Python
for everybody

Environment Setup

- Go to https://www.python.org
- Download and install the latest version of Python
- Type and run your very first line of code:

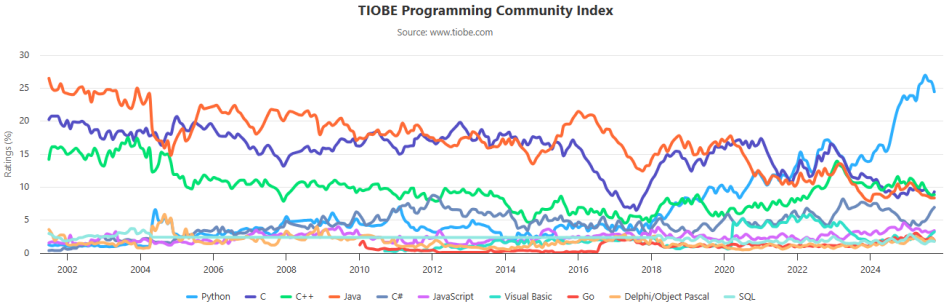
```
1 print("hello world!")
```

Introduction

Python is an interpreted, high-level, general-purpose programming language.
It is known for being beginner-friendly with a simple, readable syntax.
Its extensive standard library makes it very popular for everything :

- automation
- web development
- data analysis
- scripting
- machine learning

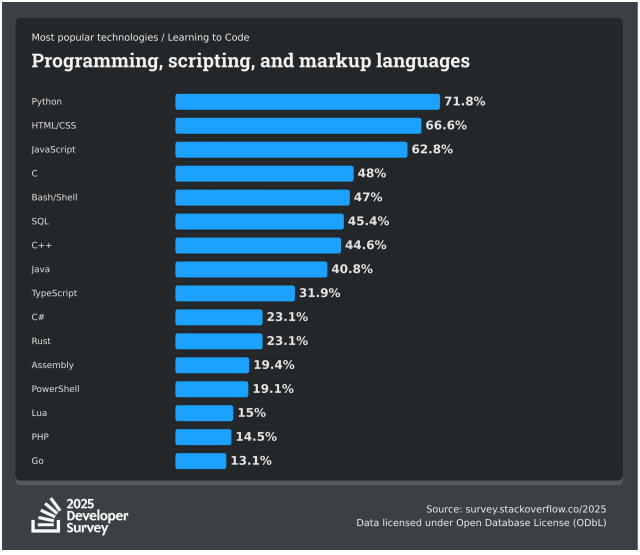
Some motivation



Important :

It is important to note that the TIOBE index is not about the best programming language or the language in which most lines of code have been written. The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system.

Some motivation



Some motivation

```c
C

#include <stdio.h>
int main() {
    printf("Hello world\n")
    ;
    return 0;
}
```

```java
JAVA

public class App {
    public static void main(String[] args)
     {
        System.out.println("Hello world");
    }
}
```

```python
Python

print("Hello world")
```

1. Variables

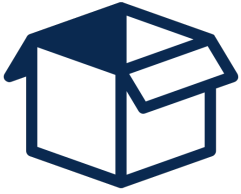
1. Variables

1.1. Definition

Variables

A variable is an allocated space in the computer's memory used to store data that can change during a program's execution.

Simple analogy:
You can think of a variable as a box where you can store a value.
You can later open the box and replace the value with another one.



1. Variables

1.2. Data types

Data types

A variable can hold different types of data, including:

- **int** : integer numbers (e.g., 5, -12, 100)
- **float** : decimal numbers (e.g., 3.14, 0.5, -2.7)
- **str** : text or string of characters (e.g., "Hello", 'Python')
- **bool** : logical values (True or False)

Practice

1. Task: Assign Multiple Values of Different Types

```
Club = "AppsClub"
is_my_favorite_club = True
members = 1000
price = 10.00
```

2. Task: Use the print() function to display the values of your variables.

```
print(Club)
print("AppsClub has",members,"members")
```

3. Task: Use the type() function to check the data type of each variable.

```
print(type(Club))
print(type( is_my_favorite_club))
```

4. Task: Write a short program that swaps the values of two variables.

1. Variables

1.3. Data manipulation

Data manipulation

A. Arithmetic Operations:

```
a  = 15
b = 6
c  = a + b   =>  21
a / b ===> 2.5
a // b ====> 2
a % b ====> 3
c = c + 1
2**3 = 8
```

Data manipulation

B. String Operations:

```
f_name = "ahmed"
l_name = "karim"
full_name = f_name + " " + l_name
line = "-"
line * 20
```

C. Boolean and comparison Operations:

```
a , b = 10 , 5
compare = (a == b)
compare = (a >= b)
print( a < b)
print( a != b )
```

1. Variables

1.4. Casting a variable

Casting a variable

Try this example:

```
1 a = '1'
2 print(a + 1)
```

What happens ?

Casting a variable

Try this example:

```
a = '1'
print(a + 1)
```

What happens ? ==> Error

Solution:
Casting a variable means converting the type of its value from one data type to another using built-in functions such as int(), str(), float(), or bool().

