Transcript

Module 1: Fundamentals for Beginners

Video 1: Module Overview

Welcome to our "Fundamentals for Beginners" module! Before we dive into the content, let me share an interesting story that illustrates the power and excitement of programming and web development.

Imagine a young entrepreneur named Jane. A few years ago, Jane had a simple idea to create an online platform where anyone could showcase and sell their handmade crafts. Passionate about supporting local talent but with limited technical knowledge, Jane faced a daunting challenge. However, with determination and a keen interest in learning, Jane decided to dive into the world of programming and web development.

Starting with basic programming concepts, Jane gradually learned to build a simple website. As the platform evolved, Jane implemented advanced features like secure payment processing, real-time chat for buyers and sellers, and personalized recommendations based on user behaviours. Through continuous learning and application of web development technologies, Jane transformed a small idea into a thriving online marketplace, connecting anyone with customers from all over the world.

Jane's story highlights the incredible potential of programming and web development. Whether you have a groundbreaking idea or a passion for solving problems, the skills you acquire in this field can empower you to turn your visions into reality.

In this first module, we will begin by understanding what programming is and its importance. You'll get an overview of various programming languages and learn what an algorithm is.

Next, we will explore development environments. You will learn what they are and why they are important. This knowledge will help you create and manage your code efficiently.

We will then move on to the basic building blocks of programming languages such as variables, operators, expressions, control structures, and functions.

After laying the foundation for programming, we will provide an overview of web development technologies and full stack development. You will learn what the components of full stack development and why it is called full stack development.

By the end of this module, you will have a solid understanding to the fundamentals of programming and web development. You will be equipped with knowledge to start building your own projects. Thank you for joining this journey, and I look forward to seeing your progress.

Video 2: Introduction to Programming

Welcome to the lecture on "Introduction to Programming".

Here are the learning objectives. By the end of this video, you should understand what programming is, get an overview of different programming languages, understand what an algorithm is, understand what algorithmic thinking is, and appreciate what development environments are.

Programming is essentially the process of creating instructions for computers to follow. Think of it as giving your computer a list of tasks to do. These instructions help automate tasks, solve problems, and create the software and applications that we use daily, from mobile apps to complex systems.

Why should you learn programming? There's a high demand for programmers in various industries, making it a lucrative career choice.

But it's not just about job opportunities. Programming also enhances your logical thinking and problem-solving skills, which are valuable in any field.

Moreover, it gives you the power to bring your project ideas to life, whether personal or professional.

A programming language is essentially a set of instructions crafted by a developer to direct a computer to perform specific tasks.

Think of it as a special language that allows us to communicate with machines, giving them precise directions on what to do.

They fall broadly into two categories: high-level languages and low-level languages.

High-level programming languages are designed to be more human-readable.

For example, here's a JavaScript code that shows a pop up message in the web browser.

High-level languages are easier to learn and use because they abstract away the hardware details and use English-like syntax.

Examples include JavaScript and Python. They're commonly used for web development, software development, data analytics, and more.

High-level languages eventually get translated into low-level languages, which the machine can directly understand.

These provide finer control over the hardware.

Here's an example of what low-level programming looks like in Assembly, which adds two numbers.

The MOV instruction copies the value 1 to the register AX and 2 to the register BX.

The ADD instruction then adds these values and stores the result back into register AX.

Here's a comparison between high-level and low-level programming languages.

High-level languages are easy to learn. They provide less control over hardware, and are highly portable. Examples include JavaScript and Python.

Low-level languages are harder to learn. They provide more control, but are less portable. Examples are Assembly and Machine code.

Different high-level languages serve different purposes. For web development, there are JavaScript, HTML, and CSS.

For analytics, Python is very popular , and for data manipulation, SQL is widely used.

Now, let's talk about algorithmic thinking. This is an essential mindset for programmers.

An algorithm is a set of step-by-step instructions to solve a problem. It helps break down complex problems into manageable steps, making it easier to write efficient programs.

Suppose we want to find the largest number in a list. An algorithm for this would involve:

First, initializing a variable to hold the largest number. Second, Looping through the list.

Finally, Comparing each number with the current largest number. Here's how this could be implemented in Python.

When developing software, we utilize a variety of software tools to make our tasks more efficient and manageable.

