**Programming Fundamentals - Week 2 Lectures**

**Lecture 1: Introduction to Python & Sequential Programming**

**Lecture Goals:**

* Familiarize students with the Python programming environment.
* Understand the basic structure and syntax of a Python program.
* Learn how to perform sequential operations using variables, input/output, and conditionals.
* Develop foundational skills for writing basic Python programs involving decisions.

**Topic 1: Installing Python and VS Code**

* **What is Python?** Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used in web development, data analysis, artificial intelligence, automation, and education. Python emphasizes code readability and has a large, supportive community.
* **Install Python from python.org**: Guide students to [python.org](https://www.python.org/) to download and install the latest version of Python. Explain the importance of adding Python to the system PATH during installation.
* **Install and configure VS Code with Python extension**: Introduce Visual Studio Code (VS Code) as a lightweight and powerful code editor. Demonstrate how to install the Python extension and configure it to detect the Python interpreter. Highlight features such as syntax highlighting, IntelliSense, and debugging tools.
* **Run a sample file**: Create and run a simple Python file in VS Code:

print("Hello, world!")

Explain how the interpreter processes the code line by line and produces output.

**Topic 2: Run a Simple Program + Syntax/Runtime Errors**

* **Discuss the structure of a simple program**: Describe how Python executes code sequentially, and explain the importance of indentation and syntax.
* **Syntax errors:** Errors that occur when the code violates the rules of the language. Examples:
* print("Hello" # missing closing parenthesis

x == 5 # using comparison without context

Use these to explain error messages and how to interpret them.

* **Runtime errors:** Errors that occur while the program is running. Examples:
* x = int("abc") # ValueError because 'abc' cannot be converted to an integer

y = 1 / 0 # ZeroDivisionError

Encourage students to read error messages carefully and use them for debugging. For instance, the ValueError in x = int("abc") occurs because the program attempts to cast the string 'abc' to an integer, which is not permitted—Python cannot interpret non-numeric characters as a valid integer. Understanding the cause of such errors helps students correct their code more efficiently.

**Topic 3: Sequential Programming**

* **Variables and data types:** Introduce variable declaration and the three basic data types: int, float, and str. Emphasize Python's dynamic typing.
* **Assign values, take input, print output:** Show how to assign values to variables, read from the user using input(), and display output with print():
* name = input("Enter your name: ")

print("Hello", name)

* **Basic calculations:** Demonstrate arithmetic operations like addition, subtraction, multiplication, and division:
* a = float(input("Enter a: "))
* b = float(input("Enter b: "))

print("Sum:", a + b)

* **Exercises:**
  + Read two numbers and calculate their sum and average.
  + Convert a temperature from Celsius to Fahrenheit using the formula: F = C \* 9/5 + 32.
  + Calculate and print the perimeter of a rectangle from user input (2 × (length + width)).

**Topic 4: If Statements**

* **Conditional statements overview:** Explain decision-making in programs using if, if-else, and if-elif-else. Clarify the role of conditions in altering program flow.
* **Types of if statements:**
  + if: Executes a block of code if a condition is true.
  + if-else: Provides an alternative block if the condition is false.
  + if-elif-else: Allows checking multiple conditions in sequence.
* **Conditions and operators:** Introduce comparison operators: ==, !=, <, >, <=, >= Explain logical operators: and, or, not Mention identity (is) and membership (in) operators briefly for awareness.
* **Example:**
* if score >= 90:
* print("A")
* elif score >= 80:
* print("B")
* else:

print("C or below")

* **Exercises:**
  + Compare two numbers and print the greater.
  + Check if a character is inside a string or not.
  + Create a condition where a user is granted access only if both the username and password match predefined values. Use logical operators to combine conditions:
  + username = input("Enter username: ")
  + password = input("Enter password: ")
  + if username == "admin" and password == "1234":
  + print("Login successful")
  + else:

print("Invalid credentials")

Explain that the and operator ensures both conditions must be true. Alternatively, discuss how or could be used to allow access if either a primary or backup username is accepted.

**Lecture 2: Arrays and Loops**

**Lecture Goals:**

* Learn to define and manipulate collections using arrays (lists).
* Understand and apply iteration using for and while loops.
* Gain the ability to combine lists and loops to solve common programming problems.

**Topic 1: Arrays**

* **Introduction to lists:** Lists in Python are ordered, mutable collections that can hold elements of any data type. They are one of the most versatile and commonly used data structures in Python. Lists can hold integers, floats, strings, or even a mixture of types.
* nums = [1, 2, 3] # list of integers
* names = ["Ali", "Sara"] # list of strings

mixed = [42, "Hello", 3.14] # mixed-type list

* **Accessing and modifying elements:** You can access list elements by their index (starting from 0) and modify them directly:
* print(nums[0]) # prints first element: 1
* nums[1] = 20 # changes second element to 20

print(nums) # [1, 20, 3]

* **Iterating through a list:** Lists are commonly used with loops to process data:
* for name in names:

print("Hello", name)

* **Input from user to fill a list:** We can dynamically construct a list by appending user input. This is useful when the size or contents of the list are not known in advance:
* arr = []
* for i in range(3):

arr.append(int(input("Enter number: ")))

* **Common operations on lists:** Introduce a few useful methods and functions:
* print(len(arr)) # returns number of elements
* arr.append(99) # adds 99 to the end
* arr.remove(99) # removes first occurrence of 99
* arr.sort() # sorts the list in ascending order

print("Sara" in names) # checks if "Sara" is in the list

These foundational concepts with lists will be used and expanded upon in subsequent lessons, especially when introducing functions, conditionals within loops, and more advanced data structures.

**Topic 2: Loops**

* **For loop basics:** A for loop is typically used when the number of iterations is known. It can iterate over a range of numbers or a collection like a list.
* for i in range(5):

print(i) # prints 0 to 4

Using range(start, stop, step) allows greater control:

for i in range(1, 10, 2):

print(i) # prints 1, 3, 5, 7, 9

* **While loop basics:** A while loop continues to execute as long as the condition remains true. It is ideal for cases where the number of iterations is not predetermined.
* x = 0
* while x < 5:
* print(x)

x += 1

* **Combining loops with arrays:** You can iterate over a list using a loop to process each element:
* arr = [5, 3, 8]
* for value in arr:

print(value)

* **Input validation example:** This example demonstrates using a loop to repeatedly request valid input:
* password = ""
* while password != "1234":
* password = input("Enter password: ")

print("Access granted")

* **Exercises:**
  + Print even numbers up to a user-defined number N.
  + Generate and print the multiplication table for a given number.
  + Search for a specific value in a list.
  + Find and print the smallest number in a list.
  + Find the maximum value in a list.
  + Compute the sum of all list elements.
  + Count how many numbers are positive or negative.
  + Print the list in reverse order.