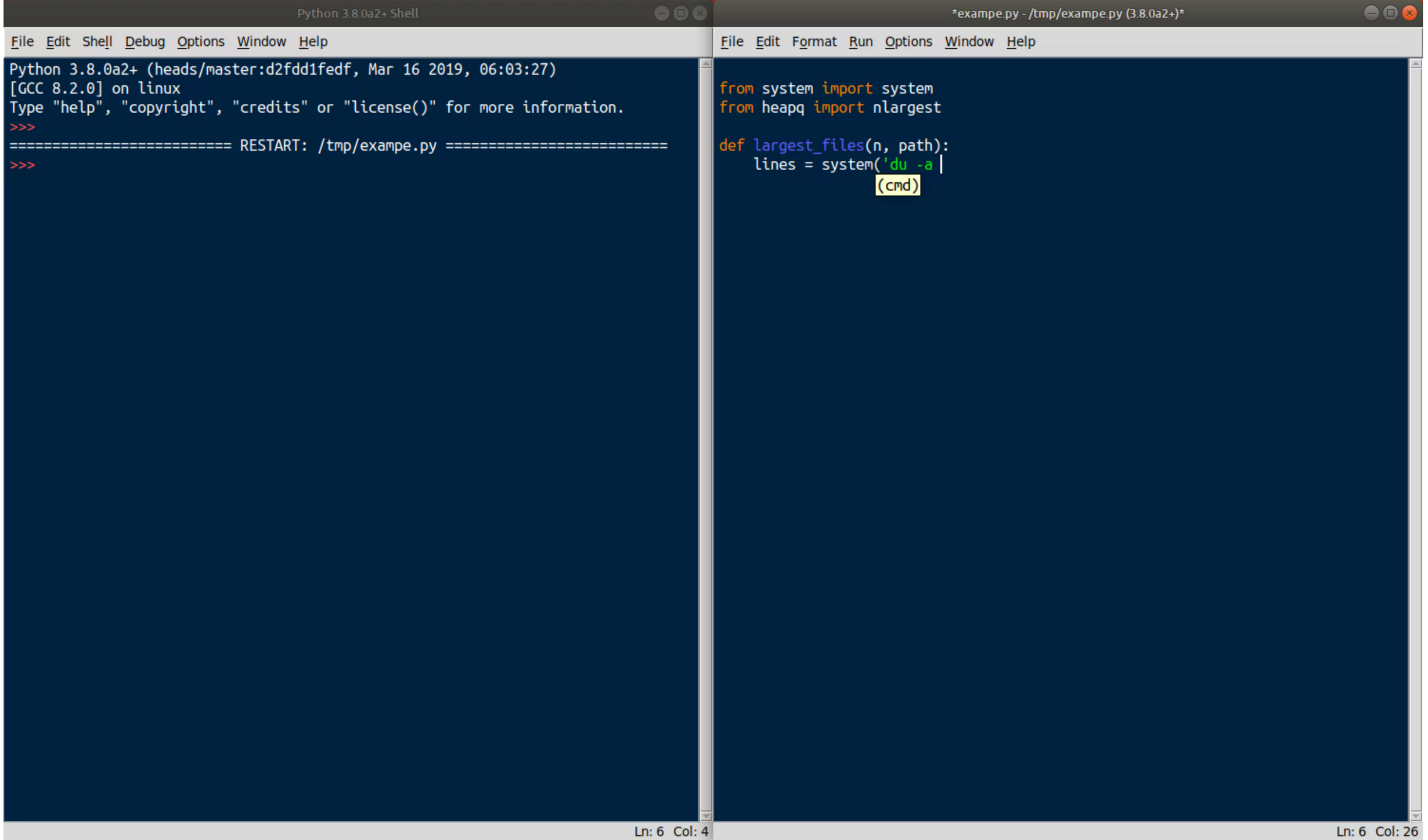
** Executive Control of Cognitive Processes in Task Switching*

Fluency is the ability
to do an activity with **little**, or
no conscious effort

7 ± 2

Relation to the Development Process

Immediate Feedback


Python 3.8.0a2+ Shell
File Edit Shell Debug Options Window Help
Python 3.8.0a2+ (heads/master:d2fdd1fedf, Mar 16 2019, 06:03:27)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: /tmp/exampe.py ==========
>>>
Ln: 6 Col: 4
exampe.py - /tmp/exampe.py (3.8.0a2+)
File Edit Format Run Options Window Help
from system import system
from heapq import nlargest
def largest_files(n, path):
lines = system('du -a
(cmd)
Ln: 6 Col: 26

```
def get_biggest_files(n, path='.'):
    lines = system(f'du -a {path}').splitlines()
    pairs = [line.split('\t') for line in lines]
    return [name for size, name in nlargest(n, pairs)]
```

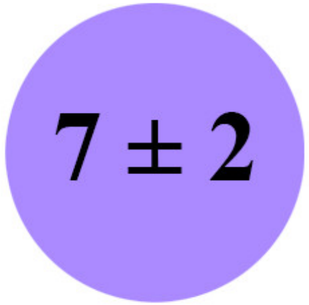
```
>>> n, path = 2, 'small folder'
>>> lines = system(f'du -a {path}').splitlines()
>>> lines[:2]
['8\t./darker.css', '32\t./index.html']
```

```
>>> n, path = 2, 'small folder'
>>> lines = system(f'du -a {path}').splitlines()
>>> lines[:2]
['8\t./darker.css', '32\t./index.html']
>>> pairs = [line.split('\t') for line in lines]
>>> pairs[:2]
[['8', './darker.css'], ['32', './index.html']]
```

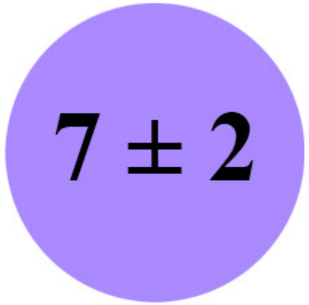
```
>>> n, path = 2, 'small folder'
>>> lines = system(f'du -a {path}').splitlines()
>>> lines[:2]
['8\t./darker.css', '32\t./index.html']
>>> pairs = [line.split('\t') for line in lines]
>>> pairs[:2]
[['8', './darker.css'], ['32', './index.html']]
>>> nlargest(n, pairs)
[['96', './.git/objects/d1/31af6800b725b05b...'],
 ['912', './images/repr.jpg'],
 ['900', './.git/objects/20']]
```