These tools fall into two main categories: Integrated Development Environments, also known as IDEs, and Version Control tools.

Integrated Development Environments or IDEs are powerful tools that help developers write, test, and debug their code more efficiently. Some popular IDEs include Visual Studio Code and PyCharm.

For instance, Visual Studio Code or VS Code, offers numerous features such as syntax highlighting, code completion, version control integration, and a rich ecosystem of extensions to enhance your development experience.

Version control is a fundamental aspect of modern software development. Version control allows you to keep track of the changes made to your codebase over time.

Here's a screenshot illustrating how VS Code can allow us to perform version control directly from the IDE.

To summarize, we have covered what programming is, the differences between high-level programming languages vs low-level programming languages, how to apply algorithmic thinking, and development environment.

I hope this video has given you a solid foundation in the basics of programming. Let's continue learning together!

Video 3: Working with Variables and Operators

Welcome to the lecture on "Working with Variables and Operators".

Here are the learning objectives. By the end of this video, you should be able to understand what variables and data types are, and use operators to manipulate values and variables.

Variables are essentially containers for storing data values. Think of them as boxes that can hold specific pieces of information. We use identifier names to denote these containers.

These identifiers are the names we give to variables so that we can refer to the stored data later on.

There are two key steps when working with variables: declaring a variable and assigning a value to it. Declaring a variable means to create a variable. On the other hand, assigning a value means to put data into that variable.

For example, in JavaScript, we write "var a" to declare a variable named a. We can then assign the value 5 to the variable a by writing a = 5;

Reference Error: It's important to note that if you try to reference a variable that hasn't been declared, you'll encounter an error. This is because the program doesn't know about the variable yet.

For example, in this code example, if we were to try to access the value of “c” but “c” is not declared, we will encounter an error.

Data type: Each variable has a specific data type. Data types specify the type of data that is stored in a variable. They are crucial because they help the programming language understand what kind of operations can be performed on the variable. Common data types include number, string, and Boolean. Numbers can be whole numbers or floating point numbers. Strings are text, and Boolean represent true or false values.

Web Browser Developer Console: To get started with programming with JavaScript, we can use the developer console in our web browser. This is a handy tool that lets us write and execute JavaScript code directly within the browser. To access the developer console, press F12 on Firefox or Chrome and select the Console tab.

Remember, nobody learns better than actually doing the coding. So, pause the video at any time to try out any code examples. This hands-on practice will help solidify your understanding.

console.log(): Let's look at a simple Javascript code example. We can use the “console.log” function to print out values in JavaScript. This is particularly useful for debugging and seeing the output of your program. For example, if we write “console.log("Hello World")”, it will print the text "Hello World" in the console.

Variable Declaration: Now, let's revisit variable declaration with an example in JavaScript. We can declare a variable using the “var” keyword. For example, writing “var a” declares a variable named a.

Identifiers are the names given to variables or functions. The naming follows specific rules. For example, they must start with a letter or the underscore character. They can be followed by letters, digits or the underscore character. Identifiers must also not be keywords or reserved words in the programming language.

It's important to note that many programming languages are case-sensitive. This means that Num1 with an upper case N and num1 with a lower case n would be two distinct identifiers representing two different variables. It is also a good practice to use meaningful variable names.

Assignment Operator: In most programming languages, we use the equal sign as the assignment operator to assign a value to a variable. The identifier is placed on the left-hand side, and the value to be assigned is on the right-hand side. For example, “var a = 5” assigns the value 5 to the variable a.

Getting the Data Type: In JavaScript, we can use the typeof operator to get the data type of a value or variable. This is particularly useful for debugging and understanding what kind of data you're working with.

For example, if we declare a and assign a value of 5 to it, we can then use the console.log function to print its value and data type.

Dynamically Typed Languages: JavaScript and Python are examples of dynamically typed languages. In dynamically typed languages, we do not need to specify the data type explicitly. Instead, the data type of each variable is inferred during runtime dynamically.

This means that the type of a variable can change as the program runs.

Variable Reassignment: Variables in JavaScript can be reassigned to another value using the assignment operator. Note that variables in JavaScript can also be reassigned to values of different types. For example, we can change the variable “a” from a number to a string.

