Day -1

Introduction :

Goals and Expectations:

- [ ] Understand **basic syntax/principles of Python programming.**
- [ ] Write **Python code to solve moderate complex problems**.
- [ ] Exposure to **real-world applications of Python.**

What is Programming ?

Just like we use Nepali or English to communicate with each other, we use a programming language like python to communicate with the computer.
https://hill-leopon-d16.notion.site/Python-Workshop-4-Days-Workshop-148d9122f9974fab8997a520063cc4b9

What is Python ?

Python is a simple and easy to understand language which feels like reading simple english. This pseudocode nature of python makes it easy to learn and understandable by beginners.

- Python is a high-level, interpreted programming language known for its simplicity and readability.
- Guido van Rossum created Python in the late 1980s, and it has since become one of the most popular languages among developers.
- Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Features:

- Easy to understand
- Free and open source
- High level language
- Portable - works on mac/linux/windows
- Fun to work with

Applications:

- Web development (Django, Flask)
- Data science and machine learning (NumPy, Pandas, TensorFlow)
- Scientific computing (SciPy)
- Artificial intelligence and natural language processing (NLTK, SpaCy)
- Automation and scripting
- Game development (Py-game)
- Finance and trading
- Education and research
- Blockchain development

Installation

Python can be easily installed from python.org

- Check using **python –version** in CMD
- Any to every version works
- Use **python** in CMD to run python one line codes
- Use **exit()** to exit the python interpreter.

IDE: Integrated Development Environment

- Vscode, Pycharm, Vim (linux), Notepad++

- Vscode and Jupyter Notebook Extension

Virtual Environment in Python

- self-contained directory tree that contains its own Python installation
- can have its own set of Python packages/modules installed.
- create isolated environments for different projects, each with its own dependencies, without affecting the system-wide Python installation or other projects.
- Create using **python3 -m venv env_name** in CMD/terminal
- Activate the file: **myenv\Scripts\activate.ps1**
- Deactivate using: **deactivate**

Benefits:

- Isolated from the system-wide Python installation and other virtual environments. (prevents conflicts between different projects)
- Allow you to manage dependencies for each project separately. (You can install specific versions of packages without affecting other projects.)
- Easy to reproduce the development environment on different systems. (can share the project with others, and they can create the same environment using the requirements.txt file)

PiP (Python Package Index):

- Package manager for Python, used for installing and managing Python packages
- Simplifies the process of installing third-party libraries and packages that are not included in the standard Python distribution.
- Check using **pip –version** in CMD
- Upgrade using **pip install --upgrade pip** in CMD

Modules:

- A file containing Python code, which can define functions, classes, and variables.
- Need to import it into your Python script using the **import** statement. Eg
  import math
  print(math.sqrt(16))

Package: A collection of modules organized in a directory structure.
Libraries:

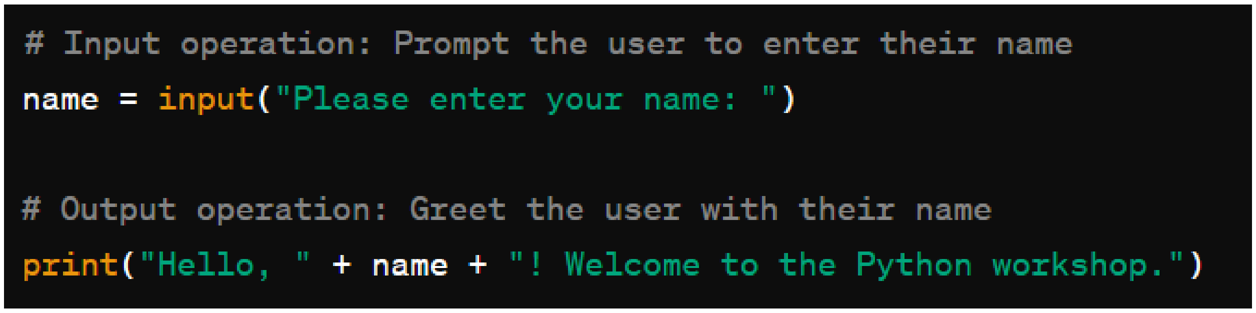
- Collections of modules or packages that provide specific functionality for tasks such as data manipulation, web development, scientific computing, etc.
- Third-party libraries are developed by the Python community and provide additional functionality beyond what is available in the standard library.
  Commonly Used Libraries:

- **NumPy**: For numerical computing and arrays.
- **Pandas**: For data manipulation and analysis.
- **Matplotlib**: For creating static, interactive, and animated visualizations.
- **Scikit-learn**: For machine learning algorithms and tools.
- **TensorFlow, PyTorch**: For deep learning and neural networks.
- **Flask, Django**: For web development.

Start from Basics

print("Hello World")

Execute this file(day1.py) by typing **python day1.py** and you will see "Hello World" printed on the screen.
Syntax for input and print function with example:



```python
# Input operation: Prompt the user to enter their name
name = input("Please enter your name: ")

# Output operation: Greet the user with their name
print("Hello, " + name + "! Welcome to the Python workshop.")
```

Comments :

Comments are used to write something which the programmer does not want to execute.

