**Introduction to Programming**

|  |
| --- |
| This curriculum is designed for beginners with no prior programming experience. It introduces fundamental concepts applicable to most programming languages and gradually builds upon them. Afterward, beginners are encouraged to pick a programming language of interest. |
| **Week 1: Fundamentals of Programming**   * What is programming?: Demystifying the concept, showcasing real-world applications, and highlighting problem-solving potential. * Basic computational terms: Understanding variables, data types, operators, expressions, and functions. |
| **Week 2: Fundamentals of Programming (Cont’d)**   * Thinking like a programmer: Algorithmic thinking, breaking down problems into steps, and designing solutions. * Introduction to programming languages: definition, concept, types of programming language. * Selecting a language: Considering career goals, project interests, and available resources. * Coding environment: Introduction to a chosen Integrated Development Environment (IDE) or online platform, basic navigation, and features. * Exploring language learning platforms and resources: Online courses, tutorials, documentation, and communities. |
| **Week 3: Control Flow and Logic**   * Conditionals: Understanding if statements, else clauses, and comparison operators. * Loops: Introducing for and while loops, iteration concepts, and controlling loop behavior. * Decision-making and branching: Building logic structures with conditionals and loops, making choices within programs. * Practice through programming challenges: Simple exercises involving conditionals and loops to reinforce understanding. |
| **Week 4: Functions**   * Functions: Defining and calling functions, understanding reusability and modularity in code. |
| **Week 5: Data structures**   * Data structures: Introducing lists, dictionaries, and basic operations like adding, removing, and accessing elements. * Organizing and manipulating data: Using data structures to store and work with information efficiently. * Project: Building a simple data management program: Applying functions and data structures to solve a practical problem (e.g., managing friends' contact information). |
| **Week 6: User Input and Output**   * Getting user input: Using prompts and functions to read data from the user. * Displaying information: Printing text, numbers, and variables to the console or GUI. * Interactive programs: Combining user input and output to create responsive and engaging experiences. * Project: Interactive quiz or guessing game: Building a program that interacts with the user and provides feedback. |
| **Week 7: Debugging and Error Handling**   * Common errors: Understanding syntax errors, logic errors, and runtime errors. * Debugging techniques: Using print statements, stepping through code, and analyzing error messages. * Error handling: Implementing mechanisms to gracefully handle and report errors. * Project: Debugging challenge: Fixing errors in a pre-written program to understand error detection and correction. |
| **Week 8: Object-Oriented Programming (OOP) Concepts**   * Classes and objects: Modeling real-world entities with classes, defining attributes and methods. * Inheritance and polymorphism: Understanding how objects inherit properties and behavior from other classes. * Encapsulation and abstraction: Hiding internal details and focusing on essential functionalities. * Exploring object-oriented languages: Introducing popular OOP languages like Python, Java, C++, and their basic syntax. |
| **Week 9: Capstone Project**   * Final project: Choosing a project idea that applies learned concepts and showcases individual interests. * Project development: Planning, coding, testing, and polishing the final project. * Presentation and reflection: Sharing the project with peers and instructors, discussing challenges and achievements. * Next steps: Setting learning goals for the future, exploring specific languages and development areas. |