Topic 1, Part 1: Introduction to Python for Computer Vision

Improves Work for Everyone

Improves Work for Everyone

Widely used top programming language

Improves Work for Everyone

Widely used top programming language

Huge growing ecosystem due to its open source nature

Improves Work for Everyone

Widely used top programming language

Huge growing ecosystem due to its open source nature

Almost every industry is on board

Descriptive Analytics and Dashboards

Descriptive Analytics and Dashboards

Exploratory data analysis

Descriptive Analytics and Dashboards

Exploratory data analysis

Manipulation of data

Descriptive Analytics and Dashboards

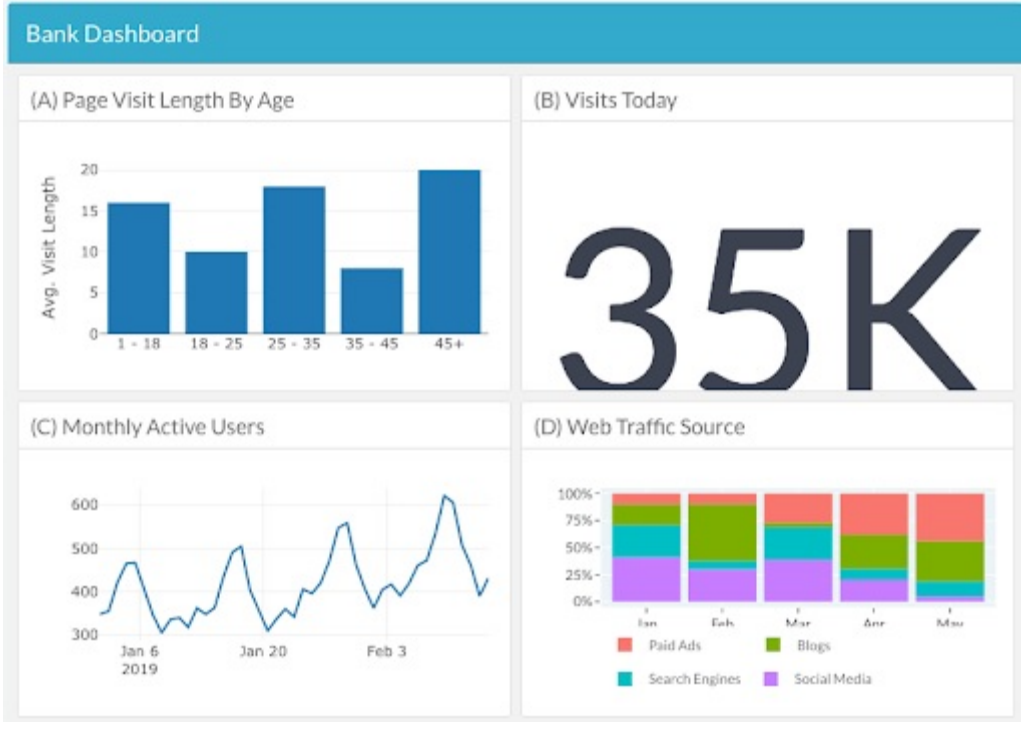
Exploratory data analysis

Manipulation of data

Streamline work flows

Creating visualisations/dashboards (i.e. plotly, streamlit)

Creating visualisations/dashboards (i.e. plotly, streamlit)



Machine Learning

Machine Learning

Predicting and classifying new data

Machine Learning

Predicting and classifying new data

Recommender systems

Machine Learning

Predicting and classifying new data

Recommender systems

Can work with popular Google machine learning libraries (such as Tesseract and Tensorflow)

Predictive/Prescriptive Analytics

Predictive/Prescriptive Analytics

Decision science

- Anticipate what, when and why certain outcome will happen
- What to do with information

Predictive/Prescriptive Analytics

Decision science

- Anticipate what, when and why certain outcome will happen
- What to do with information

