# Introduction to Computer Science

# Lecture 1: Fundamentals of Programming

## 1. Introduction

Computer science is the study of computation, automation, and information. Computer science spans theoretical disciplines (such as algorithms, theory of computation, and information theory) to practical disciplines (including the design and implementation of hardware and software). In this course, we will explore the fundamentals of programming and computational thinking, which form the foundation of computer science.

## 2. What is Programming?

Programming is the process of creating a set of instructions that tell a computer how to perform a task. Programming can be done using a variety of computer programming languages, such as Python, Java, C++, and many others.

### 2.1 Key Programming Concepts

• Variables: A variable is a storage location, paired with an associated symbolic name, which contains a value. Variables in programming are analogous to 'containers' that hold information.
• Data Types: A data type is a classification of data which tells the compiler or interpreter how the programmer intends to use the data. Common data types include integers, floating-point numbers, characters, and strings.
• Control Structures: Control structures are programming constructs that allow for the control of the flow of execution. Examples include conditionals (if-else statements) and loops (for and while loops).
• Functions: A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and reuse of code.

# 3. Introduction to Algorithms

An algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation. Algorithms are always unambiguous and are used as specifications for performing calculations, data processing, automated reasoning, and other tasks.

## 3.1 Characteristics of Algorithms

- Input: An algorithm has input values from a specified set.
- Output: An algorithm produces output values from a specified set. The output values are related to the input values.
- Definiteness: Each instruction must be clear and unambiguous.
- Finiteness: The algorithm must terminate after a finite number of steps.
- Effectiveness: Each instruction must be basic enough to be carried out by a person using only pencil and paper.

## 3.2 Algorithm Analysis

The analysis of algorithms is the determination of the amount of resources (such as time and storage) necessary to execute them. Most algorithms are designed to work with inputs of arbitrary length. Usually, the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as time complexity, or volume of memory, known as space complexity.

# 4. Introduction to Programming Languages

A programming language is a formal language comprising a set of instructions that produce various kinds of output. Programming languages are used to implement algorithms.

## 4.1 Types of Programming Languages

• Low-level languages: These languages provide little or no abstraction from a computer's instruction set architecture. They include machine code and assembly language.
• High-level languages: These languages enable a programmer to write programs that are more independent of a particular type of computer. They are more abstract and easier to use than low-level languages. Examples include Python, Java, C++, and many others.
• Scripting languages: These are high-level programming languages that are interpreted rather than compiled. Examples include JavaScript, PHP, and Python.

## 4.2 Programming Paradigms

• Imperative programming: This paradigm uses statements that change a program's state. It focuses on how to achieve a goal step-by-step.
• Declarative programming: This paradigm expresses the logic of a computation without describing its control flow. It focuses on what the program should accomplish, rather than how to accomplish it.
• Object-oriented programming: This paradigm is based on the concept of 'objects', which can contain data and code. Data in the form of fields, and code, in the form of procedures.
• Functional programming: This paradigm treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

## 5. Conclusion

In this introductory lecture, we have covered the fundamental concepts of computer science and programming. We have explored what programming is, the key concepts in programming, introduction to algorithms, and different types of programming languages and paradigms. In the next lecture, we will delve deeper into the specifics of a particular programming language and start writing our first programs.

## 6. References

1. Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein
2. Structure and Interpretation of Computer Programs by Abelson and Sussman
3. The Art of Computer Programming by Donald Knuth