Learn faster

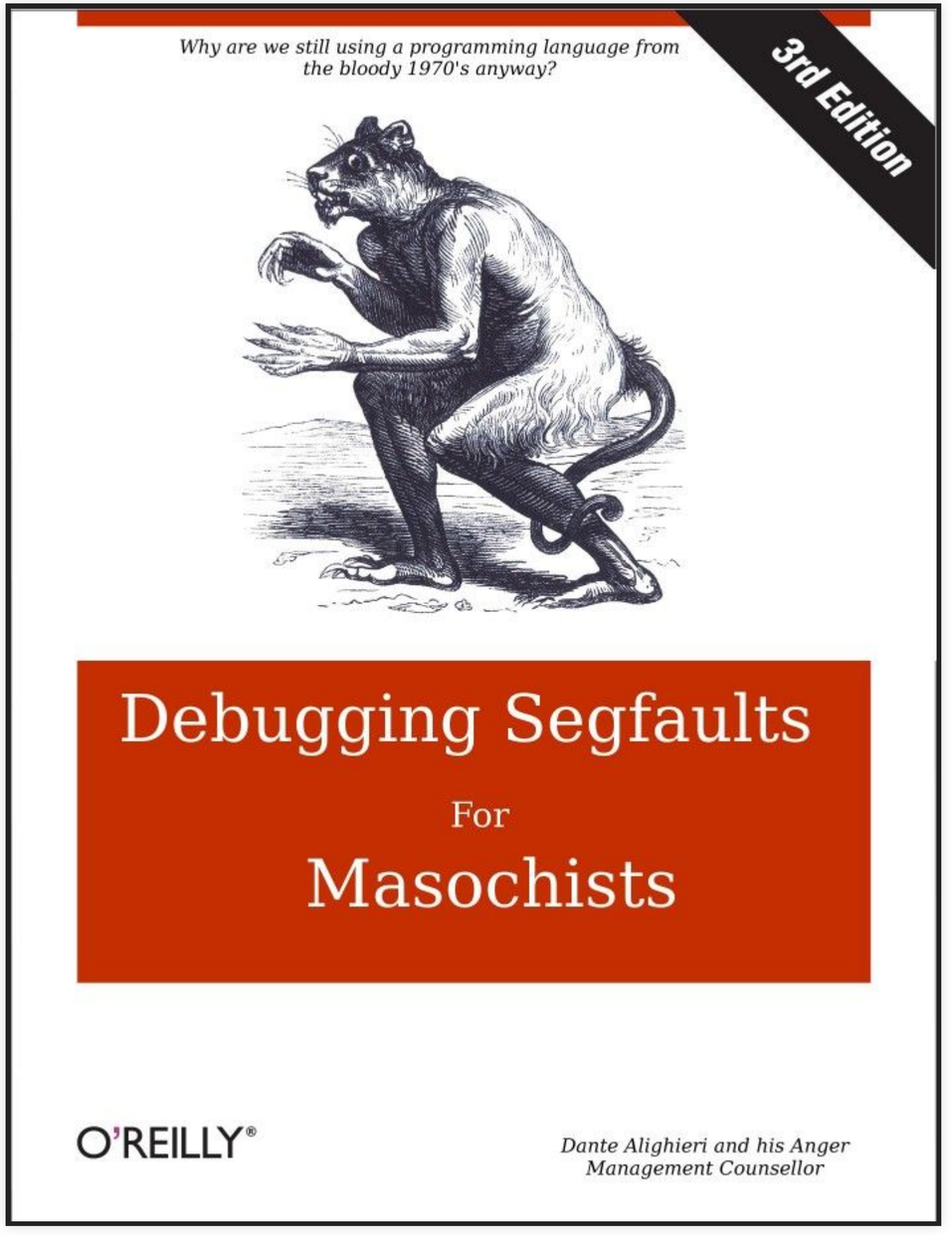
Catching bugs earlier **reduces** task switching



Confidence that your code works,



Confidence that your code works, without concious effort


Why are we still using a programming language from the bloody 1970's anyway?
3rd Edition
Debugging Segfaults
For
Masochists
O'REILLY®
Dante Alighieri and his Anger Management Counsellor

7 ± 2

Standard Representation

```
[
  { "name": "Tyler Durden",
    "age": 35,
    "sibling":[]},
  { "name": "Brad Pitt",
    "age": 56,
    "sibling":["Doug", "Julie"]},
  { "name": "Mia Wallace",
    "age": 25,
    "sibling":[]},
  { "name": "Uma Thurman",
    "age": 49,
    "sibling":[ "Dechen", "Taya", "Ganden", "Mipam"]},
]
```

A list of strings, optimized for
filtering items matching a regular expression

A list of strings, optimized for
filtering items matching a regular expression

```
"Tyler Durden\nBrad Pitt\nMia Wallace\nUma Thurman\n"
```

A list of strings, optimized for
filtering items matching a regular expression

```
"Tyler Durden\nBrad Pitt\nMia Wallace\nUma Thurman\n"
```

```
["Tyler Durden", "Brad Pitt", "Mia Wallace", "Uma Thurman"]
```

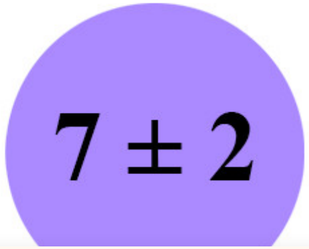
A dictionary with keys that can be searched by regular expression.

```
"Brad Pitt\nMia Wallace\nTyler Durden\nUma Thurman\n"
{ 22: "Dead",
  35: "Alive",
  0: "Alive",
  10: "Alive" }
```

A dictionary with keys that can be searched by regular expression.

```
"Brad Pitt\nMia Wallace\nTyler Durden\nUma Thurman\n"
{ 22: "Dead",
  35: "Alive",
  0: "Alive",
  10: "Alive" }
```

```
{ "Tyler Durden": "Dead",
  "Uma Thurman": "Alive",
  "Brad Pitt": "Alive",
  "Mia Wallace": "Alive" }
```




```
(gdb) p my_dict
$1 = {
  keys = "Brad Pitt\nMia Wallace\nTyler Durden\nUma T
hurman\n",
  hash_table = std::unordered_map with 4 elements = {
    [10] = PersonState::alive,
    [0] = PersonState::alive,
    [35] = PersonState::alive,
    [22] = PersonState::dead}}
```