Deep learning to optimise outcomes

Some statistics

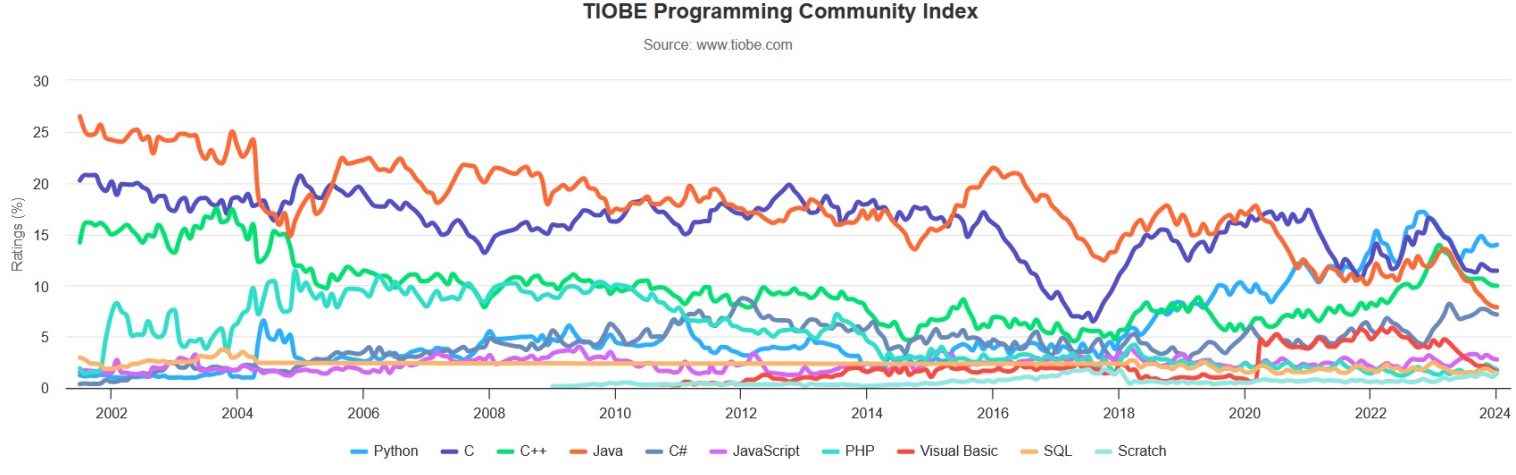
Popularity

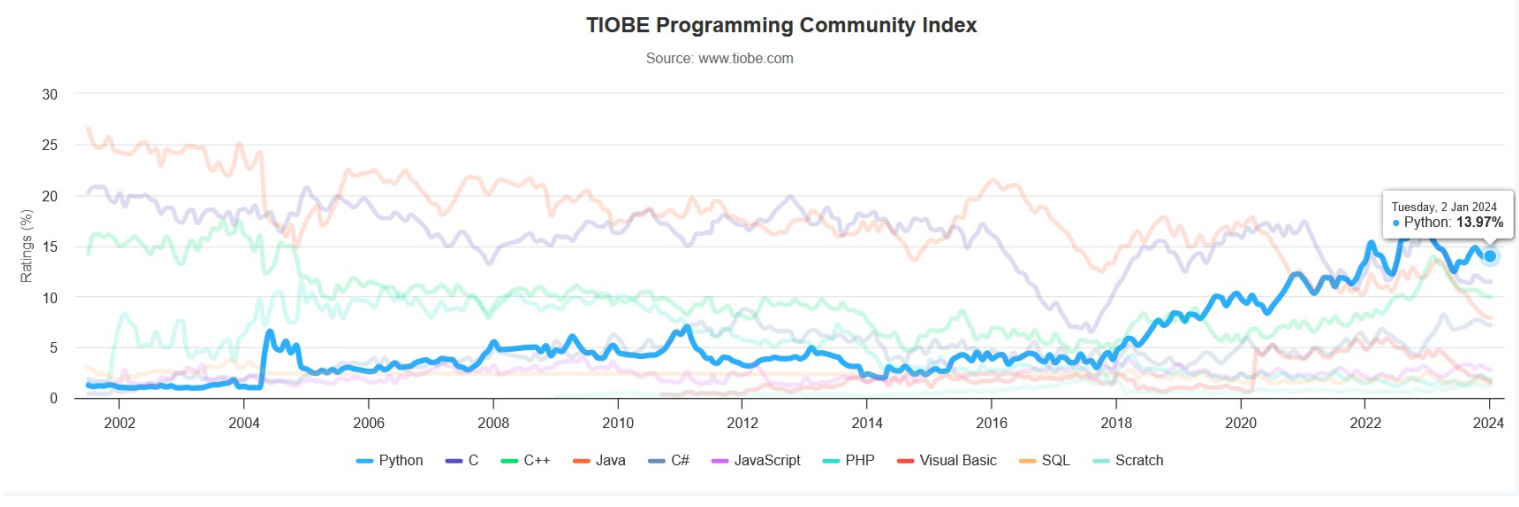
Popularity

Most popular language according to the TIOBE index

May 2024	May 2023	Change	Programming Language	Ratings	Change
1	1		Python	16.33%	+2.88%
2	2		C	9.98%	-3.37%
3	4	↑	C++	9.53%	-2.43%
4	3	↓	Java	8.69%	-3.53%
5	5		C#	6.49%	-0.94%
6	7	↑	JavaScript	3.01%	+0.57%
7	6	↓	Visual Basic	2.01%	-1.83%
8	12	⇈	Go	1.60%	+0.61%
9	9		SQL	1.44%	-0.03%
10	19	⇈	Fortran	1.24%	+0.46%

Fastest growing for the current year

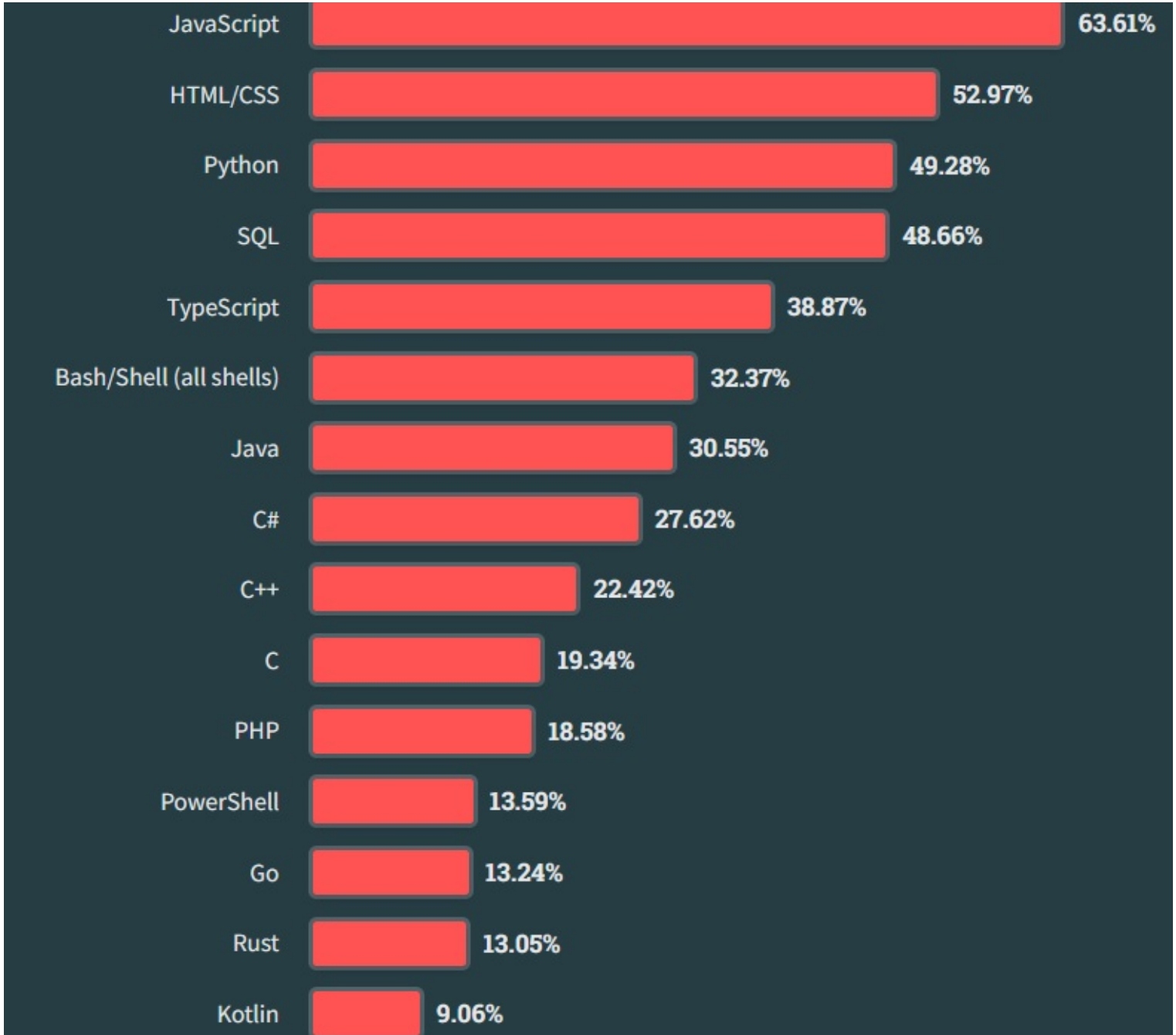

TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
2024
Python
C
C++
Java
C#
JavaScript
PHP
Visual Basic
SQL
Scratch


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
2024
Tuesday, 2 Jan 2024
Python: 13.97%
Python
C
C++
Java
C#
JavaScript
PHP
Visual Basic
SQL
Scratch

According to Stackoverflow, it is the 3rd most popular programming language in the world!

