What is Python? What are the benefits of using Python

Python is a high-level, interpreted, general-purpose programming language. Being a general-purpose language, it can be used to build almost any type of application with the right tools/libraries. Additionally, python supports objects, modules, threads, exception-handling, and automatic memory management which help in modelling real-world problems and building applications to solve these problems.

Benefits of using Python:

- Python is a general-purpose programming language that has a simple, easy-to-learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Moreover, the language is capable of scripting, is completely open-source, and supports third-party packages encouraging modularity and code reuse.
- Its high-level data structures, combined with dynamic typing and dynamic binding, attract a huge community of developers for Rapid Application Development and deployment.

What is a dynamically typed language?

Before we understand a dynamically typed language, we should learn about what typing is. **Typing** refers to type-checking in programming languages. In a **strongly-typed** language, such as Python, **"1" + 2** will result in a type error since these languages don't allow for "type-coercion" (implicit conversion of data types). On the other hand, a **weakly-typed** language, such as Javascript, will simply output **"12"** as result.

Type-checking can be done at two stages -

- **Static** - Data Types are checked before execution.
- **Dynamic** - Data Types are checked during execution.

Python is an interpreted language, executes each statement line by line and thus type-checking is done on the fly, during execution. Hence, Python is a Dynamically Typed Language.

What is an Interpreted language?

An Interpreted language executes its statements line by line. Languages such as Python, Javascript, R, PHP, and Ruby are prime examples of Interpreted languages. Programs written in an interpreted language runs directly from the source code, with no intermediary compilation step.

What is PEP 8 and why is it important?

PEP stands for **Python Enhancement Proposal**. A PEP is an official design document providing information to the Python community, or describing a new feature for Python or its processes. **PEP 8** is especially important since it documents the style guidelines for Python Code.

What is Scope in Python?

Every object in Python functions within a scope. A scope is a block of code where an object in Python remains relevant. Namespaces uniquely identify all the objects inside a program. However, these namespaces also have a scope defined for them where you could use their objects without any prefix. A few examples of scope created during code execution in Python are as follows:

- A **local scope** refers to the local objects available in the current function.
- A **global scope** refers to the objects available throughout the code execution since their inception.
- A **module-level scope** refers to the global objects of the current module accessible in the program.
- An **outermost scope** refers to all the built-in names callable in the program. The objects in this scope are searched last to find the name referenced.

Note: Local scope objects can be synced with global scope objects using keywords such as **global**

What are lists and tuples? What is the key difference between the two?

Lists and **Tuples** are both s**equence data types** that can store a collection of objects in Python. The objects stored in both sequences can have **different data types**. Lists are represented with **square brackets** `['sara', 6, 0.19]`, while tuples are represented with **parantheses** `('ansh', 5, 0.97)`.
But what is the real difference between the two? The key difference between the two is that while **lists are mutable**, **tuples** on the other hand are **immutable** objects. This means that lists can be modified, appended or sliced on the go but tuples remain constant and cannot be modified in any manner. You can run the following example on Python IDLE to confirm the difference:

What are the common built-in data types in Python?

There are several built-in data types in Python. Although, Python doesn't require data types to be defined explicitly during variable declarations type errors are likely to occur if the knowledge of data types and their compatibility with each other are neglected. Python provides `type()` and `isinstance()` functions to check the type of these variables. These data types can be grouped into the following categories-

- **None Type:**
  `None` keyword represents the null values in Python. Boolean equality operation can be performed using these NoneType objects.

Class Name	Description
NoneType	Represents the **NULL** values in Python.

- **Numeric Types:**
  There are three distinct numeric types - **integers, floating-point numbers**, and **complex numbers**. Additionally, **booleans** are a sub-type of integers.

Class Name	Description
int	Stores integer literals including hex, octal and binary numbers as integers
float	Stores literals containing decimal values and/or exponent signs as floating-point numbers
complex	Stores complex numbers in the form (A + Bj) and has attributes: `real` and `imag`
bool	Stores boolean value (True or False).