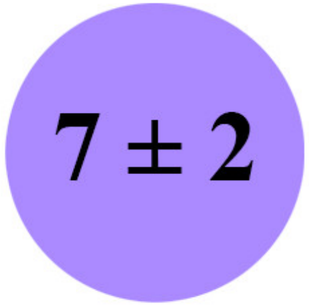
```
{ "Tyler Durden":
    <PersonState object at 0x7fd2622fbd60>,
  "Uma Thurman":
    <PersonState object at 0x7fd2622fbc40>,
  "Brad Pitt":
    <PersonState object at 0x7fd26231a160>,
  "Mia Wallace":
    <PersonState object at 0x7fd26231a100> }
```

```
{ "Tyler Durden":
    <PersonState object at 0x7fd2622fbd60>,
  "Uma Thurman":
    <PersonState object at 0x7fd2622fbc40>,
  "Brad Pitt":
    <PersonState object at 0x7fd26231a160>,
  "Mia Wallace":
    <PersonState object at 0x7fd26231a100> }
```

```
{ "Tyler Durden": PersonState(0),
  "Uma Thurman": PersonState(1),
  "Brad Pitt": PersonState(1),
  "Mia Wallace": PersonState(1) }
```

If you can read it
then you can visualize it, think about it,
and discuss it with other developers

Standard API




```
Counter({
   "Walking Dead": 19,
   "Alive": 7,
   "Dead": 2,
   "Not Born": 1,
})
```

{ }

I want to store something in a **dictionary**...

I want to store something in a std::**map**...

Composability

```
def f(nums):
    return [str(n) for n in sorted(nums) if valid(n)]
```

```
def f(nums):
    return [str(n) for n in sorted(nums) if valid(n)]
```

```
def f(nums):
    xs = list(nums)
    sort(xs)
    ys = filter(valid, xs)
    zs = map(str, ys)
    return zs
```

```
Mix the ingredients in a bowl.
Pour the bowl’s contents into a mould.
Bake the mould along with its content.
```

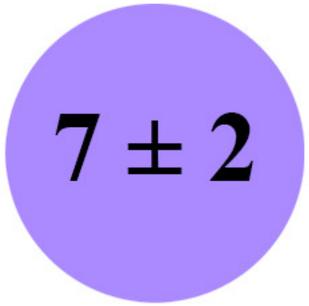
```
Mix the ingredients in a bowl.
Pour the bowl’s contents into a mould.
Bake the mould along with its content.
```

```
Bake the mixed ingredients.
```

```
def f(nums):
    return [str(n) for n in sorted(nums) if valid(n)]
```

```
def f(nums):
    xs = list(nums)
    sort(xs)
    ys = filter(valid, xs)
    zs = map(str, ys)
    return zs
```

```cpp
vector<string> f(const vector<int>& nums) {
  vector<int> xs = nums;
  sort(xs.begin(), xs.end());

  vector<int> ys;
  copy_if(
    xs.begin(),
    xs.end(),
    back_inserter(ys),
    valid
  );

  vector<string> zs;
  transform(
    ys.begin(),
    ys.end(),
    back_inserter(zs),
    [](int n){ return to_string(n); }
  );

  return zs;
}
```



It is easy to think
with **composable** tools

Simple is better than Complicated

```
void f(Object obj)                  // pass by value
void f(Object& obj)                 // pass by reference
void f(Object* obj)                 // pass by raw pointer
void f(Object&& obj)                // pass by rvalue
void f(shared_ptr<Object> obj) // pass by shared pointer
void f(unique_ptr<Object> obj) // pass by unique pointer
```

Simple is better than complex

```
void f(Object obj)                  // pass by value
void f(Object& obj)                 // pass by reference
void f(Object* obj)                 // pass by raw pointer
void f(Object&& obj)                // pass by rvalue
void f(shared_ptr<Object> obj) // pass by shared pointer
void f(unique_ptr<Object> obj) // pass by unique pointer
```

```
void f(Object obj)                // pass by value
void f(Object& obj)               // pass by reference
void f(Object* obj)               // pass by raw pointer
void f(Object&& obj)              // pass by rvalue
void f(shared_ptr<Object> obj) // pass by shared pointer
void f(unique_ptr<Object> obj) // pass by unique pointer
```

```
void f(shared_ptr<Object> or unique_ptr<Object> obj) // ??
```

```
void f(Object obj)                // pass by value
void f(Object& obj)               // pass by reference
void f(Object* obj)               // pass by raw pointer
void f(Object&& obj)              // pass by rvalue
void f(shared_ptr<Object> obj)    // pass by shared pointer
void f(unique_ptr<Object> obj)    // pass by unique pointer
```

```
void f(shared_ptr<Object> or unique_ptr<Object> obj) // ??
```

```
void f(const Object* obj)          // object is immutable
void f(Object* const obj)          // pointer is immutable
void f(const Object* const obj)    // both are immutable
```

```
void f(Object obj)                // pass by value
void f(Object& obj)               // pass by reference
void f(Object* obj)               // pass by raw pointer
void f(Object&& obj)              // pass by rvalue
void f(shared_ptr<Object> obj) // pass by shared pointer
void f(unique_ptr<Object> obj) // pass by unique pointer
```

```
void f(shared_ptr<Object> or unique_ptr<Object> obj) // ??
```

```
void f(const Object* obj)         // object is immutable
void f(Object* const obj)         // pointer is immutable
void f(const Object* const obj) // both are immutable
```

```
void f(Object const* obj)          // what is immutable?
```

Complex is better than complicated

```
void f(Object obj)                  // pass by value
void f(Object& obj)                 // pass by reference
void f(Object* obj)                 // pass by raw pointer
void f(Object&& obj)                // pass by rvalue
void f(shared_ptr<Object> obj) // pass by shared pointer
void f(unique_ptr<Object> obj) // pass by unique pointer
```

```
void f(shared_ptr<Object> or unique_ptr<Object> obj) // ??
```

```
void f(const Object* obj)           // object is immutable
void f(Object* const obj)           // pointer is immutable
void f(const Object* const obj) // both are immutable
```

```
void f(Object const* obj)           // what is immutable?
```

```
void f(shared_ptr<Object>& obj) // pass shared pointer
                                // by reference
```

```
void f(Object obj)                // pass by value
void f(Object& obj)               // pass by reference
void f(Object* obj)               // pass by raw pointer
void f(Object&& obj)              // pass by rvalue
void f(shared_ptr<Object> obj)    // pass by shared pointer
void f(unique_ptr<Object> obj)    // pass by unique pointer
```

```
void f(shared_ptr<Object> or unique_ptr<Object> obj) // ??
```