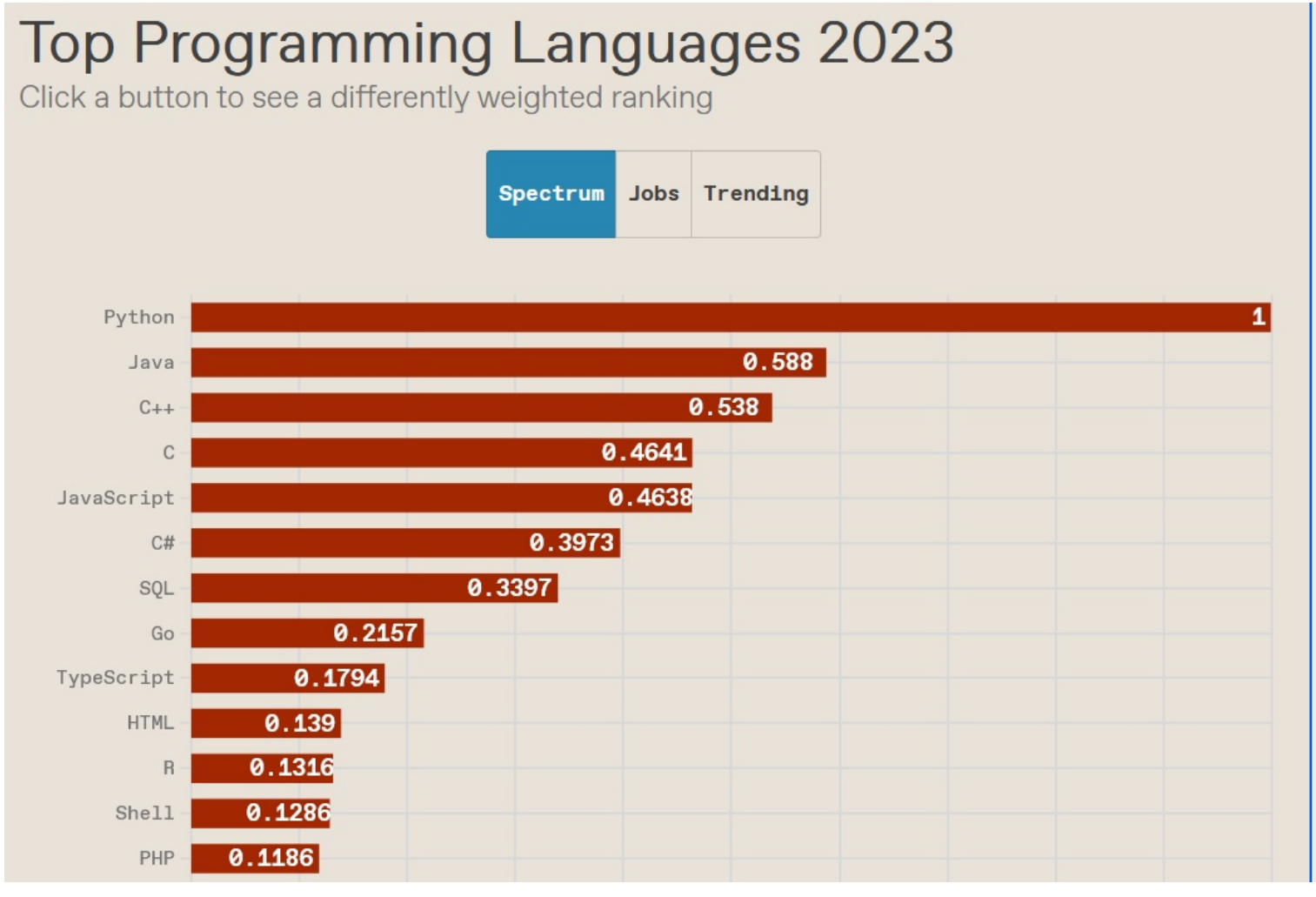
According to Stackoverflow, it is the 3rd most popular programming language in the world!

Both for general public and for professional developers


JavaScript 63.61%
HTML/CSS 52.97%
Python 49.28%
SQL 48.66%
TypeScript 38.87%
Bash/Shell (all shells) 32.37%
Java 30.55%
C# 27.62%
C++ 22.42%
C 19.34%
PHP 18.58%
PowerShell 13.59%
Go 13.24%
Rust 13.05%
Kotlin 9.06%

The best ranked language according to the Institute of Electrical and Electronics Engineers (IEEE)

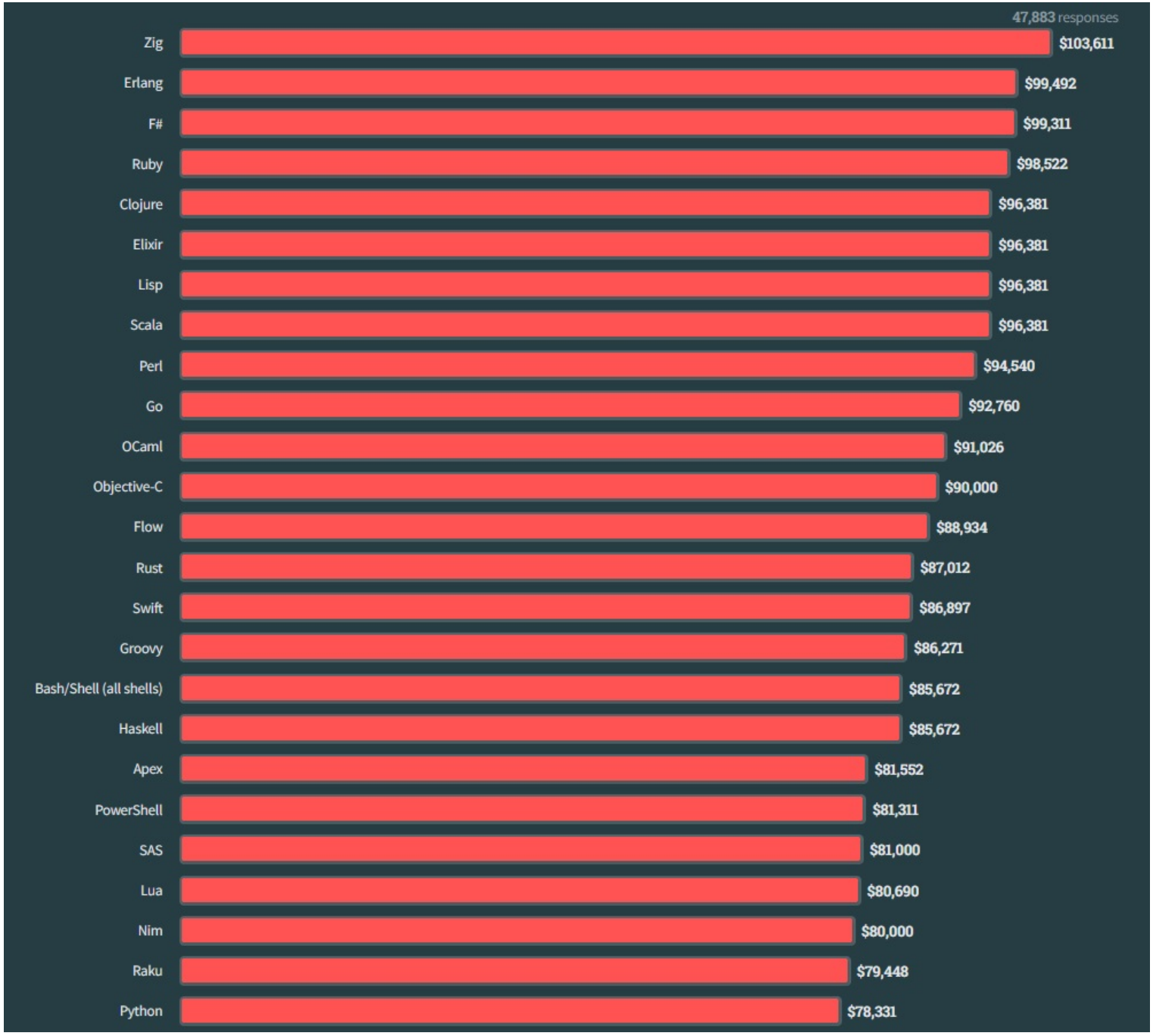
The best ranked language according to the Institute of Electrical and Electronics Engineers (IEEE)



Employabilty

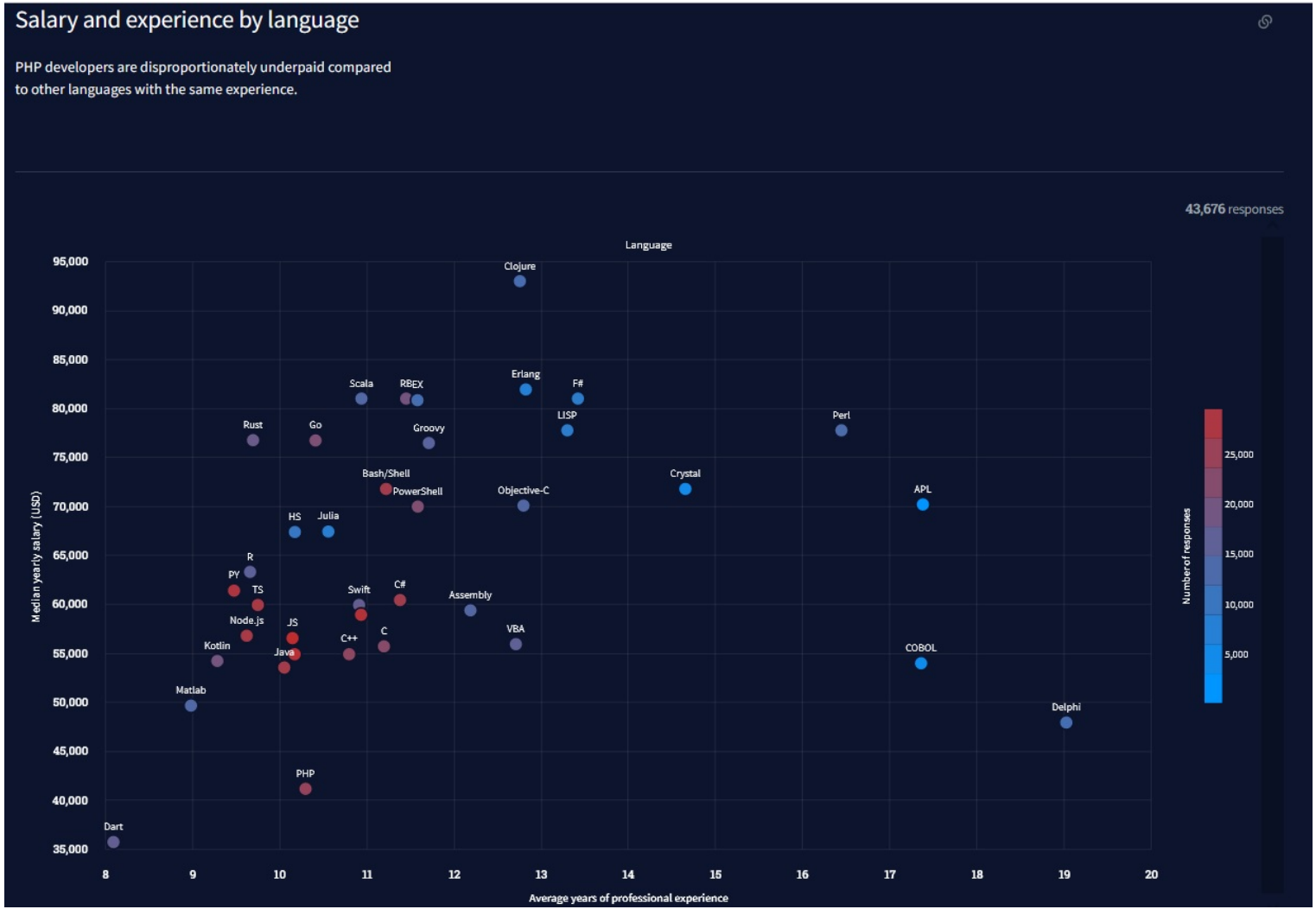
It is the **25th** "best" paid language...

It is the **25th** "best" paid language...


47,883 responses
Zig
$103,611
Erlang
$99,492
F#
$99,311
Ruby
$98,522
Clojure
$96,381
Elixir
$96,381
Lisp
$96,381
Scala
$96,381
Perl
$94,540
Go
$92,760
OCaml
$91,026
Objective-C
$90,000
Flow
$88,934
Rust
$87,012
Swift
$86,897
Groovy
$86,271
Bash/Shell (all shells)
$85,672
Haskell
$85,672
Apex
$81,552
PowerShell
$81,311
SAS
$81,000
Lua
$80,690
Nim
$80,000
Raku
$79,448
Python
$78,331

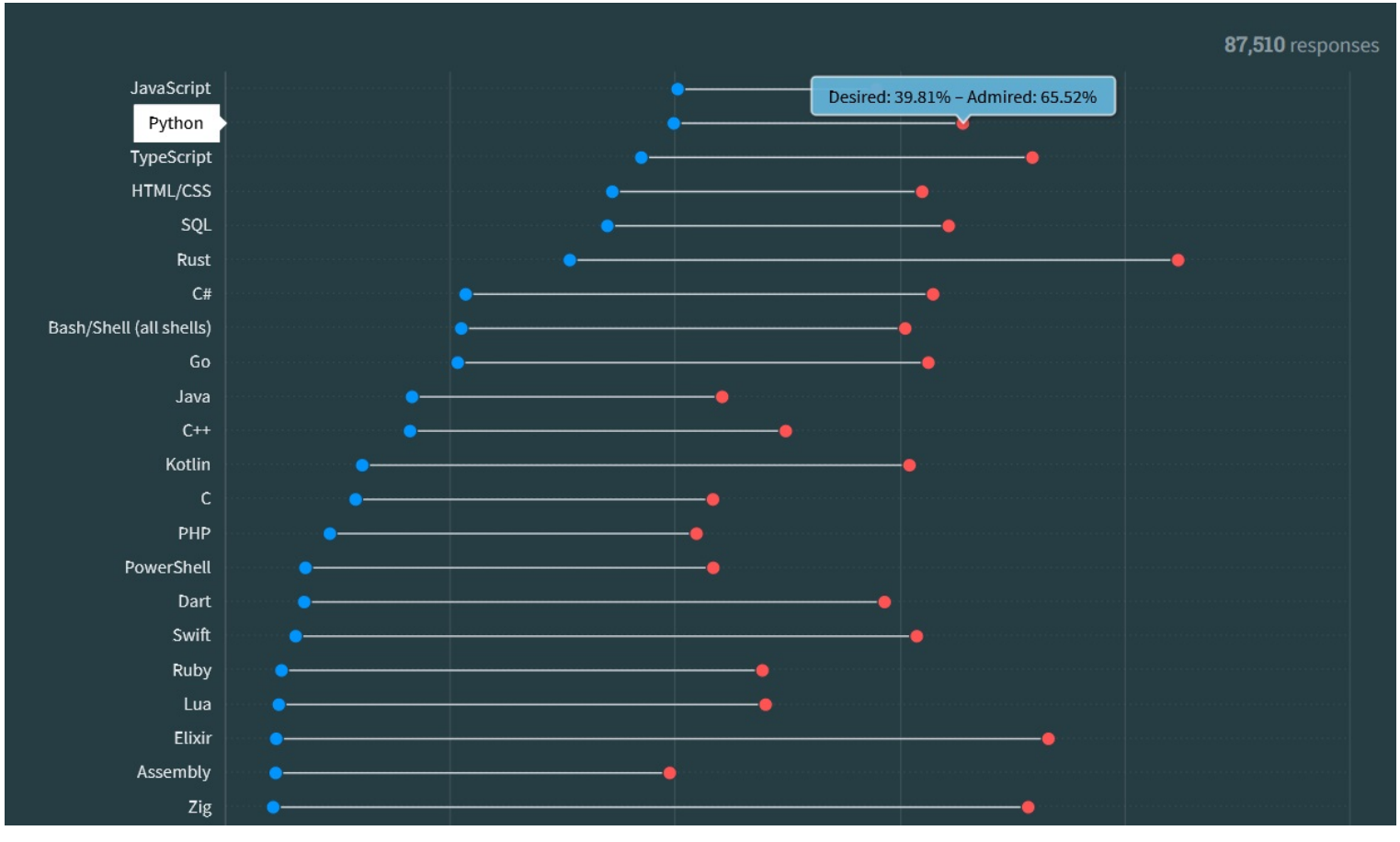
...but one of the fastest ones to adopt (from 2022)...

