**Python Basics:**

- **What is Python?**

Python is a high-level, general-purpose programming language known for its readability and ease of use. It emphasizes clear syntax and utilizes indentation to define code blocks, making it beginner-friendly.

- **Key Features:**
    - **Readability:** Python code resembles plain English, making it easier to learn and maintain.
    - **Versatility:** Python can be used for web development, data analysis, machine learning, scripting, and more.
    - **Large Standard Library:** Python comes with a rich library of pre-written modules and functions, reducing development time.
    - **Cross-Platform Compatibility:** Python code can run on various operating systems like Windows, macOS, and Linux with minimal changes.
- **Use Cases:**
    - **Web Development:** Frameworks like Django and Flask use Python for building robust web applications.
    - **Data Science and Machine Learning:** Libraries like NumPy, Pandas, and Scikit-learn make Python powerful for data analysis and creating machine learning models.
    - **Automation and Scripting:** Python is excellent for automating tasks and creating scripts to interact with other programs.
    - **Scientific Computing:** Tools like SciPy and Matplotlib allow for complex scientific computations and visualizations.

**Installing Python:**

- **Windows:** Download the latest Python installer from https://www.python.org/downloads/ and run the executable. Ensure "Add Python to PATH" is checked during installation.
- **macOS:** Open Terminal and run `brew install python`.
- **Linux:** The installation method varies depending on your distribution. Use your package manager (e.g., `apt-get install python3` on Ubuntu/Debian).

**Verification:**

Open a terminal/command prompt and type `python --version`. If installed correctly, it should display the Python version.

**Virtual Environments:**

Use tools like `venv` or `virtualenv` to create isolated environments for different projects, preventing conflicts between project dependencies.

Python Syntax and Semantics:

print("Hello, World!")

This program:

- Uses the `print` function to display a message on the console.
- "Hello, World!" is a string enclosed in double quotes.
- Indentation defines the code block executed by the `print` function.

**Data Types and Variables:**

- **Basic Data Types:**
  - **int:** Integers (whole numbers) - e.g., `10, -5`
  - **float:** Floating-point numbers (decimals) - e.g., `3.14, -2.5e2` (scientific notation)
  - **str:** Strings (text) - e.g., `"Hello"`, 'This is a string' (both single and double quotes allowed)
  - **bool:** Boolean values (True or False)

Python

age = 25  # int

pi = 3.14159  # float

name = "Alice"  # str

is_student = True  # bool

**Control Structures:**

- **Conditional Statements (if-else):**

Python

age = 18

if age >= 18:

    print("You are eligible to vote.")

```
else:
    print("You are not eligible to vote.")
```

- **Loops (for):**

```
Python
for i in range(5):  # iterates 5 times
    print(i)  # prints 0, 1, 2, 3, 4
```

**Functions in Python:**

Functions are reusable blocks of code that perform specific tasks.

```
Python
def add(x, y):
  """This function adds two numbers and returns the sum."""
  return x + y

result = add(5, 3)  # Call the function with arguments
print(result)  # Output: 8
```

**Lists and Dictionaries:**

- **Lists:** Ordered collections of items, enclosed in square brackets `[ ]`.

```
Python
numbers = [1, 2, 3, 4, 5]
print(numbers[2])  # Access element at index 2 (output: 3)
```

- **Dictionaries:** Unordered collections of key-value pairs, enclosed in curly braces `{}`.

```
Python
person = {"name": "Bob", "age": 30, "city": "New York"}
```

print(person["name"])  # Access value by key (output: Bob)

**What is exception handling?**

Exception handling is a mechanism in Python to manage errors that may occur during program execution. It allows you to write robust code that can anticipate and gracefully handle potential issues, preventing the program from crashing unexpectedly.

**Using try-except-finally blocks:**

The `try-except-finally` block structure is the primary tool for exception handling in Python. Here's how it works:

Python

```
try:
  # Code that might raise an exception
except ExceptionType:
  # Code to handle the exception
finally:
  # Code that always executes, regardless of exceptions
```

**try block:** This block contains the code that might potentially raise an exception.

**except block:** This block handles the exception if one occurs within the `try` block. You can specify the type of exception to handle (e.g., `ZeroDivisionError`) or use a general `ExceptionType` to catch any exception.

**finally block:** This block executes **always**, regardless of whether an exception is raised or not. It's commonly used for cleanup tasks like closing files or database connections.

Python

```
def divide(x, y):
  try:
    return x / y
  except ZeroDivisionError:
    print("Error: Cannot divide by zero")
```

```
    return None  # Or handle the error differently


result = divide(10, 2)

print(result)  # Output: 5.0


result = divide(10, 0)

print(result)  # Output: Error: Cannot divide by zero

            #       None
```

# Modules and Packages

**Modules and Packages:**

- **Modules:** Reusable blocks of Python code containing functions, variables, and classes. You can import modules into your script to access their functionalities.
- **Packages:** Hierarchical collections of modules that organize code into a logical structure. Packages can contain sub-packages for further organization.

**Importing and Using Modules:**

Python

```
import module_name  # Import the entire module


# Access functions, variables, or classes from the module

result = math.sqrt(16)  # Using the math.sqrt function from the math module
```

Python

```
import math


# Use functions from the math module

area_of_circle = math.pi * (radius**2)
```

```
print(area_of_circle)
```

# File I/O

**Reading from Files:**

1. Open the file in read mode (`"r"`) using the `open()` function.
2. Read the contents using methods like `read()` (reads entire file) or `readline()` (reads line by line).
3. Close the file using the `close()` method.

Python

```
with open("myfile.txt", "r") as file:
  contents = file.read()  # Read the entire file
  print(contents)
```

**Writing to Files:**

1. Open the file in write mode (`"w"`) or append mode (`"a"`) using `open()`.
2. Write content to the file using the `write()` method.
3. Close the file using the `close()` method.

Python

```
with open("data.csv", "w") as file:
  data_list = ["apple", "banana", "cherry"]
  file.writelines(data_list)  # Write each item in the list to a new line
```

The `with` statement ensures proper file handling and automatically closes the file even if an exception occurs.