

# CS-391

<HTML/>



Web begins with: <HTML/>

What Does **HTML** Stand For? Hypertext Markup Language

What is **Hypertext**? A text that contains links to other texts, images, videos, or media

What Is a **Markup Language**? A computer language that uses **tags** to generate content within a document

What are **HTML Tags**? Predefined content formatters in an HTML document

So, instead of creating websites, one-pixel at a time, you can use **HTML tags** to generate formatted content, much more efficiently.

```
<html> <head> <title> <body> <h1> <h2> <h3> <h4> <h5> <h6> <p> <a> <img> <ul>
<ol> <li> <table> <tr> <td> <div> <span> <form> <input> <button> <select> <option>
<textarea> <label> <iframe> <script>
```

# Introduction to HTML

<HTML/>



Another example:

You are hired by a famous chef to create a simple HTML page for one of his famous recipes. He wants you to write the name of his dish, in bold letters, on top of the page and then list the ingredients of the dish right below it. Then he wants you to chronologically, list the procedural steps of preparing this dish. He wants his recipe online so that his fans could all access it and enjoy it.

Something like this:

A screenshot of a Firefox browser window titled "ex-2 | lec-1". The URL is "localhost:63342/test/lec-1-ex-2/index.html?\_jtt=kttnpsqo76fl6pdh". The page content is:

## Ratatouille

### Ingredients

- 2 eggplants
- 6 roma tomatoes
- 2 yellow squashes
- 2 zucchinis

### Preparation

1. Preheat the oven for 375°F (190°C).
2. Using a sharp knife or a mandoline, slice the eggplant, tomatoes, squash, and zucchini into approximately  $\frac{1}{16}$ -inch (1-mm)-thick rounds, then set aside.
3. Make the herb seasoning: In a small bowl, mix together the basil, garlic, parsley, thyme, salt, pepper, and olive oil. Spoon the herb seasoning over the vegetables.
4. Cover the pan with foil and bake for 40 minutes. Uncover, then bake for another 20 minutes, until the vegetables are softened.
5. Serve hot as a main dish or side. The ratatouille is also excellent the next day—cover with foil and reheat in a 350°F (180°C) oven for 15 minutes, or simply microwave to desired temperature.
6. Enjoy!

```
<h1>Ratatouille</h1>
<h3>Ingredients</h3>
<ul>
  <li>2 eggplants</li>
  <li>6 roma tomatoes</li>
  <li>2 yellow squashes</li>
  <li>2 zucchinis</li>
</ul>
<h3>Preparation</h3>
<ol>
  <li>Preheat the oven for 375°F (190°C).</li>
  <li>Using a sharp knife...</li>
  <li>Make the herb...</li>
  <li>Cover the pan with...</li>
  <li>Serve hot as a main...</li>
  <li>Enjoy!</li>
</ol>
```

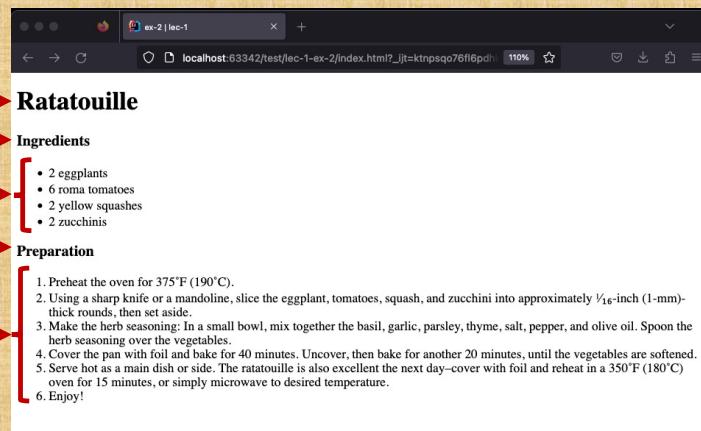
# Introduction to HTML

<HTML/>



Let's analyze the code:

```
<h1>Ratatouille</h1>
<h3>Ingredients</h3>
<ul>
  <li>2 eggplants</li>
  <li>6 roma tomatoes</li>
  <li>2 yellow squashes</li>
  <li>2 zucchinis</li>
</ul>
<h3>Preparation</h3>
<ol>
  <li>Preheat the oven for 375°F (190°C).</li>
  <li>Using a sharp knife...</li>
  <li>Make the herb...</li>
  <li>Cover the pan with...</li>
  <li>Serve hot as a main...</li>
  <li>Enjoy!</li>
</ol>
```



The **heading** tags range from 1 to 6:    <h1></h1>....<h6></h6>

Some tags are **paired** tags:    <h1></h1>    Some tags are **stand-alone** tags:    <hr>

Some tags must be **nested** in other tags:    <li></li>

<h1></h1>    Top-level Heading

<h3></h3>    Third-level Heading

<ul>
 <li></li>
 <li></li>
 <li></li>
 <li></li>
</ul>

Un-ordered List    List Items

<ol>
 <li></li>
 <li></li>
 <li></li>
 <li></li>
</ol>

Ordered List    List Items

# Introduction to HTML

<HTML/>



Let's analyze the webpage:

A screenshot of a web browser window titled "ex-2 | lec-1". The URL is "localhost:63342/test/lec-1-ex-2/index.html?\_jt=kttnpsqo76fl6pdh". The page content is a recipe for Ratatouille, divided into "Ingredients" and "Preparation" sections. The "Ingredients" section lists: 2 eggplants, 6 roma tomatoes, 2 yellow squashes, and 2 zucchinis. The "Preparation" section provides a step-by-step guide from preheating the oven to serving.

<head></head>

Title tag

<body></body>



We are all covered in skin, head to toe. In websites HTML would be the skin.

<html></html>

Just like a skeleton, a website has a **head** and a **body**.

HTML Skeleton

# Introduction to HTML

<HTML/>



Same code but inside the **skeleton**:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title> ex-2 | lec-1 </title>
  </head>
  <body>
    <h1>Ratatouille</h1>
    <h3>Ingredients</h3>
    <ul>
      <li>2 eggplants</li>
      <li>6 roma tomatoes</li>
      <li>2 yellow squashes</li>
      <li>2 zucchinis</li>
    </ul>
    <h3>Preparation</h3>
    <ol>
      <li>Preheat the oven for 375°F (190°C).</li>
      <li>Using a sharp knife...</li>
      <li>Make the herb...</li>
      <li>Cover the pan with...</li>
      <li>Serve hot as a main...</li>
      <li>Enjoy!</li>
    </ol>
  </body>
</html>
```

<!DOCTYPE html>

<html></html>

<head></head>

<body></body>

Identifies the version of the  
of the HTML code (**HTML-5**).

The **lang** attribute, identifies  
the language of the content.

The **skeleton (boilerplate)** by itself:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title> ex-2 | lec-1 </title>
  </head>
  <body>
  </body>
</html>
```

**Meta** stands for metadata.  
<meta> tags provide  
additional info, (always the  
**1<sup>st</sup>** tag in the <head>).

The **charset** attribute,  
decodes characters and  
symbols.

“**UTF-8**” covers almost all  
characters and symbols.

The **title** tag should always have two parts, in a child–parent formation, and  
they must be separated, (always the last tag in <head>).

# Introduction to HTML

<HTML/>



Let's talk about **Attributes**:

Some HTML tags have **required attributes**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title> ex-2 | lec-1 </title>
  </head>
  <body>
  </body>
</html>
```

*Note*  
Stand-alone tags:

**<img src="">**

Paired tags:

**<a href=""> </a>**

Some HTML tags have **optional attributes**

```
<table border="">
  <tr>
    <th>Where</th>
    <th>When</th>
    <th>How</th>
  </tr>
  <tr>
    <td>India</td>
    <td>Last Year</td>
    <td>Plane</td>
  </tr>
  <tr>
    <td>Turkey</td>
    <td>2 years ago</td>
    <td>Plane</td>
  </tr>
</table>
```

Without border

Where	When	How
India	Last Year	Plane
Turkey	2 years ago	Plane

With border

Where	When	How
India	Last Year	Plane
Turkey	2 years ago	Plane

# Introduction to HTML

<HTML/>



Let's talk about the **Anchor** tag:

```
<a href="https://www.bu.edu/"> Boston University </a>
```

**href** is a required attribute of the **anchor** tag. Its value must start with **http://** or **https://**

```
<a href="https://www.bu.edu/" target="_blank"> Boston University </a>
```

In **External Navigation**, if you want the link to be opened in a new page you must add the **target attribute**, with **\_blank** as its value (an optional attribute).

A screenshot showing a code editor on the left and a browser window on the right. The code editor displays the following HTML code:

```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <meta charset="UTF-8">
5          <title>child | parent</title>
6      </head>
7      <body>
8          <a href="https://www.bu.edu/"> Boston University </a>
9      </body>
10 </html>
```

The browser window shows the resulting page with a single link: "Boston University".

# Introduction to HTML

<HTML/>



More on Anchor tags:

```
<a href="https://www.google.com/">Google</a>
```

**Linked text** is the text between the `<a>` and the `</a>` tags.

Your **linked text** impacts the **security**, **usability**, **accessibility**, and **findability** of your website.

## Security

Make sure your **linked text** accurately describes where the link will go when clicked on.

This is to build credibility and to maintain your users' trust.

Phishing emails do the opposite.

## Findability

Search engines use **linked text** to help determine what the page you are linking to is about. Good **linked text** on your pages improves the findability of the pages.

## Rules of Good Link Text

1. **Accurately describe the destination of the link.**
2. **Make it as long as necessary, but not too long.**
3. **Avoid uninformative phrases (e.g. "click here")**
4. **Avoid using URLs as link text unless they are very short and easily understood.**
5. **If possible, put the most important information first.**

## Usability

People skim through web pages by jumping from link to link. Clear **linked text** gives a good idea of what a page is about. Ambiguous **linked texts** (e.g., "click here") do not.

## Accessibility

Screen readers provide the ability to tab from link to link. Many screen readers provide a way to see only the links on a page. better **linked text** improves the usefulness of these tools.

# Introduction to HTML

<HTML/>



More on Anchor tags:

Anchor tags can be used for **in-page navigation**

Example:

Let say you have a webpage with lots and lots of content, and you don't want to use the scroll bar to navigate up and down continuously.

You can add an **id** attribute at the destination:

```
<h2 id="dest">The page start here</h2>
```

And reference it in an anchor tag (with a #).