...but one of the fastest ones to adopt (from 2022)...



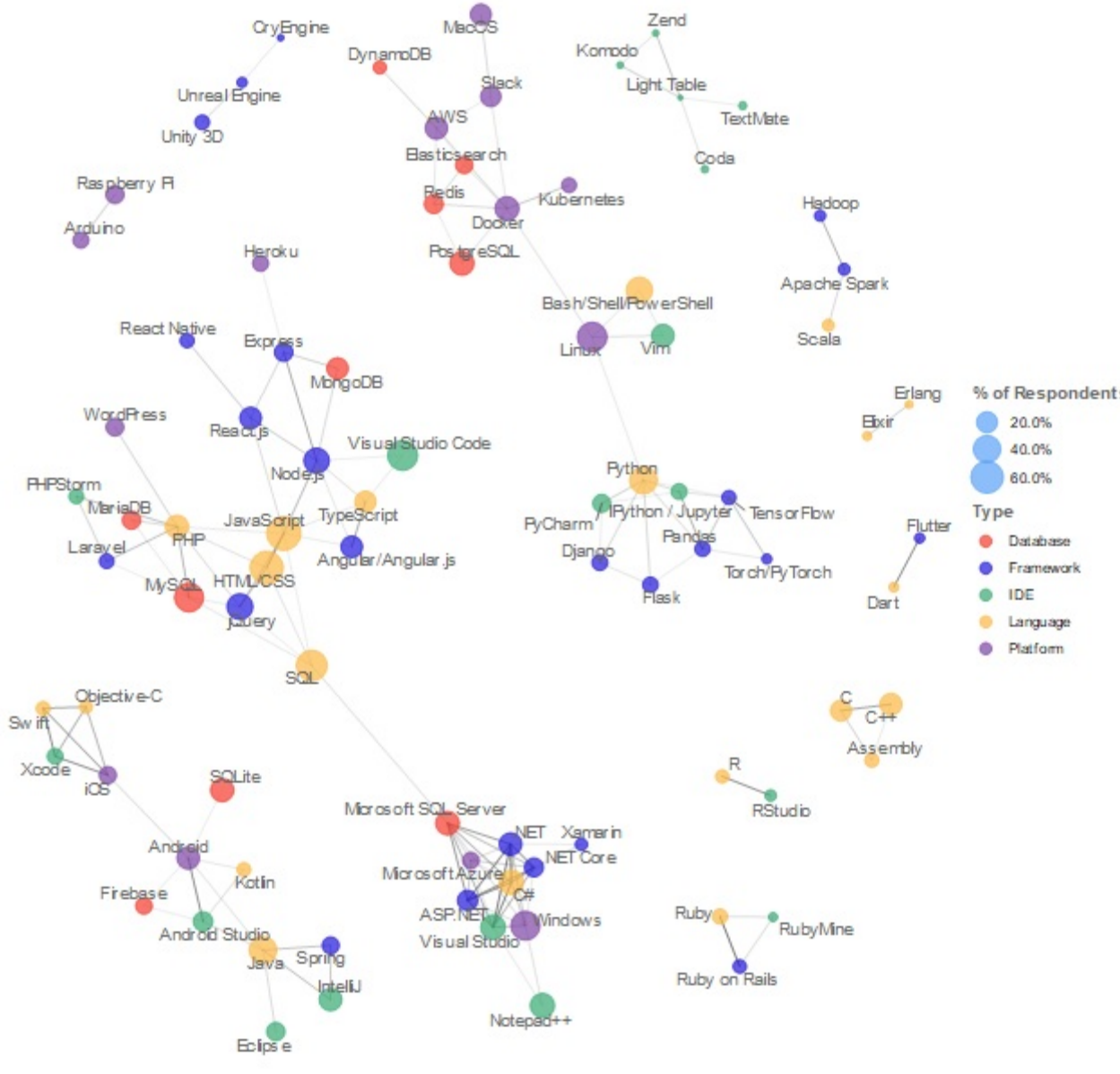
...and the 2nd most desired one!

...and the 2nd most desired one!

