Python Fundamental

What is Python?

- Python is a `widely-used programming language` (popular) that is designed in such a way that it is easy to read and write.
- It is called `"high-level"` because human readable unlike other languages. It allows programmers to write code that is closer to human language and further from machine language, making it simpler to understand and work with.

Why Python is Considered High-Level:

- **Abstracts Complexity:** Python handles many of the complexities, such as memory management and data type conversions, so the programmer doesn't have to.
- **Easy Syntax:** The syntax of Python is straightforward and intuitive, resembling plain English, which makes it easier for developers to write and read code.

- **Built-in Functions and Libraries:** Python comes with a rich set of built-in functions and libraries that allow developers to perform a wide range of tasks without needing to write additional code from scratch.
- **Cross-Platform Compatibility:** Python code can run on various operating systems with minimal or no modification, making it versatile and convenient for developers.

How does the Python Language work?

- Python code is first compiled to bytecode
- which is then interpreted and executed by the Python Virtual Machine (PVM).

- **Source code** is converted into **bytecode**, a lower-level, platform-independent representation of the code.
- Bytecode files have a `.pyc` extension and are stored in a `__pycache__` directory.
- The **Python Virtual Machine (PVM)** executes the bytecode. The PVM is an interpreter that reads the bytecode and performs the corresponding operations on the system's hardware.

Key Features of Python

a. High-Level Language : Python abstracts many of the complex details of the computer, such as memory management, making it easier to write and understand.

b. Interpreted Language : Python is interpreted, meaning it executes the code line by line. This makes debugging easier but can be slower compared to compiled languages.

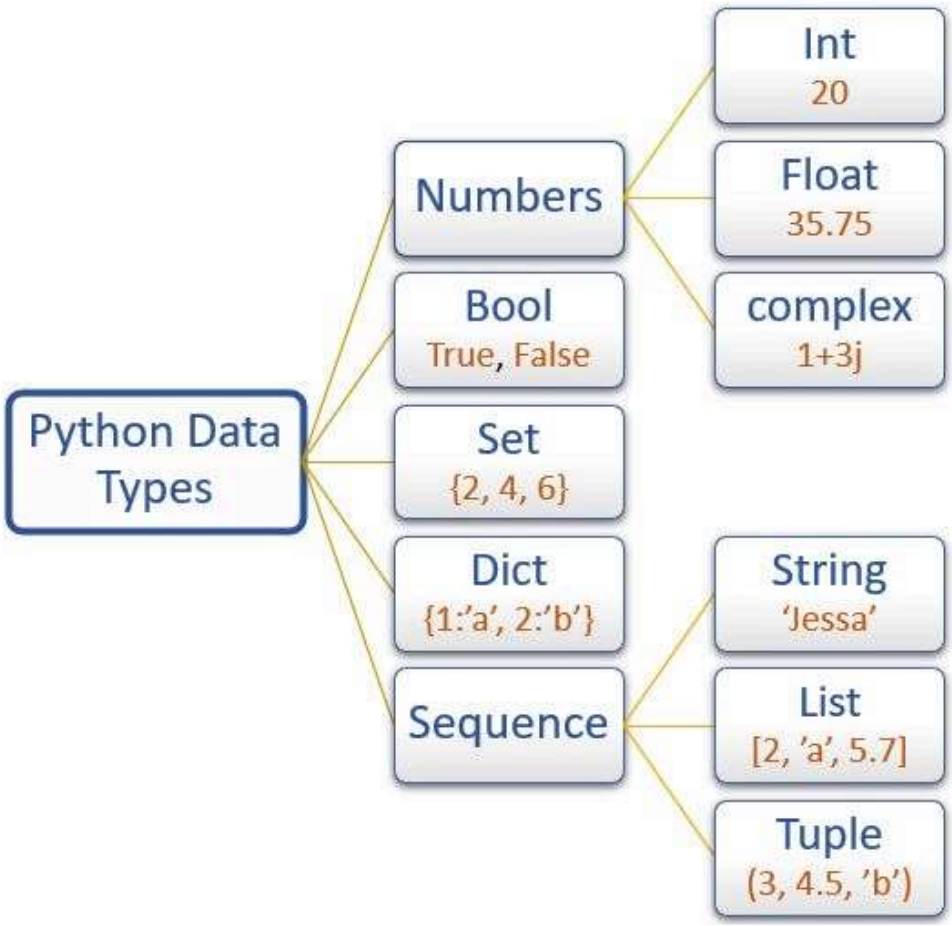
c. Dynamically Typed : Variable types are determined at runtime, which allows for more flexibility in coding but can lead to runtime errors if not managed carefully.

d. Garbage Collection : Python automatically manages memory allocation and deallocation through a garbage collector, which frees up memory that is no longer in use.

e. Extensive Standard Library : Python comes with a large standard library that provides modules and functions for many common tasks, such as file I/O, system calls, and

What are Python's built-in data types?

Integers, Floats, Strings, Lists, Tuples, Sets, Dictionaries, Booleans




```
# Boolean :
===========
x = True
x = False


===========
# None :
===========
 x= None


===========
# Numeric:
===========
x = 1  # Integer
x = 1.2 # Float
x = x = 1 + 2j #Complex

===========
# Sequence:
===========
x = "a"   # String
x = ["a","b", "c"] or [1,2,3] or [1.2, 1, "a", True]   # List
x = ("a","b","c") or(1,2,3) etc.   # Tuple


=======
```

```
# Set :
=======
x = {1, "s", 3, 5,"e" } # unique numbers
x =  frozenset([1, 2, 3, 4, 5]) # immutable_set


=======
# Map:
=======
x = {"k1" :1, "k2":2,"k3":3}  #Dictionary
example : student_grades = {"Alice": 90, "Bob": 85, "Charlie": 92}
key : value pair
```

Why Python is dynamically typed langauge?