```
<a href="#dest">Go back up</a>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>child | parent</title>
  </head>
  <body>
    <h2 id="dest">The page start here</h2>
    <a href="#dest">Go back up</a>
  </body>
</html>
```



# Introduction to HTML

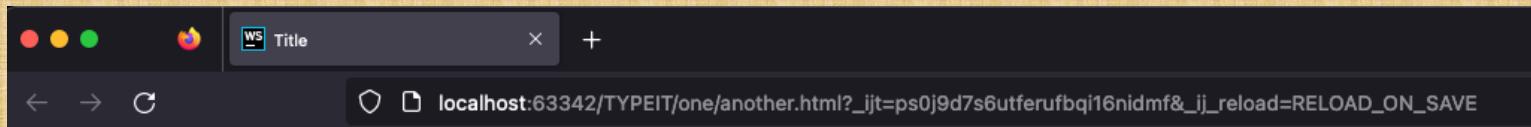
<HTML/>



Let's talk about the **Paragraph** tag:

The `<p></p>` tag is a paired tag and it should be used for textual content.

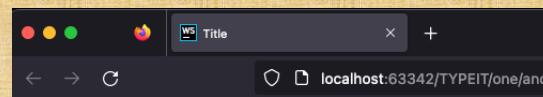
```
<p>Chinchillas eat many different things like pellets, hay, grass, and leafy greens.</p>
```



Chinchillas eat many different things like pellets, hay, grass, and leafy greens.

You can nest a `<u></u>` tag inside the `<p></p>` tag to underline a section or word.

```
<p><u>Chinchillas</u> eat many different things</p>
```



Chinchillas eat many different things

# Introduction to HTML

<HTML/>



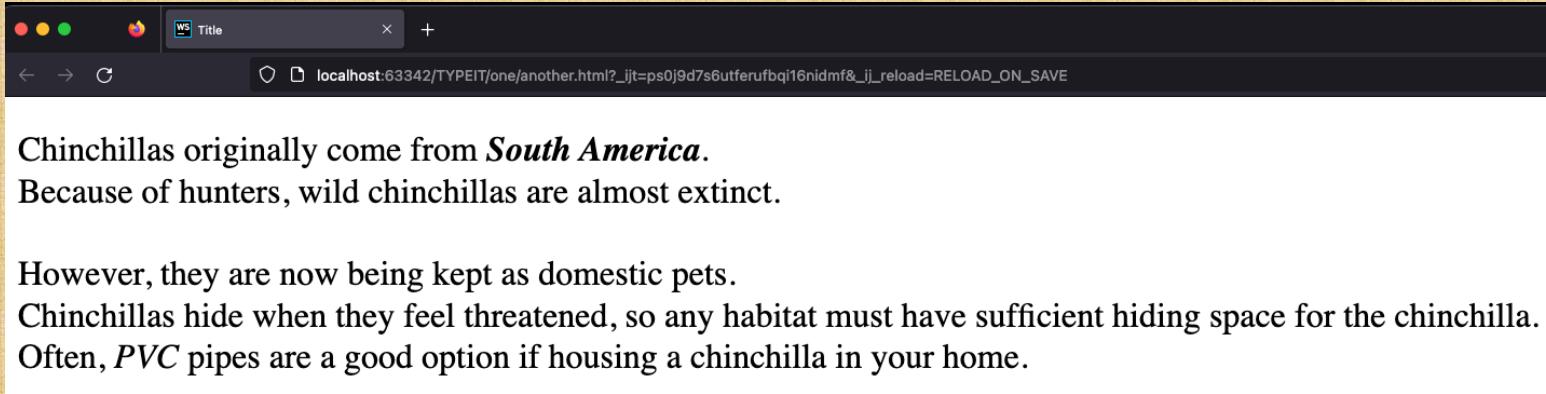
More about the <p></p> tag:

to emphasize on the importance of certain words, you can nest that section in an <em></em> or a <strong></strong> tag. You can also nest a <br> tag to brake a line if its getting to long.

```
<p>
```

```
Chinchillas originally come from <em><strong>South America</strong></em>. <br>Because of hunters, wild chinchillas  
are almost extinct. <br><br> However, they are now being kept as domestic pets. <br>Chinchillas hide when they feel  
threatened, so any habitat must have sufficient hiding space for the chinchilla. <br>Often, <em>PVC</em> pipes are  
a good option if housing a chinchilla in your home.
```

```
</p>
```



The screenshot shows a web browser window with the title bar "WS Title". The address bar displays "localhost:63342/TYPETIT/one/another.html?\_jt=ps0|9d7s6utferufbq|16nidmf&\_ji\_reload=RELOAD\_ON\_SAVE". The main content area of the browser shows the following text:

Chinchillas originally come from ***South America***.  
Because of hunters, wild chinchillas are almost extinct.

However, they are now being kept as domestic pets.  
Chinchillas hide when they feel threatened, so any habitat must have sufficient hiding space for the chinchilla.  
Often, *PVC* pipes are a good option if housing a chinchilla in your home.

# Introduction to HTML

<HTML/>



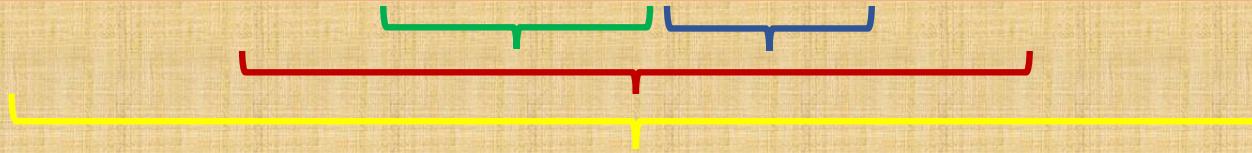
Nesting Technique:

You have already witnessed **nesting**, but you should be careful with this technique.

Check the order

Close the child tag before the parent tag, but also remember that the parent tag must be closed as well.

```
<p>Einstein's <strong><em>E=MC</em><sup>2</sup></strong> formula...</p>
```



# Introduction to HTML

<HTML/>



Indentation:

In HTML coding **indentation** is not mandatory, but its conventional

Multiple child tags should be **indented**

Each `<li></li>` tag has one nested `<a></a>` tag, so indentation is not required in the `<li></li>` tag.

```
<ul>
  <li> <a href="#">Home</a> </li>
  <li> <a href="#">Films</a> </li>
  <li> <a href="#">Photography</a> </li>
</ul>
```

But the `<ul></ul>` tag has three nested `<li></li>` tags, so indentation is required in the `<ul></ul>` tag.

**Indentation** increases the readability and the Maintainability of your code

Other tags may be **indented**, if doing so would improve the readability and the Maintainability as well

Failure to indent is considered as a **conventional error**

# Introduction to HTML

<HTML/>



Commenting:

The `<!-- -->` tag is stand-alone tag and its used for two purposes.

Other developers may end up working with your code. It would be wise to guide them through your code using the `<!-- -->` tag.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>lab-1 | part-a</title>
</head>
<body>
    <!--Main content of the website-->
    <!--Starts here-->
        <h1>Jack's first website</h1>
        <h3>Hi my name is Jack.</h3>
        <p>I am a fictional character.</p>
    <!--Ends here-->
</body>
</html>
```

When you code is not finished, it should not be used, but you don't want to delete the code either.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>lab-1 | part-a</title>
</head>
<body>
    <h1>Jack's first website</h1>
    <!-- <h3>Hi my name is Jack.</h3> -->
    <!-- <p>I am a fictional character.</p> -->
</body>
</html>
```

# Introduction to HTML

<HTML/>



Last week you learned about **HTML's Skeleton**:

```
<!DOCTYPE html>  
  
<html lang="en">  
  
    <head>  
        <meta charset="UTF-8">  
        <title>Child | Parent</title>  
    </head>  
  