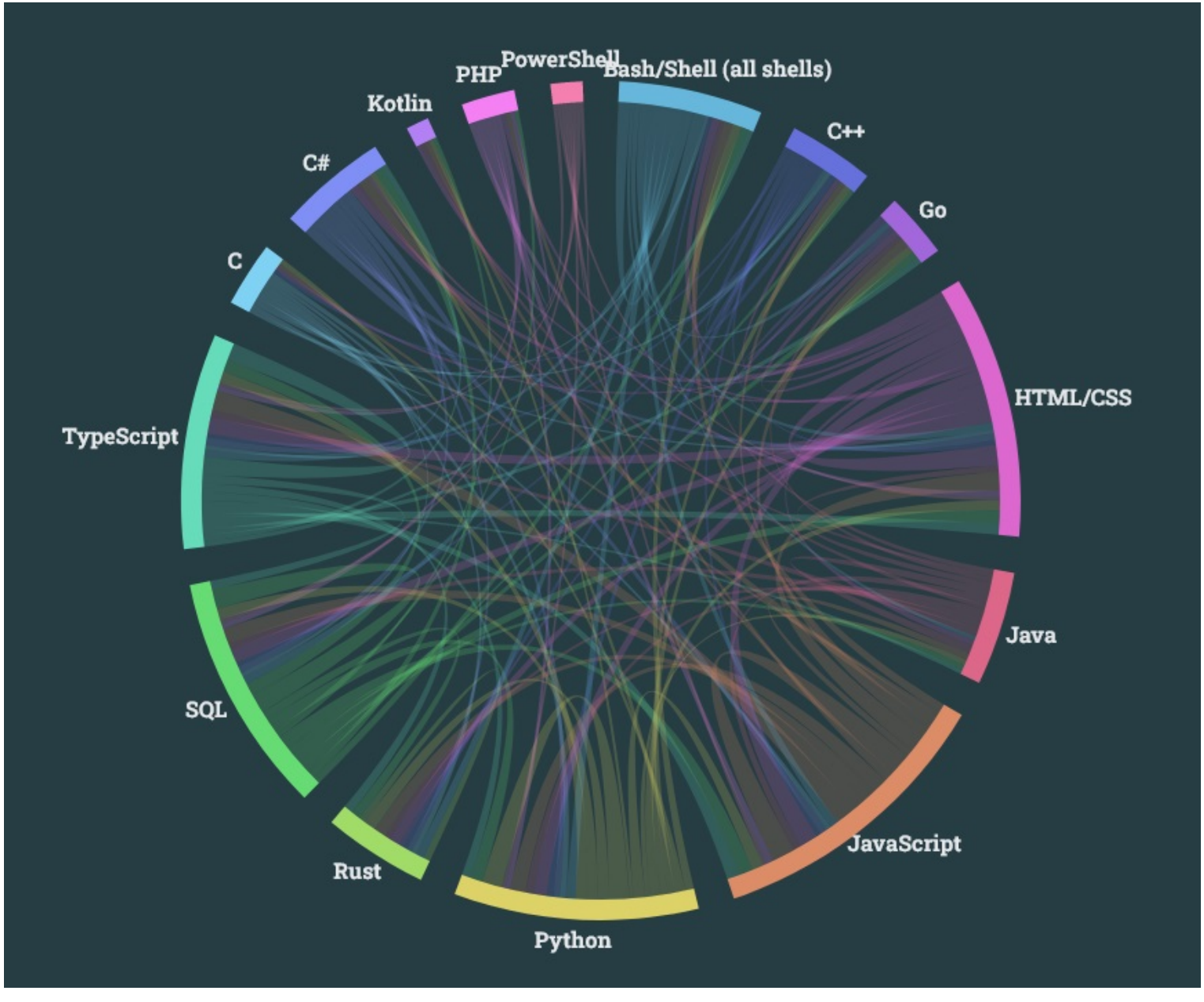


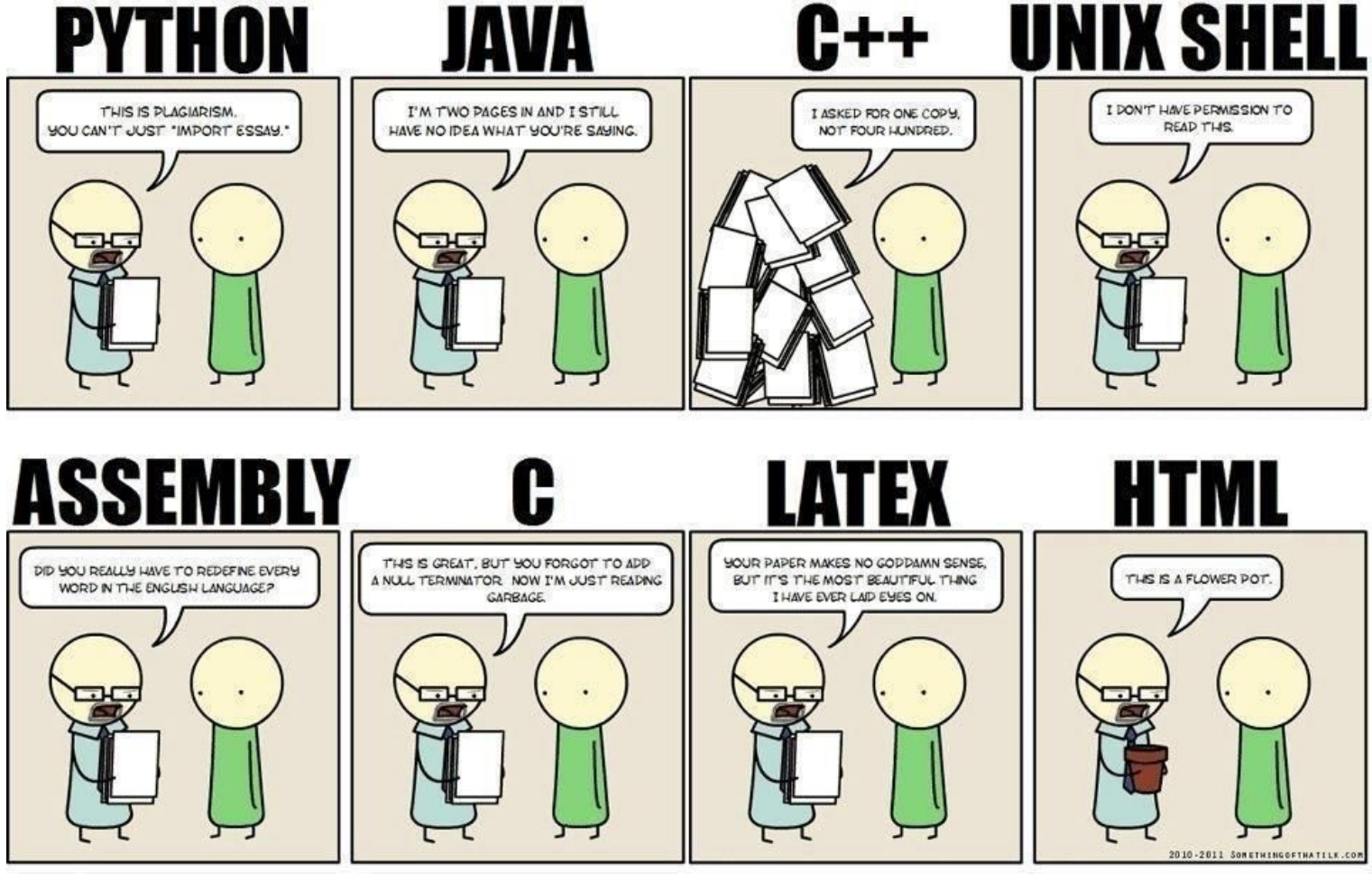
Reach/Scalability

Original figure here


CryEngine
Unreal Engine
Unity 3D
MacOS
DynamoDB
Slack
AWS
Elasticsearch
Redis
Kubernetes
Docker
PostgreSQL
Zend
Komodo
Light Table
TextMate
Coda
Raspberry Pi
Arduino
Heroku
Hadoop
Apache Spark
Scala
Bash/Shell/PowerShell
Linux
Vim
React Native
Express
MongoDB
WordPress
ReactJS
Node.js
Visual Studio Code
PHPStorm
MariaDB
PHP
Laravel
JavaScript
TypeScript
Angular/Angular.js
MySQL
HTML/CSS
jQuery
SQL
Erlang
Elixir
Python
PyCharm
IPython / Jupyter
TensorFlow
Pandas
Django
Torch/PyTorch
Flask
Flutter
Dart
% of Respondents
20.0%
40.0%
60.0%
Type
Database
Framework
IDE
Language
Platform
Objective-C
Swift
Xcode
iOS
SQLite
Android
Kotlin
Firebase
Android Studio
Java
Spring
IntelliJ
Eclipse
Microsoft SQL Server
.NET
Xamarin
.NET Core
Microsoft Azure
C#
ASP.NET
Windows
Visual Studio
Notepad++
C
C++
Assembly
R
RStudio
Ruby
RubyMine
Ruby on Rails

This figure from 2023 comes close!

