**Week 2 Python**

**Control Flow Statement**

A program’s control flow is the order in which the program’s code executes.

The control flow of a python program is regulated by conditional statements, loops and function calls.

Python has three types of control structures:

Sequential – default mode

Selection – used for decisions and branching.

Repetition – used for looping i.e., repeating a piece of code multiple times.

1 1. Sequential

Sequential statements are a set of statements whose execution process happens in a sequence. The problem with sequential statements is that if the logic has broken in any one of the lines, then the complete source code execution will break.

A computer screen shot of a black and white screen

Description automatically generated

2. Selection/Decision control statements

In Python, the selection statements are also known as Decision control statements or branching statements.

The selection statement allows a program to test several conditions and execute instructions based on which condition is true.

Some Decision Control Statements are:

Simple if

if-else

nested if

if-elif-else

Simple if: If statements are control flow statements that help us to run a particular code, but only when a certain condition is met or satisfied. A simple if only has one condition to check.

A diagram of a condition

Description automatically generated

A black and white screen with orange and white text

Description automatically generated

**Day 2**

**Functions**

Functions in Python are blocks of reusable code that perform a specific task. They allow you to organize your code, make it more modular, and avoid code duplication. Here's how you can define, call, and use functions in Python.

1. **General-Purpose Programming**: Python is a general-purpose programming language, which means it can be used for a wide variety of tasks, from web development to data analysis to scientific computing.
2. **Easy-to-Read Syntax**: Python has a clean and easy-to-read syntax, which makes it a great choice for beginners and experienced developers alike.
3. **Interpreted Language**: Python is an interpreted language, which means you don't need to compile your code before running it. You can write and execute code interactively, which makes it great for scripting and prototyping.
4. **Cross-Platform Compatibility**: Python is available on various platforms (Windows, macOS, Linux), making it easy to write code that can be run on different operating systems.
5. **Large Standard Library**: Python comes with a large standard library that includes modules and packages for various tasks, such as file handling, networking, regular expressions, and more. This allows developers to build applications more efficiently.
6. **Object-Oriented Programming (OOP)**: Python supports OOP principles, allowing developers to write code in a structured and organized manner using classes and objects.
7. **High-Level Language**: Python is a high-level language, which means it provides abstractions that allow you to focus on the problem you're trying to solve rather than the low-level details of the computer.
8. **Extensible and Embeddable**: Python can be extended by writing modules in C or C++ and can be embedded in other applications to provide scripting capabilities.
9. **Community Support**: Python has a large and active community, which means you can find a wealth of resources, libraries, and documentation to help you solve problems and learn.
10. **Web Development**: Python has several frameworks (Django, Flask, etc.) for web development, making it a popular choice for building web applications and APIs.
11. **Data Science and Machine Learning**: Python has a rich ecosystem for data analysis and machine learning, with libraries like NumPy, pandas, scikit-learn, TensorFlow, and more.
12. **Automation and Scripting**: Python is widely used for automating tasks and scripting. You can write scripts to perform repetitive tasks, manipulate files, or interact with other software.
13. **Scientific Computing**: Python is used in various scientific fields for tasks such as simulations, data analysis, and visualization.
14. **Desktop GUI Applications**: You can create desktop applications with graphical user interfaces (GUI) using libraries like Tkinter, PyQt, or wxPython.
15. **Network Programming**: Python is suitable for network programming, including socket programming for creating networked applications.

**Day 3**

Welcome to Day 3! Today we will be looking at Modules.

* Understand the different Modules that Python uses.
* Know the differences between the types of Modules.
* Be able to write programs that require input and conversions**.**

**Introduction to Modules**

Modules are the pre-defined files that contain the python codes which depict the basic functionalities of class, methods, variables, etc

1. **OS**: In Python, the **os** module provides a way of using operating system-dependent functionality. You can use it to interact with the file system, manipulate file paths, manage directories, and execute system commands.
2. **Time**: Python's **time** module is used for time-related operations. It allows you to work with time and date values, measure the execution time of code, create delays, and handle timestamps.
3. **Math**: The **math** module in Python provides mathematical functions and constants. It includes functions for basic arithmetic operations, trigonometry, logarithms, and more. It's a handy library for mathematical calculations.
4. **Matplotlib**: Matplotlib is a popular Python library for creating static, animated, and interactive visualizations in Python. It is commonly used for data visualization, allowing you to create a wide range of charts, plots, and graphs.

We have types of Modules in Python

* **OS** - provides a way of using operating system-dependent functionality.
* **Time** – It is used for time-related operations.
* **Math** – It provides mathematimatical functions and constants.
* **Metplotlib** – It is creating animated and Interactive Visualizations.

**Mechanism of Python Modules**

Listing of Modules: Modules are separate Python files containing code for reuse. You can list available modules using the help() function or by exploring the Python standard library modules.

Importing Modules from Python Standard Path: Python searches for modules in directories specified in the sys.path list, which includes the current directory and Python's standard library locations.

Importing Modules from other Sources: You can import modules from other directories by appending their paths to sys.path or using relative or absolute import statements.

Variable in a Module: Modules can contain variables, functions, and classes, which can be accessed using the module's name. For example, my\_module.my\_variable.

Difference between a module and a package in Python: A module is a single Python file, while a package is a directory containing multiple modules and an \_\_init\_\_.py file to indicate it as a package. Packages help organize related modules and sub-packages.

**Day 4**

Regular expressions (often referred to as "regex" or "regexp") in Python are used for pattern matching in strings. Python's re module provides support for working with regular expressions. Here's a brief overview of how to use regular expressions in Python:

**Import the re module:**

First, you need to import the re module:

import re

**Creating a Pattern**:

A regular expression pattern is a sequence of characters that defines a search pattern. For example, to match a simple word like "apple," you can create a pattern like this:

pattern = r"apple"

The 'r' before the string is used to indicate that it's a raw string and helps escape backslashes.

**Matching Patterns**:

You can use the re.search() function to find a match for the pattern in a string. For example:

text = "I have an apple and a banana."

match = re.search(pattern, text)

if match:

print("Match found:", match.group())

else:

print("No match found.")

This code searches for the pattern "apple" in the text string and prints the matched text.

**Metacharacters:**

Regular expressions use metacharacters that have special meanings. Some common metacharacters include:

.: Matches any character except a newline.

\*: Matches zero or more occurrences of the preceding character.

+: Matches one or more occurrences of the preceding character.

?: Matches zero or one occurrence of the preceding character.

[]: Specifies a character class.

|: Acts like a logical OR for matching different patterns.

^: Matches the start of a string.

$: Matches the end of a string.

**Groups and Capturing:**

You can use parentheses () to create groups within a pattern. These groups can be used to capture and extract matched portions of a string. For example:

pattern = r"(apple|banana)"

text = "I have an apple and a banana."

matches = re.findall(pattern, text)

for match in matches:

print("Match found:", match)

This code matches either "apple" or "banana" in the text and prints the matches.

**Substitution:**

You can use the re.sub() function to replace matched patterns with a specified string:

text = "I have an apple and a banana."

new\_text = re.sub(pattern, "fruit", text)

print(new\_text)

This code replaces occurrences of "apple" or "banana" with "fruit" in the text.