Exemple:

```
a =  10
a = str(a)    ==>  "10"
b = 6.53
b = int(b)   ==>  "6"
```

1. Variables

1.5. Pythonic tricks

Pythonic Tricks

```python
# Assign the same value to multiple variables
a = b = c = 1

# Assign multiple values in a single line
name, age = "Ali", 16

# Swap values between two variables
a, b = b, a
```

Practice

2. Contol Flow Structures

2. Contol Flow Structures

2.1. Conditional Statements

Conditional Statements

A. Conditional execution

```
Club = input("Enter your favorite club: ")

if Club == "AppsClub":
    print("Great choice! AppsClub is awesome!")
```

B. Alternative execution:

```
number = int(input("Enter a number: "))

if number % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

Conditional Statements

C. Chained conditionals

```
temperature = float(input("Enter the temperature: "))

if temperature < 15:
    print("It's a cold day!")
elif temperature < 30:
    print("Nice weather today!")
else:
    print("It's a hot day!")
```

Session 2

2. Contol Flow Structures

2.2. Loops

Introduction to Loops

Sometimes, we need to repeat a set of instructions several times in the same way.
Python provides two main looping structures: **for** and **while**.

While Loops

Syntax :

```
while CONDITION:
    #instructions
```

All instructions inside the while loop are executed repeatedly until the condition becomes False.
Example :

```
n = 0
while n < 5:
    print(n)
    n = n + 1
print('End')
```

Note : Each time the block of instructions inside the loop runs, it's called an iteration.

While Loops

If the stopping condition never becomes False, the loop will run forever.

```
x = 0
while x >= 0:
    print(x)
    x = x + 1
print("End")
```

==> This creates an infinite loop.

While Loops

- **break** : stops the loop completely

```
# Look for N in [0, 100] where N ** 2 = 36
x = 0
while x < 100:
    print("x =", x)
    if x**2 == 36:
        print("N is:", x)
        break
    x = x + 1
```

While Loops

- **continue** : skips the current iteration and goes to the next one.

```
# Print the square of every even number below 16
n = 0
while n < 16:
    if n % 2 == 1:
        n = n + 1
        continue
    print(n ** 2)
    n = n + 1
```

Real Example: Simple Chatbot

```
# Simple Chatbot
print("Hello, I'm your Chatbot!")
user_query = input("What do you want: ")

while True:
    if user_query == "":
        continue
    if user_query == "END":
        break
    print("Okay, I will do:", user_query)
    user_query = input("What's next: ")

print("Bye!")
```

For loops

In the previous example, we didn't know when the user would stop, so a while loop was suitable.
But when we know in advance how many iterations are needed (for example, summing numbers from 1 to n), a for loop is more convenient and easier to use.

Syntax:

```
for i in range(start, end, step):
    # instructions
```

For loops

Example 1:

```
for i in range(0, 5, 1):
    print(i)

# Default start=0 and step=1
for i in range(5):
    print(i)
```

For loops

Example 2: Let's calculate the term of a sequence:

$$U_n = \sum_{k=1}^{n} k^2$$

```
n = 20
result = 0

for k in range(1, n + 1):
    result = result + k ** 2

print("U(" + str(n) + ") = " + str(result))
```

3. Lists

3. Lists

3.1. What is a List in Python?

What is a List in Python?

A list is a collection (or sequence) of elements.

```
1 clubs = ['AppsClub', 'CRRT', 'CreArt']
```

In python:

1. A list can contain elements of **different** data types.
2. A list can contain **nested lists** (lists inside lists).
3. Lists are **dynamic**, meaning we don't need to resize them when adding new items.

3. Lists

3.2. List Manipulation

List manipulation

```python
names = ["Mohamed", "Ahmed", "Ali", "Khalid"]

# Accessing Elements
print(names[0])    # First element (index starts at 0)
print(names[-1])   # Last element

# Modifying a List Element
names[2] = "Lahoucine"

# Checking if an Element Exists
check = "Mohamed" in names
print(check)  # True

```

List manipulation

Traversing a List

- **Method 1 (read-only):**

```
for name in names:
    print(name)
```

- **Method 2 (can modify elements):**

```
for i in range(len(names)):
    print(names[i])
```

List manipulation

List Slicing: Sometimes we need just a part of the list — we can easily copy it using slicing.

Syntax

```
list[start:end:step]
```

Examples:

```
numbers = [9, 1, 5, 3, 7]
print(numbers[0::2])  # [9, 5, 7]
print(numbers[1:3])   # [1, 5]
```

3. Lists

3.3. List Methods

List methods

```
numbers = [9, 1, 2, 7, 6, 4]

# Add one element
numbers.append(10)

# Add multiple elements
numbers.extend([5, 7, 2])

# Remove the first occurrence of 9
numbers.remove(9)

# Remove the element at index 0
numbers.pop(0)

# Sort the list
numbers.sort()
```

4. String

4. String

4.1. String vs List

String vs List

A string is a sequence of characters.

```
fruit = "banana"
```

Strings are **not** lists, but they share many features:

```
letter = fruit[1]
print(letter)

print('b' in fruit)

sliced_word = fruit[0:3]

for char in fruit:
    print(char)
```

String vs List

Strings are **immutable**, which means you can't change an existing string :

```
fruit = "banana"
fruit[0] = 'v' #Error
```

The best you can do is create a **new string** that is a variation on the original:

```
fruit = "banana"
new_fruit = 'v' + fruit[1:]
print(new_fruit)
```

4. String

4.2. String Methods

String methods

```
fruit = "banana"

index = fruit.find('b')       # Returns index of 'b'which is 0

characters = list(fruit)      # ['b', 'a', 'n', 'a', 'n', 'a']

full_name = "Mohamed Ali Mahmoud"
names = full_name.split(" ")  # ['Mohamed', 'Ali', 'Mahmoud']

words = ['look', 'for', 'a', 'job']
sentence = " ".join(words)    # 'look for a job'
```