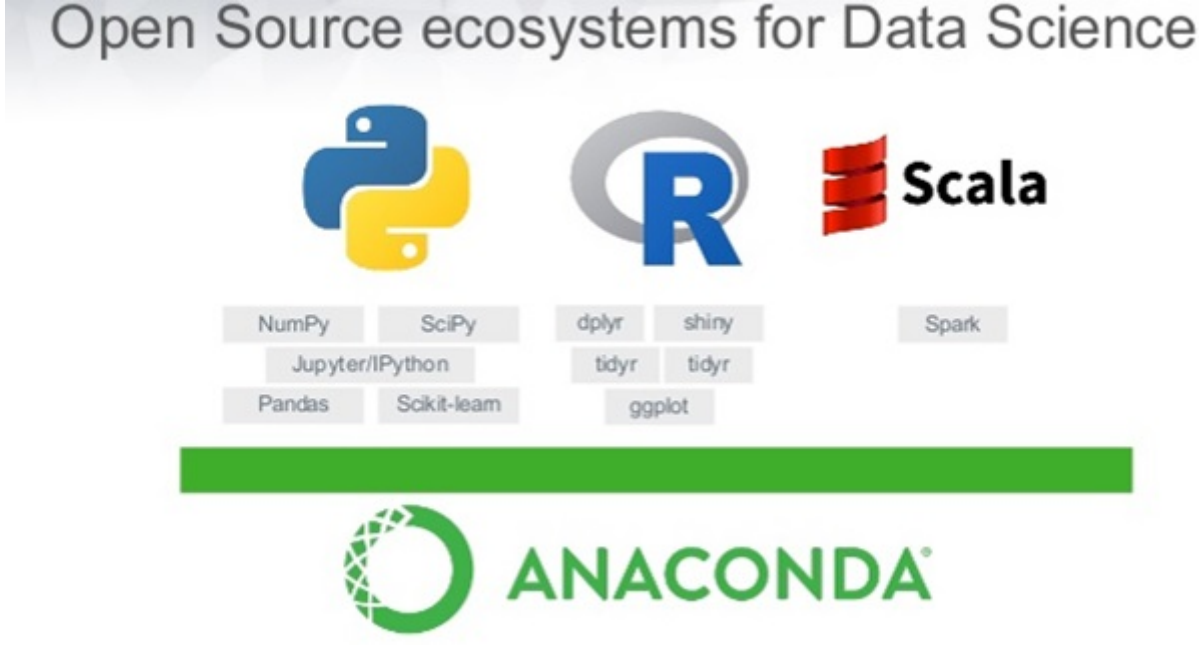

Bash/Shell (all shells)
PowerShell
PHP
Kotlin
C#
C
C++
Go
HTML/CSS
Java
JavaScript
Python
Rust
SQL
TypeScript


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.
2010-2011 SOMETHINGOFTHATILK.COM

The long and hard way

1. Install Python (https://www.python.org).
2. Install a Python Integrated Development Environment (IDE) such as IDLE (available when installing Python), Pycharm (https://www.jetbrains.com/pycharm/) or Spyder (https://pypi.org/project/spyder/).
3. Install Jupyter Notebook (http://jupyter.org/).

The best way: Anaconda Navigator



In its most simplistic state, Python acts like a calculator. You simply write one calculation, and Python gives you the answer!

In its most simplistic state, Python acts like a calculator. You simply write one calculation, and Python gives you the answer!

In [1]:
```
1+2
```

Out[1]:
```
3
```

Moreover, you can also do some coding!

Moreover, you can also do some coding!

In [2]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Notice the simplicity of the Python syntax, in the sense that we do not need to define classes or use a complex and strict structure of parenthesis!

Notice the simplicity of the Python syntax, in the sense that we do not need to define classes or use a complex and strict structure of parenthesis!

In this course we will use Jupyter Notebook not only for lectures, but also for laboratory activities and to code/present the coursework.

Python contains a pre-defined set of **classes** which can contain certain data types or data structures

Python contains a pre-defined set of **classes** which can contain certain data types or data structures

Instead of defining objects/variables/classes/constructors, one may simply type a number and Python will assume the type of this value

Python contains a pre-defined set of **classes** which can contain certain data types or data structures

Instead of defining objects/variables/classes/constructors, one may simply type a number and Python will assume the type of this value

In [3]:
```
## Type anything and then run the cell to see if Python recognises the object
type(1)
```

Out[3]:
```
int
```

Numerical Data Types

```
* Integers
* Float
* Booleans
* Hex
* Oct
* Complex
* and many more to import...
```

Integers

Integers

The most basic data type in Python

Integers

The most basic data type in Python

A number by default is an integer if no decimal value is specified

The `type()` **function** can be used to discover the type of a variable or a number

The `type()` **function** can be used to discover the type of a variable or a number

You can use comparison operators to evaluate integer values

In [4]:
```
type(3)
```

Out[4]:
```
int
```

In [4]:
```
type(3)
```

Out[4]:
```
int
```

In [5]:
```
## Writing exponential numbers
int(2e5)
```

Out[5]:
```
200000
```

In [4]:
```
type(3)
```

Out[4]:
```
int
```

In [5]:
```
## Writing exponential numbers
int(2e5)
```

Out[5]:
```
200000
```

In [6]:
```
x=3
print(type(x))
```

```
<class 'int'>
```

In [7]:
```
# See if two ints are equal
3==5
```

Out[7]:
```
False
```

In [7]:
```
# See if two ints are equal
3==5
```

Out[7]: False

In [8]:
```
# See if two ints are not equal
3!=5
```

Out[8]: True

In [7]:
```
# See if two ints are equal
3==5
```

Out[7]: False

In [8]:
```
# See if two ints are not equal
3!=5
```

Out[8]: True

In [9]:
```
# See if a number is smaller than another
3<5
```

Out[9]: True

In [10]:
```
# See if a number is smaller or equal to another
3<=5
```

Out[10]:
```
True
```

In [10]:
```
# See if a number is smaller or equal to another
3<=5
```

Out[10]: True

In [11]:
```
# See if a number is larger than another
3>5
```

Out[11]: False

In [10]:
```
# See if a number is smaller or equal to another
3<=5
```

Out[10]: True

In [11]:
```
# See if a number is larger than another
3>5
```

Out[11]: False

In [12]:
```
# See if a number is larger or equal to another
3>=5
```

Out[12]: False

Booleans (logical operators)

Booleans (logical operators)

In [13]:
```
type(False)
```

Out[13]:
```
bool
```

Booleans (logical operators)

In [13]:
```
type(False)
```

Out[13]:
```
bool
```

In [14]:
```
z = True
print(type(z))
```

```
<class 'bool'>
```

Float

Float

Is how we call decimals in Python

Float

Is how we call decimals in Python

Up to 15 decimal places

In [15]:
```
y=6.912897398127847
```

In [15]:
```
y=6.912897398127847
```

In [16]:
```
print(y)
```

```
6.912897398127847
```

In [15]:
```
y=6.912897398127847
```

In [16]:
```
print(y)
```

```
6.912897398127847
```

In [17]:
```
# If more than 15 decimal places are used, python truncates
7.07893492368947398473989723489742389 47
```

