

# Web Design VS Web Development

**Made By**

Ahmed Mohamed Abubakr

Electronics engineer  
IC designer

Full stack web developer

Ambassador at TIEC-IA







# THE WEB ?!!

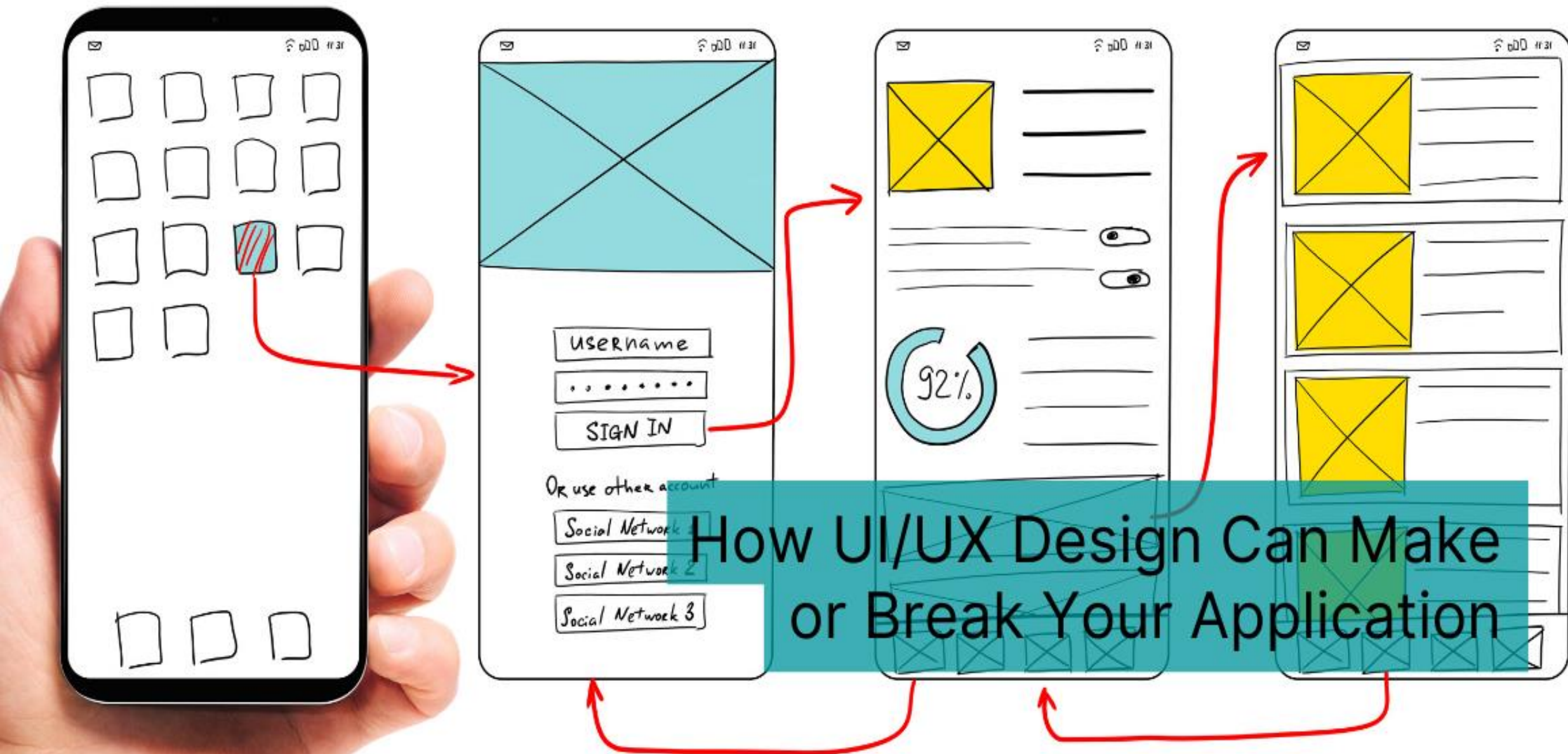
---

- ✓ What is websites and why we make it ?
- ✓ What is the sections inside it ?
- ✓ How I work ?
- ✓ What do I need to start ?

# Developers VS Designers

---

- ✓ What do we call designers and developers ?
- ✓ What tools they use ?
- ✓ Who starts the project ?





# What's the Difference?



## HTML

Hypertext Markup Language

### *Create the structure*

- Controls the layout of the content
- Provides structure for the web page design
- The fundamental building block of any web page



## CSS

Cascading Style Sheet

### *Stylize the website*

- Applies style to the web page elements
- Targets various screen sizes to make web pages responsive
- Primarily handles the "look and feel" of a web page



## Javascript

### *Increase interactivity*

- Adds interactivity to a web page
- Handles complex functions and features
- Programmatic code which enhances functionality

# UX/UI DESIGN VS WEB DEVELOPMENT

IF YOU HAVE THESE QUALITIES,  
YOU MAY BE A FUTURE ...

## UX/UI DESIGNER



- You're interested in how to improve things for others.
- You're open to different ways of looking at the world.
- You see improvements to be made everywhere – at school, at home, at work, at the gym, etc.