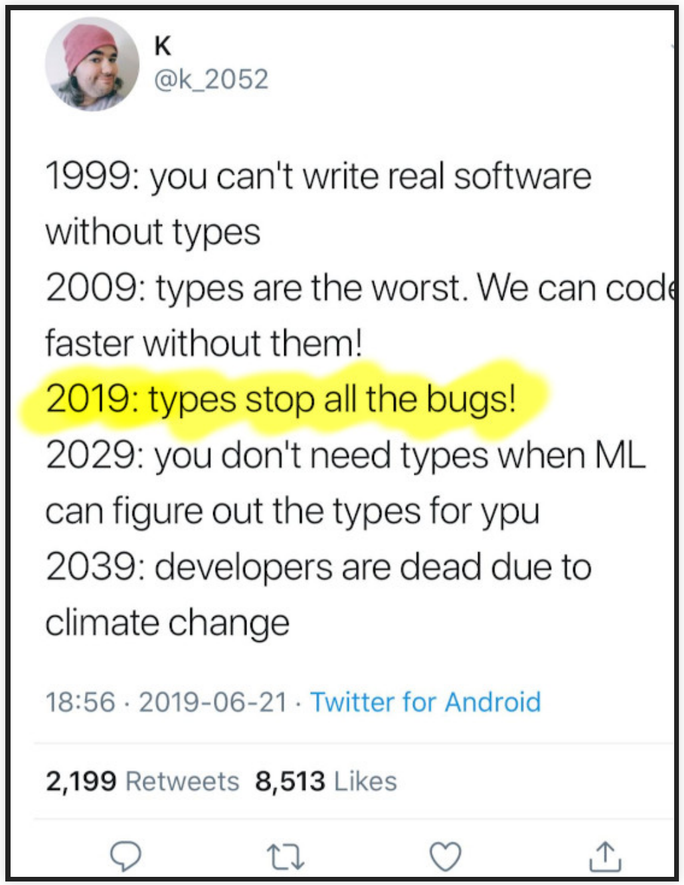
```
void f(const Object* obj)         // object is immutable
void f(Object* const obj)         // pointer is immutable
void f(const Object* const obj)   // both are immutable
```

```
void f(Object const* obj)         // what is immutable?
```

We can use shared pointers everywhere

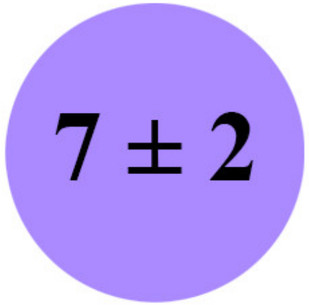
But, we **cannot** stop ***thinking*** about...

Type Hints


K
@k_2052
1999: you can't write real software without types
2009: types are the worst. We can code faster without them!
2019: types stop all the bugs!
2029: you don't need types when ML can figure out the types for ypu
2039: developers are dead due to climate change
18:56 · 2019-06-21 · Twitter for Android
2,199 Retweets 8,513 Likes

Do we realy want to
define types and structures before
understanding the problem
and the solution?

Constantly task switching between: Coding and Type-defining



How many bits would
I like this integer to have?

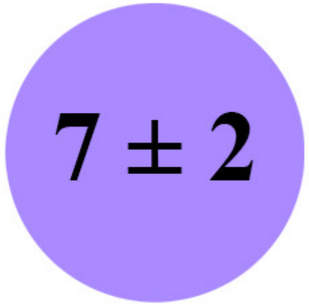
What happens when you are wrong?

Lets just use int,
and deal with it later.

Prototyping

Prototype is a model
built to **test** a concept,
and to be **learned** from

You write it once,
gaining experience in both
understanding the problem, and
understanding a solution



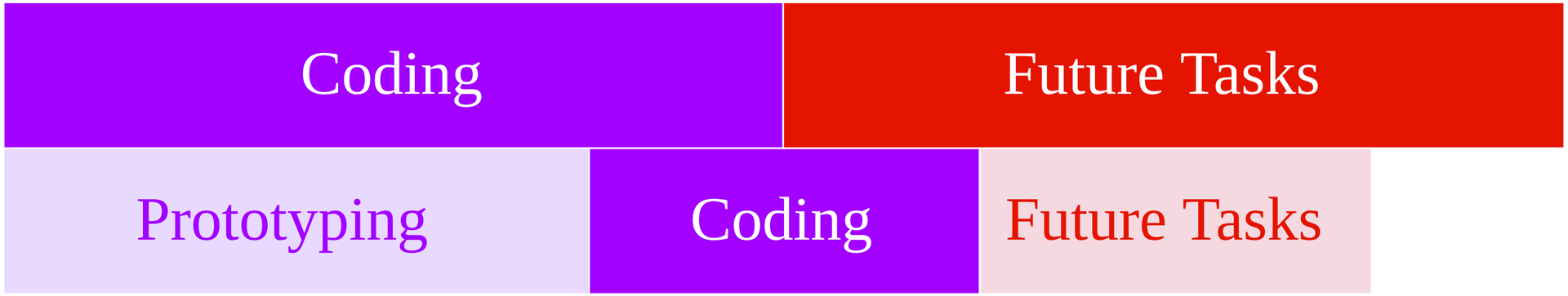
You write it again,
with less things to worry about
and attention to finer details

