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

Why do you Need 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

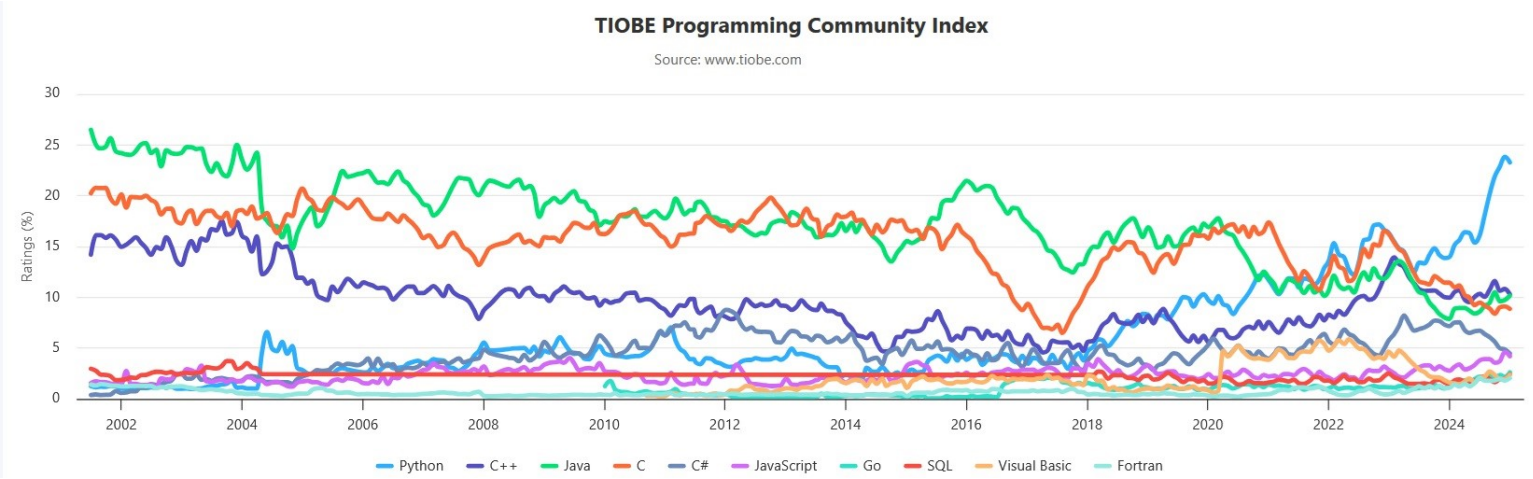
Most popular language according to the TIOBE index

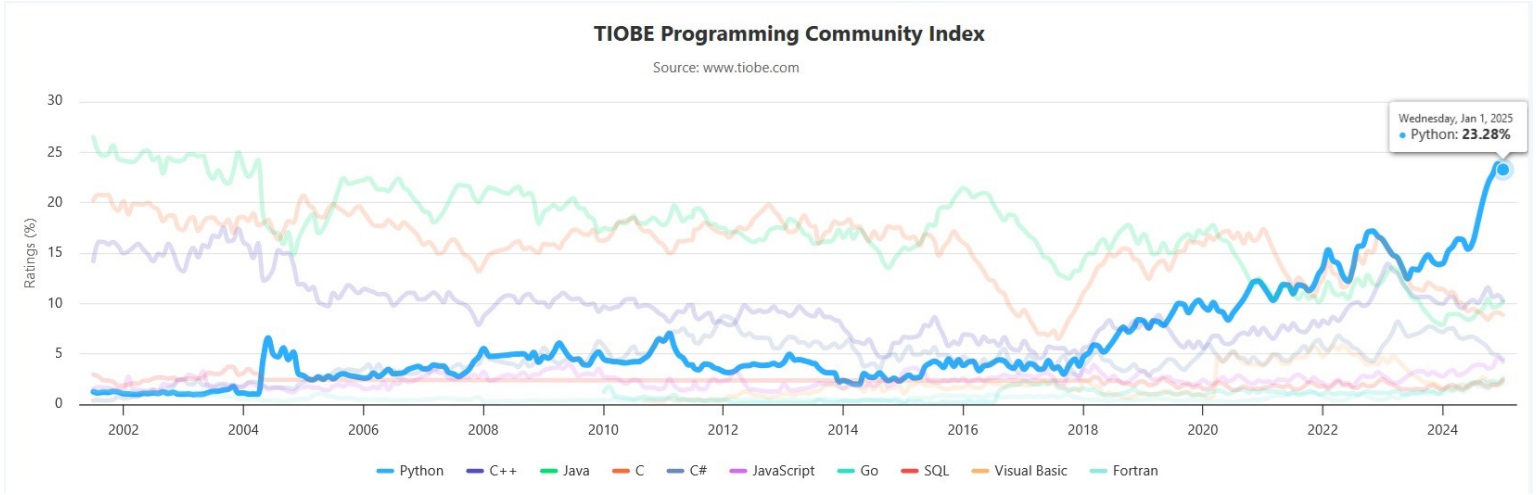
Most popular language according to the TIOBE index

Jan 2025	Jan 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.28%	+9.32%
2	3	↑	C++	10.29%	+0.33%
3	4	↑	Java	10.15%	+2.28%
4	2	↓	C	8.86%	-2.59%
5	5		C#	4.45%	-2.71%
6	6		JavaScript	4.20%	+1.43%
7	11	⇈	Go	2.61%	+1.24%
8	9	↑	SQL	2.41%	+0.95%
9	8	↓	Visual Basic	2.37%	+0.77%
10	12	↑	Fortran	2.04%	+0.94%