    <body>  
    </body>  
  
</html>
```

You also learned that the **H** in **HTML** stands for **Hypertext**

"A software system that links topics on the screen to related information and graphics, which are typically accessed by a point-and-click method."  
- *Oxford Dictionary*

Tags that go inside the **<head></head>** are mainly declarative

<meta>, <title></title>, <link>, <script></script>, etc.

Tags that go inside the **<body></body>** are mainly content based

<hr>, <a></a>, <br>, <p></p>, <ul></ul>, <ol></ol>, <table></table>, etc.

Inside the **<body></body>**, conventionally, other tags are grouped in to 4 deferent sections:

- Header
- Navigation
- Main-content
- Footer

# Introduction to HTML

<HTML/>

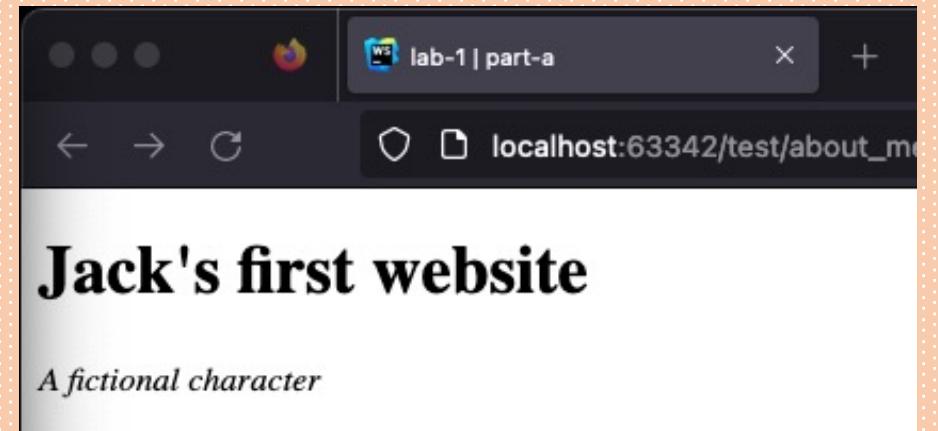


## Header:

The Header section has its own designated tag and it is called `<header></header>`

Inside the `<header></header>` tag you should give a title to your website (with an `<h1></h1>` tag), and a subtitle, (with a `<p></p>` tag)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>lab-1 | part-a</title>
  </head>
  <body>
    <!--Header starts here-->
    <header>
      <h1>Jack's first website</h1>
      <p><em>A fictional character</em></p>
    </header>
    <!--Header ends here-->
  </body>
</html>
```



The `<header></header>` tag helps search engines figure out what your website is about. It improves the Findability of your website.

# Introduction to HTML

<HTML/>



## Navigation:

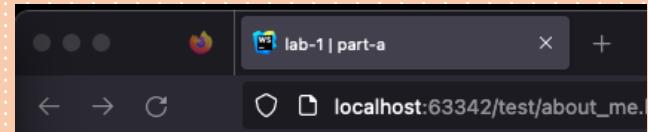
In web, we use `<a></a>` tags to create Links, and they are used for:

- **Navigating** from one website to another (**External Navigation**).
- **Navigating** in the same page, like scrolling up and down, (**In-page Navigation**).
- **Navigating** from one webpage to another webpage inside the same website (**Internal Navigation**).

Most Internal Navigations links should be grouped in a `<nav></nav>` tag.

`<nav></nav>` tag is a paired tag, and is conventionally parent to a `<ul></ul>` tag that contains navigational anchor `<a></a>` tags.

```
<!--Navigation starts here-->
<nav>
  <ul>
    <li><a href="">Jack's Profile</a></li>
    <li><a href="">Jack's Pics</a></li>
    <li><a href="">Jack's Friends</a></li>
    <li><a href="">Jack's subscriptions</a></li>
  </ul>
</nav>
<!--Navigation ends here-->
```



## Jack's first website

*A fictional character*

- [Jack's Profile](#)
- [Jack's Pics](#)
- [Jack's Friends](#)
- [Jack's subscriptions](#)

# Introduction to HTML

<HTML/>

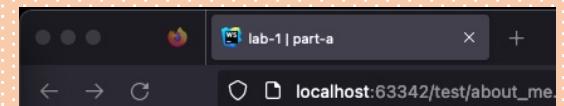


## Main-Content:

Most important content of a webpage should be displayed in the Main-content area of that page, inside the `<main></main>` tag.

The `<main></main>` tag should conventionally include a title in form of a **heading** `<h1></h1>`, `<h2></h2>`, `<h3></h3>`, etc.

```
<!--Main-Content starts here-->
<main>
    <!--Main-Content's title-->
    <h3>Jack's Pics</h3>
    
    
    <br>
    
    
</main>
<!--Main-Content ends here-->
```



## Jack's first website

*A fictional character*

- [Jack's Profile](#)
- [Jack's Pics](#)
- [Jack's Friends](#)
- [Jack's subscriptions](#)

### Jack's Pics



# Introduction to HTML

<HTML/>



## Footer:

The **<footer></footer>** section of a webpage is conventionally reserved for the **copyrights** statement.

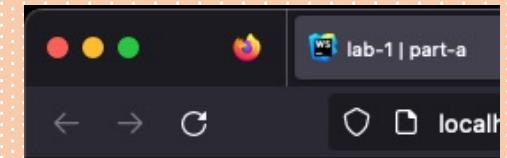
```
<!--Footer starts here-->  
  
  <footer>  
    <p>All rights reserved: <a href="">Credits</a> ©</p>  
  </footer>  
  
<!--Footer ends here-->
```

The **copyrights** statement conventionally contains a link to the Credits page.

Any type of work that requires a credit statement must be linked to its credit statement in the Credits page

The **copyrights** statement and a Credits page, in the **<footer></footer>** are very important, both from a **legal** and a **conventional** perspective.

The **<footer></footer>** may also contain less important navigational links, (when there is not enough space).



## Jack's first website

*A fictional character*

- [Jack's Profile](#)
- [Jack's Pics](#)
- [Jack's Friends](#)
- [Jack's subscriptions](#)

### Jack's Pics



All rights reserved: [Credits](#) ©

# Introduction to HTML

<HTML/>



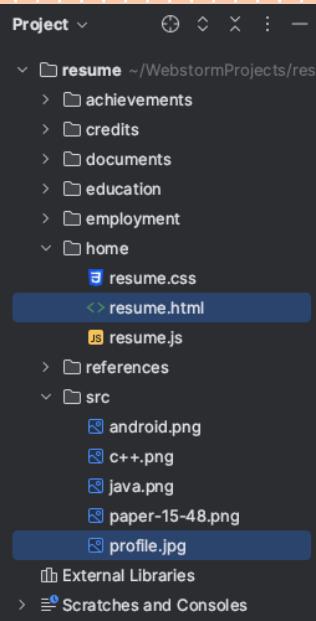
You have learned about the main parts of an **HTML** page like the **skeleton, header, navigation, main-content, footer**, etc.

So we will spend this week wrapping up on **HTML** and cover things we might have missed.

For example we have learned how to add **images** to our websites, but we have not talked about **videos** or **songs**.

Before we move on to **videos** and **songs**, let's wrap up on **images** first

`<img src="" alt="" />`



The `<img>` tag is a stand-alone tag, and it has 2 required attributes

`src=" "` will point to where you have saved the image in your project.

Given this project structure, if you wanted to use the **profile.jpg** in **resume.html**, what would be your `src=" "` attribute's value?

``

One dot means my folder; two means my parent folder

`..` (which would be resume).



# Introduction to HTML

<HTML/>



<img/>

**alt=""** has three purposes:

1. Displays a message, instead of the image, if and when the image fails to load (**Usability**).
2. The voice-over or screen-reader features, informs people who can not see, as to what the content of the image is, based on the value of the **alt** attribute (**Accessibility**).
3. Search engines also use the value of the alt attributes to show users images they had searched for (**SEO or Findability**).

More about **alt** attribute:

- Every img tag must have an alt attribute.
- For images that are purely **decorative** or have been **described in a caption** (text below the image), use null (**alt=""**).
- Refrain from using **image-text**, but if you did, alt attribute should include those words.
- Do not use "**image of ...**", "**graphic of ...**" in alt attribute. The screen reader will announce the presence of an image, so "image of" and "graphic of" are redundant.
- If the image is a **link** (image tag is nested in an anchor tag), alt attribute should indicate the destination of the link (copy the value of href in to alt).
- Do not use words like "**link to**" or "**click this image**" for alternative text of images that are used as links.



# Introduction to HTML

<img/>

Web browsers will only display images that are saved in these formats:

- **JPEG** (pronounced JAY-peg)
- **PNG** (pronounced PING)
- **GIF** (pronounced Jiff)
- **SVG** (pronounced "S" "V" "G")

Image formats such as Tiff (short for Tagged Image File Format) and BMP (Short for Bitmap) **cannot** be displayed by web browsers, (these formats are used for printing devices).

Each image format has certain characteristics that would make that format better or worse.

- Number of supported colors.
- Compression type (lossy or lossless)
- Supporting Transparency
- Supporting animations
- Being Scalable (infinite resolution)
- File size of the image (in Megabyte)

Check out this URL to learn more about these differences:

<https://visual.ly/community/infographic/computers/beginners-guide-image-formats-web>

# Introduction to HTML

<HTML/>



<img/>

## JPEG

- Stands for “Joint Photographic Experts Group”
- 16 million colors
- Compression is lossy
- Relatively small file size
- Best for photos



## PNG

- Stands for “Portable Network Graphics”
- 16 million colors
- Compression is lossless
- Supports transparency
- Large file size
- Good for drawings, logos, and charts regardless of the number of colors

# Introduction to HTML

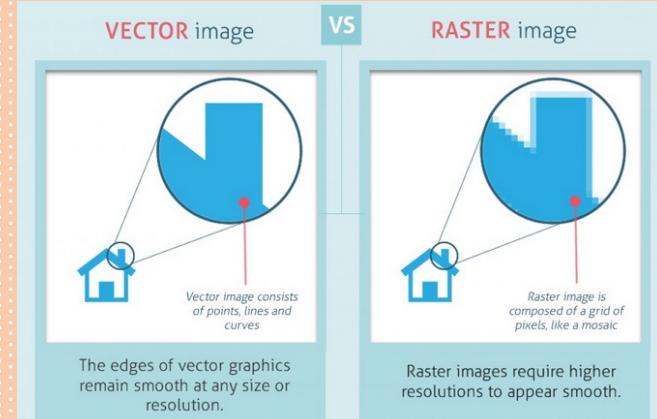
<HTML/>



<img/>

## SVG

- Stands for "Scalable Vector Graphics".
- Infinitely scalable resolution.
- Vector-based format, perfect for graphics of any size.
- Compression is lossless.
- Supports transparency and interactivity.
- Relatively smaller file size compared to bitmap formats.
- Ideal for logos, icons, illustrations, and interactive web graphics.
- Retains crisp quality on various display devices and sizes.