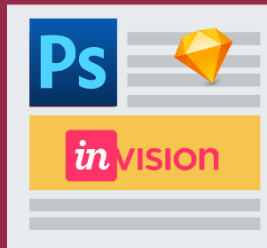
## WEB DEVELOPER



- You enjoy problem solving.
- You like to find solutions AND implement them.
- You look at error messages as a good thing.
- You think logically.

## ADD THESE TECH SKILLS...

- Adobe Photoshop
- Illustrator
- InDesign
- Sketch
- InVision



- Front End Languages like **HTML/CSS** + **JavaScript**
- Back End Languages like **Ruby** or **Python**
- Database technologies like **SQL** or **MongoDB**
- Fluency with **APIs**, **GIT**, basic **algorithms** + **data structures**



## AND YOU COULD GET JOBS LIKE ...

- PRODUCT DESIGNER
- UX RESEARCHER
- UX CONSULTANT
- UX DESIGNER

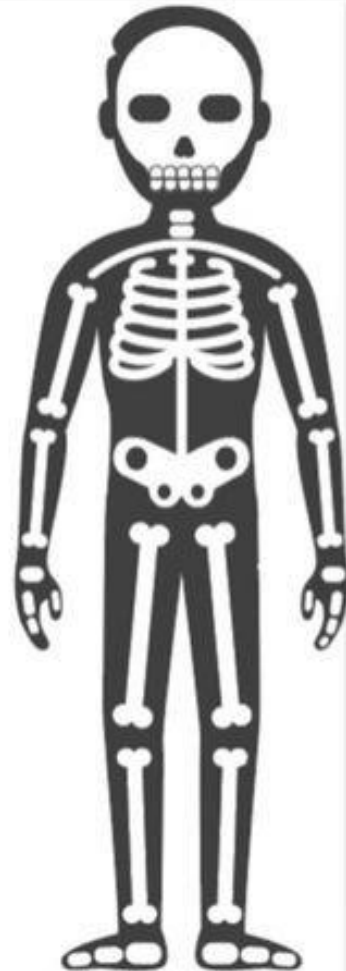


- TECHNICAL PRODUCT MANAGER
- FULL STACK DEVELOPER
- SALES ENGINEER
- QA ENGINEER
- IOS DEVELOPER

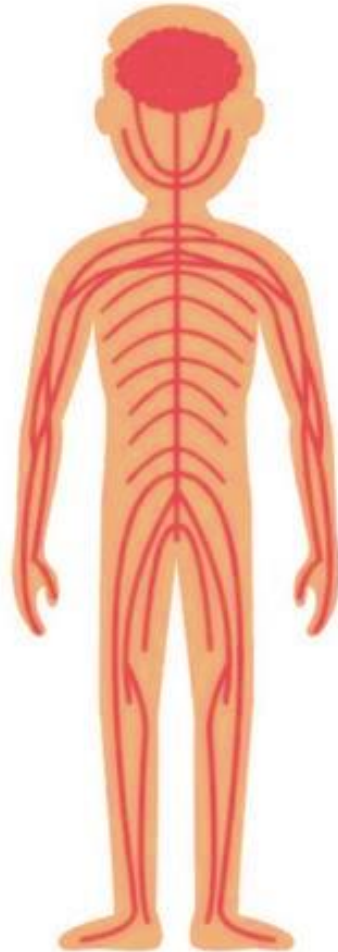




HTML



JS



CSS



Compiled VS Markup VS script languages

# **Mark-up language**

**Descriptions of how text and other forms of components should appear in a certain context (i.e. HTML, hypertext mark-up language - is the instructions for the context of a web page). Mark up languages tend to provide structure to data and/or determine how it is displayed.**

- **HTML- hypertext mark-up language**
- **Dynamic HTML**
- **XML-extensible mark-up language**
- **XHTML- extensible hypertext mark-up language**
- **CSS**



# Compiled language

A programming language is a language used by humans to give instructions to a machine.

The other main type is compiled programming languages. C, C++, Objective-C, Swift. These take text files, run them through a compiler, and the compiler creates binary instruction files (binaries). These are the lower level languages, they "talk to" the memory and processor.

Programming languages (almost always) need to be compiled before running and instruct the computer to perform tasks/calculations, including how to perform them (although if you're using a high-level language the low level instructions are hidden in the syntax).

- JAVA, C++, COBOL, C++ and VB, C#, etc.

## **Scripting language**

Usually use an interpreter or some running application to take programming commands and turn them into instructions to be executed. Scripting languages are programming languages, but they fit into a category called interpreted languages (i.e. Python, Ruby and PHP). We can write full featured applications with scripting languages.

A scripting language is a subset of programming language that is used to produce scripts, which are sets of instructions that automate tasks that would otherwise be performed manually by a human. Of course, these "tasks" are essentially a human giving instructions to a machine.



# **Difference between Programming Language and Mark-up Language**

Sometimes the terms programming language and scripting language are used colloquially to describe compiled programming languages and interpreted programming languages, respectively. Compiled programming languages are languages whose instructions are translated (compiled) directly into machine code, whereas interpreted languages are those that require a program known as an interpreter, which interprets instructions in terms of previously compiled machine code.

Mark up languages annotate the content of a document with information on the document's structure or presentation. More recently, the task of annotating documents with information about their presentation has been delegated to style sheet languages, such as Cascading Style Sheets (CSS). Both are related to historical, physical mark up and style sheets as seen in publishing and other fields.



# HTML (Hyper Text Markup Language)

---

- Meaning of HTML
- What can it do
- Syntax
- Tags
- Sourcing files
- Files HTML deal with
- Opening the code

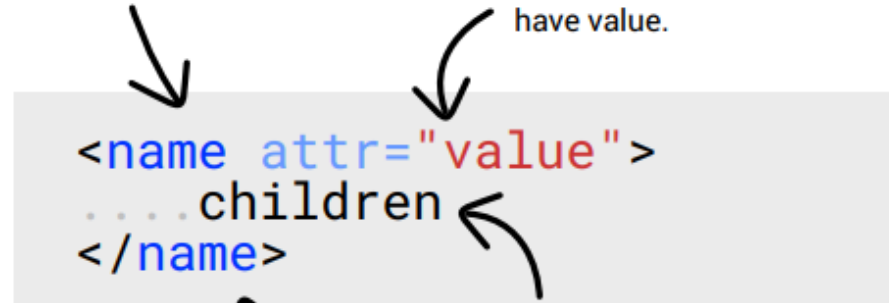
## ① Syntax to write an HTML element

### Opening Tag

Every element has an opening tag with the name of the element at its start.

### Attribute and its value (optional)

Attributes are like options of an element. Attributes have value.



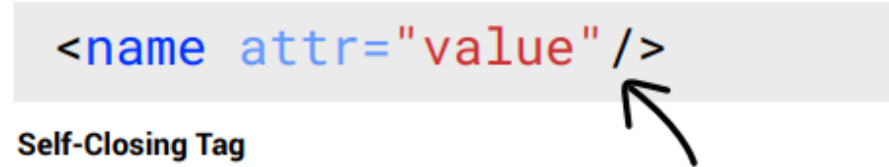
```
<name attr="value">
...children
</name>
```

### Closing Tag

A closing tag has the name of the element with a forward slash "/" before it.

### Children (optional)

Between the opening and closing tags are the children of the element. This can be more elements or just plain text.



```
<name attr="value" />
```

### Self-Closing Tag

Some elements that do not have children do not need a closing tag. In this case a forward-slash "/" is added to the element's opening tag to denote a self-closing tag.

Some examples of self-closing elements are:

**img, br, hr, meta, input ....**

## ② Basic markup of every HTML page



```
<!DOCTYPE html>
<html lang="en">
... <head>
... <title>
... page title here
... </title>
... </head>
... <body>
...
... page content goes here
... </body>
</html>
```

### Code Formatting

For good code formatting, remember to indent the children by 2 or 4 spaces.

## ③ Commonly used HTML elements

**headings:**  
<h1> ... <h6>

<p>

<i> <strong>

<a>

<div>

<span>

### lists:

<ul> <ol> <li>

### Special formatting

<blockquote> <pre>

### multimedia:

<img /> <video>

<audio>

### separators:

<hr /> <br />

# Lets have example

---

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hi-from-title</title>
  </head>
  <body>
    <h1>This is a page</h1>
    <p>a very simple page</p>
  </body>
</html>
```



# CSS (Cascading Style Sheet)

---

- Meaning of CSS
- What can it do
- Syntax
- Where can I write the code
- Accessing the tag
- Animations

# 1 Syntax to write CSS

**Selectors**  
The element(s) on which the style should be applied

**Property and its value**  
This is the actual style to be applied to the element(s)



# 3 Selectors and their syntax

Basic Selectors	Combinators	Pseudo Selectors
elementname	selectorA + selectorB Adjacent sibling	:active
.classname	selectorA ~ selectorB General sibling	:hover
#idname	parent > child Direct child	:visited
[attr=value]	parent descendant Descendent	:focus
*		

# 2 3 places to write CSS

- (A) Inline styles
- ```
<element style="property: value;">
```
- (B) In the <style> element
- ```
<head>  
....<style>  
..... selectors { property: value; }  
....</style>  
</head>
```
- (C) In a dedicated file (style.css)  
& refer that file via the <link> element
- ```
<head>  
....<link rel="stylesheet"  
..... href="style.css" />  
</head>
```

# 4 Common CSS properties (by group)

**TEXT:**

- color
- font
- font-family
- font-size
- font-weight
- letter-spacing
- line-height
- text-align
- text-decoration
- text-indent
- text-transform
- vertical-align

**LIST:**

- list-style
- list-style-image
- list-style-position
- list-style-type

**BACKGROUND:**

- background
- background-attachment
- background-color
- background-image
- background-position
- background-repeat

**DISPLAY:**

- display
- float
- clear
- overflow
- visibility

**OTHER:**

- cursor

**margin**

**border**

**padding**

**content**

**BOX:**

- border
- border-color
- border-style
- border-width
- height
- margin
- padding
- width
- box-sizing

**POSITION:**

- position
- top
- bottom
- left
- right
- z-index

# Style the page

---

```
<style>  
  h1 {  
    color: red;  
  }  
</style>
```



# Lets talk about javaScript

---

- Syntax
- Variables / Constants & datatypes
- Conditional Statements
- Loops
- Arrays
- Functions

## 1 Seven (7) Types

Six Primitive Types

1. String  
"Any text"
  2. Number  
123.45
  3. Boolean  
true or false
  4. Null  
null
  5. Undefined  
undefined
  6. Symbol  
Symbol('something')
- 
7. Object
    - Array  
[1, "text", false]
    - Function  
function name() { }

## 2 Basic Vocabulary

### Variable

A named reference to a value is a variable.

### Operator

Operators are reserved-words that perform action on values and variables. Examples: + - = \* in === typeof != ...

### Statement

A group of words, numbers and operators that **do a task** is a statement.

```
var a = 7 + "2";
```

**Note:** var, let & const are all valid keywords to declare variables. The difference between them is covered on page 7 of this cheatsheet.

### Keyword / reserved word

Any word that is part of the vocabulary of the programming language is called a keyword (a.k.a reserved word). Examples: var = + if for...

### Expression

A reference, value or a group of reference(s) and value(s) combined with operator(s), **which result in a single value**.

## 3 Object

An object is a data type in JavaScript that is used to store a combination of data in a simple key-value pair. That's it.

### Key

These are the keys in user object.

```
var user = {  
  name: "Aziz Ali",  
  yearOfBirth: 1988,  
  calculateAge: function(){  
    // some code to calculate age  
  }  
}
```

### Value

These are the values of the respective keys in user object.

### Method

If a key has a function as a value, it's called a method.

## 4 Function

A function is simply a bunch of code bundled in a section. This bunch of code ONLY runs when the function is called. Functions allow for organizing code into sections and code reusability.

Using a function has ONLY two parts. (1) Declaring/defining a function, and (2) using/running a function.

### Name of function

That's it, it's just a name you give to your function. Tip: Make your function names descriptive to what the function does.

### Return (optional)

A function can optionally spit-out or "return" a value once it's invoked. Once a function returns, no further lines of code within the function run.

### Invoke a function

Invoking, calling or running a function all mean the same thing. When we write the function name, in this case `someName`, followed by the brackets symbol `()` like this `someName()`, the code inside the function gets executed.

```
// Function declaration / Function statement
function someName(param1, param2){

    // bunch of code as needed...
    var a = param1 + "love" + param2;
    return a;
}

// Invoke (run / call) a function
someName("Me", "You")
```

### Parameters / Arguments (optional)

A function can optionally take parameters (a.k.a arguments). The function can then use this information within the code it has.

### Code block

Any code within the curly braces `{ ... }` is called a "block of code", "code block" or simply "block". This concept is not just limited to functions. "if statements", "for loops" and other statements use code blocks as well.

### Passing parameter(s) to a function (optional)

At the time of invoking a function, parameter(s) may be passed to the function code.



## 5 Vocabulary around variables and scope

```
var a;
```

### Variable Declaration

The creation of the variable.

```
a = 12;
```

### Variable Initialization

The initial assignment of value to a variable.

```
a = "me";
```

### Variable Assignment

Assigning value to a variable.

```
console.log(a);  
var a = "me";
```

### Hoisting

Variables are declared at the top of the function automatically, and initialized at the time they are run.

### Scope

The limits in which a variable exists.

#### Global scope

The outer most scope is called the Global scope.

#### Functional scope

Any variables inside a function is in scope of the function.

#### Lexical Environment (Lexical scope)


The physical location (scope) where a variable or function is declared is its lexical environment (lexical scope).

#### Rule:

(1) Variables in the outer scope can be accessed in a nested scope; But variables inside a nested scope CANNOT be accessed by the outer scope. (a.k.a private variables.)

(2) Variables are picked up from the lexical environment.

```
var a = "global";  
  
function first(){  
    var a = "fresh";  
  
    function second(){  
        console.log(a);  
    }  
}
```



### Scope chain

The nested hierarchy of scope is called the scope chain. The JS engine looks for variables in the scope chain upwards (it its ancestors, until found)

## 6 Operators

Full list of JavaScript operators <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators>

Operators are reserved-words that perform action on values and variables.

### Arithmetic

.. + .. Add  
.. - .. Subtract  
.. \* .. Multiply  
.. / .. Divide  
.. % .. Remainder  
.. \*\* .. Exponential

### Assignment

.. = .. Assign value  
.. += .. Add then assign  
.. -= .. Subtract then assign  
.. \*= .. Multiply then assign

### Logical

.. || .. Or  
.. && .. And

### Equality

.. === .. Equality  
.. == .. Equality with coercion

### Conversion

+ .. Convert to number  
- .. Convert to number then negate it  
! .. Convert to boolean then inverse it

### Relational / Comparison

.. >= .. Greater than or equal to  
.. <= .. Less than or equal to  
.. != .. Not equal after coercion  
.. !== .. Not equal

### Increment / Decrement

.. ++ Postfix increment  
.. -- Postfix decrement

++ .. Prefix increment  
-- .. Prefix decrement

### Others

typeof ..  
.. instanceof ..  
(..) ..  
...spread-operator  
.  
..[..]  
new ..  
delete ..  
(.. ? .. : ..)

### Operator Precedence

Given multiple operators are used in an expression, the "Operator Precedence" determines which operator will be executed first. The higher the precedence, the earlier it will get executed.

### Operator Associativity

Given multiple operators have the same precedence, "Associativity" determines in which direction the code will be parsed.

See the **Operator Precedence and Associativity table** here:

<http://bit.ly/operatortable>

## 7 Coercion

When trying to compare different "types", the JavaScript engine attempts to convert one type into another so it can compare the two values.

### Type coercion priority order:

1. String
2. Number
3. Boolean

### Coercion in action

Does this make sense?

```
2 + "7"; // "27"  
true - 5 // -4
```



## 8 Conditional Statements

Conditional statements allow our program to run specific code only if certain conditions are met. For instance, let's say we have a shopping app. We can tell our program to hide the "checkout" button if the shopping cart is empty.

**If -else Statement:** Run certain code, "if" a condition is met. If the condition is not met, the code in the "else" block is run (if available.)

```
if (a > 0) {  
    // run this code  
} else if (a < 0) {  
    // run this code  
} else {  
    // run this code  
}
```

**Ternary Operator:** A ternary operator returns the first value if the expression is truthy, or else returns the second value.

```
(expression)? ifTrue: ifFalse;
```

**Switch Statement:** Takes a single expression, and runs the code of the "case" where the expression matches. The "break" keyword is used to end the switch statement.

```
switch (expression) {  
    case choice1:  
        // run this code  
        break;  
  
    case choice1:  
        // run this code  
        break;  
  
    default:  
        // run this code  
}
```

## 9 Truthy / Falsy

There are certain values in JavaScript that return true when coerced into boolean. Such values are called **truthy** values. On the other hand, there are certain values that return false when coerced to boolean. These values are known as **falsy** values.

### Truthy Values

true  
"text"  
72  
-72  
Infinity  
-Infinity  
{ }  
[ ]

### Falsy Values

false  
""  
0  
-0  
NaN  
null  
undefined



## 10 Loop Statements

Loops are used to do something repeatedly. For instance let's say we get a list of 50 blog posts from the database and we want to print their titles on our page. Instead of writing the code 50 times, we would instead use a loop to make this happen.

### For loop

```
for (initial-expression; condition; second-expression){  
  // run this code in block  
}
```

**Step 1:** Run the initial expression.

**Step 2:** Check if condition meets. If condition meets, proceed; or else end the loop.

**Step 3:** Run the code in block.

**Step 4:** Run the second-expression.

**Step 5:** Go to Step 2.

### While loop

```
while (i<3){  
  // run this code in block  
  i++;  
}
```

**Step 1:** If the condition is true, proceed; or else end the loop.

**Step 2:** Run the code in block.

**Step 3:** Go to Step 1.

### Do while loop

```
do {  
  // run this code in block  
  i++;  
} while (i<3);
```

**Step 1:** Run the code in block.

**Step 2:** If the condition is true, proceed; or else end the loop.

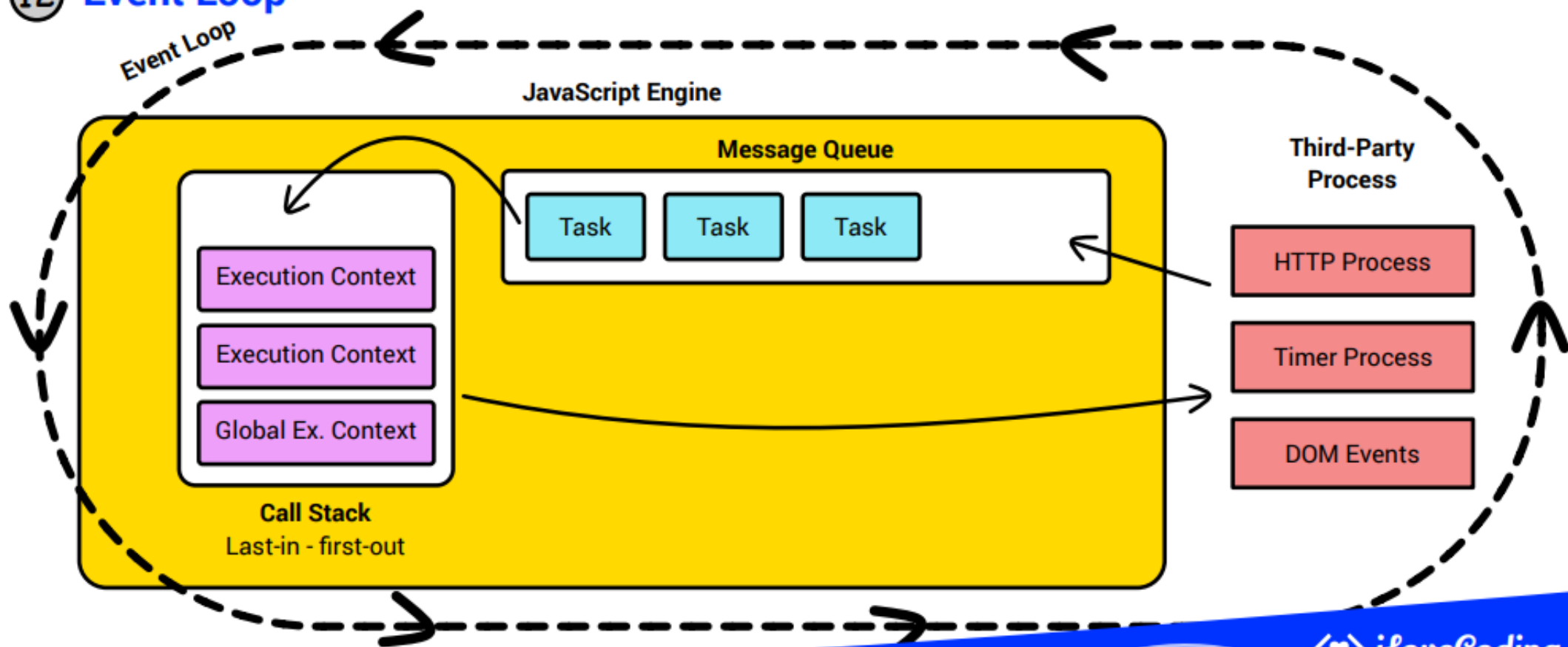
**Step 3:** Go to Step 1.

## 11 Ways to create a variable

There are 3 ways to create variables in JavaScript: **var**, **let** and **const**. Variables created with **var** are in scope of the function (or global if declared in the global scope); **let** variables are block scoped; and **const** variables are like **let** plus their values cannot be re-assigned.

```
var a = "some value"; // functional or global scoped
let b = "some value"; // block scoped
const c = "some value"; // block scoped + cannot get new value
```

## 12 Event Loop





## 13 Browser

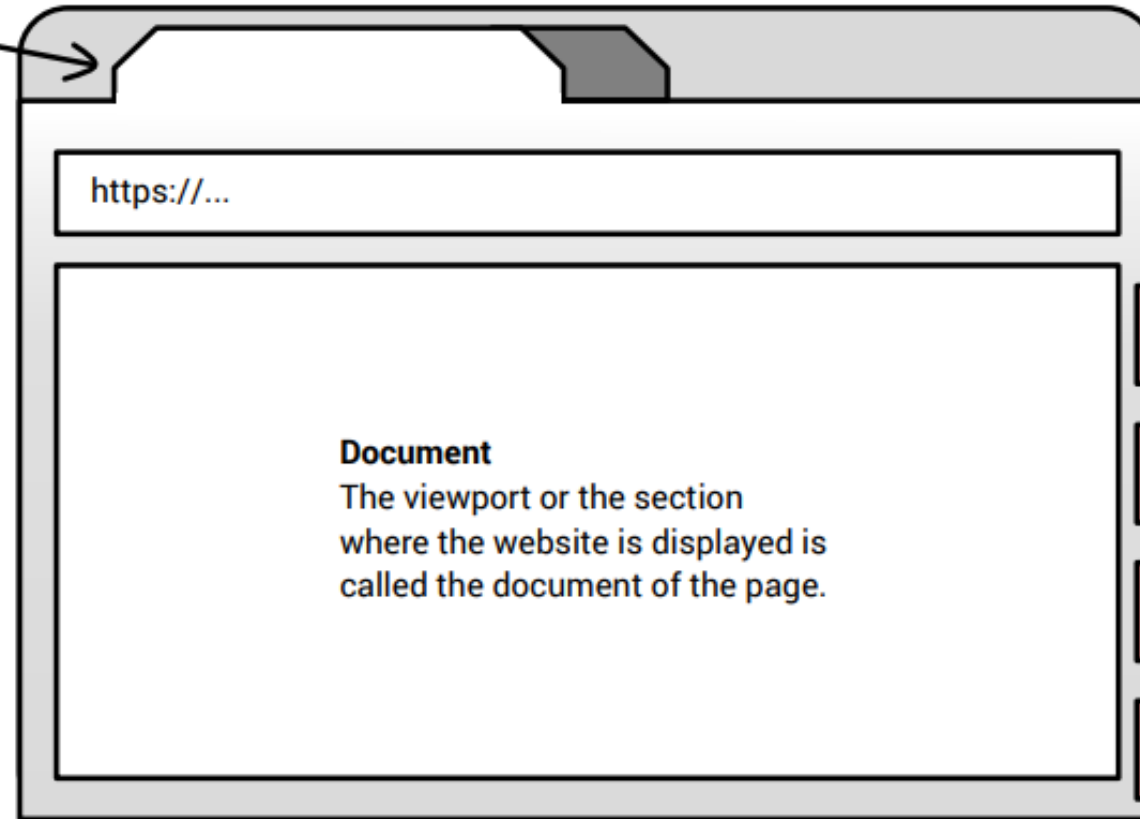
A web browser is a pretty advance piece of software which contains a lot of components. Many of these components are accessible to a web developer, so we can create complex web apps. At the same time a lot of components are kept out of reach of the web developer for security purposes. For instance, we as web developers can get access to the user's location, but we cannot get access to the user's saved passwords or browsing history. **Let's see below how a browser is structured:**

### Window

Each tab of a browser is considered the window.

This is the outer most container that a web-app can access.

Notice: A website opened in one tab CANNOT access the window object of another tab. Pretty cool right?



### Document

The viewport or the section where the website is displayed is called the document of the page.

The browser contains a lot of components that a Front-End Developer may need, such as **Navigator**, **JavaScript Engine** and **Dev Tools**.


Navigator

HTML / CSS Processor






JavaScript Engine


Dev Tools








# Ahmed Abubakr




 Home

 About





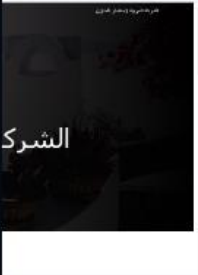
 Portfolio

 Services

 Contact

works and you can get more when you contact me

AKAR EDUCATION HEKAYAT SALMA



Elements Console Sources Network