Improved Readability

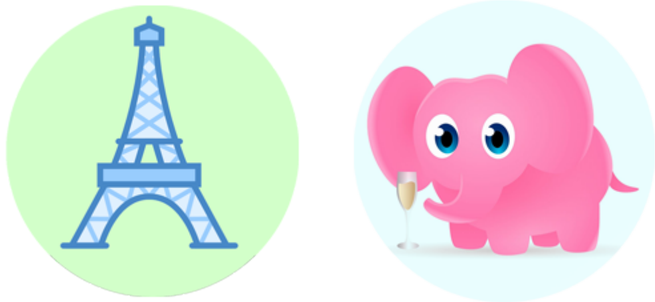
Improved Maintainability

Fewer Bugs


Coding
Prototyping
Coding


Coding
Future Tasks
Prototyping
Coding
Future Tasks

Some things you can do only in Python



Use a dictionary

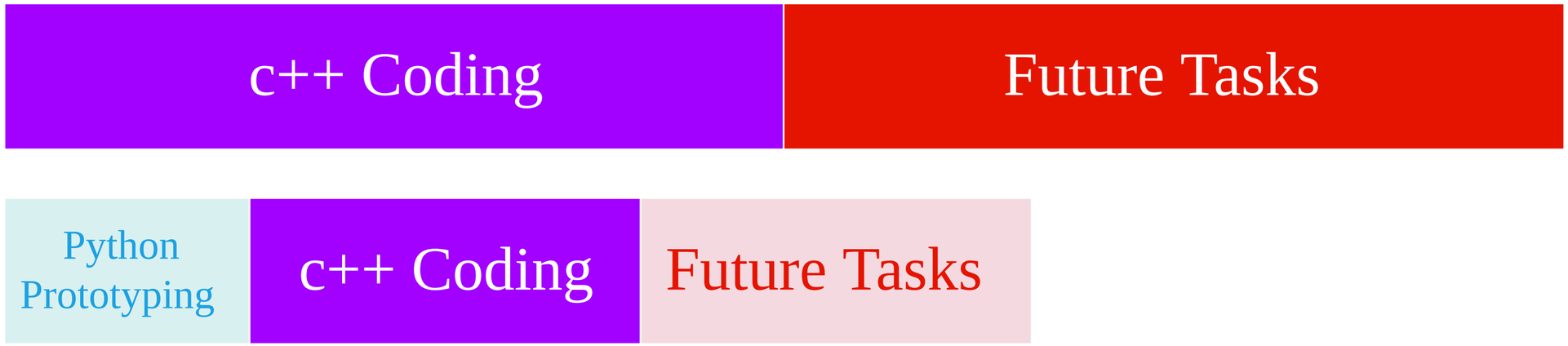
Define a function

7 ± 2

Think in the language you write

Handle type checking, seperately

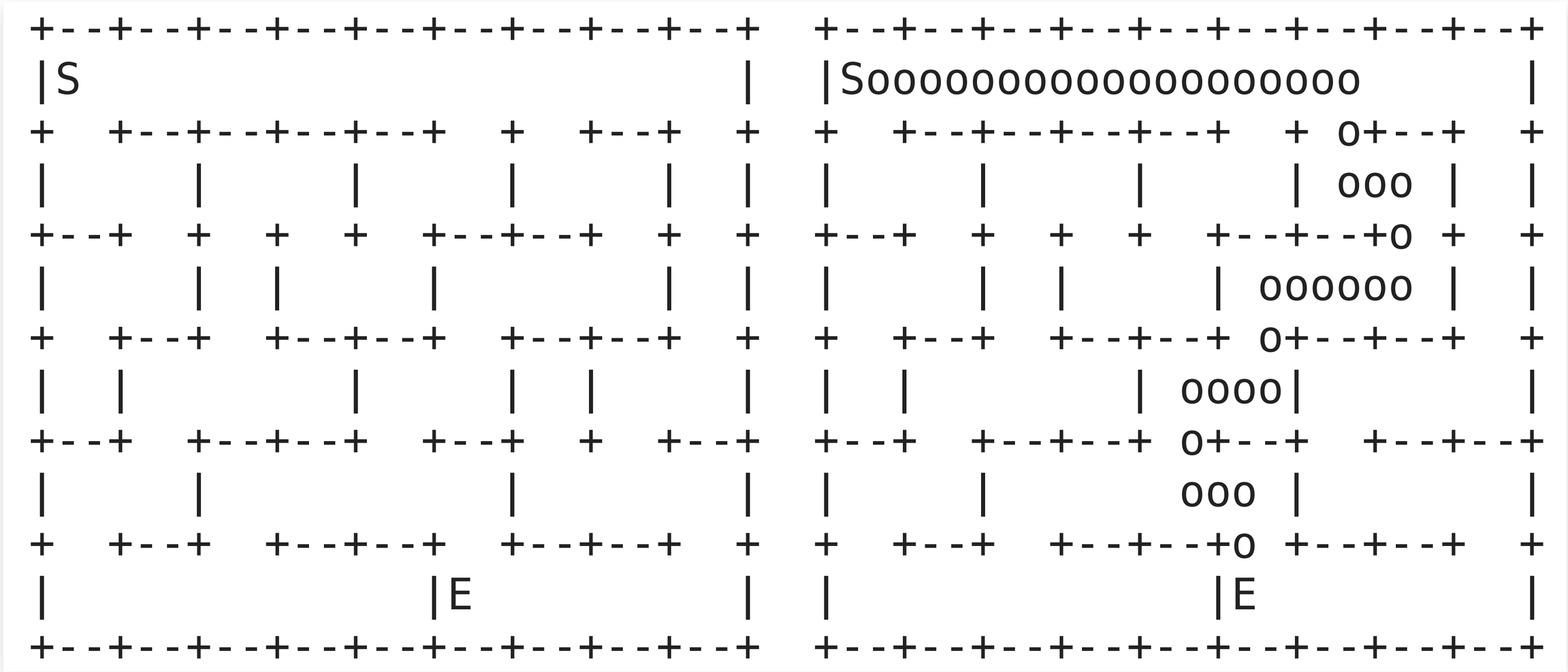
Along with many other reasons.



An empirical comparison of c, c++, java, perl, python, ...

How much of the speedup do we get from ***thinking*** faster?