Fastest growing for the current year

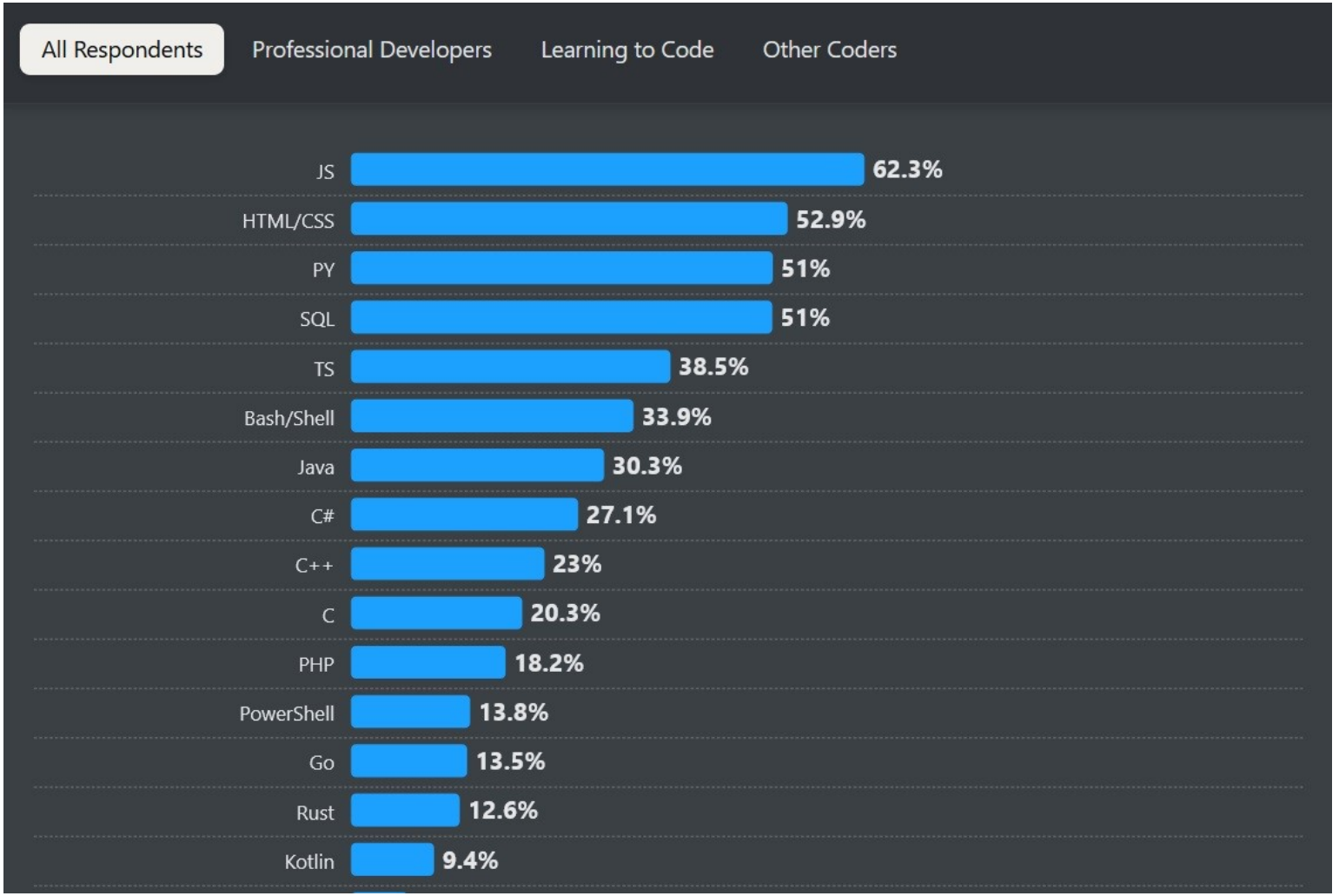
Fastest growing for the current year




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
Wednesday, Jan 1, 2025
Python: 23.28%
Python
C++
Java
C
C#
JavaScript
Go
SQL
Visual Basic
Fortran

According to Stackoverflow's 2024 survey, it is the 3rd most popular programming language in the world!

According to Stackoverflow's 2024 survey, it is the 3rd most popular programming language in the world!