Out[17]:
```
7.078934923689474
```

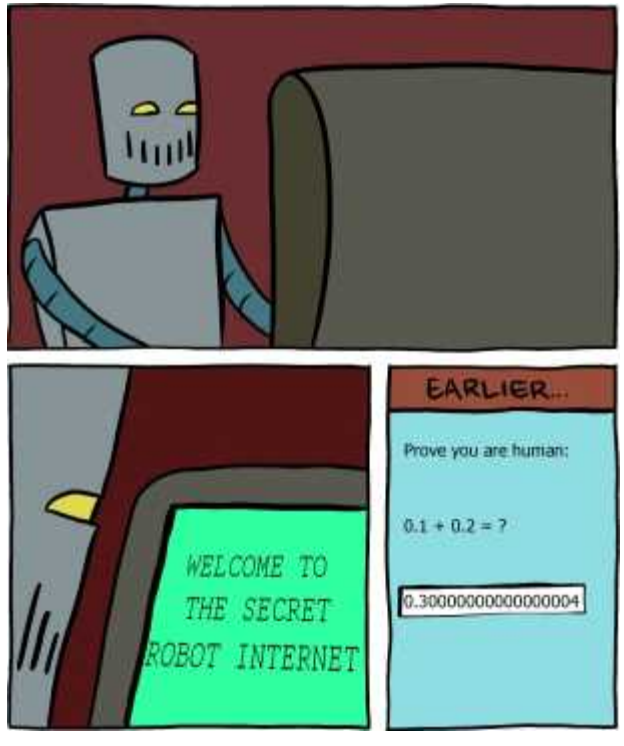
In [18]:
```
# We can also compare floats, but funny things may happen!
0.1+0.2==0.3
```

Out[18]:
```
False
```

In [18]:
```
# We can also compare floats, but funny things may happen!
0.1+0.2==0.3
```

Out[18]:
```
False
```

Do you know why?


WELCOME TO
THE SECRET
ROBOT INTERNET
EARLIER...
Prove you are human:
0.1 + 0.2 = ?
0.30000000000000004

In [19]:
```
from decimal import *
getcontext().prec = 1
Decimal(0.1) + Decimal(0.2) == Decimal('0.3')
```

Out[19]:
```
True
```

The `isinstance()` function

The `isinstance()` function

In [20]:
```
isinstance(x,float)
```

Out[20]:
```
False
```

The `isinstance()` function

In [20]:
```
isinstance(x,float)
```

Out[20]:
```
False
```

In [21]:
```
isinstance(y,int)
```

Out[21]:
```
False
```

You can also work with complex, binary, octal and hexadecimal numbers in Python.

Strings

Strings

In [22]:
```
# Defining a string
a = "apple"
a
```

Out[22]:
```
'apple'
```

Strings

In [22]:
```
# Defining a string
a = "apple"
a
```

Out[22]:
```
'apple'
```

In [23]:
```
# Any number inside quotations becomes a string
b = "45"
type(b)
```

Out[23]:
```
str
```

Strings

In [22]:
```
# Defining a string
a = "apple"
a
```

Out[22]:
```
'apple'
```

In [23]:
```
# Any number inside quotations becomes a string
b = "45"
type(b)
```

Out[23]:
```
str
```

In [24]:
```
# A boolean inside quotations becomes a string
c = "True"
print(c)
```

```
True
```

Strings

In [22]:
```
# Defining a string
a = "apple"
a
```

Out[22]:
```
'apple'
```

In [23]:
```
# Any number inside quotations becomes a string
b = "45"
type(b)
```

Out[23]:
```
str
```

In [24]:
```
# A boolean inside quotations becomes a string
c = "True"
print(c)
```

```
True
```

In [25]:
```
# What happens if I want to write the string "don't"?
d = '''"don't"'''
print(d)
```

```
"don't"
```

In [26]:
```
print(type(a),type(b),type(c),type(d))
```

```
<class 'str'> <class 'str'> <class 'str'> <class 'str'>
```

Conversion Functions

In [27]:
```
# the int() function can convert any number into an integer.
# Mostly used to truncate floats
int(7.8)
```

Out[27]:
```
7
```

In [27]:
```
# the int() function can convert any number into an integer.
# Mostly used to truncate floats
int(7.8)
```

Out[27]:
```
7
```

In [28]:
```
# Converting booleans into ints
int(True)
```

Out[28]:
```
1
```

In [27]:
```
# the int() function can convert any number into an integer.
# Mostly used to truncate floats
int(7.8)
```

Out[27]:
```
7
```

In [28]:
```
# Converting booleans into ints
int(True)
```

Out[28]:
```
1
```

In [29]:
```
# float function adds a 0 decimal to an int
float(9)
```

Out[29]:
```
9.0
```

In [30]:
```
# bool() turns any number to TRUE (other than 0)
bool(0.1)
```

Out[30]:
```
True
```

In [30]:
```
# bool() turns any number to TRUE (other than 0)
bool(0.1)
```

Out[30]:
```
True
```

In [31]:
```
bool(0)
```

Out[31]:
```
False
```

Data Structures

Tuples

Tuples

IMMUTABLE collection of elements

Tuples

IMMUTABLE collection of elements

Defined using parenthesis and separating elements with commas

Tuples

IMMUTABLE collection of elements

Defined using parenthesis and separating elements with commas

Not all elements in a tuple have to be of the same type

In [32]:
```
tuple1 = (1,2,3)
tuple1
```

Out[32]:
```
(1, 2, 3)
```

In [32]:
```
tuple1 = (1,2,3)
tuple1
```

Out[32]:
```
(1, 2, 3)
```

In [33]:
```
tuple2 = (1,'g',4.4,True)
tuple2
```

Out[33]:
```
(1, 'g', 4.4, True)
```

Lists

Lists

MUTABLE collection of elements

Lists

MUTABLE collection of elements

Defined using squared brackets and separating elements with commas

Lists

MUTABLE collection of elements

Defined using squared brackets and separating elements with commas

Not all elements in a list have to be of the same type

In [34]:
```
list1 = [1,2,3]
list1
```

Out[34]:
```
[1, 2, 3]
```

In [34]:
```
list1 = [1,2,3]
list1
```

Out[34]:
```
[1, 2, 3]
```

In [35]:
```
list2 = [1,'g',4.4,True]
list2
```

Out[35]:
```
[1, 'g', 4.4, True]
```

Why do we need two very similar structures such as tuples and lists?

The `len()` function

The `len()` function

In [36]:
```
len(tuple1)
```

Out[36]:
```
3
```

The `len()` function

In [36]:
```
len(tuple1)
```

Out[36]: 3

In [37]:
```
len(list2)
```

Out[37]: 4

Accessing an element in a tuple/list

Accessing an element in a tuple/list

We can access to all positions in a tuple/list by using squared brackets **after** the tuple/list

Accessing an element in a tuple/list

We can access to all positions in a tuple/list by using squared brackets **after** the tuple/list

Indexes in Python begin in 0!

In [38]:
```
# Access the first element of list1
list1[0]
```

Out[38]:
```
1
```

In [38]:
```
# Access the first element of list1
list1[0]
```

Out[38]:
```
1
```

In [39]:
```
# Access the second element of tuple2
tuple2[1]
```

Out[39]:
```
'g'
```

In [40]:
```
# changing an element of a list
list1[0]=3
list1
```

Out[40]:
```
[3, 2, 3]
```

In [40]:
```
# changing an element of a list
list1[0]=3
list1
```

Out[40]:
```
[3, 2, 3]
```

In [41]:
```
# changing an element of a tuple -> ERROR!
tuple2[0]=3
tuple2
```

```
---------------------------------------------------------------------
------
TypeError                                 Traceback (most recent call
last)
Cell In[41], line 2
      1 # changing an element of a tuple -> ERROR!
----> 2 tuple2[0]=3
      3 tuple2

TypeError: 'tuple' object does not support item assignment
```