My Experiment


S
E
Sooooooooooooooooooo
o
ooo
o
ooooo
o
oooo
o
ooo
o
E

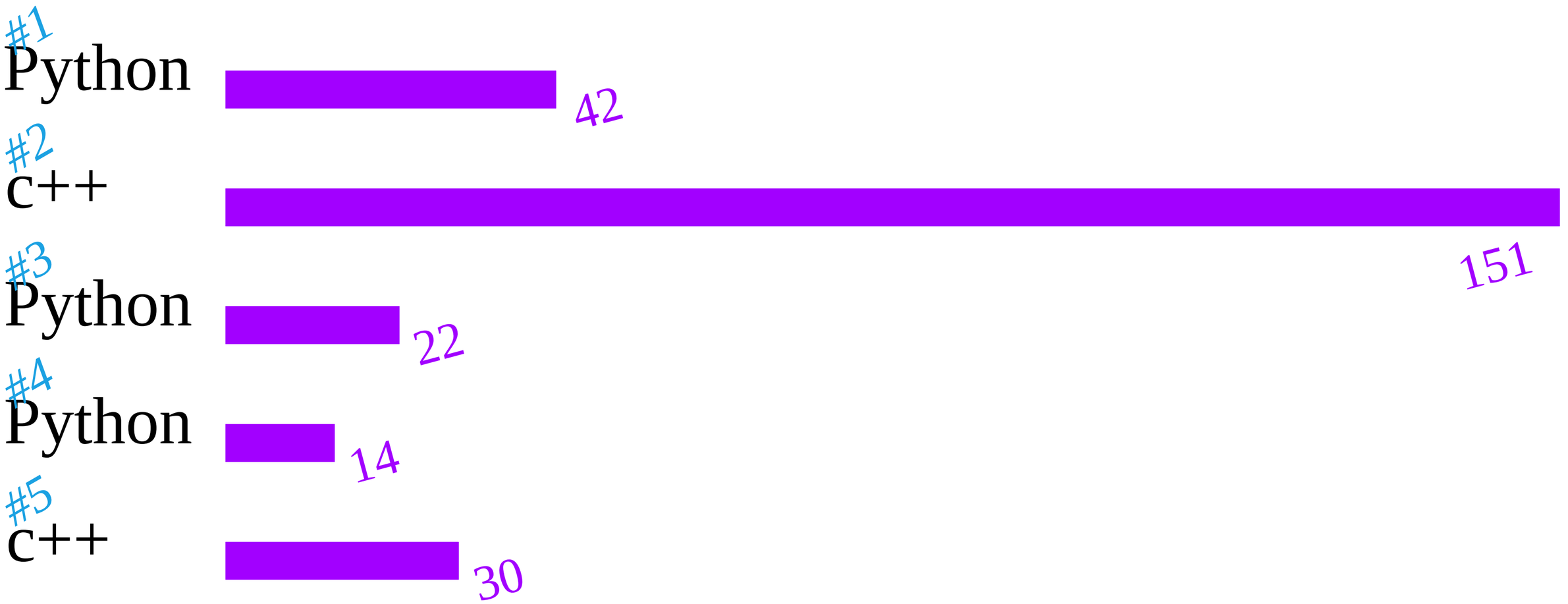
#1
Python

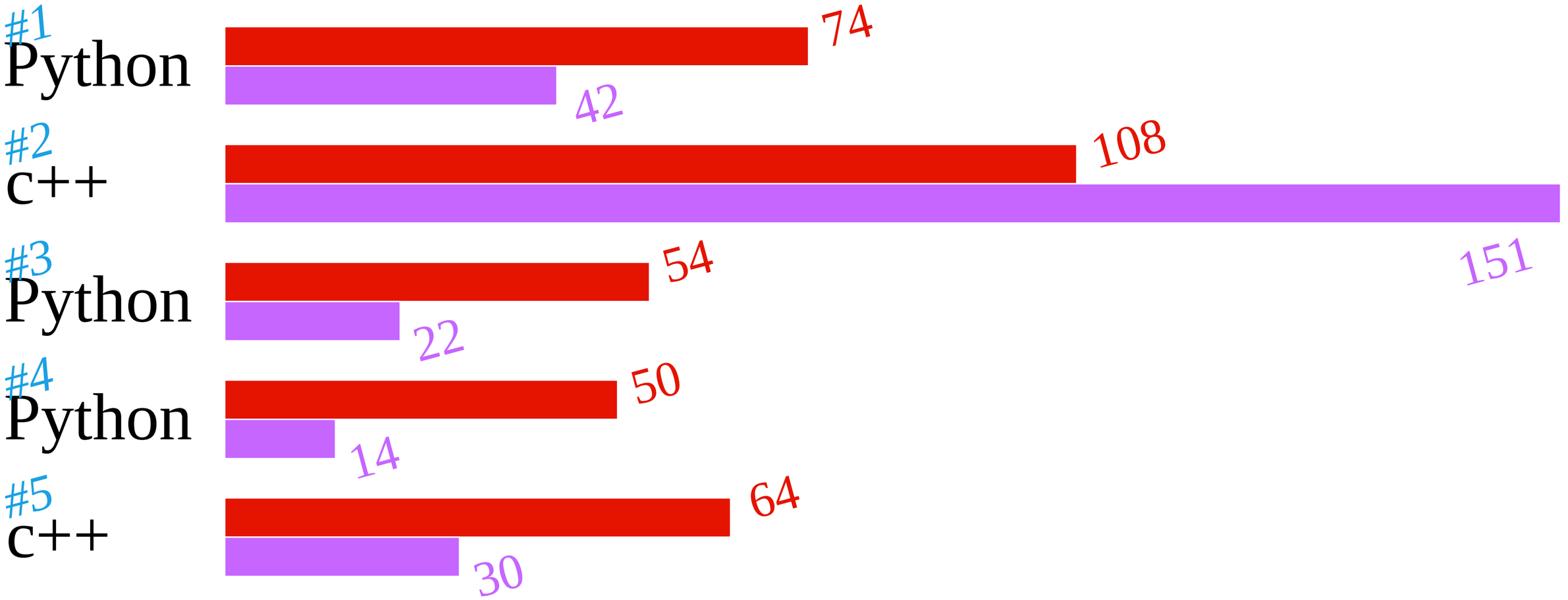
#2
c++

#3
Python

#4
Python

#5
c++


#1
Python
42
#2
c++
151
#3
Python
22
#4
Python
14
#5
c++
30
Work Time in Minutes



Work Time in Minutes

Source Lines of Code

Excluding: comment, empty lines, bracelets

#4

Python

50

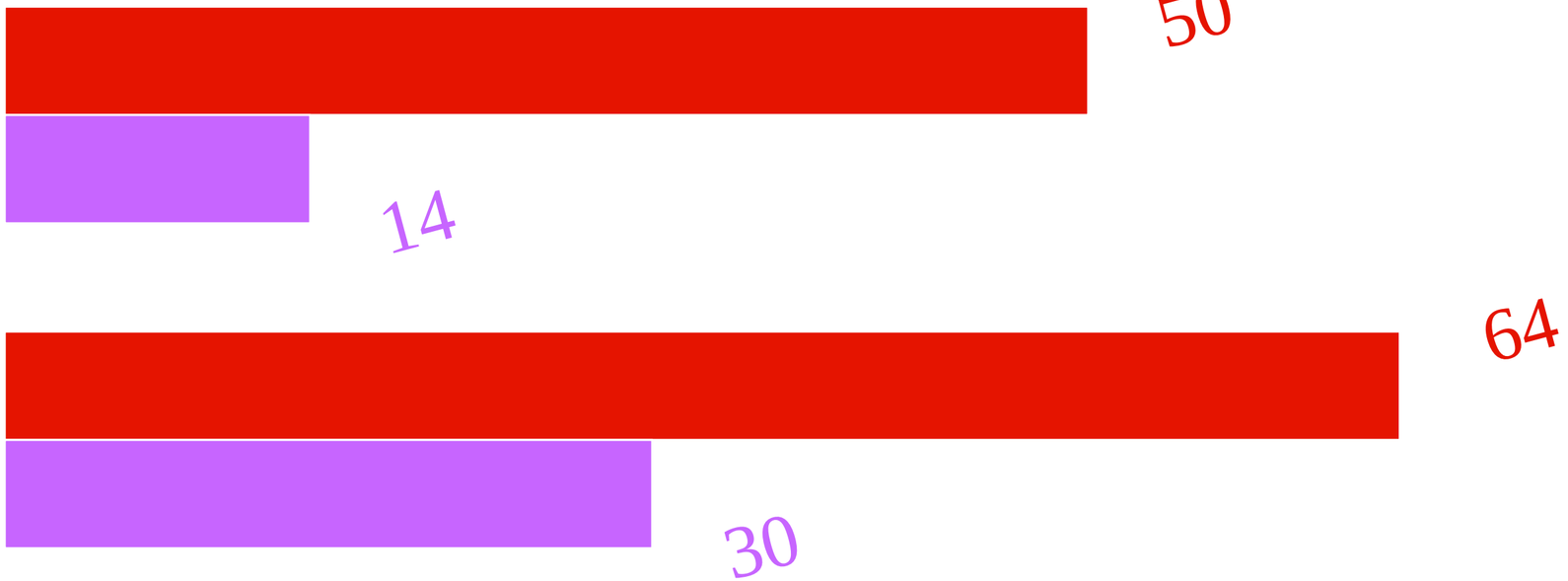
14

#5

c++

64

30

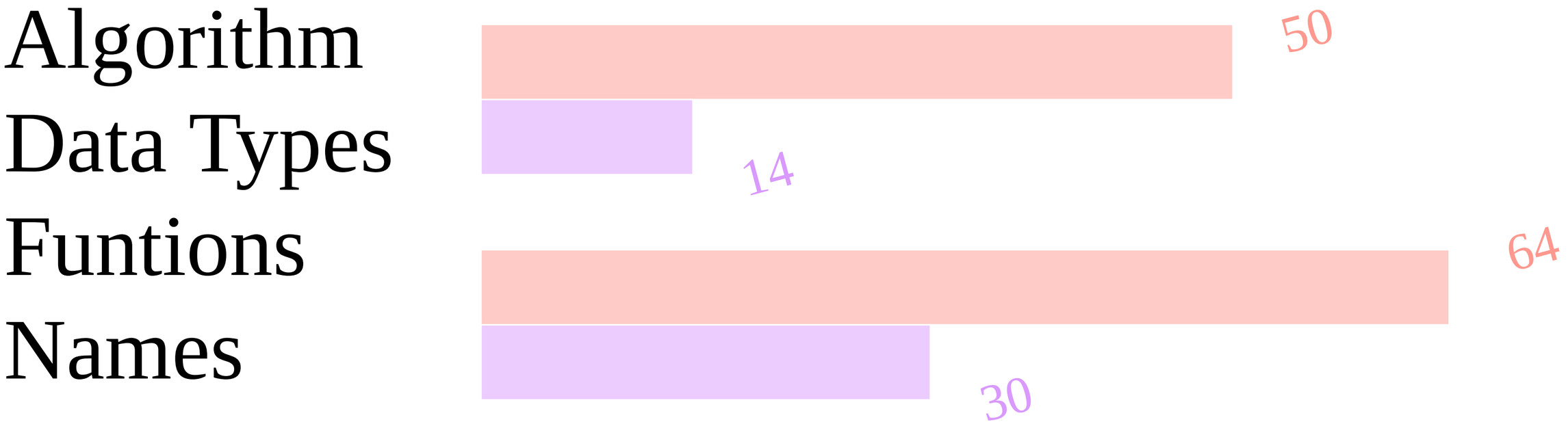


Work Time in Minutes

Source Lines of Code

Excluding: comment, empty lines, bracelets