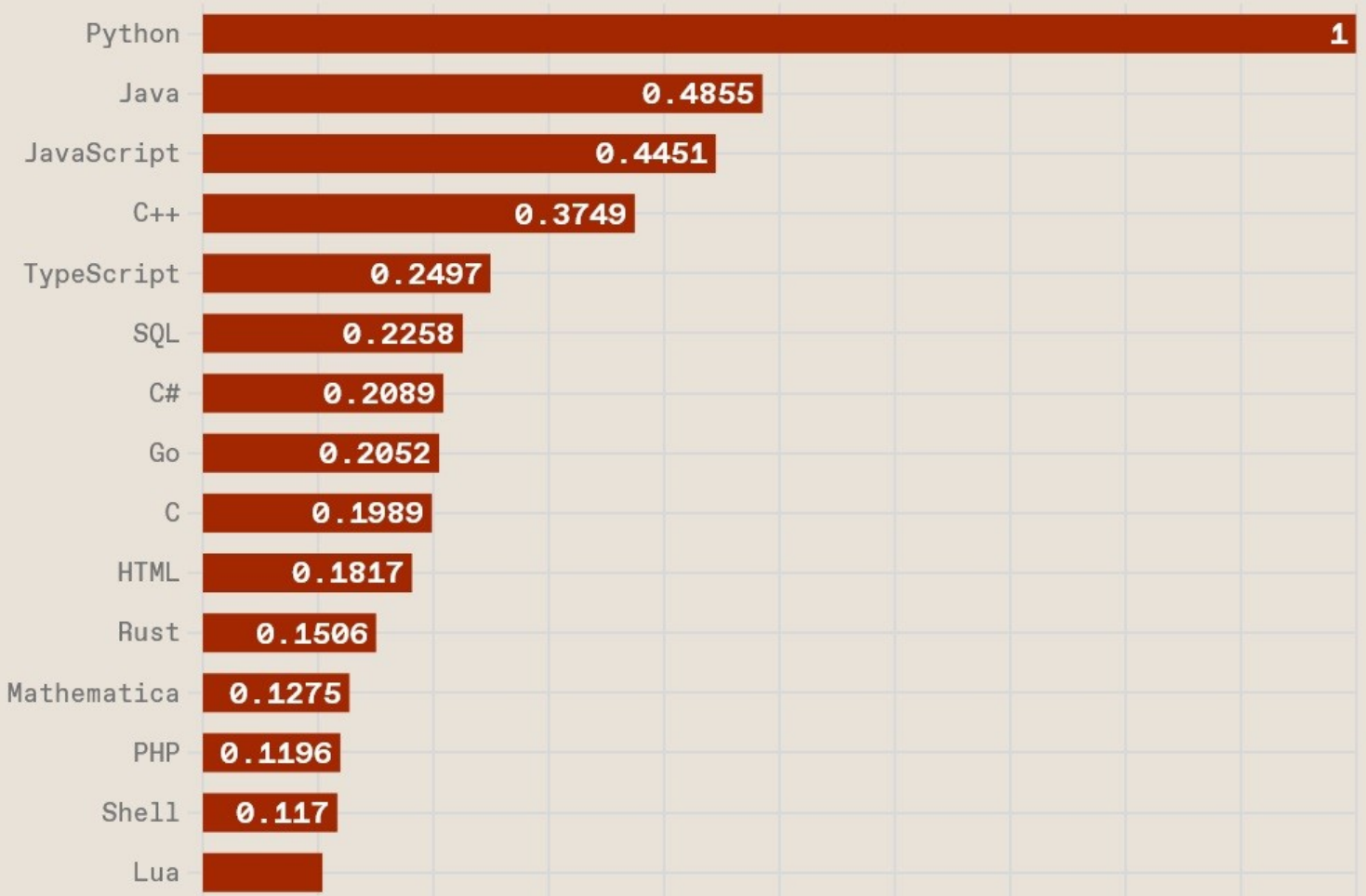
It is the best ranked language according to the Institute of Electrical and Electronics Engineers (IEEE)

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

Top Programming Languages 2024

Click a button to see a differently weighted ranking

Spectrum | Trending | Jobs



Employabilty

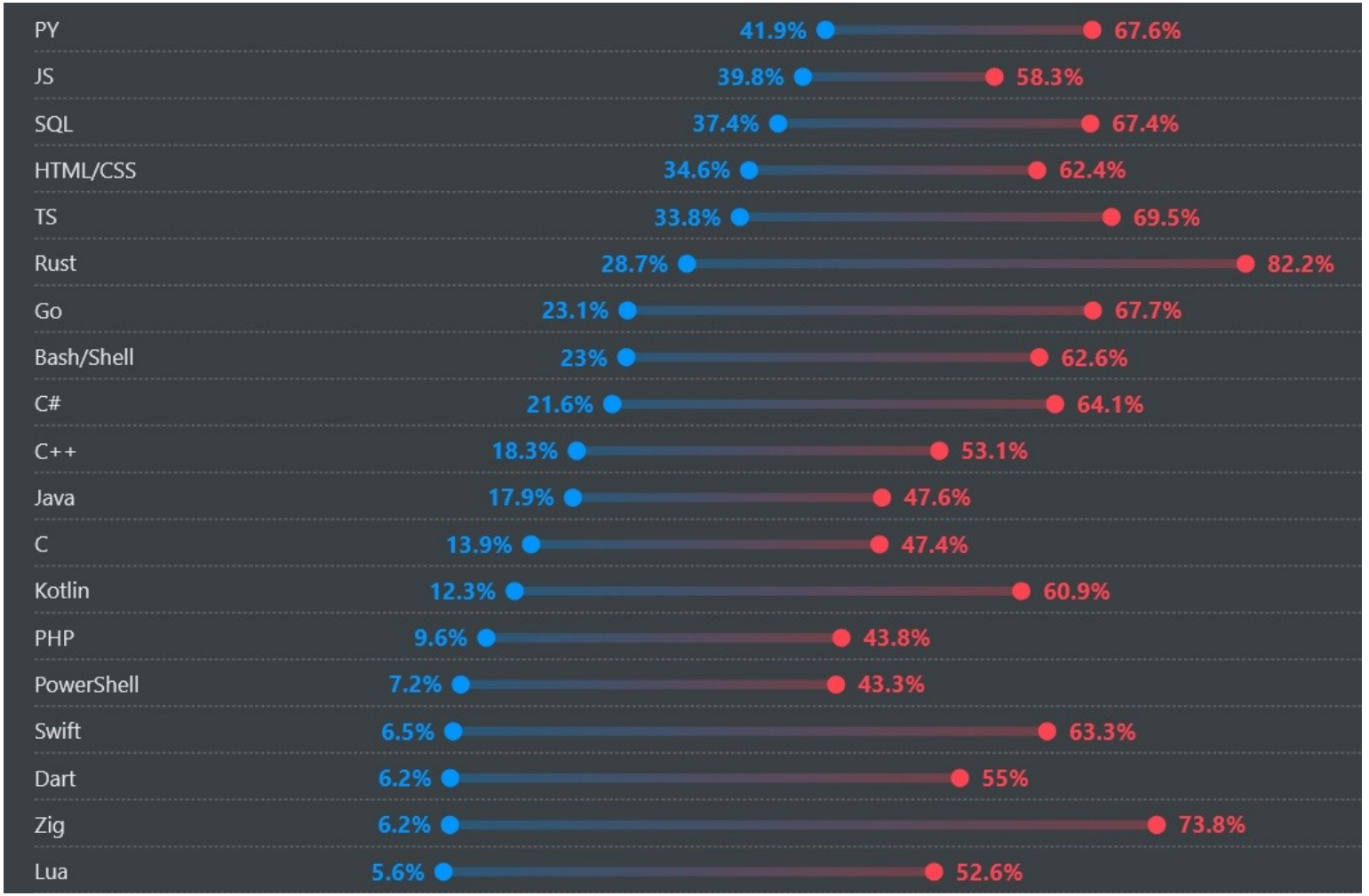
It is the **27th best** paid language...

It is the **27th best** paid language...


Erlang $100,636
Elixir $96,000
Clojure $95,541
Nim $94,924
Ruby $90,221
Perl $90,000
Scala $88,619
Apex $82,500
F# $80,555
Lisp $80,555
Groovy $79,330
Go $76,433
Rust $76,292
Zig $75,332
Bash/Shell $75,184
MicroPython $75,184
OCaml $75,184
Swift $75,184
Objective-C $73,648
Crystal $73,036
PowerShell $72,673
Lua $72,542
Fortran $70,351
Cobol $70,000

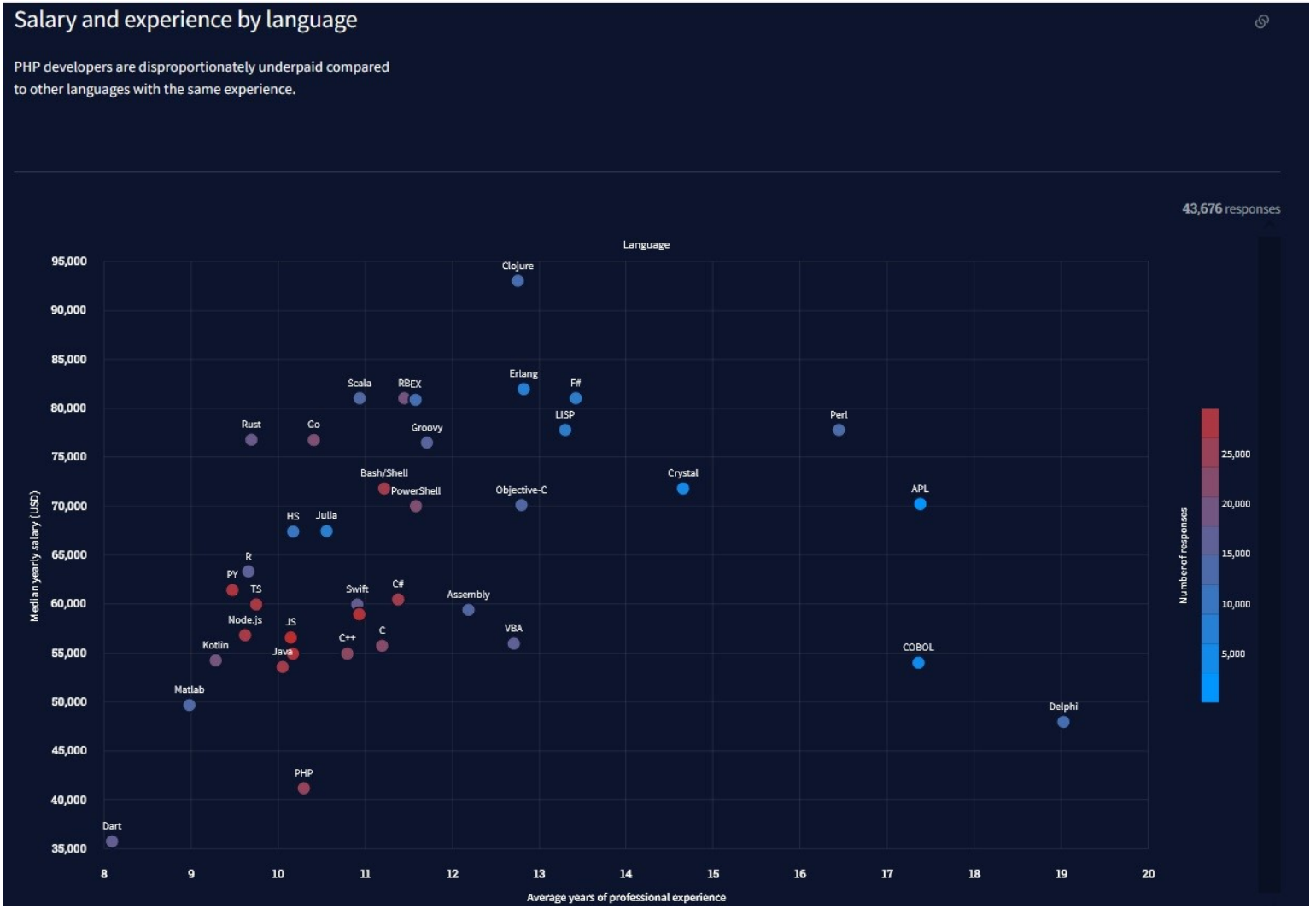
...but the most desired...

...but the most desired...



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

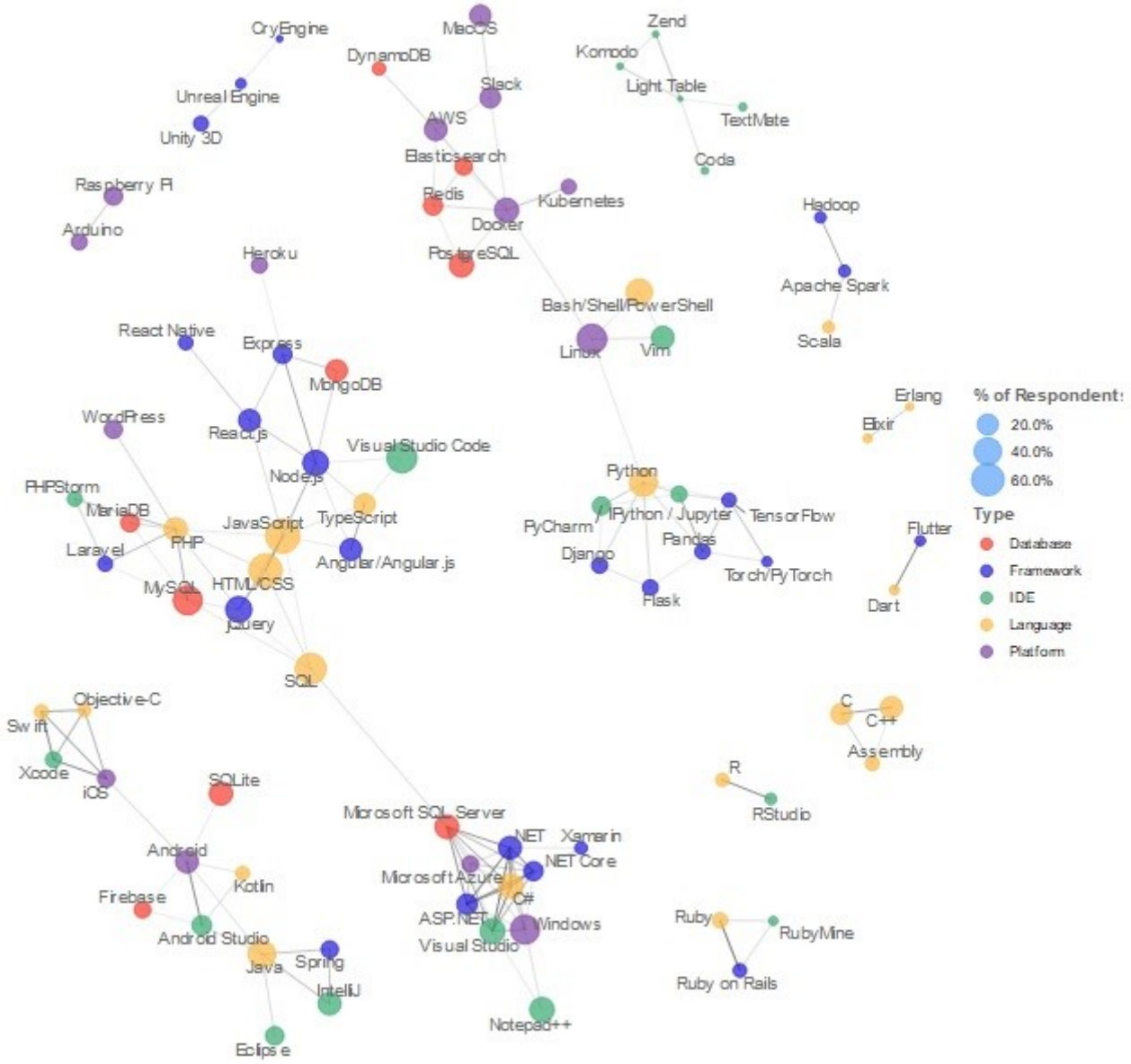
...and one of the fastest ones to adopt (from the 2022 survey)



