CORE PYTHON SYLLABUS

Module 1: Introduction and Environment Setup

- **1.1 Python Fundamentals**
  - What is Python? (Features, History, Applications)
  - Interpreted, Interactive, Object-Oriented, High-Level Language
  - The Python Interpreter (Interactive Mode/REPL)
  - Differences between Python 2 and Python 3 (Focus on Python 3)
- **1.2 Setup**
  - Installing Python (Official installer)
  - Setting up the environment (IDE/Code Editor like VS Code, PyCharm)
  - Writing and Executing the First Python Program ("Hello, World!")
  - Understanding the role of the Python Virtual Machine (PVM) and Bytecode (.pyc files)
- **1.3 Basics of Syntax**
  - Python Identifiers and Keywords
  - **Indentation** (Significance in Python)
  - Comments (Single-line # and multi-line """…""")
  - Statements and Expressions

Module 2: Variables, Data Types, and Operators

- **2.1 Variables and Assignment**
  - Defining variables (No explicit declaration needed)
  - Variable Naming Rules and Conventions (e.g., PEP 8)
  - Dynamic Typing (Type is checked at runtime)
  - The id() and type() functions
- **2.2 Built-in Data Types (Primitives)**
  - **Numeric Types:** int, float, complex
  - **Boolean Type:** bool (True, False)
  - **Type Conversion (Type Casting):** int(), float(), str(), etc.
- **2.3 Operators**
  - **Arithmetic Operators:** +, -, *, /, // (floor division), % (modulus), ** (exponent)
  - **Comparison Operators:** ==, !=, >, <, >=, <=
  - **Assignment Operators:** =, +=, -=, *= etc.
  - **Logical Operators:** and, or, not
  - **Identity Operators:** is, is not (Comparing memory location/object identity)
  - **Membership Operators:** in, not in (Checking presence in a sequence)
  - Operator Precedence and Associativity
- **2.4 Input and Output**
  - Using the print() function (Formatting output, sep, end)
  - Using the input() function to get user input
  - String Formatting: % operator, .format() method, **f-strings**

Module 3: Control Flow Statements

- **3.1** Conditional Statements (Decision Making)
  - The `if` statement
  - The `if-else` statement
  - The `if-elif-else` chain
  - Nested `if` statements
- **3.2 Looping Statements (Iteration)**
  - The `while` loop
  - The `for` loop (Iterating over sequences/iterables)
  - The `range()` function
  - Nested loops
- **3.3 Loop Control Statements**
  - `break` (Exiting the loop)
  - `continue` (Skipping the current iteration)
  - `pass` (Null operation statement)
  - `else` clause with loops (Executes if the loop finishes without a `break`)

Module 4: Functions

- **4.1 Defining and Calling Functions**
  - Syntax: `def` keyword
  - Function Call and Execution Flow
  - Docstrings (`"""…"""`)
  - The `return` statement
- **4.2 Arguments and Parameters**
  - Positional Arguments
  - Keyword Arguments
  - Default Arguments
  - Variable-Length Arguments: `*args` (Non-Keyword), `**kwargs` (Keyword)
- **4.3 Scope and Lifetime**
  - Local vs. Global variables
  - LEGB Rule (Local, Enclosing, Global, Built-in)
  - The `global` and `nonlocal` keywords
- **4.4 Functional Programming Concepts (Core)**
  - **Lambda Functions** (Anonymous functions)
  - Built-in higher-order functions: `map()`, `filter()`
  - Introduction to `reduce()` (from `functools` module)
  - Recursion (Basic concept and examples)

Module 5: Core Data Structures (Collections)