## GIF

- Stands for “Graphics Interchange Format”
- 256 colors
- Compression is lossless if 256 colors or less are used
- Supports transparency
- Supports animation
- Small file size
- Good for drawings, logos, and charts with less than 256 colors.

# Introduction to HTML

<HTML/>



<video></video>

Now let's talk about the **video** tag

<video></video>

The <video></video> tag is a paired tag, and it has 2 required attributes

<video src="" controls> </video>

Just like in <img> the **src=" "** attribute will point to the location in which the video has been saved, (inside the project folders).

The **controls** attribute has no values, but if you forget it, your video won't play

Only 3 video format are playable in the web, (MP4, WebM, and Ogg).



With **controls**

Browser	MP4	WebM	Ogg
Edge	YES	YES	YES
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES
Explorer	YES	NO	NO



Without **controls**

# Introduction to HTML

<HTML/>



<audio></audio>

Now let's talk about the **audio** tag

<audio></audio>

The <audio></audio> tag is a paired tag, and it has 2 required attributes

<audio src="" controls></audio>

Just like in <img/> the **src=" "** attribute will point to the location in which the **audio** has been saved, (inside the project folders).

The **controls** attribute has no values, but if you forget it, your **audio** won't play

Only 3 **audio** format are playable in the web, (MP3, WAV, and OGG).

Listen and enjoy :)

OUTPUT

▶ 0:00 / 0:02 ━━ ⏪ ⏹ ⏷

With **controls**

Browser	MP3	WAV	OGG
Explorer	YES	YES	YES
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

Listen and enjoy :)

Without **controls**

<audio><audio/>  
<video><video/>

# Introduction to HTML

Optional Attributes

<HTML/>



Attribute	Value	Description
<a href="#">autoplay</a>	autoplay	Specifies that the video/audio will start playing as soon as it is ready
<a href="#">controls</a>	controls	Specifies that video/audio controls should be displayed (such as a play/pause button etc).
<a href="#">height</a>	pixels	Sets the height of the video/audio player
<a href="#">loop</a>	loop	Specifies that the video/audio will start over again, every time it is finished
<a href="#">muted</a>	muted	Specifies that the audio output of the video/audio should be muted
<a href="#">poster</a>	URL	Specifies an image to be shown while the video/audio is downloading, or until the user hits the play button
<a href="#">preload</a>	auto metadata none	Specifies if and how the author thinks the video/audio should be loaded when the page loads
<a href="#">src</a>	URL	Specifies the URL of the video/audio file
<a href="#">width</a>	pixels	Sets the width of the video/audio player

Ex:

```
<video src="movie.mp4" width="320" height="240" controls autoplay muted> </video>
```

Ex:

```
<audio src="horse.ogg" controls autoplay loop> </audio>
```

<audio><audio/>  
<video><video/>

# Introduction to HTML

<HTML/>



Unfortunately managing **audio** and **video** in web is somewhat problematic.

Problem	Solution
<u>Resizing</u> the <b>video</b> player's window vertically will distort the video.	Always adjust the player's window using the <b>width=" "</b> attribute.
The <u>video format</u> may be one of approved formats, but it still won't play, because of its encoding.	<b>Use encoding software</b> to convert the video to another format.
<u>Audio auto-play</u> is deactivated in some browsers because of copyrights issues.	None.

These issues are only the tip of the iceberg, there are many other problems, (the explanation of which would exceed the scope of this class)

But, there is one solution that web developers could utilize to avoid all of these issues

<iframe><iframe/>

With **<iframe><iframe/>** developers can embed **Audio** and **Video** content in their websites without having to worry about formatting, resizing, copyrights issues (credit must be given), etc.

<iframe></iframe>

# Introduction to HTML

<HTML/>



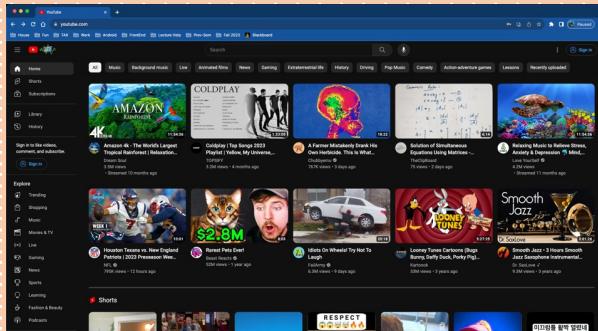
So, how does **embedding** work?

The steps are easy

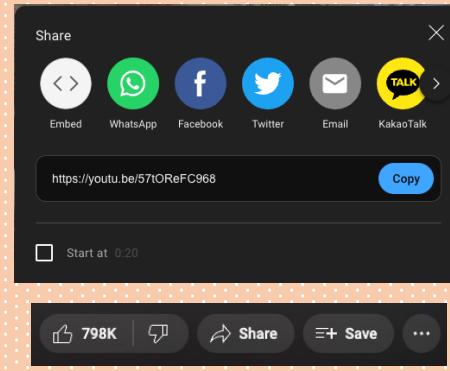
1. Go to the website and pick up an audiovisual content.
2. Click on the “Share” button.
3. Click on the “Embed” button.
4. Copy the provided html code and past it into your html file.

YouTube Example:

Chose a video



Click on Share and select Embed



Copy and paste it in your code

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/57
t0ReFC968" title="YouTube video
player" frameborder="0"
allow="accelerometer; autoplay;
clipboard-write; encrypted-media;
gyroscope; picture-in-picture; web-
share" allowfullscreen></iframe>
```



# Introduction to HTML

Block-level/inline

Okay, let's move on to another topic we missed previously

Sometimes **HTML tags** are referred to as Elements and web developers often use this term when they want to refer to a tag's **appearance**

For example you might hear or read online about **block-level elements** or **inline elements**

## block-level elements

Tags that occupy the entire width of the screen

- Starts on a new line.
- Forces the next element on a new line.

<p>  
<ul>  
<ol>  
<li>  
<table>  
<tr>  
<h1>...<h6>  
<hr>

BLOCK ELEMENTS EXPAND NATURALLY →



AND NATURALLY DROP BELOW OTHER ELEMENTS ↘



## inline elements

Tags that don't occupy the entire width of the screen.

- They **do not start** on a new line.
- **Do not force** the next element on a new line.

<img> <a> <strong> <em> <td> <th>

### Note:

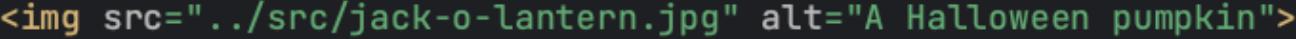
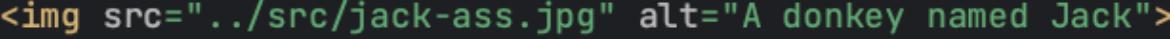
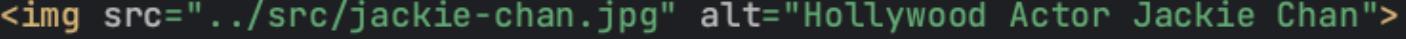
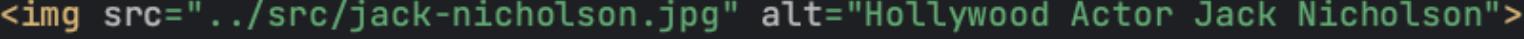
The **block-level** or **inline** behavior could be altered (changed) using **CSS**. This means that a block-level element could be turned into an inline element and vice versa.



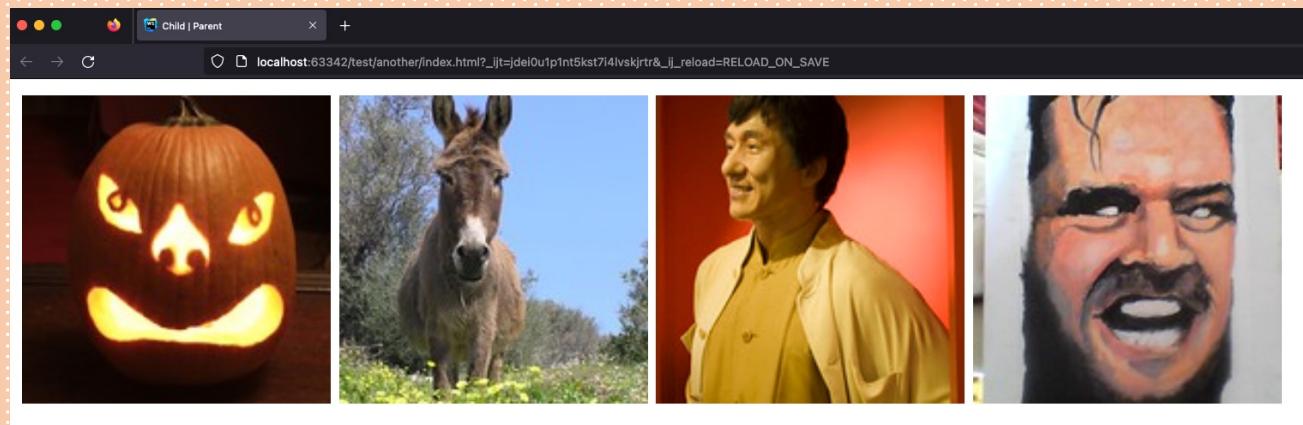
Block-level/inline

# Introduction to HTML

inline elements