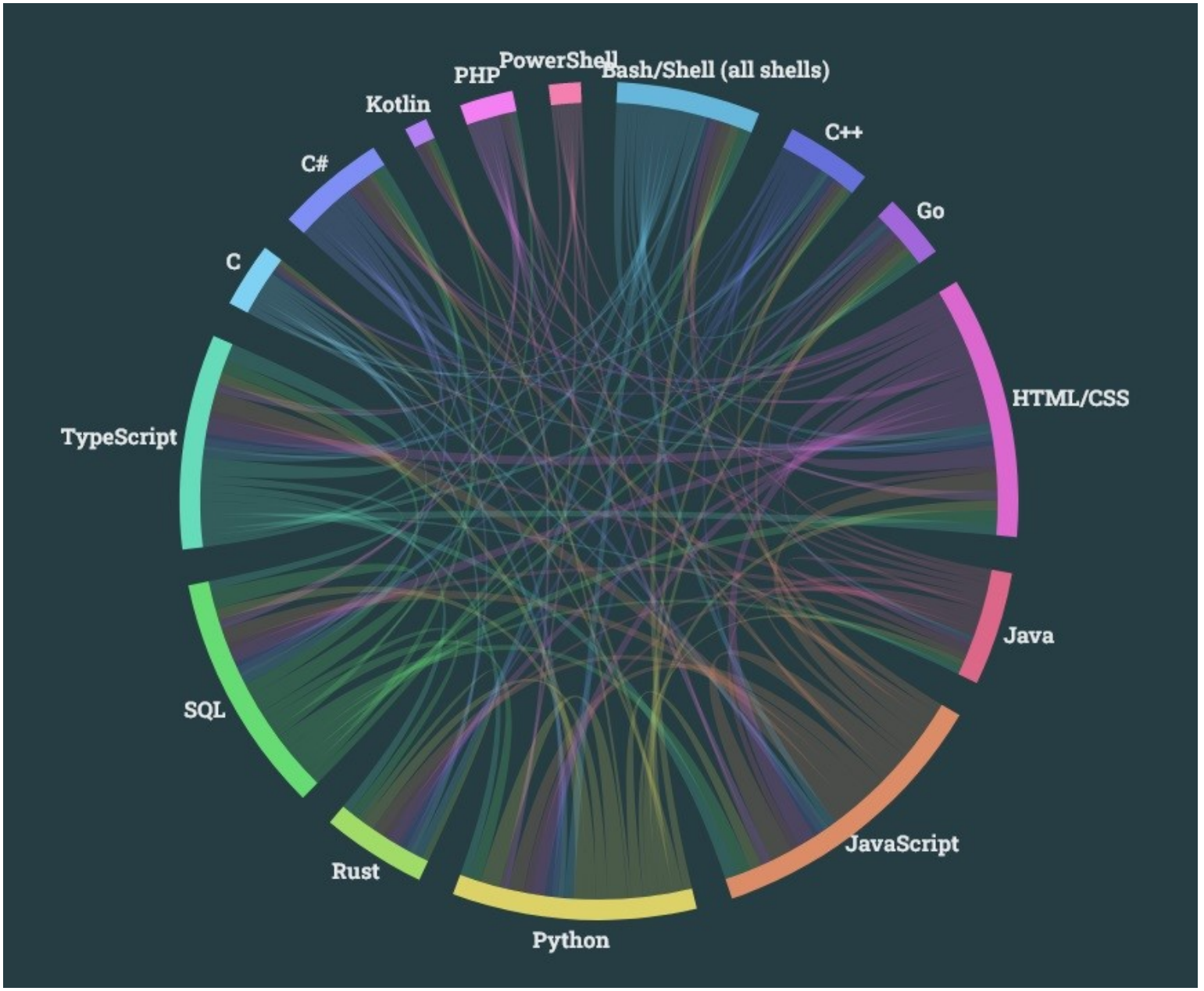
Reach/Scalability

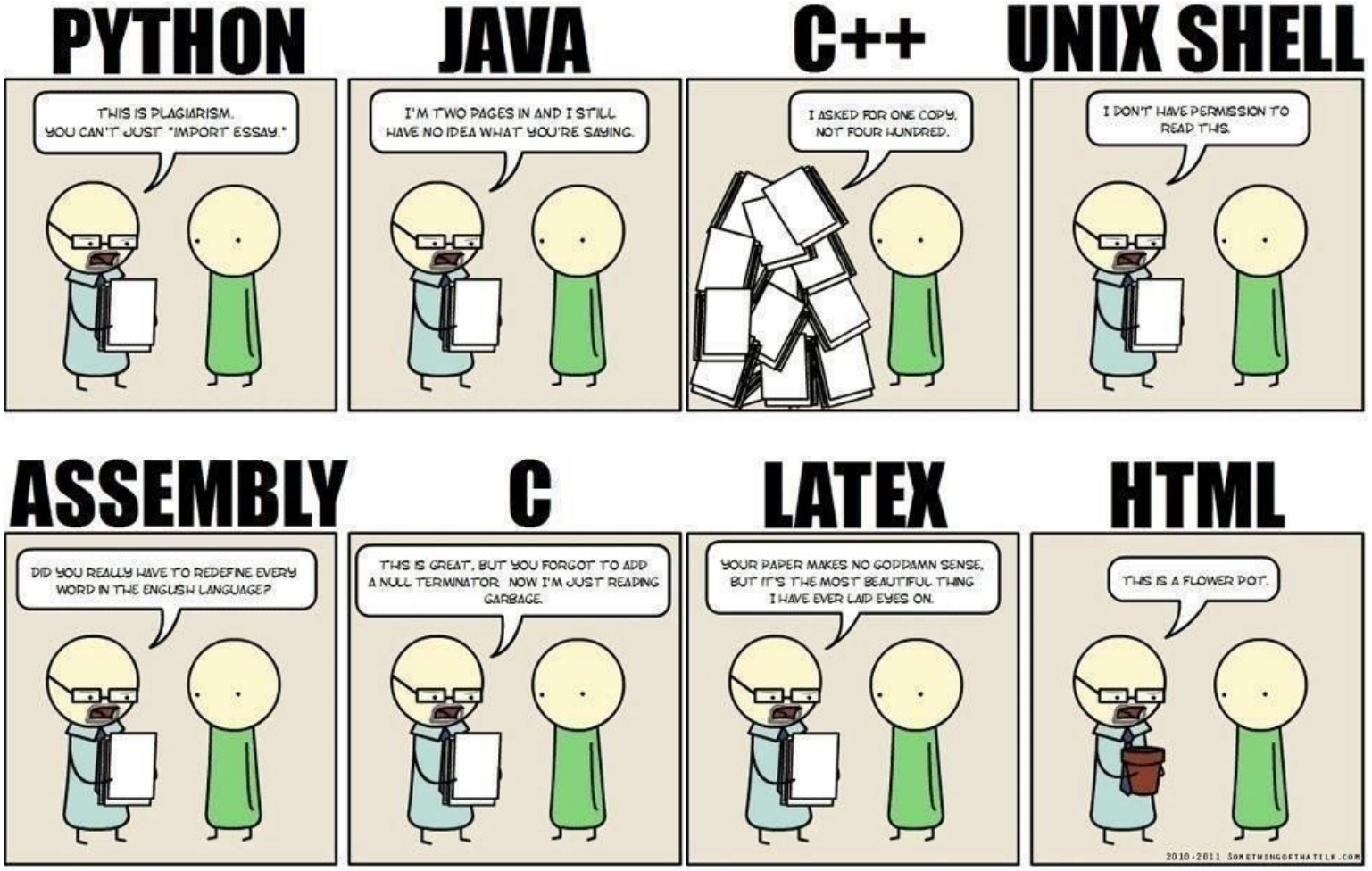
Figure from the 2019 survey

Figure from the 2019 survey


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
React.js
Node.js
Visual Studio Code
PHPStorm
MariaDB
PHP
JavaScript
TypeScript
Laravel
Angular/Angular.js
MySQL
HTML/CSS
jQuery
SQL
Erlang
Elixir
Python
PyCharm
IPython / Jupyter
TensorFlow
Pandas
Django
Flask
Torch/PyTorch
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!


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