- **5.1 Strings (Immutable Sequence)**
  - Creating strings (Single, double, triple quotes)
  - Indexing (Positive and Negative)
  - **Slicing** [start:stop:step]
  - String Methods (e.g., len(), lower(), upper(), strip(), split(), join(), find(), replace())
  - String Formatting: % operator, .format() method, **f-strings (Formatted String Literals)**
  - String Immutability
- **5.2 Lists (Mutable Sequence)**
  - Creating and accessing lists
  - List Indexing and Slicing
  - List Methods (e.g., append(), insert(), extend(), remove(), pop(), sort(), reverse())
  - List Mutability and Copying (Shallow vs. Deep Copy)
  - **List Comprehensions** (Basic syntax)
  - Using Lists as Stacks and Queues (Brief)
- **5.3 Tuples (Immutable Sequence)**
  - Creating and accessing tuples
  - Packing and Unpacking tuples
  - Tuple Immutability
  - Single-item tuple
- **5.4 Dictionaries (Mutable Mapping)**
  - Creating and accessing dictionaries (Key-Value pairs)
  - Keys and Values (Properties of keys)
  - Dictionary Methods (e.g., keys(), values(), items(), get(), pop(), update())
  - **Dictionary Comprehensions** (Basic syntax)
- **5.5 Sets (Mutable, Unordered Collection of Unique Elements)**
  - Creating sets (Differences from dictionary creation)
  - Set Operations: Union (|), Intersection (&), Difference (-), Symmetric Difference (^)
  - Set Methods (e.g., add(), remove(), discard())
  - frozenset (Immutable version of set)

Module 6: Modules and Packages

- **6.1 Modules**
  - What is a Module? (A Python file with definitions and statements)
  - Creating a Module
  - Importing Modules: import, from…import
  - Module Search Path (sys.path)
  - The dir() function
  - Using __name__ == '__main__'
- **6.2 Packages**
  - What is a Package? (A collection of modules in directories)
  - Creating a Package (Role of __init__.py)
  - Importing from Packages

Module 7: Object-Oriented Programming (OOP) - The Core Pillars

- **7.1 Classes and Objects**
  - Defining a **Class** (The blueprint)
  - Creating an **Object** (The instance)
  - Instance Variables vs. Class Variables
  - The self variable
  - **Constructors (__init__)** and Destructors (__del__)
- **7.2 Encapsulation**
  - Bundling data and methods together
  - Access Specifiers (Public, Protected _, Private __ conventions)
  - Using **Getters and Setters** (Property decorators for advanced control)
- **7.3 Inheritance**
  - Single, Multi-Level, and **Multiple Inheritance** (MRO)
  - Method Overriding
  - Using the **super()** function
- **7.4 Polymorphism**
  - Concept of taking "many forms"
  - **Method Overloading** (Simulated using default arguments)
  - **Method Overriding** (Run-time Polymorphism)
  - **Operator Overloading** (e.g., using __add__ to redefine +)
- **7.5 Data Abstraction**
  - Hiding implementation details
  - Using the **abc module** (Abstract Base Classes) for creating Abstract Methods and Classes

Module 8: Errors, Exceptions, and File Handling

- **8.1 Errors and Exceptions**
  - Syntax Errors (Compile-time)
  - Logical Errors and Runtime Errors (Exceptions)
  - Common built-in exceptions (e.g., TypeError, ValueError, ZeroDivisionError, IndexError)
- **8.2 Exception Handling**
  - The try, except, else, and finally blocks
  - Handling specific exceptions
  - Raising Exceptions (raise keyword)
  - User-Defined Exceptions (Basic)
- **8.3 File Handling**
  - File Operations: Open, Read, Write, Close
  - File Modes ('r', 'w', 'a', 'r+', 'w+', 'b', 't')
  - Reading Methods: read(), readline(), readlines()
  - Writing Methods: write(), writelines()
  - The **with statement** and Context Managers (Recommended practice for file handling)
- **8.4 Working with Standard Library Modules (Core)**
  - **os:** Basic functions for interacting with the operating system (e.g., path manipulation, file/directory checks)
  - **sys:** System-specific parameters and functions (e.g., sys.argv)
  - **math:** Mathematical functions
  - **datetime:** Working with dates and times