Arithmetic Operators: In addition to the assignment operator, we can use other arithmetic operators that we are familiar with in Math to perform calculations. For example, if we declare and assign 2 numbers: num1 and num2, we can use the + operator to add them up and assign the sum to a new variable called sum.

In this lecture, we have covered what variables and data types are, how to declare and assign values to variables, variable reassignment and its effect on data types, and basic arithmetic operators.

Video 4: Working with Control Flows

Welcome to the lecture on "Working with Control Flows”.

Here are the learning objectives. By the end of this video, you should be able to understand the need for control flows, how to define Boolean expressions, and how to work with both conditional flows and iterative conditional flows.

Control Flows: A program is essentially a series of instructions executed sequentially. However, there are situations where you need to selectively execute instructions based on specific conditions or repeat instructions multiple times. This is where control flows come into play.

To illustrate control flow, consider the example of performing data modelling. Depending on whether we have data or not, we can conditionally execute different blocks of instructions. In this code, if HAVE_DATA is true, we perform data modelling. Otherwise, we execute instructions to find data.

Boolean Expressions: Control flows rely on Boolean expressions, which are expressions that evaluate to either true or false. These Boolean expressions form the foundation for making decisions in your code.

Comparison Operators: Here are some examples of Boolean expressions using comparison operators. These operators compare values and return a Boolean result. These comparison operators help determine relationships between values.

In addition to comparison operators, logical operators are used to build more complex Boolean expressions. Here are some examples.

Logical operators allow you to combine or invert Boolean expressions.

Building on our earlier example, let's see how we can use control flows to execute multiple instructions conditionally. Initially, we had.

Now, we can break down PERFORM_DATA_MODELLING into more detailed steps.

This makes it clear what actions are taken when we have data, including data cleaning, generating data models, and reporting results.

Control flows can also involve repeating a block of instructions as long as a condition remains true. For example, we can use a while loop to continuously find data until we have data:

This loop will keep executing FIND_DATA as long as NO_DATA is true.

Here's a visual representation of a while loop. The while statement checks a condition before executing the block of code. If the condition is true, the code block executes, and the loop repeats. If the condition is false, the loop terminates.

Combining everything together, this is a flowchart that visually represents the process of performing data modelling.

It starts with checking if we have data. If we do, we proceed with data modelling steps; if not, we find data.

In this lecture, we have covered the basics of control flows, including defining Boolean expressions and implementing conditional flows and iterative conditional flows. These concepts are fundamental in making your programs more dynamic and responsive to different conditions.

Video 5: Working with Functions

Welcome to the lecture on "Working with Functions". Here are the learning objectives. By the end of this video, you should be able to understand what functions are, how to call them in your programs, and how to define your own functions.

A function is a block of code designed to perform a specific task. Functions help us reuse code, making our programs easier to read and maintain. Instead of writing the same code over and over again, we can define a function once and use it whenever needed.

Imagine you need to greet users multiple times in your program. Without functions, you might write something like this.

As you can see, we are repeating similar lines of code. This can make our program longer and harder to manage.

Using a function, we can greet users without repeating code. Here are some code snippet how we can achieve this. With this approach, our code is shorter and much easier to manage. We define the greet function once and reuse it by calling it with different names.

In JavaScript, a function is defined using the function keyword followed by a name, parentheses, and a block of code. Here's a simple example. This defines a function named greet that, when called, prints "Hello, World!" to the console.

To run a function, we need to call the function using its name followed by parentheses. Functions can take input values called parameters. Parameters are placed inside the parentheses when defining the function. In this example, name is a parameter. This greet function uses this parameter to personalize the greeting.

To call a function with parameters, we pass the values inside the parentheses. By calling greet with different names, we get personalized greetings for each name.

Functions can return a value back to the caller using the “return” statement. In this example, the “add” function takes two parameters, adds them together, and returns the result. We can then use this result in our program.

In this lecture, we have covered what functions are and why they are useful, how to define and call functions, and using parameters and return values. Understanding functions is a crucial step in becoming a proficient programmer. Functions make your code more organized, reusable, and easier to understand.

Video 6: Overview of Web Development

Welcome to the lecture on "Overview of Web Development”. Here are the learning objectives. By the end of this video, you should be able to appreciate the different web development technologies, understand the difference between static and dynamic web pages, and appreciate the various web development approaches.

Web development involves several key technologies. HTML or HyperText Markup Language, defines the structure of web pages. CSS or Cascading Style Sheets, defines the style and layout of these pages. JavaScript which is often abbreviated as JS, adds behaviour and interactivity to web pages.

HTML is a text-based document that specifies the structure and content of a webpage. Here is a simple example of an HTML document. As you can see here, developers write HTML documents using tags. For example, every HTML page should start and ends off with the html tag. The head section is defined using the head tag and the body section is defined using the body tag. Within the body of page, we can define more HTML tags to specify the structure and content. For example, in here we write the contents of a paragraph within a p tag.

While HTML provides the structure, CSS enhances the visual appearance of HTML elements. It is used to style and layout web pages. For example, in this snippet, we define the background color, font size, and text color of the body element to apply this styling to all the contents of the body section.

JavaScript adds interactivity and dynamic behaviour to web pages. It can manipulate HTML and CSS to update content in real-time. Here is an example of a simple JavaScript function that validates a form. The function checks if the name field is empty, and if so, displays an alert and prevents form submission. JavaScript is crucial for creating responsive and interactive user experiences.

Web pages can be static or dynamic. Static web pages have fixed content. To update them, we manually edit the files and upload them to the server. On the other hand, dynamic web pages change content based on user interaction. Most modern websites are dynamic, allowing for a more personalized and interactive user experience.

There are two main ways to generate dynamic pages: server-side and client-side generation.

Server-side generation involves the server fetching content, such as from a database, and sending a complete page to the client.

Client-side generation involves the browser fetching data and dynamically updating the page using JavaScript.

Both approaches have their own use cases and benefits.

In server-side generation, the server fetches the necessary content and sends a fully rendered page to the client. From the browser's perspective, this process is no different from receiving a static page. The server handles all the logic and data fetching.

In client-side generation, the browser fetches data and renders content dynamically. JavaScript is used to manipulate the DOM, or Document Object Model, to update the page. This approach allows for a more responsive and interactive user experience, as the page can update without requiring a full page reload.

Creating web applications goes beyond just creating the structure and styling of web pages. We need to incorporate greater interactivity, such as filtering data and handling form submissions, and develop the logic to manage this interactivity. Typically, a server powers these applications, handling the backend logic and data storage.

Web development can be divided into backend and frontend development.

Backend development focuses on server-side logic, such as database interactions and API development.

Frontend development focuses on the client-side interface, including webpage design, styling, and interactive elements.

Both are essential for building a complete web application.

Developers use various tools to streamline their workflow. Version control systems like Git and GitHub help manage code changes and collaboration. Code editors and integrated development environments (IDEs) like Visual Studio Code, Sublime Text, and Atom provide a robust environment for writing and testing code. Browser developer tools, such as inspect element and console, are invaluable for debugging and optimizing web pages.

To summarize, we've covered the different web development technologies: HTML, CSS, and JavaScript. We've discussed the difference between static and dynamic web pages and explored the different approaches to web development, including server-side and client-side generation. These foundational concepts are crucial for understanding modern web development.

Video 7: Introduction to Full Stack Development

Welcome to the lecture on "Introduction to Full Stack Development". Here are the learning objectives. By the end of this video, you should be able to understand what Full Stack Development is, identify the components of Full Stack Development, identify the popular Full Stack technologies, appreciate the benefits and challenges of Full Stack Development, and understand the learning path to becoming a Full Stack Developer.

Full Stack Development refers to the development of both front-end and back-end technologies. It involves working on both the client side and the server side of an application. Having knowledge in both areas allows a developer to be dynamic and effective.

A Full Stack Developer is responsible for designing complete user experiences, developing both front-end and back-end applications, and managing databases. They are versatile professionals who can work across the entire technology stack, making them invaluable in the development process.

Let's briefly compare the front-end development and back-end development. Front-end development is related the client side, and back-end development is related to the server side. Front-end development involves using HTML, CSS, and JavaScript to create nice user interface and good user experience. The back-end involves server logic, database interactions, and handling requests and data processing. Both sides need to work together seamlessly to create a functional application.

Now, let's look at the technologies used in front-end development. HTML defines the structure of web pages. CSS handles the styling and layout. JavaScript adds interactivity and dynamic behaviour. Popular frameworks used for frontend development include CSS frameworks such as Bootstrap, and Javascript development libraries or frameworks such as React.

On the backend, we have several key technologies. Server-side languages that power the server logic. This can be written in various programming languages such as Python and Javascript. Usually backend developers would also use a framework when working with these server-side languages to make the development process more efficient and systematic. The server logic usually involves connecting to a database to store data. There are various database options such as MySQL and MongoDB. The server applications would then need to run on an application server such as Apache and Nginx.

You might have heard of the term tech stack which depicts the choice of technology used in full stack development. Popular tech stacks include the MERN stack, MEAN stack, and the LAMP stack. These tech stacks are adopted because they usually integrate well with each other. Furthermore, as they are common tech stacks used, it also means that when you encounter an issue, it is easier to seek support on the web.

Let's take a closer look at the MERN stack. It consists of MongoDB, a NoSQL database. Express.js, a backend web application framework. React, a frontend library for building user interfaces, and Node.js, a JavaScript runtime for server-side development. This combination allows for efficient and scalable application development and is probably the most popular tech stack at the moment.

Full Stack Development offers several benefits. It ensures that the developers are versatile, being able to work on both the front-end and back-end. It increases efficiency by streamlining the development process, ensuring that there is clear separation of concerns of different aspects of the development. As full stack developers are able to handle both the front-end and back-end, it can be more cost-effective for companies as each developer can be utilized depending on the development needs. Being proficient with both the front-end and back-end technologies also means that developers can communicate more effectively with co-workers who is working on the other part of the stack.

However, full stack development also comes with challenges. It requires broad knowledge, as developers need to stay updated with multiple technologies. The complexity of handling both front-end and back-end can be demanding. However, we believe that the training that we provide here will help prepare you to tackle these challenges.

Here is a high-level learning path to prepare you for full stack development. We will be covering different aspects in different modules. In general, start off with the basics, get a solid foundations in HTML, CSS, Javascript will help you greatly in the long run. Next, you should learn about frontend development frameworks or libraries and backend development technologies. Understanding database technologies is also essential. After mastering the basics, try building projects to gain hands-on experience and stay updated with the latest trends and updates in web development.

To summarize, we have covered what Full Stack Development is, the components of full stack development, popular full stack technologies, the benefits and challenges of full stack development, and the learning path to becoming a Full Stack Developer. I hope this lecture has provided you with a comprehensive understanding of Full Stack Development.

Video 8: Module Summary

Welcome to the summary for this module on "Fundamentals for Beginners". In this lecture, we will recap the key points from each topic covered in this module. We began with an introduction to programming, defining it as the process of creating instructions for computers. We see that programming is becoming one of the useful skills relevant to any profession. It also trains someone to be able to think logically.

We explored programming languages, distinguishing between high-level and low-level languages. High-level languages, like JavaScript and Python, are more user-friendly, while low-level languages, like Assembly, offer greater hardware control.

We introduced algorithmic thinking, a method to solve problems through step-by-step instructions. This is crucial for breaking down complex tasks into manageable parts.

We briefly discussed development environments, focusing on Integrated Development Environments like Visual Studio Code and version control systems like Git, which streamline coding and collaboration.

We then discussed basic programming concepts such as variables, operations, expressions, control structures, and functions.

Variables store data values. We covered declaring variables, assigning values, and understanding data types, like numbers, strings, and Booleans.

Operators manipulate values and variables. Arithmetic operators, such as plus and minus, perform calculations, while comparison and logical operators help in the decision-making processes.

Control flows guide the execution of your program. Conditional statements such as the if-else statement execute code based on conditions. Iterative conditional statements are loops that repeat code blocks as long as a condition is met.

Functions are reusable code blocks that perform specific tasks. They help in organizing code, improving readability, and reducing repetition. We also covered how to define and call functions, and use parameters and return values.

Web development involves HTML for structure, CSS for styling, and JavaScript for interactivity. We also discussed the difference between static and dynamic web pages.

Full Stack Development covers both front-end and back-end technologies. We highlighted popular tech stacks like the MERN stack.

These fundamentals form the foundation for your journey in programming.