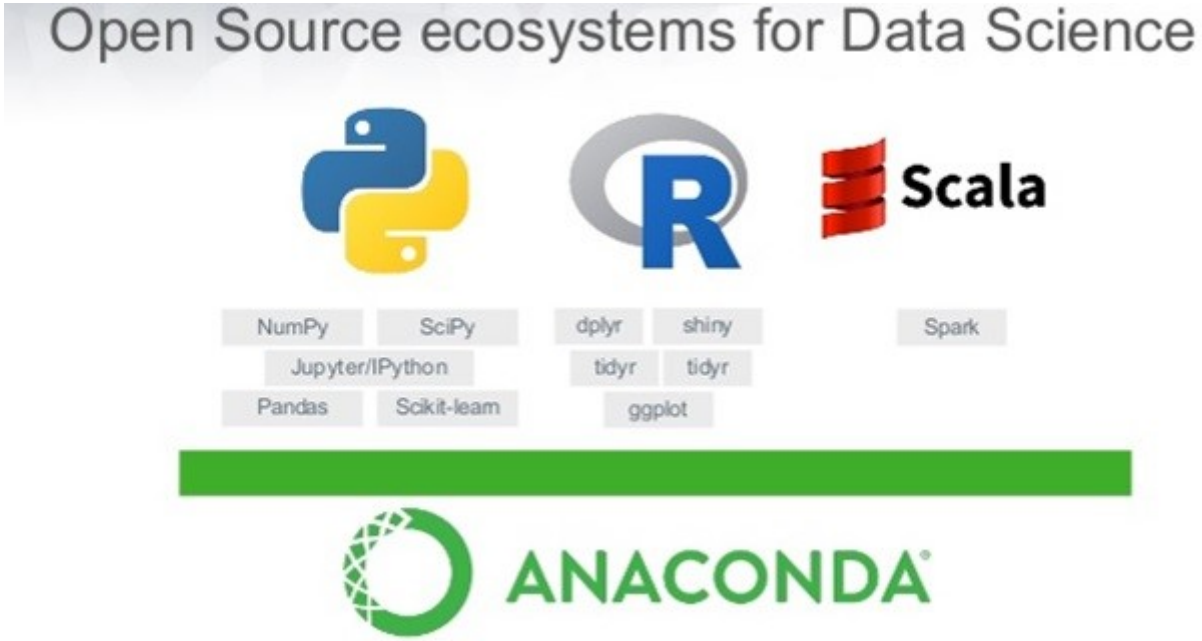

PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS 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

Installing Python

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



How Does Python Look Like?

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+1
```

Out[1]:
```
2
```

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.

Data Types and Data Structures

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

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
```

