**Introduction to Computation and Programming**

**1. Introduction**

Computation is a fundamental concept that drives modern technology. This course aims to introduce students to computational thinking, programming, and problem-solving. The main goal is to help students think like computer scientists and understand how computation can be used to tackle complex technical problems.

This course is designed for students with little to no prior programming experience and will primarily use Python as the programming language.

**2. Course Objectives**

**Strategic Goals:**

1. Prepare students for computational problem-solving.
2. Equip students with programming skills for technical applications.
3. Instill confidence in students to write and read small pieces of code.
4. Teach students the capabilities and limitations of computation.
5. Provide skills that help students secure research opportunities and internships.

**Tactical Goals:**

By the end of the course, students should be able to:

1. Apply computational thinking to solve small-scale problems.
2. Read, write, and debug simple programs in Python.
3. Understand the fundamental principles of computation.
4. Translate real-world problems into computational models.
5. Use programming as a tool for scientific and engineering applications.

**3. What is Computation?**

At its core, computation involves following a well-defined sequence of steps (an algorithm) to transform input data into useful output. The study of computation focuses on developing techniques to express these steps effectively in a language that a machine can understand.

**Declarative vs. Imperative Knowledge**

* **Declarative Knowledge:** Statements of fact, such as mathematical formulas.
  + Example: "The square root of x is a number y such that y² = x."
* **Imperative Knowledge:** Step-by-step instructions to perform a task.
  + Example: Algorithm to compute square roots using Heron's method:
    1. Start with an initial guess (g).
    2. If g² is close to x, return g.
    3. Otherwise, update g using (g + x/g)/2.
    4. Repeat until the desired accuracy is achieved.

Computational thinking is largely based on **imperative knowledge**, as it provides a structured way to solve problems programmatically.

**4. Evolution of Computation**

Early computers were designed to perform specific tasks, such as:

* **Fixed Program Computers:** Could execute only predefined computations (e.g., early calculators, Enigma code-breakers).
* **Stored Program Computers:** Could execute different computations by reading sequences of instructions (modern computers follow this model).

**Basic Architecture of a Stored Program Computer**

A modern computer consists of:

1. **Memory**: Stores data and instructions.
2. **Control Unit**: Directs the execution of instructions.
3. **Arithmetic Logic Unit (ALU)**: Performs calculations and logical operations.
4. **Program Counter**: Keeps track of which instruction is being executed.

A program is a **sequence of instructions** stored in memory that guides the computer’s operations.

**5. Programming Languages**

**Characteristics of Programming Languages:**

1. **High-Level vs. Low-Level:**
   * High-level languages (e.g., Python) are easier to write and understand.
   * Low-level languages (e.g., Assembly) provide more control over hardware.
2. **General-Purpose vs. Targeted:**
   * General-purpose (e.g., Python) is suitable for various applications.
   * Targeted languages (e.g., MATLAB) are specialized for certain domains.
3. **Interpreted vs. Compiled:**
   * Interpreted languages execute instructions directly (e.g., Python).
   * Compiled languages require pre-processing before execution (e.g., C, Java).

Python is an **interpreted, high-level, general-purpose** language, making it an excellent choice for beginners.

**6. Python Basics**

**Basic Data Types**

* **Numbers**: Integers (int) and floating-point numbers (float)
* x = 5 # Integer

y = 3.14 # Float

* **Strings**: A sequence of characters enclosed in quotes

text = "Hello, World!"

**Operators**

Python supports arithmetic and string operations:

* Arithmetic:
* sum = 3 + 5 # Addition
* diff = 7 - 2 # Subtraction
* product = 4 \* 6 # Multiplication

quotient = 8 / 2 # Division

* String Concatenation:

greeting = "Hello" + " " + "World!"

**Variables and Assignments**

* Variables store values for later use.
* name = "Alice"

age = 25

**7. Understanding Syntax and Semantics**

**Syntax**

* Defines the structure of valid expressions.
* Example of correct syntax:

print("Hello, World!")

* Example of incorrect syntax:

print("Hello, World!) # Missing closing quote

**Static vs. Full Semantics**

* **Static Semantics:** Determines whether an expression is meaningful before execution.
* **Full Semantics:** Describes what a program does when executed.

**Errors in Python**

* **Syntax Errors:** Mistakes in code structure (e.g., missing brackets).
* **Semantic Errors:** Code runs but produces incorrect results.
* **Runtime Errors:** Occur during execution (e.g., division by zero).

**8. Key Takeaways**

1. **Computation** is about systematically solving problems using a structured approach.
2. **Imperative knowledge** describes step-by-step methods to solve problems.
3. **Stored Program Computers** allow programs to be written and executed dynamically.
4. **Programming languages** provide an interface for writing computation recipes.
5. **Python** is an interpreted, high-level language that simplifies computation.
6. **Syntax and semantics** are crucial for writing correct programs.
7. **Errors in programming** can be syntactic, semantic, or runtime-related.

**9. Practice Questions**

1. **Conceptual Questions:**
   * What is the difference between declarative and imperative knowledge?
   * Why are stored program computers more powerful than fixed program computers?
   * What are the advantages of using Python for beginners?
2. **Coding Challenges:**
   * Write a Python program that calculates the square root of a number using Heron's method.
   * Write a program that concatenates two user-inputted strings and prints the result.

**Conclusion**

This lecture introduced the foundational ideas of computation, programming, and problem-solving. Understanding these concepts will help students develop computational thinking, an essential skill for tackling real-world problems with code.