Types:

Single line comments -> Written using #comment

Multi line comments -> `Written using '''Comment'''`

Use **‘ctrl+?’** for commenting in VS Code

Variable, Data Types and Operators:

A variable is the name given to a memory location in a program. For example:

a = 30

b = “Harry”

c = 5.5

Bool = True

Variable : Container to store a value.

Keyword: Reserved words in python.

Data Types :

Variables can store different types of data. Some common data types -

1. Integer (**int**): Represents whole numbers, e.g., 10, -5, 1000.
2. Float (**float**): Represents floating-point numbers, i.e., numbers with a decimal point, e.g., 3.14, -0.5, 2.0.

3. String (**str**): Represents text, enclosed in single quotes (' ') or double quotes (" "), e.g., 'Hello', "Python", '123'.
4. Boolean (**bool**): Represents True or False values, used for logical operations and comparisons, e.g., True, False.

Operators:
Operators are symbols or keywords that perform operations on items.

1. **Arithmetic** Operators:
    - Addition (+),
    - Subtraction (-),
    - Multiplication (*),
    - Division (/),
    - Modulus (%),
    - Exponentiation (**)
2. **Comparison** Operators:
    - Equal to (==),
    - Not equal to (!=),
    - Greater than (>),
    - Less than (<),
    - Greater than or equal to (>=),
    - Less than or equal to (<=)
3. **Logical** Operators:
    - AND (and),
    - OR (or),
    - NOT (not).
4. **Assignment** Operators:
    - Assignment (=),
    - Addition assignment (+=),
    - Subtraction assignment (-=),
    - Similar arithmetic assignments

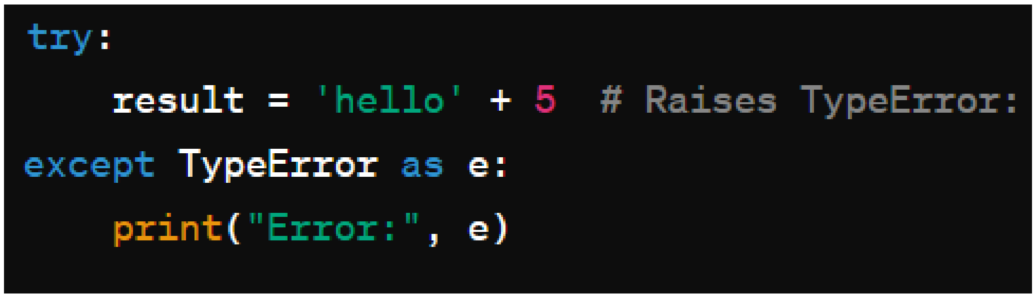
5. **Bitwise** Operators: (to work with binary bits)
    - AND (&),
    - OR (|),
    - XOR (^),
    - NOT (~),
    - Left shift (<<),
    - Right shift (>>).

Exception handling:

- allows you to handle errors or exceptional situations that may occur during the execution of a program
- prevents the program from crashing
- provides a mechanism to catch and handle exceptions using **try, except, else,** and **finally** blocks.

