Python

Python - Introduction

Python is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python emphasizes code readability with its clear and concise syntax, making it an ideal language for beginners and experienced developers.

Features and Advantages:

Simple and Readable Syntax: Python code is easy to read and understand, reducing the cost of program maintenance and development time.

Extensive Standard Library: Python comes with a comprehensive standard library, providing modules and packages for a wide range of tasks, from file I/O to networking.

Cross-platform: Python is available on all major operating systems (Windows, macOS, Linux), making it highly portable.

Large Community and Ecosystem: Python has a vast community of developers who contribute libraries, frameworks, and resources, fostering innovation and collaboration.

Versatility: Python can be used for various purposes, including web development, data analysis, artificial intelligence, scientific computing, automation, and more.

Why Python?

Large standard library and active community support

Versatility: used for web development, data analysis, machine learning, and more

Cross-platform compatibility

Strong integration capabilities with other languages and tools

Python Installation

Installing Python on different platforms (Windows, macOS, Linux)

Managing Python versions (Python 2 vs. Python 3)

Setting up Python environment (IDEs, text editors, virtual environments)

Different ways of Installing Python - Official Python Installer, Anaconda Distribution, Jupyter, Spyder, PyCharm & Visual Studio

Basic Concepts

Variables, data types (integers, floats, strings, Booleans), and **type conversion**.

Operators (arithmetic, comparison, logical).

Arithmetic Operator: Used to perform Mathematical Operations (+/*-=)

Comparison Operators: Used to compare values (e.g., ==, !=, <, >).

Logical Operators: Combine conditions (e.g., `and`, `or`, `not`).

Input and output (using `input()` and `print()`).

Data Structures

Lists, Tuples, and Dictionaries, Sets:

Lists: Lists are ordered collections of items. They are mutable, meaning you can modify their contents after creation. Lists are defined using square brackets `[]`.

Tuples: Tuples are similar to lists but are immutable (cannot be changed after creation). They are defined using parentheses `()`.

Dictionaries: Dictionaries store key-value pairs. They are unordered and mutable. Keys must be unique, and they are defined using curly braces `{}`.

Sets and Their Operations: Sets are unordered collections of unique elements. They are defined using curly braces `{}` or the `set()` constructor.

List Comprehensions:

List comprehensions provide a concise way to create lists based on existing lists (or other iterable objects).

```
squares = [x**2 for x in range(10)]
print(squares)
```

Control Structures

- **if…else Statements**: Execute code blocks based on conditions.
- **for Loop with** `range()`: Execute a code block a fixed number of times.
- **while Loop**: Execute a code block as long as a condition is true.
- **break**: Exit a loop prematurely.
- **continue**: Skip the current iteration and proceed to the next one.
- **pass**: Use as a placeholder.

Functions & Modules

Python functions allow you to organize code into reusable blocks.

Default Parameters: Specify default values for function parameters.

Keyword Arguments: Make function calls more explicit.

Recursive Functions: Define functions that call themselves.

Lambda Expressions: Create anonymous functions.

Docstrings: Document functions using docstrings.

Object Oriented Programming

OOP is a powerful paradigm that allows you to model real-world entities as objects, organize code into classes, and define relationships between them

Classes and Objects:

- A **class** is a blueprint for creating objects. It defines attributes (data members) and methods (functions) that the objects of that class will have.
- An **object** is an instance of a class. You can create multiple objects from the same class, each with its own state and behavior.

Attributes (Instance Variables): Attributes represent the data associated with an object. They are defined within the class and are specific to each instance.

Methods (Instance Methods): Methods are functions defined within a class. They operate on the object's data.

Inheritance: Inheritance allows you to create a new class (the subclass) based on an existing class (the base class).

Encapsulation: Encapsulation refers to hiding the internal details of an object and exposing only necessary methods and attributes.

Polymorphism: Polymorphism allows objects of different classes to be treated uniformly.

Modules and Libraries

Importing modules.

Common libraries (e.g., math, random, datetime).

Regular expressions. (re)

Decorators in Python are a powerful way to modify or enhance the behavior of functions or methods. They allow you to wrap a function with additional functionality without modifying its source code.

Generators are a type of iterator that allow lazy evaluation. They produce values one at a time, on demand, instead of creating an entire list or sequence in memory.

Iterator is a more general concept. An iterator is any object that implements the methods __iter__() and __next__()

Multithreading involves creating multiple threads within a single process, allowing different parts of the code to run concurrently. It's useful for tasks like I/O-bound operations or GUI applications.