```
<!--Inline Elements starts-->




<!--Inline Elements ends-->
```

Even though I have placed each image tag on its own line, the images ended up stacking up against each other



<HTML/>



## Block-level/inline

# Introduction to HTML

So, how did I get two images to move bellow? CSS?

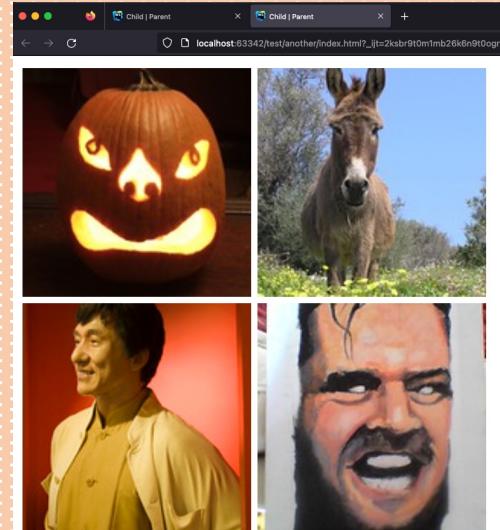
YES  NO

I placed `<br>` (an invisible **block-level element**) between them

```
<!--Inline Elements starts-->


<br>


<!--Inline Elements ends-->
```



## block-level elements

```
<!--Block-level Elements starts-->
<h1>T </h1> <h2>a </h2> <h3>y </h3> <h4>m </h4> <h5>a </h5> <h6>z </h6> <p>!</p>
<!--Block-level Elements ends-->
```

Even though I have stacked all of the tags above in one line, each and every one of them is occupying an entire line (**Block**) in the browser, and is pushing the next one down to the next line.

T  
a  
y  
m  
a  
z  
!

# Introduction to HTML



<div></div> tags are very handy. They can replace many other tags, but the W3 consortium advises developers to refrain from using the div tag when there could be a better suited tag.

*"The div element has no special meaning at all...Authors are strongly encouraged to view the div element as an element of last resort, for when no other element is suitable. Use of more appropriate elements instead of the div element leads to better accessibility for readers and easier maintainability for authors." W3C*

Starting next week we will begin our CSS portion of this class. To be able to style a group of tags in a certain manner, we would have to group them into a parent element. <div></div> tags would allow us to do so, but we should first make sure that there is no other parent tag that could help us achieve that.

For example <div></div> tags can be used instead of a Header, a Nav, a Main-content, or a Footer tag, to group other tags, but doing so would decrease your websites' **Accessibility**, and **Findability (SEO)**.

Navigation using the <nav></nav> tag

Example

Navigation using the <div></div> tag

```
<!--Navigation Start-->
<nav>
  <ul>
    <li><a href="resume.html">Home</a></li>
    <li><a href="../education/education.html">Education</a></li>
    <li><a href="../employment/employment.html">Employment</a></li>
    <li><a href="../achievements/achievements.html">Achievements</a></li>
    <li><a href="..//references/references.html">References</a></li>
    <li><a href="..//documents/documents.html">Documents</a></li>
  </ul>
</nav>
<!--Navigation End-->
```

```
<!--Navigation Start-->
<div>
  <ul>
    <li><a href="resume.html">Home</a></li>
    <li><a href="..//education/education.html">Education</a></li>
    <li><a href="..//employment/employment.html">Employment</a></li>
    <li><a href="..//achievements/achievements.html">Achievements</a></li>
    <li><a href="..//references/references.html">References</a></li>
    <li><a href="..//documents/documents.html">Documents</a></li>
  </ul>
</div>
<!--Navigation End-->
```

## Search Engine Optimization

So far you have learned how to create websites. Okay what's next? Well Creating a website is half of the battle, now you need to advertise it to your target audience. Placing an add though, could cost you lots of money, especially if your website would have to compete with existing giants, you may not be able to afford the competition.

So what is the solution?

SEO

"**A methodology of strategies, techniques and tactics used to increase the number of visitors to a website by obtaining a high-ranking placement in the search result page of a search engine (such as Google Search Engine).**" -Webopedia

And it won't cost you a penny

## Search Engine Optimization

So what are these **strategies, techniques and tactics**?

Well you are already familiar with some of them

You already know that the search engines acquire info from your website using the HTML tags. For example, if you used a **<div></div>** instead of a **<nav></nav>**, for your navigational menu, you would essentially confuse the search engine as to what the content of that tag is or should be.

So using the right tag for the right purpose is one of many **strategies** that you should take into account

In fact if you use incorrect tags too many times or you purposely use misleading tags, your website's ranking will be reduced. That means search engines would not show your website in their search results.

## Search Engine Optimization

Before we continue with **strategies, techniques** and **tactics**, we should figure out how search process works

When we search for something online, search engines would have to figure out what we are searching for.

- There are no pre-defined vocabulary of keywords
- Synonymy – there can be multiple words for the same thing
- Polysemy – one term can have multiple meanings
- Diversity of authors and searchers
- Web content is constantly changing
- Huge number of pages to index

To be able to show results efficiently (quickly and correctly), search engines constantly monitor websites, index them, and rank them. This process is known as the **Discovery Step**.

## Search Engine Optimization

In the **Discovery** process, small computer programs (called Bots or Spiders), which belong to search engines crawl the web, jumping from link to link, exploring, logging, and ranking each page.

Bots from Google, Bing, and many international search engines are constantly crawling the web.

In lab-0 you created a “.txt” file called **robots.txt** to prevent these searching bots from indexing your website or parts of your website. Today we are going to do the opposite.

While some parts of your website will remain private, we do want the bots to access and index other parts. To do so we need a new file called

### **sitemap.xml**

“A **Sitemap** is an XML file that lists URLs for a site along with additional metadata about each URL, so that search engines can more intelligently crawl the site.”—*sitemaps.org*

# sitemap

## Search Engine Optimization

So, how do we build a sitemap?

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset schemaLocation="https://www.sitemaps.org/schemas/sitemap/0.9 ">
  <url>
    <loc>https://www.resume.com/home</loc>
    <lastmod>2008-01-01</lastmod>
    <changefreq>weekly</changefreq>
    <priority>1</priority>
  </url>

  <url>
    <loc>https://www.resume.com/certs</loc>
    <lastmod>2008-12-23</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.8</priority>
  </url>

  <url>
    <loc>https://www.resume.com/credits</loc>
    <lastmod>2008-12-23</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.5</priority>
  </url>
</urlset>
```

The Sitemap must:

- Begin with an opening `<urlset>` tag and end with a closing `</urlset>` tag.
- Specify the namespace (protocol standard) within the `<urlset>` tag.
- Include a `<url>` entry for each URL, as a parent XML tag.
- Include a `<loc>` child entry for each `<url>` parent tag.

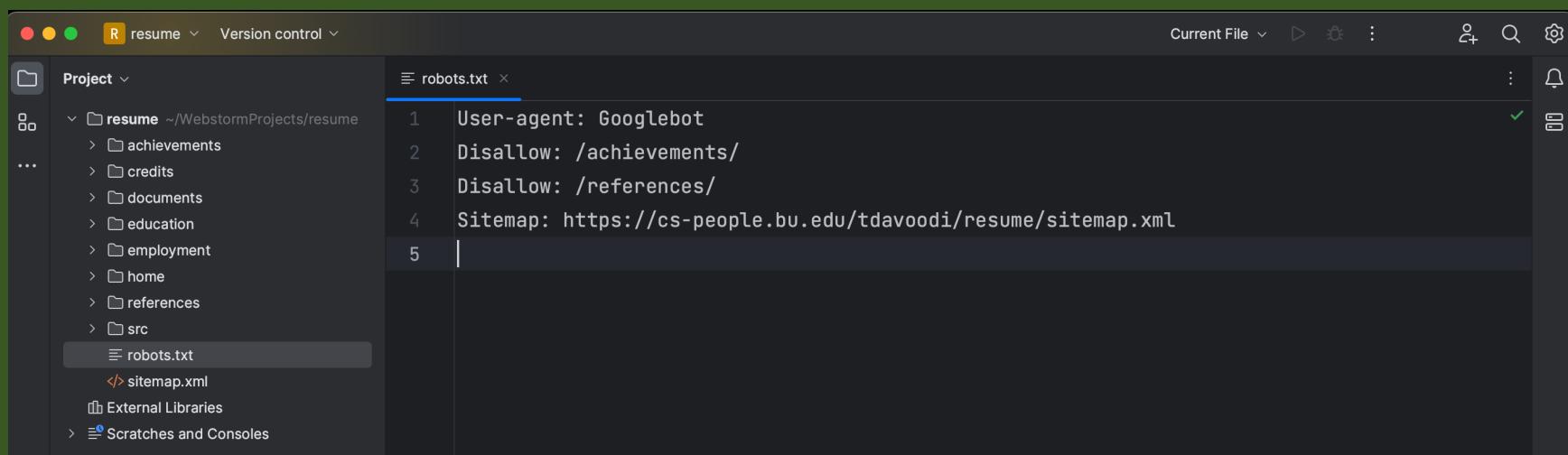
All other tags are optional.

sitemap

## Search Engine Optimization

The **sitemap.xml** file should be placed in the top-level domain (parent folder); this is where you saved your **robots.txt** file as well.