```
<body inmaintabuse="1" data-aos-easing="ease-in-out-back" data-aos-duration="1000" data-aos-delay="0" class="mobile-nav-active"> == $0
```

html body.mobile-nav-active

Styles Computed Layout Event Listeners DOM Breakpoints Properties Accessibility

Filter

element.style {

.mobile-nav-active {

body {

body {

Console Issues

☐ Group by kind ☒ Include third-party cookie issues

9 5 2

An element doesn't have an autocomplete attribute

Content Security Policy of your site blocks the use of 'eval' in JavaScript

Cookie sent in cross-site context will be blocked in future Chrome versions

Incorrect use of <label for=FORM ELEMENT>

## 14 DOM - Document Object Model

### Query/Get Elements

```
// Preferred way:
document.querySelector('css-selectors')
document.querySelectorAll('css-selectors', ...)

// Old ways, and still work:
document.getElementsByTagName('element-name')
document.getElementsByClassName('class-name')
document.getElementById('id')
```

### Create / clone Element

```
document.createElement('div')
document.createTextNode('some text here')
node.cloneNode()
node.textContent = 'some text here'
```

### Add node to document

```
parentNode.appendChild(nodeToAdd)
parentNode.insertBefore(nodeToAdd, childNode)
```

### Get Element Details

```
node.nextSibling
node.firstChild
node.lastChild
node.parentNode
node.childNodes
node.children
```

### Modify Element

```
node.style.color = 'red'
node.style.padding = '10px',
node.style.fontSize = '200%'

node.setAttribute('attr-name', 'attr-value')
node.removeAttribute('attr-name')
```

### Get and Modify Element Class

```
node.classList
node.classList.add('class-name', ...)
node.classList.remove('class-name', ...)
node.classList.toggle('class-name')
node.classList.contains('class-name')
node.classList.replace('old', 'new')
```

### Remove Node

```
parentNode.removeChild(nodeToRemove)
// Hack to remove self
nodeToRemove.parentNode.removeChild(nodeToRemove)
```

### Events

```
node.addEventListener('event-name', callback-function)
node.removeEventListener('event-name', callback-function)
```

List of Events: <https://developer.mozilla.org/en-US/docs/Web/Events>  
or google "Mozilla event reference"

### What is a "Node"? (in the context of DOM)

**Node:** Every item in the DOM tree is called a node. There are two types of node - A text node, and an element node:

**Text Node:** Node that has text.

**Element Node:** Node that has an element.

**Child Node:** A node which is a child of another node.

**Parent Node:** A node which has one or more child.

**Descendent Node:** A node which is nested deep in the tree.

**Sibling Node:** A node that share the same parent node.

# Script it

---