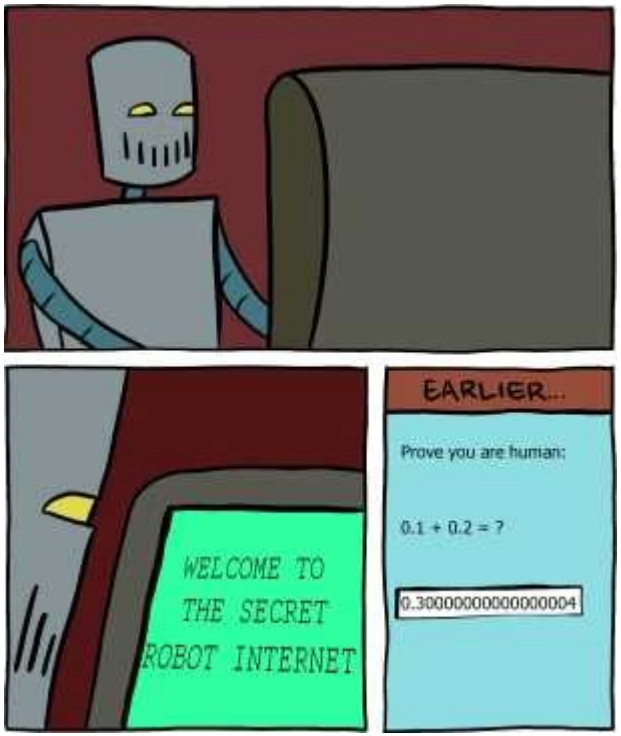
In [18]:
```
# We can also compare floats, but funny things may happen!
0.1+0.2==0.3
```

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 [41]:
```
from decimal import *
getcontext().prec = 1
Decimal(0.1) + Decimal(0.2) == Decimal('0.3')
```

Out[41]:
```
True
```

The `isinstance()` function

The `isinstance()` function

In [19]:
```
isinstance(x,float)
```

Out[19]:
```
False
```

The `isinstance()` function

In [19]:
```
isinstance(x,float)
```

Out[19]:
```
False
```

In [20]:
```
isinstance(y,int)
```

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

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

Strings

Strings

In [21]:
```
# Defining a string
a = "apple"
a
```

Out[21]:
```
'apple'
```

Strings

In [21]:
```
# Defining a string
a = "apple"
a
```

Out[21]:
```
'apple'
```

In [22]:
```
# Any number inside quotations becomes a string
b = "45"
type(b)
```

Out[22]:
```
str
```

Strings

In [21]:
```
# Defining a string
a = "apple"
a
```

Out[21]:
```
'apple'
```

In [22]:
```
# Any number inside quotations becomes a string
b = "45"
type(b)
```

Out[22]:
```
str
```

In [23]:
```
# A boolean inside quotations becomes a string
c = "True"
print(c)
```

```
True
```

Strings

In [21]:
```
# Defining a string
a = "apple"
a
```

Out[21]:
```
'apple'
```

In [22]:
```
# Any number inside quotations becomes a string
b = "45"
type(b)
```

Out[22]:
```
str
```

In [23]:
```
# A boolean inside quotations becomes a string
c = "True"
print(c)
```

```
True
```

In [24]:
```
# What happens if I want to write the string "don't"?
d = '''"don't"'''
print(d)
```

```
"don't"
```

In [25]:
```
print(type(a),type(b),type(c),type(d))
```

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

Conversion Functions

Conversion Functions

In [26]:
```
# the int() function can convert any number into an integer.
# Mostly used to truncate floats
int(7.8)
```

Out[26]:
```
7
```

In [26]:
```
# the int() function can convert any number into an integer.
# Mostly used to truncate floats
int(7.8)
```

Out[26]:
```
7
```

In [27]:
```
# Converting booleans into ints
int(True)
```

Out[27]:
```
1
```

In [26]:
```
# the int() function can convert any number into an integer.
# Mostly used to truncate floats
int(7.8)
```

Out[26]:
```
7
```

In [27]:
```
# Converting booleans into ints
int(True)
```

Out[27]:
```
1
```

In [28]:
```
# float function adds a 0 decimal to an int
float(9)
```

Out[28]:
```
9.0
```

In [29]:
```
# bool() turns any number to TRUE (other than 0)
bool(0.1)
```

Out[29]:
```
True
```

In [29]:
```
# bool() turns any number to TRUE (other than 0)
bool(0.1)
```

Out[29]:
```
True
```

In [30]:
```
bool(0)
```

Out[30]:
```
False
```

Data Structures

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 [31]:
```
tuple1 = (1,2,3)
tuple1
```

Out[31]:
```
(1, 2, 3)
```

In [31]:
```
tuple1 = (1,2,3)
tuple1
```

Out[31]:
```
(1, 2, 3)
```

In [32]:
```
tuple2 = (1,'g',4.4,True)
tuple2
```

Out[32]:
```
(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 [33]:
```
list1 = [1,2,3]
list1
```

Out[33]:
```
[1, 2, 3]
```

In [33]:
```
list1 = [1,2,3]
list1
```

Out[33]:
```
[1, 2, 3]
```

In [34]:
```
list2 = [1,'g',4.4,True]
list2
```

Out[34]:
```
[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 [35]:
```
len(tuple1)
```

Out[35]:
```
3
```

The `len()` function

In [35]:
```
len(tuple1)
```

Out[35]:
```
3
```

In [36]:
```
len(list2)
```

Out[36]:
```
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 [37]:
```
# Access the first element of list1
list1[0]
```

Out[37]:
```
1
```

In [37]:
```
# Access the first element of list1
list1[0]
```

Out[37]:
```
1
```

In [38]:
```
# Access the second element of tuple2
tuple2[1]
```

Out[38]:
```
'g'
```

In [39]:
```
# changing an element of a list
list1[0]=3
list1
```

Out[39]:
```
[3, 2, 3]
```

In [39]:
```
# changing an element of a list
list1[0]=3
list1
```

Out[39]:
```
[3, 2, 3]
```

In [40]:
```
# changing an element of a tuple -> ERROR!
tuple2[0]=3
tuple2
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Cell In[40], line 2
      1 # changing an element of a tuple -> ERROR!
----> 2 tuple2[0]=3
      3 tuple2

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