Python Fundamentals - I

Course - Data Analysis with Python

Environment Setup

1. Install Python 3.10
2. Install Jupyter Lab / Notebook - pip install jupyterlab

Python Overview

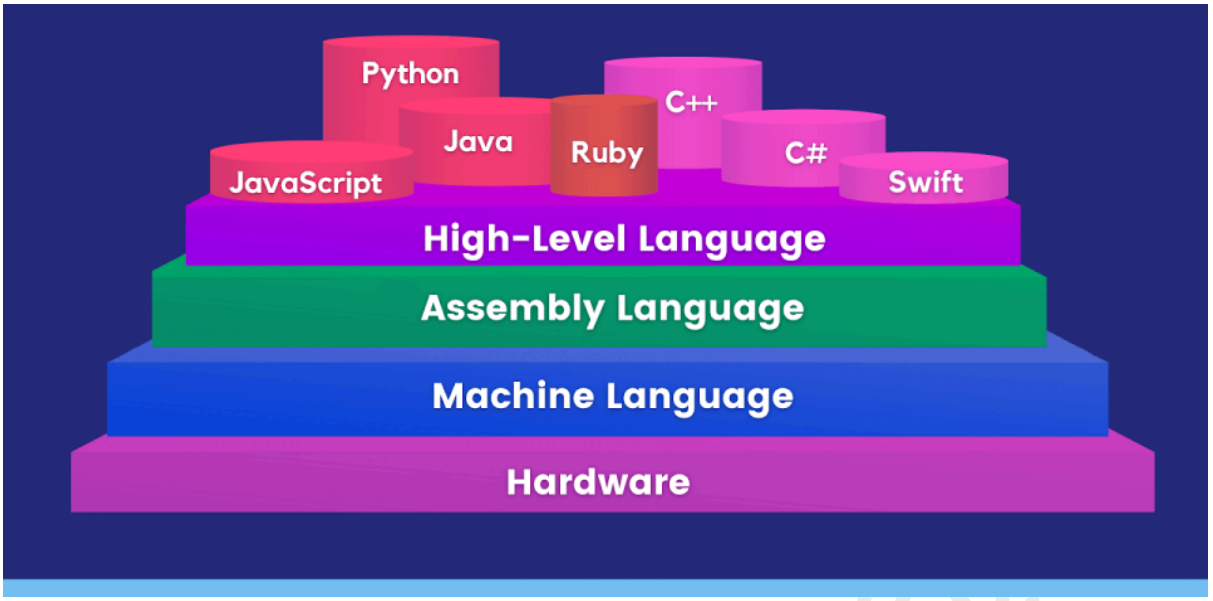
History of Python

Python, a general-purpose high-level computer programming language valued for its English-like syntax and powerful built-in data analysis and data science functions and libraries.

The Dutch programmer **Guido van Rossum** developed Python in 1991. Python, which he named after the British television series **Monty Python's Flying Circus**, was publicly released in 1994. Although Van Rossum remained deeply involved in Python's development until 2018, a robust community of other developers also made significant contributions.

Python has seen three primary evolutions: **Python 1.0, released in 1994**; **Python 2.0, in 2000**; and **Python 3.0, in 2008**. Notably, Python 3.0 is not backward-compatible with earlier versions. By the early **2020s** Python had become one of the most extensively used programming languages worldwide.

The Python Software Foundation describes Python as "an interpreted, object-oriented, high-level programming language with dynamic semantics." Unlike such languages as Java, Python is an interpreted language, indicating that its source code can be directly used and executed without needing a compiler. Python is also an object-oriented language, in contrast to functional programming languages, such as C. Object-oriented languages design software around objects, which can be real-world entities, such as cars, or abstract concepts, such as numbers. Objects are instances of a class (for example, the class "cars") and have methods and attributes. This contrasts with languages that center on a series of functions.



Moreover, Python is defined as a high-level programming language (in opposition to low-level languages, such as assembly), which corresponds to its high level of abstraction from hardware. High-level languages are designed for human understanding and must be interpreted before they are read by machines. Finally, Python is also defined as having dynamic semantics, in contrast to a statically typed language such as C, because variable names (for example, "x") can point to objects of any type. For instance, "x" can equal the number 3, but the same variable name can also be assigned the value of the string "car" or the value of the list [50, 150, 200]. (*Don't worry, we'll learn how it will happen.*)

Python's surge in popularity has been due partly to its clear and concise syntax, which enhances readability. Python emphasizes code clarity and promotes a sense of elegance. It is prescriptive in upholding these attributes. For example, Python has a maximum recommended line length of 79 characters and a specific indentation style, which encourages the use of four white spaces while prohibiting space and tab mixing. Python's versatility is another notable strength. Although it is primarily an object-oriented language, Python can also be harnessed for procedural or functional applications.