```
<script>  
    document.write("hi");  
</script>
```



# Example



Lets be web developers and create  
a website

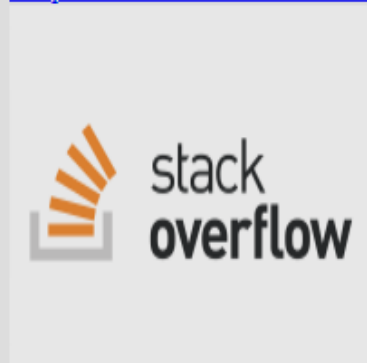


## first div

image without title and link without target

your img & link

[simple link to facebook without outer target](https://www.facebook.com/)



username

password

[link go to second div](#)

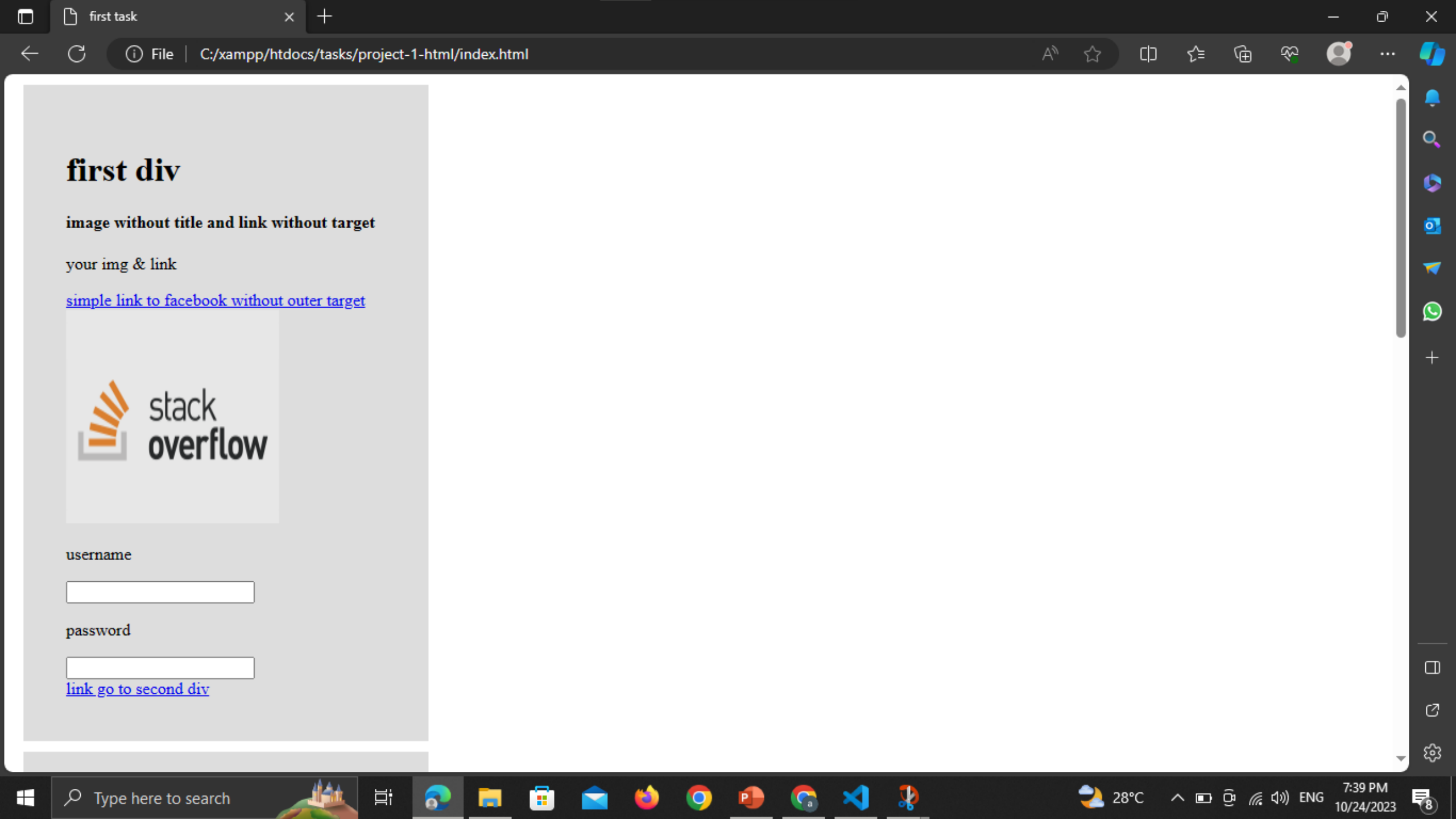
```
.first_div,
.second_div {
  width: 300px;
  margin: 10px;
  padding: 40px;
  background: #ddd;
  /* display: inline-block; */
}
```

```
<!-- we type the code here -->
<!-- start first div -->
<div class="first_div" id="fir">
  <h1>first div</h1>
  <!-- the main heading repeated again down -->
  <h4>image without title and link without target</h4>
  <!-- the second heading repeated again down -->
  <p>your img & link</p>
  <!-- paragraph about what is inside -->
  <a href="https://www.facebook.com/" target=""
  >simple link to facebook without outer target</a>
  <br />
  <br />

  <!-- sign in form -->

  <!-- start -->
  <p>username</p>
  <input type="text" name="username" value="" />
  <p>password</p>
  <input type="password" name="pass" value="" /><br />
  <!-- end -->

  <a href="#sec">link go to second div</a>
</div>
```



## first div

image without title and link without target

your img & link

[simple link to facebook without outer target](#)



username

password

[link go to second div](#)



# First Task

My name is abuBakr

i am programer

my hope is to be the gtrest programmer

My skills are :

1. html & html5

2. css & css3

3. javascript:

- jquery

- bom

- dom

4. Bootstrap

# Third Task

type the first number

type here ...

type the second number

type here ...

the plus of them is :

add

FRONT END



BACK END



comic.browserling.com



# The End

---

in the end I hope you understood all I said contact on :



<https://www.facebook.com/abobakr143>



<https://wa.me/201113284597>