Multiprocessing provides true parallelism by creating separate processes, each with its own memory space. It's suitable for CPU-bound tasks.

NumPy

Features:

- Provides support for large, multi-dimensional arrays and matrices.
- Offers mathematical functions for array manipulation (e.g., element-wise operations, linear algebra, Fourier transforms).
- Efficiently handles numerical computations.

Python

```python
import numpy as np
arr = np.array([1, 2, 3])
print(arr.mean())  # Calculates the mean
```

Pandas

Features:

- Provides data structures (Series and DataFrame) for handling tabular data.
- Supports data cleaning, transformation, and exploration.
- Integrates well with other libraries (e.g., NumPy, Matplotlib).

Python

```python
import pandas as pd
data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df = pd.DataFrame(data)
print(df.head())
```

Matplotlib

Features:

- Generates line plots, scatter plots, bar charts, histograms, etc.
- Customizable appearance (labels, colors, styles).
- Works well with Jupyter notebooks.

Python

```python
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [4, 7, 2]
plt.plot(x, y, marker='o')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Sample Plot')
plt.show()
```

Exception Handling

Try-Except Blocks:

Exception handling in Python is crucial for dealing with errors that might occur during program execution.
The try and except blocks allow you to handle exceptions gracefully.

Custom Exceptions:

Sometimes, built-in exceptions don’t cover specific scenarios in your program. In such cases, you can create your own custom exceptions.

To define a custom exception, create a new class that inherits from the built-in Exception class (or its subclasses).

```
class InvalidAgeException(Exception):

    """Raised when the input value is less than 18."""

    pass
```

File Handling

The open() function allows you to open a file and work with it.

There are different modes you can use when opening a file.

"r": Read mode (default). Opens the file for reading.

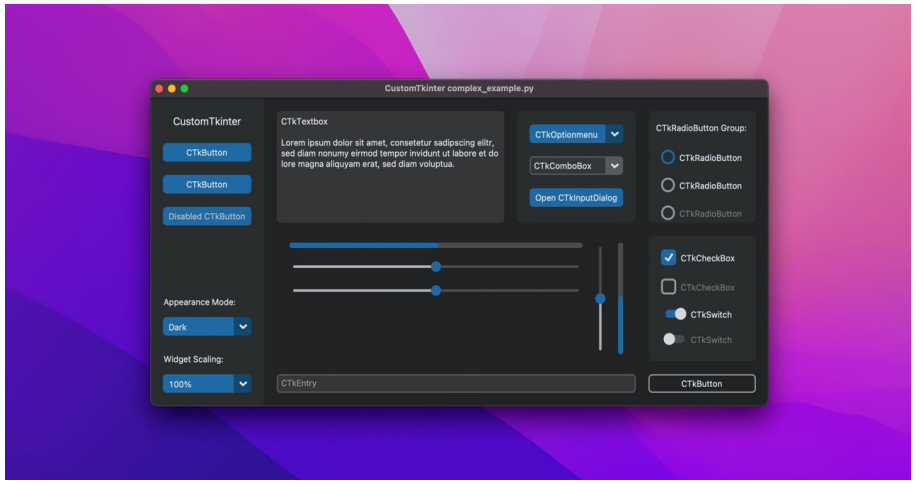
"w": Write mode. Opens the file for writing (creates a new file if it doesn't exist or truncates an existing file).

"a": Append mode. Opens the file for writing (appends data to an existing file or creates a new file).

"x": Create mode. Creates a new file for writing (raises an error if the file already exists).

Reading from and writing to files using with statements.

Remember to close the file after you're done working with it!



- **Tkinter** – Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look at this option in this chapter.
- **wxPython** – This is an open-source Python interface for wxWidgets GUI toolkit. You can find a complete tutorial on WxPython here.
- **PyQt** – This is also a Python interface for a popular cross-platform Qt GUI library. TutorialsPoint has a very good tutorial on PyQt5 here.
- **PyGTK** – PyGTK is a set of wrappers written in Python and C for GTK + GUI library. The complete PyGTK tutorial is available here.
- **PySimpleGUI** – PySimpleGui is an open source, cross-platform GUI library for Python. It aims to provide a uniform API for creating desktop GUIs based on Python's Tkinter, PySide and WxPython toolkits. For a detaile PySimpleGUI tutorial, click here.

