correlation.·one

TECH FOR JOBS

Support Session 9

Ahmad Albaqsami

Agenda

- Introduction to Python
- Data Structures in Python

About Python

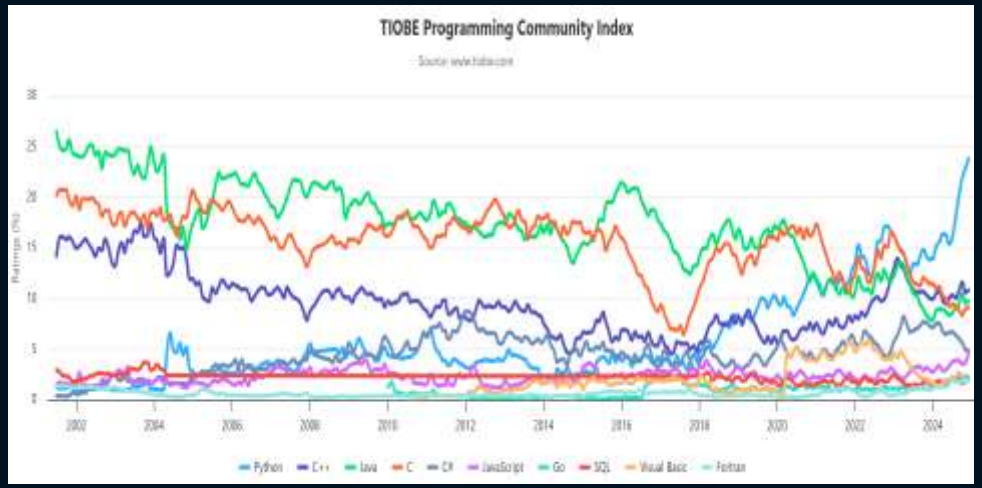
- Invented in 1989 by Guido van Rossum
- Multi-paradigm programming language
  - Object Oriented Programming
  - Structured programming
  - Functional Programming
- Dynamically typed
- Batteries included!

Where is Python used

- Artificial Intelligence and Machine Learning
- Data Science
- Web development
- Automation Testing
- Game development
- Computer Vision
- Image Processing
- Web scraping
- Finance
- Data Analytics
- Internet of Things

Python Popularity

- Source: https://www.tiobe.com/tiobe-index/



Dec 2024	Dec 2023	Change	Programming Language	Ratings	Change
1	1		Python	23.84%	+9.98%
2	3	^	C++	10.82%	+0.81%
3	4	^	Java	8.72%	+1.72%
4	2	v	C	9.11%	-2.34%
5	5		C#	4.8%	-2.43%
6	6		JavaScript	3.91%	+1.72%
7	13	^	Go	2.17%	+1.14%
8	8	^	SQL	1.98%	+0.32%
9	9	v	Visual Basic	1.96%	+0.14%

“Hello World”

In C :

```c
#include <stdio.h>

int main() {
   printf("Hello World");
   return 0;
}
```

In C# :

```csharp
namespace HelloWorld
{
    class Hello {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello World");
        }
    }
}
```

In Java :

```java
import java.io.*;

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

In Javascript:

```javascript
console.log("Hello World");
```

In Python:

```python
print("Hello World")
```

Where to download the Interpreter

- https://www.python.org/

Anaconda Navigator

- https://www.anaconda.com/download

Programming – In general!

- Data: Information that is being analyzed such as words, numbers, etc
- Algorithm: A computational Procedure for solving a problem
- Data Structure: The way the data is organized

Input -> Manipulation -> output

Data {Input} -> {data manipulation via Algorithm} -> Data {Output}

Variables

- A **variable** is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”

- Programmers get to choose the names of the variables

- You can change the contents of a variable in a later statement
- Rules of naming:
  - Must start with a letter or underscore _
  - Must consist of letters, numbers, and underscores
  - Case Sensitive

Reserved Word

- You cannot use reserved words as variable names / identifiers

```
False    class    return   is       finally
None     if       for      lambda   continue
True     def      from     while    nonlocal
and      del      global   not      with
as       elif     try      or       yield
assert   else     import   pass
break    except   in       raise
```

Assignment and Expressions

- An expression is a piece of code that produces a value
- A statement is a piece of code that performs an action
- expressions have a return value, while statements do not. This means that expressions can be used to compute a value and assign it to a variable.