Python is considered a dynamically typed language because **the type of a variable is determined at runtime** rather than at compile time. This means you don't need to explicitly declare the type of a variable when you create it. Instead, Python infers the type based on the value assigned to the variable.

In [1]:
```
a = 0
print(type(a))
```

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

In [ ]:
```
a = 3.4
print(type(a))
```

In [2]:
```
a = 1 + 2j
print(type(a))
```

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

In [3]:
```
a = "python"
print(type(a))
```

```
<class 'float'>
```

In [ ]:
```
a = True
print(type(a))
```

In [4]:
```
a =[1,2,'a']
print(type(a))
```

```
<class 'list'>
```

In [ ]:
```
a = (1,2,3,)
print(type(a))
```

In [6]:
```
a ={"a":1,"b":"ss02", "c":2.3}
print(type(a))
```

```
<class 'dict'>
```

In [10]:
```
a ={1,2,3,4}
print(type(a))
```

```
<class 'set'>
```

Python Variables and Data Type

The primary purpose of computers is to process data into useful information, for that to happen, the data needs to be stored in its memory. This is achieved using a programming language's variables and data types.

Data types in Python are particular kinds of data items, as defined by the value they can take. Variables, on the other hand, are like labeled containers that store this data. They enable you to manage and modify information using specific identifiers.

Data types are generally classified into two types:

- **Primitive (Fundamental) Data**
- **Non-Primitive (Composite) Data**

Primitive (Fundamental) Data Types in Python:

Primitive data types represent simple values. These data types are the most basic and essential units used to store and manipulate information in a program. They translate directly into low-level machine code.

Primitive data types include:

- **String (str):** Represents sequences of characters. Should be enclosed in quotes. Example: "Hello, Python!"
- **Integer (int):** Represents whole numbers without decimals. Example: 42
- **Float (float):** Represents numbers with decimals. Example: 3.14
- **Boolean (bool):** Represents either True or False.

Non-Primitive (Composite) Data Types in Python :

Non-primitive data types are structures that can hold multiple values and are composed of other data types, including both primitive and other composite types. Unlike primitive data types, non-primitive types allow for more complex and structured representations of data.

Non-primitive data types include:

- **List (list):** Represents an ordered and mutable collection of values. Example: fruits = ["apple","banana", "apple","orange"]
- **Tuple (tuple):** Represents an ordered and immutable collection of values. Example: coordinates = (3, 7)
- **Set (set) :** Respresents unordered collection of unique items. Example : fruits = {"apple", "banana", "orange"}
- **Dictionary (dict):** Represents an unordered collection of key-value pairs. Example: person = {"name": "Alice", "age": 25, "is_student": True}

Characteristics of Non-Primitive Data Types:

- Mutability: Lists are mutable, meaning their elements can be modified after creation. Tuples, on the other hand, are immutable – their elements cannot be changed. Dictionaries are mutable – you can add, modify, or remove key-value pairs.
- Collection of Values: Non-primitive data types allow the grouping of multiple values into a single structure, enabling the creation of more sophisticated data representations.
- Ordered (Lists and Tuples): Lists and tuples maintain the order of elements, allowing for predictable indexing.
- Key-Value Mapping (Dictionary): Dictionaries map keys to values, providing a way to organize and retrieve data based on specific identifiers.

The Versatility and Power of Python in the Tech Industry

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc. The biggest strength of Python is huge collection of standard library which can be used for the following:

- Machine Learning
- GUI Applications
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia
- Scientific computing
- Text processing and many more..

Python Libraries in Data Science

Python's extensive ecosystem of libraries and tools makes it a powerful language for data science. Here are some of the key libraries used in data science:

- NumPy: Fundamental package for numerical computation in Python, providing support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- Pandas: Essential library for data manipulation and analysis, offering data structures like DataFrames and functions needed to manipulate numerical tables and time series data.
- Matplotlib: Comprehensive library for creating static, animated, and interactive visualizations in Python, often used for plotting complex graphs and charts.
- Seaborn: Data visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
- SciPy: Library used for scientific and technical computing, built on NumPy and offering additional functionalities for optimization, integration, interpolation, eigenvalue problems, and more.
- Scikit-Learn: Simple and efficient tools for data mining and data analysis, built on NumPy, SciPy, and Matplotlib, and widely used for implementing machine learning algorithms.
- TensorFlow: Open-source machine learning framework developed by Google, extensively used for building and deploying machine learning models, particularly deep learning models.
- Keras: High-level neural networks API, written in Python and capable of running on top of TensorFlow, making it easier to build and train deep learning models.
- PyTorch: Open-source machine learning library developed by Facebook's AI Research lab, known for its flexibility and dynamic computation graph, making it

popular for research and production use.

- Statsmodels: Provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests and statistical data exploration.
- NLTK: Natural Language Toolkit, a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English.
- spaCy: Industrial-strength NLP library designed for efficient processing of large volumes of text, offering functionalities like tokenization, parsing, named entity recognition, and more.
- Flask : A lightweight and versatile web framework for Python, designed to make it easy to build web applications quickly and with minimal overhead.

These libraries are critical for data scientists and analysts, enabling them to efficiently manipulate data, perform complex computations, and create insightful visualizations and machine learning models.

In [ ]: