1. Introduction to programming basics (part I)

1.1 Introduction to phase 2

- HTML and CSS are not technically programming languages; they're just page structure and style information. But before moving on to JavaScript, we covered HTML and CSS under phase 1. We said HTML provides the basic structure of sites, which is enhanced and modified by other technologies like CSS and JavaScript. CSS is used to control presentation, formatting, and layout. JavaScript is a logic-based programming language that can be used to modify website content and make it behave in different ways in response to a user's actions.

1.2 Why do we need to learn JavaScript?

- The best way to answer this question would be to list out all the things you would need to do as a full stack web developer. Below, you will see list of technologies a person who wants to become a full stack web developer needs to know. The list will make it very clear on where the skill JavaScript fits in the whole picture.
    - You usually start form a design and have to convert the design (PSD) to a webpage using HTML, CSS and Bootstrap.
    - When you want to add some interactivity on the webpage (You use JavaScript)
    - If your webpage saves and retrieves data, you will need a database (MySql, MangooDB, etc.)
    - To get the data from the database, you would need to use a programing language such as JS(Node.JS), Java, etc.
    - If you host your webpage somewhere on a remote server, you will need a web server (like Apache or Node) to send the page to the user. If you use Node.js, you need JavaScript.
    - If you want to make your webpage faster and robust, you will need to use JavaScript frameworks like React or Angular
- So, if you want to be a successful full stack web developer, you really need to focus on JavaScript.

1.3 Definition of JavaScript: a language to develop interactive web pages

- **Interactive website (definition)**: A web site is called interactive, when it can communicate with the user in a meaningful way. Interactivity facilitates the user to be actively engaged

with the site. This is the main thing you need to understand as a front-end JavaScript developer.

- **What is your role as a front-end JavaScript developer?** A front-end developer lists out all the possible actions a visitor might take on your website, chooses those actions you want to respond to and finally write a script/logic that responds to whenever an action is taken by a user.
    - Responding can mean as simple as changing color or popping a text on the screen that would be meaningful to the user that took the action.
    - Action means anything the user does. It can be clicking or hovering over a specific element of your web site.
    - You can take evangadi.com's sign up/login page as an example. When you try to click on the login button without putting an email address, the email input's background color changed to red to alarm you that you need to provide your email address. The website is basically responding (by changing the background color to red) to the user's actions.

1.4 Definition of JavaScript: a programming language to instruct our compute

- **Programming language (definition)**: It is a language with set of rules to instruct a computer or computing device to perform specific tasks.
- **How is programming language used to instruct computers?** It is with the use of machine language that we can instruct a computer to do tasks. Machine language gives instructions as 0's and 1's. Let's in detail how machine code/language is used to instruct computer or other computing devises to perform tasks.
    - **Machine language (machine code)**: One thing you must understand is, computers operate in binary, meaning they store data and perform calculations using only zeros and ones. Machine language is a collection of binary digits or bits that the computer reads and interprets. Machine language is the only language a computer is capable of understanding.
    - **Binary (or base 2)**: Binary is a numeric system that only uses two digit, 0 and 1. The mechanism we store data in computers is by using transistors, the tiny switches that are activated by the electronic signals they receive. Transistor's only job is to serve as a switch to let electricity pass or not pass. The digits 1 and 0 used in binary reflect the on (passing of electricity) and off (not passing of electricity) states of a transistor. Meaning, we call the passing of electricity 1 and the not passing of

electricity 0. We can say computers only understand the passing or not passing of electricity, nothing else.

- **How can you represent the number 57 using the binary concept of letting electricity pass or not pass?**
    - First, convert this number which uses the decimal system (with digits from 0 to 10) into binary (with 0 and 1 digits).
        - 57 in binary is: 111001
    - Since the binary representation of 57 has 6 digits (111001), we need at least 6 flip-flops (collection of transistors) to represent the number 57 with electricity.
        - Reading 111001, we need 3 transistors that let electricity pass, then two transistors that don't let electricity pass and finally one transistor that lets electricity pass. Now if I plug in these 6 transistors to electricity, they will be in the order of:
            - |pass| pass| pass| no pass| no pass| pass|
            - |1 |1 |1 |0 |0 |1 |
        - These set of transistors are now located somewhere inside the memory with their own address.
    - If we do the same for the number 14, it will be:1110
    - Now let's say we want to give instruction to the computer to add 57 and 14, both of which are on its memory.
    - At this point, we need help to do the binary addition. That is why we need the help of the processor. The processor has different circuits for different tasks. Like, addition circuit, multiplication circuit, graphic processing circuit, etc. These processors have certain number of tasks that they are designed to perform. These TASKS have their own binary representation.
        - For example, the "Add" instruction has a representation of "1011010111010".
        - So, if I want to instruct the computer to add 57 and 14, what I would tell the processor would be something like this:

(1011010111010 111001 1110)

        - The above numbers now can be understood by the processor to mean “add 57 and 14”. This way of writing instruction to the computer is called Machine Code. We are talking in the language that the machine/computer really understands.
- **Assembly language (assembly code)**: Imagine how hard it would be to write instructions to the computer using 0s and 1s. To make it a little bit easier, people came up with this concept of giving a humanly readable representation to the above tasks the processor does. For example, saying “ADD” instead of 1011010111010
    - Now their instruction to ask the processor to add 57 and 14 would be something like: (ADD 111001 1110)
    - But the processor doesn't understand what ADD is. So, they wrote a converter that converts “ADD” instruction to 1011010111010 and send it to the processor.
    - That means, they started sending the instruction to the convertor instead of sending it directly to the processor.
    - **Assembly language**: is the above way of sending instruction to the processor through a convertor. Unlike machine language, in assmebly language the instruction is first sent to the converter and then the converter will send the instruction to the processor. Unlike machine language, which consists of binary characters, assembly languages are designed to be readable by humans. An assembly language is almost exactly like machine code, but it uses words in place of numbers.
- **Compilers**: The assembly language way of writing instructions to the computer was not easy enough for developers to write. They wanted to simply say “57 + 14”. Companies recognized this need and started writing software that let people write their instruction in a much simpler way that will later be converted into the binary code that the computer understands. These middle softwares that convert the humanly readable instructions to the binary instructions are called compilers. The instructions that are understood by these compilers are what we call Programing Languages
    - Different compilers have their own different standards/formats that they follow. This difference in standards led to the development of different programing languages like Java, C++, Python, JavaScript, etc.

1.5 Definition of JavaScript: a scripting language to instruct the browser

- **Scripting/interpreted languages**: scripting languages are programming languages that do not need an additional step to be converted to a language a machine or computer understand because they are written to just communicate with one software, not with a machine/computer. For example, JavaScript was originally developed to only communicate with web browsers. It is the web browsers that directly communicate with the processor. Because JavaScript needs only to instruct the browser, it originally didn't need a compiler that converts it to the machine code. It only needed an interpreter to be understood by the browser.
    - **Interpreters**: though scripting languages do not need compilers, they still need interpreters to convert them into the language the software they are created to communicate with.
        - **How does browser understand the JavaScript instruction?** Browsers have their own interpreter which understands JavaScript codes. These kinds of languages which are understood by other software are called interpreted/scripting languages.
        - Every browser has a JavaScript engine that takes our JavaScript code and converts it into something that the browser can understand. Ex. IE (Chakra), Chrome (V8), Firefox (SpiderMonkey), Safar (JavaScriptCore)
        - Here is a link to a diagram showing the interactions of an HTML file, a CSS file and a JavaScript file with the Browser.
            - https://www.oreilly.com/library/view/php-mysql/9781449355517/httpatomoreillycomsourceoreillyimages1416220.png.jpgb
- **JavaScript is now both a programing and a scripting language**: JavaScript, as its name implies is a scripting language, meaning, it is used to instruct/ communicate with the browser. When JavaScript was originally developed, it was with the intention that it was to be used to communicate with the browser only. Consequently, JavaScript's popularity was dramatically increased because of Node.js. Node allows developers to write JavaScript code that runs directly in a computer process itself instead of in a browser. Node can, therefore, be used to write server-side applications with access to the operating system, file system, and everything else required to build fully functional applications. Because Node.js made it possible for JavaScript to communicate with the computer, JavaScript also

considered as a programming language as well. No worries, we will cover Node.js in detail under phase 3 of this course.

- **Client- side vs server-side JavaScript:** You might come across with phrases like client-side or server-side scripts to explain why JavaScript is both a scripting and a programming language. when JavaScript is explained. Below, please find what is meant by client-side and server-side scripts/instructions:
  - **Client- side scripts:** these are JavaScript instructions/scripts that are only understood by the browsers. This usually is related with DOM.
  - **Server-side scripts:** these are JavaScript instructions that are mainly written to communicate directly with the computer/device.

1.6 Core JavaScript: syntax and semantics

- As stated above, JavaScript is a language we use to send instructions to both to the browser or the computer. For the interpreter or the compiler to understand our instruction/script, we need to follow a standard that is set up by the developers of the compilers and the developers of JavaScript. These standards are called the core JavaScript. What we are going to cover in here is the standard way of writing a JavaScript code in a way that can be understood both by the interpreters or compilers.
- **Syntax and semantics of a programming language:**
  - **Syntax of a programming language**: All programing languages have their own way of formatting/structuring the instructions/code. Syntax is basically a collection of rules/standards to specify the structure or format of code. You can take syntax as a grammar, which is the rule to follow in making sentences in human languages.
  - **Semantics of a programming language**: Semantics refers to the meaning of the code/instruction we write. It is the associated meaning of the symbols, characters or any part of a program. You can take semantics as meaning of the sentence in human language. Even though we write the script with the right syntax, sometimes the script might be translated in a different way than we anticipated.
- If we make the mistake of following the standard syntax, the compiler or interpreter wouldn't understand our code, that means, our code won't work. Therefore, the first thing we need to learn is the syntax of JavaScript, then the Semantics.
- **Debugging**: The best practice you should follow when writing a script is to check every single line of code as you write it, both for the syntax and semantics error. That means you

need a way to double check your line of code as you write it. This process of checking your script is called debugging.

- The only way the debugger would know if your code worked or not is by compiling or interpreting the code. That means we either need a compiler or an interpreter to help us double check our code. This would be our feedback system.
- **Browser's console window for debugging**: One thing that is easily available for us to do debugging is the interpreters that we find on our browsers. Most browsers have the console window to display syntax errors and other things we write on the console. We use console to double check the code we write.

1.7 Including JavaScript in our HTML and using the console for debugging JavaScript

- Just like HTML and CSS, JavaScript is also a text file. HTML file uses ".html" extension. CSS file uses ".css" extension. JavaScript file uses ".js" extension. There are different ways of including JavaScript in our HTML.
- **Including an external JavaScript file in HTML (Recommended):**
  - Use the script tag with the attribute src . You've already used the src attribute when using images. The value for the src attribute should be the path to your JavaScript file.
  - Include the script tag in between the <head> tags in your HTML document.
  - Once you include your external JavaScript file into your HTML, you can use the "console.log()" method to display what you want on the console. Below is an example of code you will write if you want the code to log "Hello World" on the console: console.log("Hello World")
- **Including JavaScript code inline**:
  - Insert your JavaScript code directly inside the HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc. Example:

```
<body>
   <button onclick="alert('Hello World!')">Click Me</button>
</body>
```

- Note: You should avoid placing large amount of JavaScript code inline as it clutters up your HTML with JavaScript and makes your JavaScript code difficult to maintain

- **Embedding the JavaScript Code in your HTML**:
  - Embed the JavaScript code directly within your web pages by placing it between the <script> and </script> tags.
  - The <script> tag indicates the browser that the contained statements are to be interpreted as executable script and not HTML. Here's an example:
- Note!!!!! Please spend the coming few weeks exclusively on writing JavaScript code and debugging your code using the console. Don't worry about how we will use this programing knowledge (JavaScript) to build websites for now.

```
<body>
    <script> console.log("Hello World")</script>
</body>
```

1.8 Core JavaScript (variables): declaring variables

- **Variable**:
  - Variable is a container for storing data (values). It is a place where a script temporarily saves data for later use.
  - The name variable is used because the value in a variable can vary or be changed when the script run.
  - It is basically saying, “hey variable, please remember this data because we are going to use it later”
- **Declaring a variable**:
  - This is basically creating/announcing/introducing the variable name you want to use in the script. We use the reserved keyword “var” to declare a variable in JavaScript.
  - **Syntax to declare a variable in JavaScript**: Ex: var width;
    - Width is the name of the variable we are planning to use
    - var is the JavaScript keyword we use to declare a variable

        - Note!!! There are newer ways of declaring variables introduced in 2016 (ES6) using “let” and “const”. We will discuss the difference between var, const and let when we discuss scoping later on.
    - It's a good programming practice to declare all variables at the beginning of a script.

1.9 Core JavaScript (variables): assigning value to and changing value of a variable and rules on naming variables

- **Assigning a value to a variable**: After a variable is created/declared, it has no value (technically it has the value of undefined).
    - **We assign a value to a variable using the (=) operator:** We have already discussed that in JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator. This is different from algebra.
    - **Syntax for assigning value to a variable**: If we want to give/assign value to the above width variable we declared earlier, we use: width = 10;
        - **(=)** here is used as an assignment operator
        - 10 is the value we assign
        - Width is the variable
        - You can now use the variable by using its name
    - **Shorthand for declaring and assigning variables all at onc**e: You can also assign a value to the variable when you declare it. Look at this example if you want to declare and assign a variable at the same time: Ex: var width = 10;
        - **Re-declaring JavaScript variables**: If you re-declare a JavaScript variable, it will not lose its value. In the below example, the variable “width” will still have the value 20 after the execution of these statements.

```
Example:
var width= "20";
var width;
```

- **Changing the value of a variable**: Remember that a variable holds a value, and that value vary. In JavaScript, you can reassign values to a variable you once declared with let or var.

- **Syntax for changing/reassigning value of a variable**: When you assign a new value to a variable, you write the variable name, followed by (=) sign and your new value. Ex:
    - var width = 10; (declaring a variable by the name "width" and assigning the variable a value of 10)
    - width = 11; (changing or reassigning value 11 to width variable)
    - width = 12; (changing or reassigning value 12 to width variable)
- **Latest variable value always overrides the previous values**: At this point, our width variable has a value of 12 because it is the latest value.

- **Rules for naming variables**: All JavaScript variables must be identified with unique names. The general rules for constructing names for variables (unique identifiers) are:
    - Variable names can begin with a letter, $ sign or "_" (underscore)
    - Variable names cannot start with a number
    - Variable names can contain letters, digits, underscores, and dollar signs.
    - Variable names are case sensitive (y and Y are different variables)
    - Variable names should not include hyphen (-), space
    - Reserved words (like JavaScript keywords) cannot be used as variable names. Refer this link to find the list of reserved key words
        - https://www.w3schools.com/js/js_reserved.asp
    - If the variable name uses multiple words, it is a convention to capitalize the first letter of each word except the that of the first word. We call this a camel case variable naming convention. For example, if you want the variable name to contain "first" and "name", it is a convention to use "firstName" as variable name.