```
x = 2
x = x + 2
print(x)
```

Operators

- special symbols that perform operations on values and variables

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Type

- In Python variables and literals have a “type”
- Python knows the difference between an **integer** number and a **string**
- For example “+” means “addition” if something is a number and “concatenate” if something is a string
- We can ask Python what type something is by using the type() function

Numbers

- Numbers have two main types
    - Integers are whole numbers:
      -14, -2, 0, 1, 100, 401233
    - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- When you put an integer and floating point in an expression, the integer is implicitly converted to a float
- You can control this with the built-in functions int() and float()
- You can also use int() and float() to convert between strings and integers

Collections

- Collections are container objects that group data.
- Types:
  - List: Ordered, mutable.
  - Tuple: Ordered, immutable.
  - Dict: Key-value pairs, mutable.
  - Set: Unordered, unique items.

List

- A collection that is ordered and indexed

```
# Create a list
my_list = [1, 2, 3, 4]

my_list.append(5)  # Add element
my_list.remove(2)  # Remove element
my_list[1] = 10    # Update element
print(len(my_list))  # Length of list
```

Tuple

- Concept: Mutability
    - Mutable: Can be changed after creation (add, remove, update elements).
    - Immutable: Cannot be changed after creation.

```
# Create a tuple
my_tuple = (1, 2, 3, 4)

print(my_tuple[1])    # Access by index
print(len(my_tuple))  # Length of tuple
```

Dictionary

- Keys vs Indexes:
  - Indexes: Used to access elements by position (e.g., in lists, tuples).
  - Keys: Used in dictionaries for value lookup.

```
# Create a dictionary
my_dict = {'a': 1, 'b': 2, 'c': 3}

my_dict['a'] = 10      # Update value
my_dict['d'] = 4       # Add new key-value pair
print(my_dict.keys())  # Get all keys
```

Set

- Subscriptable vs Unordered:
  - **Subscriptable**: Data structures (lists, tuples, dicts) allow access via indexes or keys.
  - **Unordered (not subscriptable)**: Sets do not support direct indexing.
- **Set**: Collection of **unique**, unordered items.

```
# Create a set
my_set = {1, 2, 3, 4}
my_set.add(5)        # Add element
my_set.remove(2)     # Remove element
print(3 in my_set)   # Membership test
```

Set Operators

- Intersection (&): Common elements between sets.
- Union (|): Combine all unique elements.
- Difference (-): Elements in one set but not the other.
- Symmetric Difference (^): Elements in either set, but not both.

Set Operators

```
set_a = {1, 2, 3, 4}
set_b = {3, 4, 5, 6}

print(set_a & set_b)  # Intersection: {3, 4}
print(set_a | set_b)  # Union: {1, 2, 3, 4, 5, 6}
print(set_a - set_b)  # Difference: {1, 2}
print(set_a ^ set_b)  # Symmetric Difference: {1, 2, 5, 6}
```

Id()

- Purpose: Returns the unique identifier (memory address) of an object.
- Use Case: To check if two variables point to the same object in memory.
- Key Concepts:
  - Mutability and id():Mutable objects (e.g., lists, dictionaries) can have the same id even after modifications.
  - Immutable objects (e.g., tuples, strings) typically have a new id when **modified indirectly**.
- **Debugging**: To understand how Python handles objects in memory.

Objects

- Everything is an Object
  - In Python, everything is an object: numbers, strings, functions, classes, and even None.
- Objects have:
  - Type: Determines the kind of object (e.g., int, str).
  - Value: The data stored by the object.
  - Identity: The memory address where the object resides (accessed via id()).

Labels (Variables)

- **Labels**: Variables in Python are references (or pointers) to objects.

- Multiple labels can refer to the **same object**.

- a = 500 # 'a' points to the object 500
- b = a # 'b' now also points to the object 500
- print(id(a), id(b)) # Both IDs are the same

Immutable Objects

- Reassigning a variable creates a **new object**.

```
x = 500          # x points to 500
print(id(x))
x = x + 100      # New object created for 600
print(id(x))   # Different from the ID of 600
```

Mutable Objects

- Modifying the object does not create a new one.

```
my_list = [1, 2, 3]
print(id(my_list))  # Original ID
my_list.append(4)   # Modified in-place
print(id(my_list))  # Same ID
```

Python Memory model

- Variables are labels attached to objects, not the objects themselves.
- Reassigning a variable does not affect the object; it simply changes what the label points to.
- Immutable objects (e.g., int, str) cannot be changed in place; new objects are created.
- Mutable objects (e.g., list, dict) can be changed in place.