The **sitemap.xml**'s URL address must be pasted at the end of the **robots.txt** file



A screenshot of the WebStorm IDE interface. On the left, the Project tool window shows a project named 'resume' with several subfolders like 'achievements', 'credits', 'documents', etc. A 'robots.txt' file is selected. On the right, the code editor shows the contents of the 'robots.txt' file:

```
User-agent: Googlebot
Disallow: /achievements/
Disallow: /references/
Sitemap: https://cs-people.bu.edu/tdavoodi/resume/sitemap.xml
```

In this example, in my **robots.txt** file, I have disallowed Google's bots from parts of my website; while at the same time, with the **sitemap**'s URL, I have given those bots a road map to follow.

meta

## Search Engine Optimization

During the **discovery** step, another technique, that could potentially increase the ranking of your website, is the use of **meta-description** tag

```
<meta name="description" content="A website about...">
```

The **meta-description** tag, helps search engines figure out what your website is about.

```
<head>
  <meta charset="UTF-8">
  <meta name="description" content="A website showcasing Taymaz Davoodi's Online Resume">
  <title>Home | Resume</title>
</head>
```

Search engines use the value of the **content** attribute to populate a search result page for users, so the accuracy of your description is very important. If you fail to be accurate, your ranking will suffer.

## Search Engine Optimization

Unfortunately in the past, some **SEO engineers** have abused their knowledge of **SEO**.

For example in fall of 2010, department store JC Penney hired an unethical SEO firm to boost their search engine rankings. Links to JC Penney pages were embedded in hundreds of low-quality, unrelated websites (for a fee). JC Penney ranked #1 in dozens of categories for several months. Google took a manual corrective action, and ratings fell

Unethical SEO techniques are known as “**black-hat**” techniques.

A **black-hat** technique is using the meta-description tag to insert excessive or irrelevant content. This unethical approach aims to make a webpage appear in search results regardless of the user's actual query, undermining the integrity of search engine results.

To mitigate this issue, most search engines use the meta-description tag in conjunction with another tag (e.g., <title></title>) to ascertain meta-description's integrity.

meta

## Search Engine Optimization

Using Google, if you search for “Boston University”, you will end up with a list of websites that either belong to BU, are affiliated with BU, or have mentioned BU in their content

Now if you look for BU’s official website in that list, depending on what search engine you use, you will see something like this:

<title></title>



Boston University  
https://www.bu.edu › homepage-alt ::

Boston University: Homepage

<meta/>

Boston University is a leading private research institution with two primary campuses in the heart of Boston and programs around the world.

```
<title>Homepage | Boston University</title>
```

```
<meta
```

```
    name="description"
```

```
    content=
```

```
        "Boston University is a leading private research institution with two primary campuses in the heart of Boston and programs around the world."
```

```
>
```

SEO

## Search Engine Optimization

Okay, so far you have learned 2 SEO techniques

**sitemap.xml**

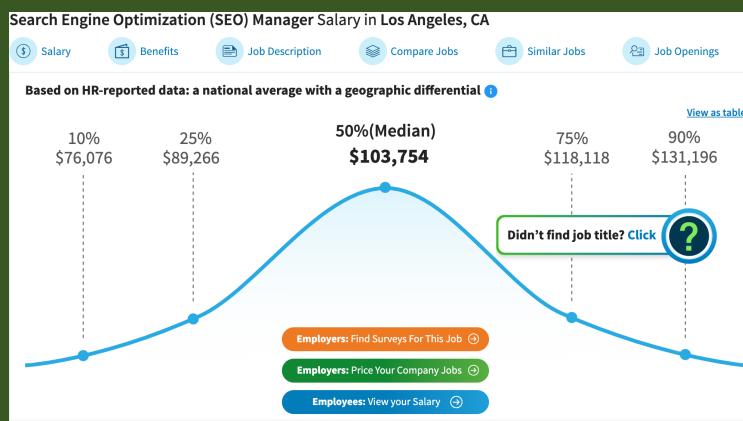
**&**

**Meta-description tag**

But there are actually more than 250 strategies, techniques and tactics. Obviously we wouldn't be able to cover all of them in CS-103. Some of them are also trade secrets, but we will cover a few additional techniques in near future.

If you are interested in SEO you could potentially become a SEO engineer

Please be a  
**white-hat** SEO  
specialist



## SEO (continued)

So far, we have learned about a few **strategies, techniques** and **tactics** that would help us increase the number of views our website.

We have learned about the **sitemap** and the **meta tag**.

But there are many other **strategies, techniques** and **tactics**, many of which are trade secrets, (so, obviously we wouldn't be able to cover those in this class).

But, we will cover one more technique before wrapping up this topic.

## PageRank™

The algorithm that first put Google on the map, and made searches useful was **PageRank™**

Named after **Larry Page**, the co-founder of Google.

## SEO (continued)

PageRank algorithm works based on the hyperlink structure of the Internet.

For each node  $x \in V$ :

$$PR_i(x) = (1 - d) + d \times \sum_{y \in InN(x)} \left( \frac{PR_{(i-1)}(y)}{\text{Outdegree}(y)} \right)$$

PR = PageRank™

$x$  = a web page

$V$  = the World Wide Web

$d$  = the damping factor (see next page)

$InN(x)$  = the set of pages that have links to page  $x$

$\text{Outdegree}(y)$  = the number of links leading from page  $y$

1<sup>st</sup>

Pages with **highest incoming links** have the highest ranking.



# PageRank

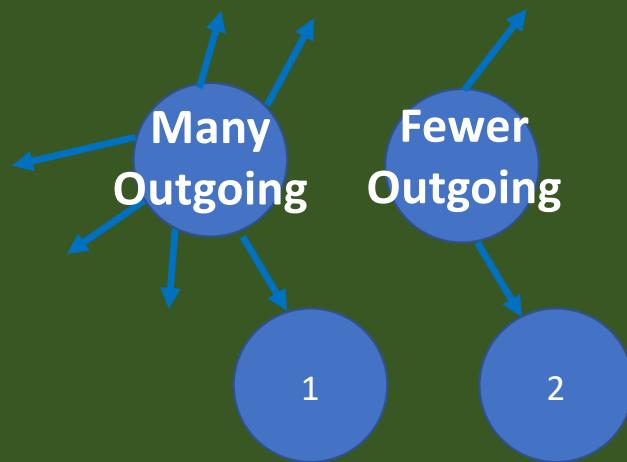
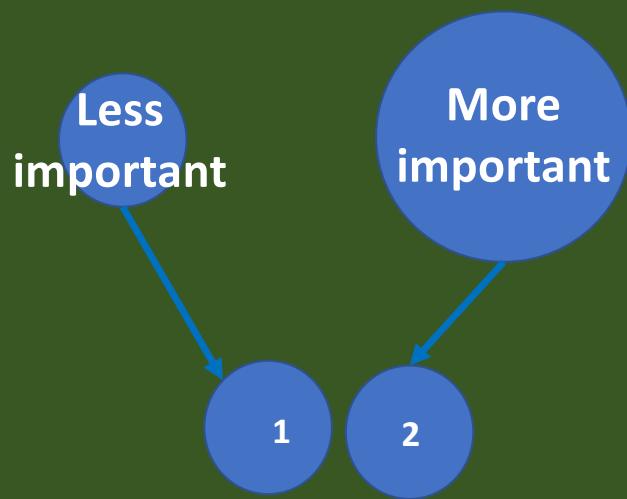
## SEO (continued)

2<sup>nd</sup>

Also, incoming links from important pages count more than incoming links from less important pages.

3<sup>rd</sup>

Pages promoted by pages with fewer outgoing links have higher ranking than pages that are promoted by pages that have lots of outgoing links



## SEO (continued)

1<sup>st</sup>

Page with **highest incoming links**

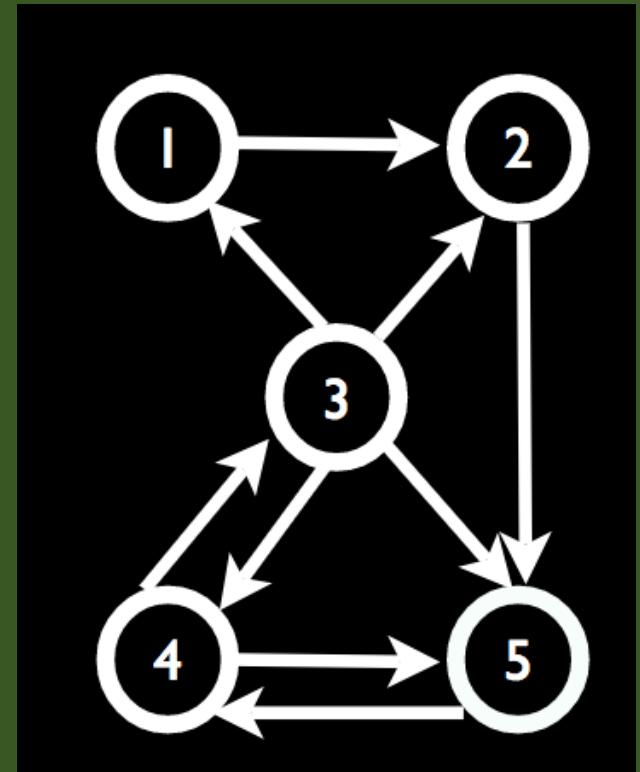
2<sup>nd</sup>

Page that has **Incoming links from important pages**

3<sup>rd</sup>

Page that is **promoted by pages with fewer outgoing links**

Answer: Page 5      Then 4, 3, 2 and 1



## SEO (continued)

When search engines suspect **black-hat** SEO activities, they will manually increase the “**d**” value of the **PageRank** algorithm

For example, if Google notices that there is an unusual improvement in the ranking of a webpage, it will increase the **damping factor** for that webpage to decrease the improvement.

