Introduction to Programming

Duration: 2 Days **Course Code: GK840203**

Overview:

Learn the essentials of programming and start your coding journey.
Introduction to Programming is perfect for anyone looking to get started with coding. Whether you're aiming to kickstart a career in software development, enhance your problem-solving skills, or simply automate everyday tasks, this course has got you covered. You'll dive into the basics of programming, learning about data types, variables, operators, and control flow. By the end of the course, you'll be able to write simple programs and understand the fundamental concepts that underpin all programming languages.
This course is designed to be accessible to beginners, so no prior programming experience is required. Through a mix of lectures, hands-on exercises, and practical labs, you'll gain a solid foundation in programming. You'll also learn about the differences between compiled and interpreted languages, and get introduced to debugging techniques to help you identify and fix errors in your code. By the end of the course, you'll have the skills and confidence to tackle more advanced programming topics and projects.

Target Audience:

Anyone wanting to learn how to program and the different programming languages

Objectives:

- Understand the purpose and importance of programming in solving real-world problems and automating tasks.
- Differentiate between compiled and interpreted programming languages and identify their common use cases.
- Use core programming concepts, including data types, variables, operators, and expressions, to write simple programs.
- Create variables to store and manage different data types in a program.
- Apply arithmetic, comparison, and logical operators to perform operations and evaluate expression
- Construct conditional statements, including if, else if, else, and switch, to handle decision-making in programs.
- Build programs that incorporate nested and chained conditionals for solving more complex problems.
- Use debugging techniques to identify and fix logical errors in simple programs.

Prerequisites:

- A desire to learn programming.
- Basic math and algorithm skills would be helpful

Content:

Introduction

- Definition ; Purpose of Programming
- What is programming and why do we do it?
- How programming solves real-world problems
- Evolution and brief history

Programming Languages ; Basic Environment Setup

- What Are Programming Languages?
- Definition and purpose of programming languages
- Types of Programming Languages - Procedural, Object-Oriented, Scripting, Functional
- Compiled vs. Interpreted languages (high-level)
- Brief mention of popular languages (Python, JavaScript, C++, etc.)

Environment Setup

- Selecting a code editor (e.g., VS Code, Sublime, online playgrounds)
- Brief mention of installing the required runtime or interpreter for a programming language (e.g., Python, Node.js, Java)
- Creating code files and saving them with appropriate extensions
- Running code from the command line vs. integrated terminals
- Simple demonstration of creating and running a generic “Hello World” file

Structure of Code

- Common parts of a program
- How programming languages have syntax (rules) and semantics (meaning)
- What is syntax?
- Importance of indentation
- Code comments
- General structure of a simple program (pseudocode)

Data Types ; Variables

- Data Type Fundamentals
- Definition and purpose
- Primitive data types (numbers, strings, booleans)
- Non-primitive or composite data types (high-level mention, e.g., arrays/objects)
- Memory concept: how data is stored (conceptually)
- Variable Declaration ; Initialization
- What are variables?
- Purpose of storing data in variables
- Common declaration keywords (e.g., var, let, const or equivalents in other languages)
- Naming conventions and best practices
- Working with Variables
- Updating and reassigning values
- Combining variables in expressions (e.g., string concatenation or numeric calculations)
- Basic examples of variable usage in a small snippet (pseudocode)

Operators ; Expressions

- What Are Operators?
- Definition and purpose
- Types of Operators
- Arithmetic Operators
- Addition, subtraction, multiplication, division, modulus
- Operator precedence (order of operations)
- Comparison Operators
- Greater than, less than, equality vs. strict equality (conceptual differences)
- Logical operators (AND, OR, NOT)
- Truthy/falsy concepts in a general sense
- Expressions ; Evaluations
- Combining variables and operators to form expressions
- Evaluating an expression to produce a result
- Example scenarios (calculations, comparisons, etc.)

Control Flow (Conditionals)

- What Are Conditionals?
- Explanation

Functions

- Introduction to Functions
- What are functions?
- Purpose of functions (reusability, organization of code)
- How functions solve problems by breaking them into smaller tasks.
- Definitions: parameters vs. arguments
- Returning values vs. side effects
- Function Syntax
- Example function declaration in pseudocode (parameters, body, return statement)
- Calling functions and passing arguments
- Best practices in function naming and scoping (local vs. global concept)
- Concept of "scoping" in a program
- Difference between the scopes

Loops ; Iteration

- What Are Loops?
- Definition and Purpose
- Why loops are used (e.g., processing collections of data, automating repetitive tasks).
- Loop Types (Conceptual)
- While loops and do-while loops (differences in checking conditions)
- For loops (count-based iteration)
- Nested loops (when to use and when to avoid complexity)
- While Loop
- Do-while loop
- Reverse Loop
- Loop Control
- Break and continue statements (high-level)
- Infinite loops (what they are and how to avoid them)

Basic Data Structures

- What Are Data Structures?
- Definition and Purpose
- Why they are important in programming
- Arrays/Lists
- Storing multiple items in a single structure
- Array/ List with singular data types
- Array/ List with multiple data types
- Indexing and accessing elements (conceptual)

Debugging

- Debugging Fundamentals
- Types of errors (syntax, runtime, logic)
- Reading error messages
- Using print/log statements for debugging

- How programs use conditions to respond to different situations
- If/Else Statements
- General structure (if, else if, else)
- When to nest vs. when to chain else-if
- Complexity considerations and code organization
- Switch/Case (Conceptual)
- Differences compared to if/else
- Typical use cases (multiple distinct cases)
- Break and default statements (behavior and necessity)

Further Information:

For More information, or to book your course, please call us on 0800/84.009

info@globalknowledge.be

www.globalknowledge.com/en-be/