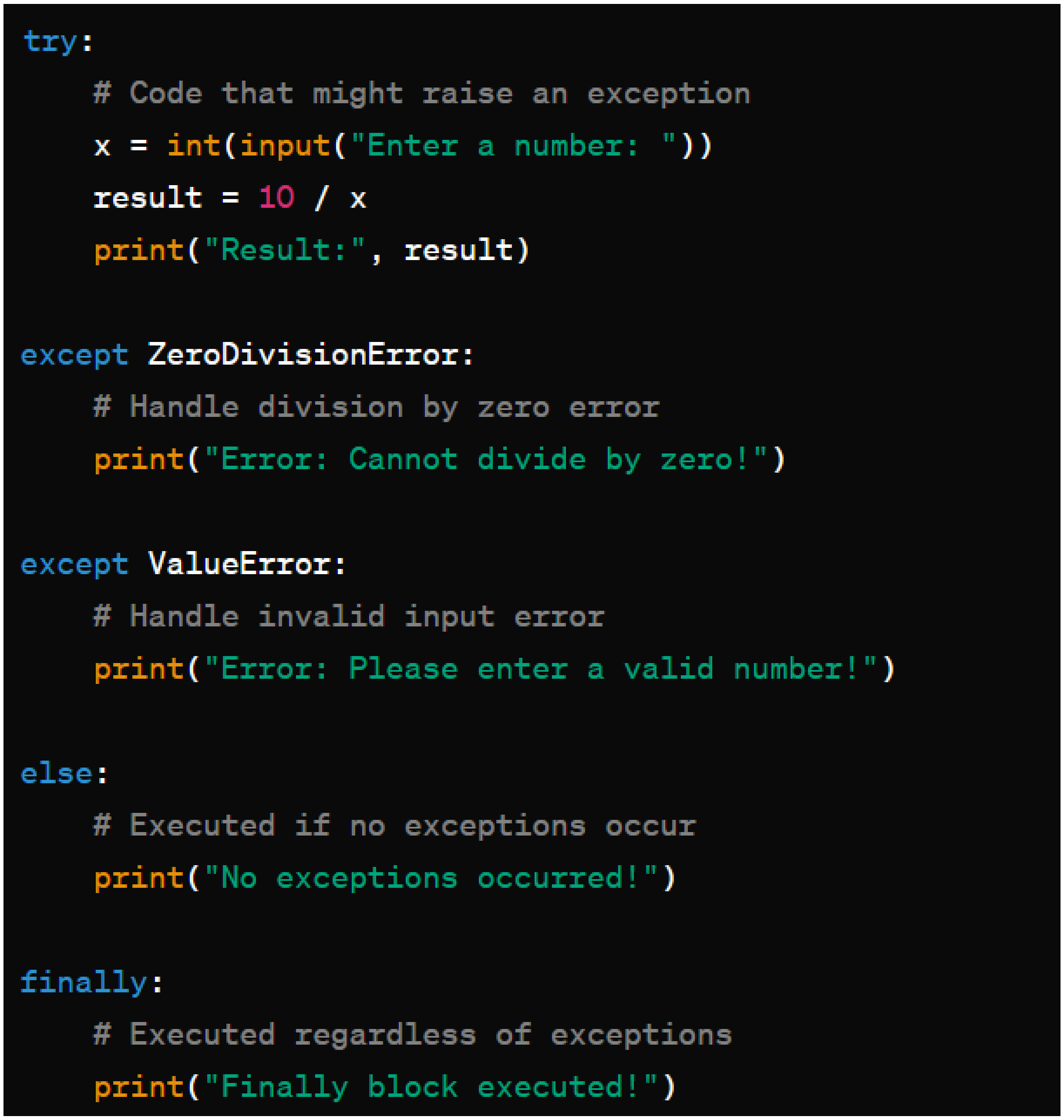
try block: The code that might raise an exception is placed within the try block.
except block: If an exception occurs within the try block, the control is transferred to the corresponding except block.
else block (optional): The else block is executed if no exceptions occur in the try block. It is typically used to perform actions that should only occur if no exceptions were raised.
finally block (optional): The finally block is executed regardless of whether an exception occurs or not. It is often used to release resources or clean up operations.
Let's take a common example:

- Type conversion error



```
try:
    result = 'hello' + 5  # Raises TypeError:
except TypeError as e:
    print("Error:", e)
```

- Divide by zero error



```python
try:
    # Code that might raise an exception
    x = int(input("Enter a number: "))
    result = 10 / x
    print("Result:", result)

except ZeroDivisionError:
    # Handle division by zero error
    print("Error: Cannot divide by zero!")

except ValueError:
    # Handle invalid input error
    print("Error: Please enter a valid number!")

else:
    # Executed if no exceptions occur
    print("No exceptions occurred!")

finally:
    # Executed regardless of exceptions
    print("Finally block executed!")
```

Simple Python Project (Homework):

- Create a program that takes inputs from a student (name, roll number, marks in 5 subjects) and then give result (Name, Total marks, average marks, percentage) as output
- Format the output **(optional)**

Title of the Session
Session 1: Welcome and overview ● Introduction to the workshop, goals, and expectations ● Overview of Python and its applications
Session 2: Setting UP Python Environment ● Installation of Python and necessary tools (e.g. Jupyter Notebook) ● Introduction to an integrated development environment (IDE) ● Concept of Virtual Environment
Session 3: Python Basics ● Variables, data types, and basic operations ● I/O statements and comments
Session 4: Control Flow ● Introduction to conditional statements (if, else if, else) ● Looping (for and while loops)
Session 5: Functions, Parameters and arguments ● Defining functions , parameters, arguments ● Return statements and function calls
Session 6 : Exception handling ● Introduction to handling errors and exceptions
Session 7 : List and Tuples ● Introduction to basic data structures ● List manipulation and tuple usage
Session 8 : Dictionaries and Sets ● Understanding dictionaries and sets ● Operations on dictionaries and sets
Session 9: File handling ● Reading and writing to files

● Working with different file formats (e.g., text files, CSV, json)
Session 10: Hands on Project ● Small project to apply the concepts learned
Session 11: Introduction to modules ● Understanding Python modules and libraries ● Exploring commonly used modules (e.g., math, random)
Session 12: Project work ● Hands-on project incorporating file handling and modules
Advanced Topics and Real-world applications **Session 13:Introduction to NumPy and Pandas** ● Basics of data manipulation using NumPy and Pandas
Session 14: Web Scraping ● Introduction to web scraping for data collection
Session 15: Introduction to Data Visualization ● Basic plotting using libraries like Matplotlib
Session 16: Final Project Q & A ● Participants work on a comprehensive final project, incorporating various concepts covered ● Q&A session and review of key takeaways
Session 17: Closing Session