Ex:

- In fall 2010, department store JC Penney hired an unethical SEO firm to boost their search engine rankings.
- Links to JC Penney pages were embedded in hundreds of low-quality, unrelated websites (for a fee).
- JC Penney ranked #1 in dozens of categories for several months.
- Google took a manual corrective action, and ratings fell.

- "The Dirty Little Secrets of Search"

David Segal, *New York Times*, February 12, 2011 <http://nyti.ms/1AlXm7g>

## Until Now:

- Any webpage should include at least 4 parts.
  - ✓ The header
  - ✓ The navigation menu
  - ✓ The main content area
  - ✓ The footer



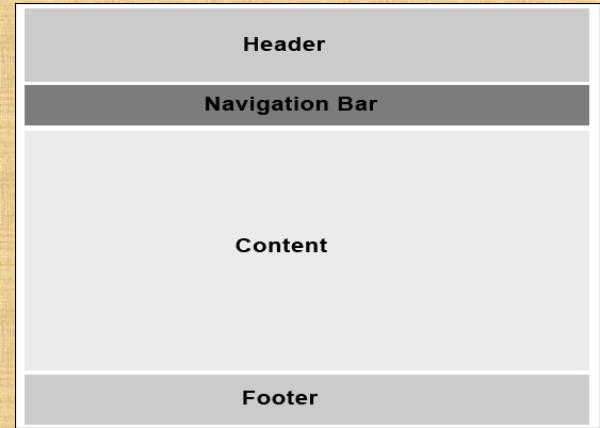
# Creating a Horizontal Navigation Menu (Setting Up the HTML & CSS)

## ➤ In the HTML file

1. Write your navigation menu as an unordered list (<ul>).
2. Add hyperlinks to the menu items.

## ➤ In the CSS file

1. Set the <li> to **display as inline**. (This makes the list horizontal.)
2. Set the **list-style** on the <ul> element to **none**. (This removes the bullets.)
3. Remove the default **left padding** on the <ul> element. Set a background color if desired.
4. Add right and left padding on the <li> element to add spacing.
5. Add top and bottom padding on the <ul> element to add spacing.
6. Remove underlines from the hyperlinks.



## “Block” & “Inline”

➤ Properties of block display

- ✓ Starts on a new line
- ✓ Forces the element that follows to start on a new line
- ✓ Takes up the whole width of the containing element unless a specific width is set

➤ Properties of inline display

- ✓ Inline elements do not start on a new line
- ✓ They do not force the next element to start on a new line (with one exception).

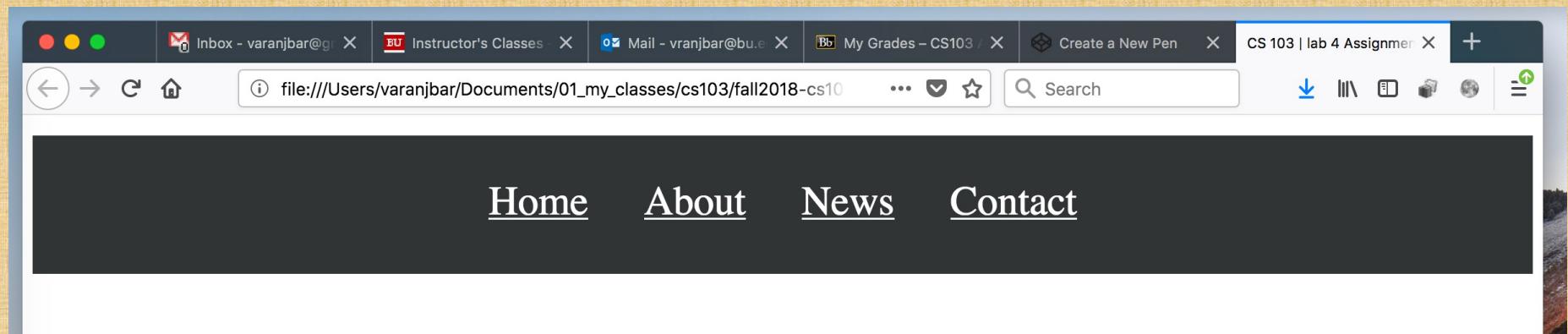
## Manipulating Display

- It is possible to override the default display of an HTML element by setting the display property in CSS.

```
display: block;      /* display as a block element */  
display: inline;    /* display as an inline element */  
display: none;     /* do not display */
```

- By changing the display property, we can make block-level elements behave like inline elements and vice versa.
- We can also make elements disappear by setting the display property to none.

# Creating a Horizontal Navigation Menu

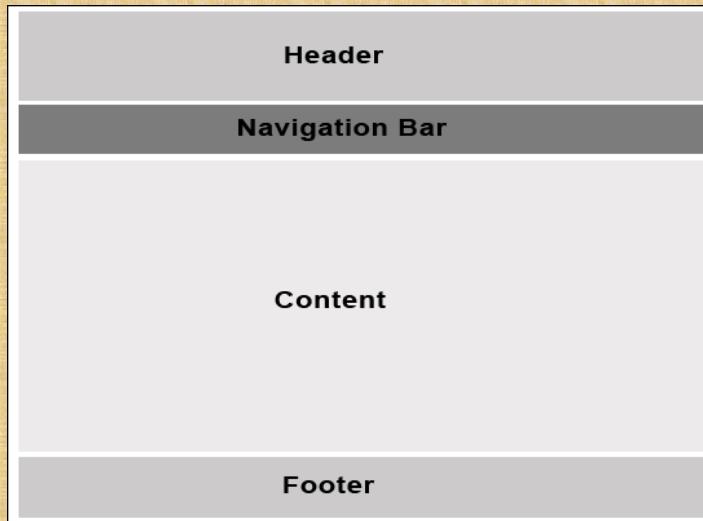


## Overview:

- Div & Span Elements – Id and Class Selectors
- Shorthand Properties
- Fonts in CSS: Font stacks and generic font families
- Visual Hierarchy
- Responsive Design

➤ As we describe earlier, a web page needs 4 parts:

- ✓ Header
- ✓ Navigation
- ✓ Main content
- ✓ Footer



➤ If these 4 parts lay down on top of each other, the web page is called one column layout.

## div and span elements

- Unlike the HTML elements such as paragraphs, lists, headings, the **<div>** and **<span>** elements do not convey any meaning or information about the content of a page.
- **<div>** is used to **group some elements**.
- **<span>** is used to **identify parts of elements**.

## div element

- A block-level element
- Can be used to put consecutive HTML elements into one group.
- Can hold other block-level elements, such as paragraphs, lists, and tables.
- Usually used with a class or ID attribute for styling.
- We are going to use it for two parts in a one-column layout:
  - ✓ Wrap around all of the elements by a div tag to make a one-column layout not using the entire width of the screen.
  - ✓ Wrap around all of the tags in the main content area after **nav** tag and before **footer** tag.

## div element example

➤ Sample HTML

```
<div class="blogpost">  
    <h3>Weather for Monday</h3>  
    <p>by Harvey Leonard</p>  
    <p>Cloudy, with temperatures in the 60s.</p>  
</div>
```

➤ Sample CSS

```
.blogpost { margin-left: 1.5em; }
```

## span element

- An inline element
- Used to single out part of an element.
- Cannot be used around block-level elements, only inside them.
- Usually used with a class or id attribute for styling.

## span element example

➤ Sample HTML

```
<p>Written by <span class="author">Harvey Leonard</span></p>
```

➤ Sample CSS

```
.author { text-transform: uppercase; }
```

## Notes about div and span elements

- The `<div>` tag is frequently used in combination with the `id` or `class` attributes to create CSS layouts
- As `<span>` and `<div>` don't convey meaning, only use when you cannot use an HTML element as a selector.

## div element

- A block-level element which is used with a class or ID attribute for styling.
- **Used to put consecutive HTML elements into one group.**
- Can hold other block-level elements, such as paragraphs, lists, and tables.

## span element

- An inline element which is used with a class or ID attribute for styling.
- **Used to single out part of an element.**
- Cannot be used around block-level elements, only inside them.

These elements do not convey meaning, so do not use div element instead of paragraphs, headings, etc ...

# Limitations of HTML Selectors

- Sometimes you want to treat the same HTML element differently within a page

```
<!DOCTYPE html>
<html>
  <head>
    <title> lecture 5-2 | CS 103 </title>
    <link rel="stylesheet" type="text/css" href="web.css">

  </head>

  <body>
    <h1>Lecture 5-2 </h1>
    <h2>More about Cascading Style Sheets (CSS)</h2>
    <h3>CSS colors</h3>

    <p>There are 4 different method to apply colors to texts
      including name of the color, hex codes, rgb colors and hsl codes</p>

    <p>Transparency is another important element of colors in HTMLs</p>
  </body>
</html>
```

```
body {background-color: hsla(193, 100%, 50%, 0.63);}

p {
  padding-left: 70px;
  font-size: 1.1em;
}
```

# Because of Limitations of HTML Selectors

- Increase specificity:
- Using two selectors together without a comma or other punctuation in between
- It will apply the rule to the second selector if and only if it is nested inside the first selector

```
ul li { color: white; background-color: red; }  
header h1 { font-weight: bold; }
```

- CSS Nesting
- Putting a comma between two selectors applies the same CSS rule to both selectors.