Both version have exactly the same

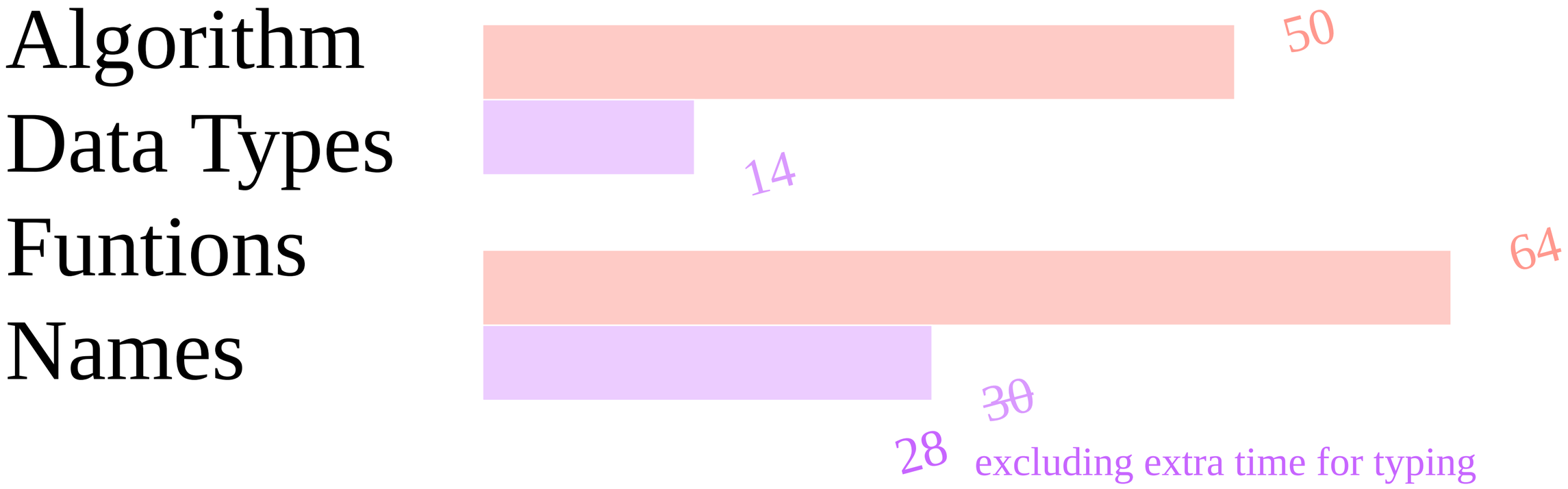


Work Time in Minutes

Source Lines of Code

Excluding: comment, empty lines, bracelets

Both version have exactly the same



Excluding: comment, empty lines, bracelets

```
set<Point> calc_path(map<Point, Point> prevs, Point point) {
    set<Point> results;
    point = prevs[point];
    while (prevs.find(point) != prevs.end()) {
        results.insert(point);
        point = prevs[point]; }
    return results; }

auto points = calc_path(prevs, end_point);
```