GUI PROGRAMMING - **PYTHON** OFFERS MULTIPLE OPTIONS FOR DEVELOPING **GUI** (**GRAPHICAL USER INTERFACE**).

Introduction to Tkinter

A standard Python library for creating graphical user interfaces (GUIs)

Why Use Tkinter?

Built-in Library: No need for additional installations.

Cross-Platform: Windows, macOS, and Linux.

Basic Concepts of Tkinter

Widgets: These are the elements that make up the user interface, such as buttons, labels, text entries, etc.

Root Window: The main window of your application.

Event Loop: Tkinter applications operate in an event-driven manner, meaning they wait for user actions like clicks or key presses and respond accordingly.

Tkinter Widgets Overview

Label: Displays text or images.

Button: Creates a button that can trigger an action when clicked.

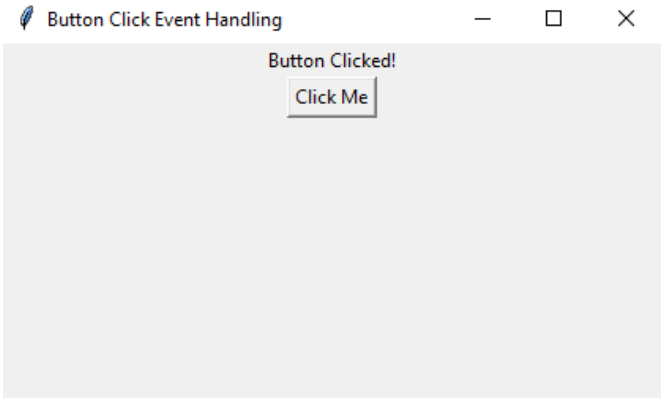
Entry: A single-line text box for user input.

Text: A multi-line text box for user input.

Frame: A container widget to hold other widgets.

Event Handling

- Event handling in Python refers to the process of capturing and responding to events, such as user input or system notifications. Different libraries and frameworks provide mechanisms for event handling.
- In **Tkinter**, event handling is done through binding events to event handlers. You can bind events to specific widgets or to the entire application window:
- Mouse Events
- Keyboard Events



Database Programming in Python

Python and SQL are powerful tools for data analysis and manipulation. Let's dive into how you can work with SQL databases using Python:

Connecting to a SQL Database:

- `sqlite3`: For SQLite databases (lightweight, file-based databases).
- `mysql-connector-python`: For MySQL databases.
- `psycopg2`: For PostgreSQL databases.

Basic Operations: Once connected, you can perform common database operations:

Inserting Data: Add new records to the database.

Updating Data: Modify existing records.

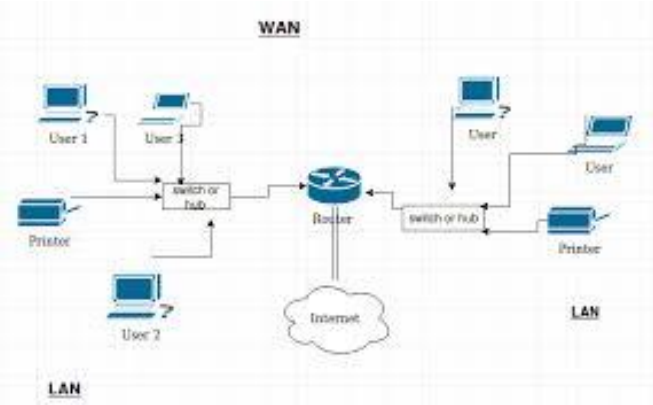
Deleting Data: Remove records.

Querying Data: Retrieve data based on specific conditions

Networking

Networking in Python allows you to create applications that communicate over a network.

Here are some resources and examples to get you started:



Socket Programming:

- **Sockets** provide a way to send messages across a network. They facilitate inter-process communication (IPC) between different endpoints.
- You can use Python's `socket` module to create client-server applications, handle multiple connections, and send data between endpoints.

Conversion of Python script to Executable Files

You can convert a Python script into an executable file for easier distribution and execution on systems that do not have Python installed. After installing PyInstaller we can create the executables,

Create an Executable: Let's assume you have a Python script named `my_script.py`. Navigate to the directory containing your script using the command prompt. Then run the following command:

```
pyinstaller my_script.py
```

This will create a `dist` folder containing your executable (`my_script.exe`). The executable will include all necessary dependencies.

Thanks