- ➔ Differences between Procedural and Object Oriented Programming - GeeksforGeeks
- ➔ Functional Programming Paradigm - GeeksforGeeks

Data and Data Type

What is data and what is its type?

Data and data types are fundamental concepts in computer science and programming. Data refers to information, facts, or values that are stored and manipulated by a computer program. Data types specify the kind of data that can be stored and the operations that can be performed on it. Here's a description of data and common data types:

Data: In computing, data is any piece of information that can be processed or manipulated by a computer. Data can be represented in various forms, including numbers, text, images, audio, video, and more.

Data Types: Data types define the type of data that a variable can hold, the operations that can be performed on the data, and the memory space required to store the data. Common data types include:

- **Integer**: Represents whole numbers without any fractional part. Examples include 0, 1, -5, 1000, etc.
- **Floating-point**: Represents numbers with a fractional part. Examples include 3.14, -0.001, 2.71828, etc.
- **Boolean**: Represents logical values indicating either true or false.
- **String**: Represents a sequence of characters. Examples include "hello", "world", "123abc", etc.
- **Character**: Represents a single character. Examples include 'a', 'B', '5', '$', etc.
- **Array**: Represents a collection of elements of the same data type, arranged sequentially in memory.
- **Struct/Record**: Represents a composite data type that groups together variables under one name in a block of memory.
- **Pointer**: Represents a memory address that points to another data type.
- **Enumeration**: Represents a set of named integer constants.
- **Void**: Represents the absence of a data type, often used to denote functions that return no value.
- **Custom/User-defined types**: Data types defined by the programmer to suit specific requirements.

What are data and data types in Python?

In Python, data and data types play a crucial role, as Python is a dynamically typed language, meaning that variables can hold values of any data type and data types are inferred at runtime. Here's an overview of data and data types in Python:

➔ **Data**: In Python, data can be any information that the program handles, such as numbers, strings, lists, tuples, dictionaries, and more. Data can also be objects and instances of classes.

➔ **Data Types** in Python:

- ◆ **Numeric Types**:
  - ○ int: Represents integer values, e.g., 5, -3, 1000.
  - ○ float: Represents floating-point numbers, e.g., 3.14, -0.001, 2.71828.
  - ○ complex: Represents complex numbers with a real and imaginary part, e.g., 3 + 4j.
- ◆ **Sequence Types**:
  - ○ str: Represents strings, sequences of characters enclosed within single quotes (' ') or double quotes (" "), e.g., "hello", 'world'.
  - ○ list: Represents mutable ordered sequences of elements, e.g., [1, 2, 3], ['a', 'b', 'c'].
  - ○ tuple: Represents immutable ordered sequences of elements, e.g., (1, 2, 3), ('a', 'b', 'c').
- ◆ **Mapping Type**:
  - ○ dict: Represents mutable collections of key-value pairs, e.g., {'name': 'Alice', 'age': 30}.
- ◆ **Set Types**:
  - ○ set: Represents mutable unordered collections of unique elements, e.g., {1, 2, 3}.
  - ○ frozenset: Represents immutable sets, e.g., frozenset([1, 2, 3]).
- ◆ **Boolean Type**:
  - ○ bool: Represents boolean values True or False.
- ◆ **None Type**:
  - ○ NoneType: Represents the absence of a value or null.
- ◆ **Binary Types**:
  - ○ bytes: Represents immutable sequences of bytes, e.g., b'hello'.
  - ○ bytearray: Represents mutable sequences of bytes, e.g., bytearray(b'hello').

➔ **Type Inference**: In Python, data types are dynamically inferred at runtime. You don't need to explicitly declare the data type of a variable; Python determines it based on the value assigned to the variable.

```
# Example demonstrating type inference in Python
x = 5              # x is an int
y = 3.14           # y is a float
name = "Alice"     # name is a str
is_valid = True    # is_valid is a bool
```

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.. For example –

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print counter
print miles
print name
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example –

a = b = c = 1

- ➔ Get the Variable Type
  - ◆ You can get the data type of a variable with the type() function.
  - ◆ Example: print(type(counter)), print(type(miles)), print(type(name))
- ➔ Variable names are case-sensitive.
  - ◆ Example:

```
m = 100
M = "Python"
#M will not overwrite m
```

Practice Problem

1. Create 3 variables randomly: an integer, a float, and a string. Then print all, and replace the all variable into a string and print it.

2. Create a variable named **bookname** and assign the value **Mockingbird** to it.
3. Create 2 variables named **first_name** and **last_name**, input them from the keyboard, and print them.