Note: *The standard library also includes* ***fractions*** *to store rational numbers and* ***decimal*** *to store floating-point numbers with user-defined precision.*

- **Sequence Types:**
  According to Python Docs, there are three basic Sequence Types - **lists, tuples,** and **range** objects. Sequence types have the `in` and `not in` operators defined for their traversing their elements. These operators share the same priority as the comparison operations.

Class Name	Description
list	Mutable sequence used to store collection of items.
tuple	Immutable sequence used to store collection of items.
str	Immutable sequence of Unicode code points to store textual data.

Note: The standard library also includes additional types for processing:
1. **Binary data** such as `bytearray` `bytes` `memoryview` , and
2. **Text strings** such as `str`.

- **Mapping Types:**

  A mapping object can map hashable values to random objects in Python. Mappings objects are mutable and there is currently only one standard mapping type, the ***dictionary***.

Class Name	Description
dict	Stores comma-separated list of **key: value** pairs

- **Set Types:**
  Currently, Python has two built-in set types - **set** and **frozenset**. **set** type is mutable and supports methods like `add()` and `remove()`. **frozenset** type is immutable and can't be modified after creation.

Class Name	Description
set	Mutable unordered collection of distinct hashable objects.
frozenset	Immutable collection of distinct hashable objects.

Note: *`set` is mutable and thus cannot be used as key for a dictionary. On the other hand, `frozenset` is immutable and thus, hashable, and can be used as a dictionary key or as an element of another set.*

- **Modules:**
  Module is an additional built-in type supported by the Python Interpreter. It supports one special operation, i.e., **attribute access**: `mymod.myobj`, where `mymod` is a module and **myobj** references a name defined in m's symbol table. The module's symbol table resides in a very special attribute of the module **__dict__**, but direct assignment to this module is neither possible nor recommended.
- **Callable Types:**
  Callable types are the types to which function call can be applied. They can be **user-defined functions, instance methods, generator functions**, and some other **built-in functions, methods** and **classes**.
  Refer to the documentation at docs.python.org for a detailed view of the **callable types**.

What is pass in Python?

The `pass` keyword represents a null operation in Python. It is generally used for the purpose of filling up empty blocks of code which may execute during runtime but has yet to be written. Without the **pass** statement in the following code, we may run into some errors during code execution.

What is break, continue and pass in Python?

Break	The break statement terminates the loop immediately and the control flows to the statement after the body of the loop.
Continue	The continue statement terminates the current iteration of the statement, skips the rest of the code in the current iteration and the control flows to the next iteration of the loop.
Pass	As explained above, the pass keyword in Python is generally used to fill up empty blocks and is similar to an empty statement represented by a semi-colon in languages such as Java, C++, Javascript, etc.

What is slicing syntex in Python?

- As the name suggests, 'slicing' is taking parts of.
- Syntax for slicing is **[start : stop : step]**
- **start** is the starting index from where to slice a list or tuple
- **stop** is the ending index or where to sop.
- **step** is the number of steps to jump.
- Default value for **start** is 0, **stop** is number of items, **step** is 1.
- Slicing can be done on **strings, arrays, lists**, and **tuples**.

What is the difference between Python Arrays and lists?

- Arrays in python can only contain elements of same data types i.e., data type of array should be homogeneous. It is a thin wrapper around C language arrays and consumes far less memory than lists.
- Lists in python can contain elements of different data types i.e., data type of lists can be heterogeneous. It has the disadvantage of consuming large memory.

What is PYTHONPATH in Python?

PYTHONPATH is an environment variable which you can set to add additional directories where Python will look for modules and packages. This is especially useful in maintaining Python libraries that you do not wish to install in the global default location

Explain how to delete a file in Python?

Use command **os.remove(file_name)**

Explain split() and join() functions in Python?

- You can use **split()** function to split a string based on a delimiter to a list of strings.
- You can use **join()** function to join a list of strings based on a delimiter to give a single string.

What are negative indexes and why are they used?

- Negative indexes are the indexes from the end of the list or tuple or string.
- **Arr[-1]** means the last element of array **Arr[]**