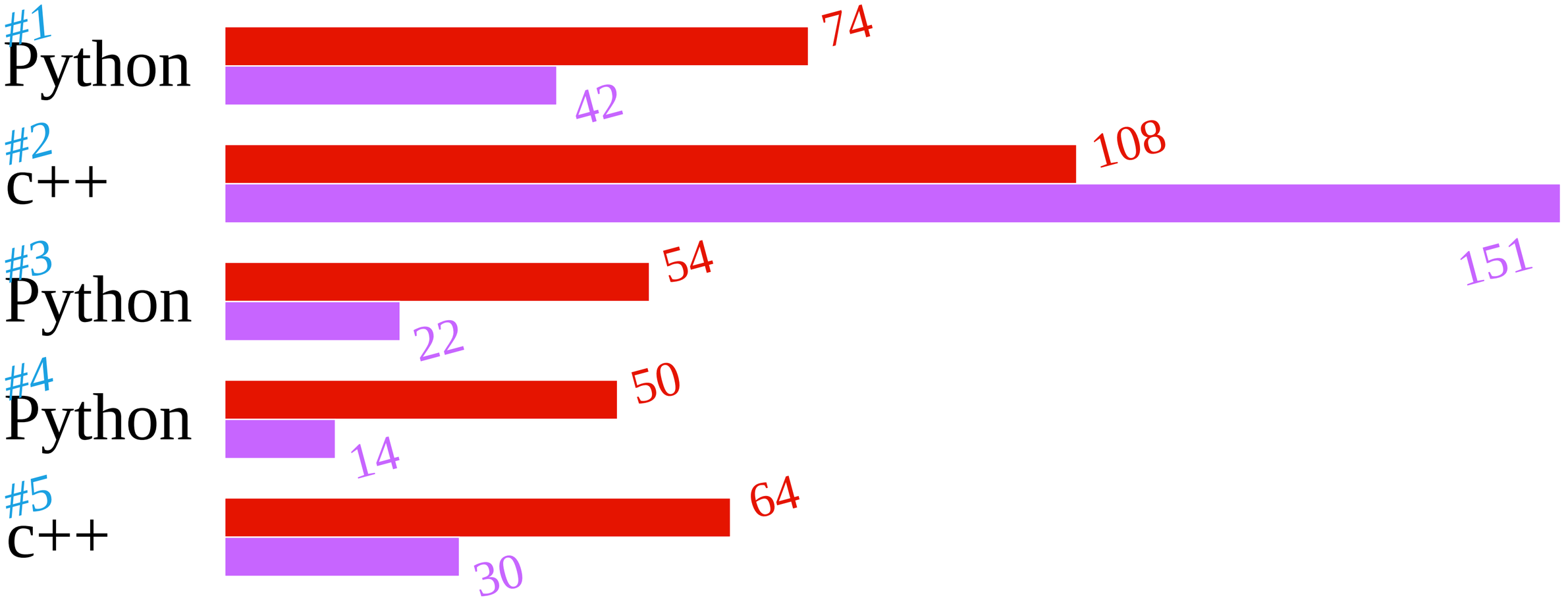
```
def calc_path(prevs, point):
    point = prevs[point]
    while point in prevs:
        yield point
        point = prevs[point]

points = set(calc_path(prevs, end_point))
```

Why?

What was I *Thinking* About?

7 ± 2

Experiment,
it's fun



Work Time in Minutes

Source Lines of Code

Excluding: comment, empty lines, bracelets

Summary

Immediate Feedback

Standard Representation & API

Composability

Prototype in Python

Summary

Immediate Feedback

Standard Representation & API

Composability

Prototype in Python

Summary

Immediate Feedback

Standard Representation & API

Composability

Prototype in Python

Summary

Immediate Feedback

Standard Representation & API

Composability

Prototype in Python

Summary

Immediate Feedback

Standard Representation & API

Composability

Prototype in Python

Think about the way you think

Think in Python

Experiment, it's fun!

Think about the way you think

Think in Python

Experiment, it's fun!

Think about the way you think

Think in Python

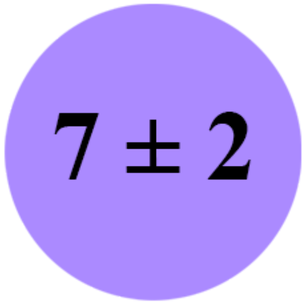
Experiment, it's fun!

Think about the way you think

Think in Python

Experiment, it's fun!

Think in Python



Thank You

Twitter: @DudeJohnny1219

email: johnny.dude@gmail.com

bit.ly/ThinkPy

**Many thanks to those who made this talk possible,
without thier help this talk might not be worth listening to.**

Nobuko Sano (佐野信子), Elizabeth Firman,

Michael Hirsch, Aharon Broduch, Eran Galon, Kobi and Suzi Lidershnider,

Boris Liberman, Ariel Weinstein, Omer Anson, Eran Galon, Aviv Kuvent, Eddy Duer, Meital Bar-Kana, Yaron Mor

References and Inspirations

Bret Victor gave an amazing talk ("Inventing on Principle") in which he mentioned immidiate reaction.

https://vimeo.com/36579366

I first learned about the magical number 7 from the famous post of **Glyph** about threading module complexities.

https://glyph.twistedmatrix.com/2014/02/unyielding.html

Alan Kay have many talks explaining science, human, machines, learning, teaching, and combining it all together.

I actually started reading "Thinking fast and slow" of **Daniel Kahneman** last month and decided to take a couple ofcouple of very good points from the first half of this book.

One of the papers about task switching I happend to find. It talks about many interesting experiments on task switiching

https://www.apa.org/pubs/journals/releases/xhp274763.pdf
Rubinstein, J. S., Meyer, D. E. & Evans, J. E. (2001). Executive Control of Cognitive Processes in Task Switching. Journal of Experimental Psychology: Human Perception and Performance, 27, 763-797.

The empirical research I like, it shows a lot of measurements comparing programing languages, it is old, but I believe nothing major changed since then. There are other reseach that shows similar results in different domains, and with different methods. (like analyzing github repositories.)

https://page.mi.fu-berlin.de/prechelt/Biblio/jccpprtTR.pdf
Prechelt, Lutz. (2000). An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program.