Instead of,

```
p { color: red; background-color: pink; }  
h2 { color: red; background-color: pink; }
```

It is better to say:

```
p, h2 { color: red; background-color: pink; }
```

## But sometimes CSS-Nesting is not enough

- Sometimes you want to style a group of elements with one set of rules
- Sometimes you want to style just part of an element
- **class** and **id** attributes help with that.
- They can be used as selectors.

## Combining CSS Selectors for More Power

- Combining HTML and Class Selectors
- Combining HTML and ID Selectors

## id attribute

```
<ul>
    <li id="favorite">Breakfast</li>
    <li>Lunch</li>
    <li>Dinner</li>
</ul>
```

- Used to style **one particular** occurrence of something on a page
- ID Name (value) must not contain spaces

# id attribute

```
<!DOCTYPE html>
<html>
  <head>
    <title> </title>
    <link rel="stylesheet" type="text/css" href="web.css">

  </head>

  <body>
    <ul>
      <li id="favorite">Breakfast</li>
      <li>Lunch</li>
      <li>Dinner</li>
    </ul>

  </body>
</html>
```

```
body {background-color: hsla(193, 100%, 50%, 0.63);}
```

```
#favorite { color: red; }
```

- Breakfast
- Lunch
- Dinner

## class attribute

- class attributes can be used to style multiple particular occurrences of something on a page.
- The name must not contain spaces either.

```
<ul>
    <li class="writing-intensive">Anthropology</li>
    <li class="writing-intensive">English</li>
    <li>Computer Science</li>
    <li>Accounting</li>
</ul>
```

## class attribute as a selector

- To apply a style to a class attribute ("writing-intensive" in this example), use a period followed by the name of the class attribute as your selector:

```
.writing-intensive { color: green; }
```

- Or, to be even more specific, use the element followed by . followed by the name of the class attribute

```
li.writing-intensive { color: green; }
```

## Using Id & Class Attributes as Selectors

```
<ul>
  <li>Breakfast</li>
  <li>Lunch</li>
  <li>Dinner</li>
</ul>
```

Problem:

- We want to assign green to Breakfast and purple to Lunch and Dinner
- Using elements (tags) as selectors can not help
  - ✓ **li {color: green;}**

## Using Id & Class Attributes as Selectors

```
<ul>
  <li id="favorite">Breakfast</li>
  <li class="food">Lunch</li>
  <li class="food" >Dinner</li>
</ul>
```

- Instead of using elements (tags) as selectors
  - ✓ ~~li {color: red;}~~
- We will use id and classes as selectors
  - ✓ #favorite{color: green;}
  - ✓ .food{color: purple;}

## Using Id & Class Attributes as Selectors

```
<ul>
  <li id="favorite">Breakfast</li>
  <li class="food">Lunch</li>
  <li class="food" >Dinner</li>
</ul>
```

➤ Instead of using elements (tags) as selectors

✓ ~~li {color: red;}~~

➤ We will use id and classes as selectors

✓ **#favorite{color: green;}**

✓ **.food{color: purple;}** or **li.food{color: purple;}**

more specific

## Using Id & Class Attributes as Selectors

One important rule:

- ✓ Id attributes can be used to style **one particular** occurrence of something on a page
- ✓ class attributes can be used to style **multiple particular** occurrences of something on a page.

```
<ul>
  <li id="favorite">Breakfast</li>
  <li class="food">Lunch</li>
  <li class="food" >Dinner</li>
</ul>
```

## A question about ID and Class selectors: How Does the Cascade Affect This?

- If an element uses both an ID attribute and a class attribute, and their style rules conflict, which rule will apply?

html: <p class="b" id="g">Will this paragraph be blue or green?</p>

css:

```
#g {color: green;}  
.b {color: blue;}
```

- Answer: green

# Combining HTML and Class Selectors

- ❖ HTML selectors can be used with class selectors to provide extra specificity.

`.column1`

any element with a class of column1

`h1.column1`

a top-level heading with a class of column1

`.column1 h1`

a top-level heading nested inside an element with a class of column1

## Combining HTML and Class Selectors

- HTML tags can be used with class selectors to provide extra specificity.

**.column1**

any element with a class of column1

**h1.column1**

a top-level heading with a class of column1

**.column1 h1**

a top-level heading nested inside an element with a class of column1

**<header>**

**<h1 class="column1"> Name of the website will sit here </h1>**

**</header>**

**<div class="column1">**

**<p> This is a paragraph. </p>**

**<h1> This is a level-one heading. </h1>**

**</div>**

Which elements will become blue in each case?

Case1: .column1{color:blue;}

Case2: h1.column1{color:blue;}

Case3: .column1 h1{color:blue;}

<header>

    <h1 class="column1"> Name of the website will sit here </h1>

</header>

<div class="column1">

    <p> This is a paragraph. </p>

    <h1> This is a level-one heading. </h1>

</div>

# Which elements will become blue in each case?

Case1:	.column1{color:blue;}	All three elements
Case2:	h1.column1{color:blue;}	The first h1 element
Case3:	.column1 h1{color:blue;}	The second h1 element

```
<header>
```

```
    <h1 class="column1"> Name of the website will sit here </h1>
```

```
</header>
```

```
<div class="column1">
```

```
    <p> This is a paragraph. </p>
```

```
    <h1> This is a level-one heading. </h1>
```

```
</div>
```

## Shorthand Properties in CSS

- CSS can get very long and to prevent this some properties can be written in shorthand form (combined or condensed).
- One example is properties related to the **border**.

## Shorthand Method #1: All Four Sides

- If you are specifying **border-style**, **border-color**, or **border-width**, you can specify all four sides of the box at once.
- Specify in this order: ***top right bottom left***. No punctuation in between.
- Example:

```
ul{border-color: red green yellow blue;}
```

- 
- Breakfast
  - Lunch
  - Dinner

## Shorthand Method #2: Two Sides

- If the **top** and **bottom** values of a border property are the same, AND the **right** and **left** values of the border property are the same, you can write it this way:

```
ul{ border-color: red green;}
```

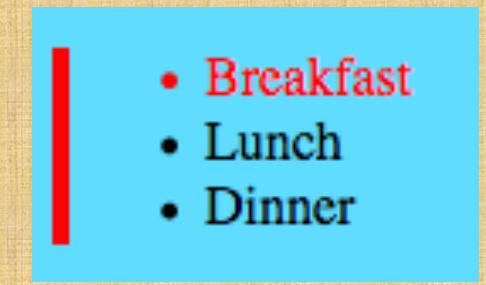
- Breakfast
- Lunch
- Dinner

- The first value represents the top and bottom. The second value will affect left and right.

## Shorthand Method #3: All properties for the same border

- If you are specifying just one side of the box, you can set all three border properties (**style**, **color**, **width**) at once.
- The properties can be listed in any order.
- Example:

```
ul{border-left: 5px solid red;}
```



## Putting It All Together

- If all four sides of your border are the same, you can specify the whole border in one short statement.

**ul{ border: red solid 2px;}**

- Breakfast
- Lunch
- Dinner

- All of these methods also work with margin and padding. When we say

**margin: 5vw;**

- It applies to all four sides.

- how about **margin: 0 20px;**

- It applies 0px to the top and bottom and 20px to the left and right.

## Fonts



## The Font Shorthand Property

- All properties beginning with “font” can be set in one rule using the font shorthand property
  - ✓ font-family
  - ✓ font-size
  - ✓ font-weight
  - ✓ font-style
  - ✓ font-variant

```
p {  
    font: small-caps italic bold 12px Georgia, serif;  
}
```

## Font stacks and generic font families

Font stack

```
body { font-family: Arial, "Helvetica Neue", Helvetica, sans-serif }
```

generic font family

## Font Stacks

- A font stack includes a list of fonts, **not a single font**.
- The browser will use the first one that it finds.
- Font stacks should end with a **generic font family**.



## Sample Font Stack

```
body { font-family: Arial, "Helvetica Neue", Helvetica, sans-serif}
```

- ✓ The browser will first look on the user's computer to see if it has Arial.

## Sample Font Stack

```
body { font-family: Arial, "Helvetica Neue", Helvetica, sans-serif}
```

- ✓ The browser will first look on the user's computer to see if it has Arial.
- ✓ If not, it will look for Helvetica Neue.

## Sample Font Stack

```
body { font-family: Arial, "Helvetica Neue", Helvetica, sans-serif}
```

- ✓ The browser will first look on the user's computer to see if it has Arial.
- ✓ If not, it will look for Helvetica Neue.
- ✓ If not, it will look for Helvetica.

## Sample Font Stack

```
body { font-family: Arial, "Helvetica Neue", Helvetica, sans-serif}
```

- ✓ The browser will first look on the user's computer to see if it has Arial.
- ✓ If not, it will look for Helvetica Neue.
- ✓ If not, it will look for Helvetica.
- ✓ If not, it will use whatever the computer has designated as its generic sans-serif font.

## Sample Font Stack

```
body { font-family: Arial, "Helvetica Neue", Helvetica, sans-serif}
```

- ✓ The browser will first look on the user's computer to see if it has Arial.
  - ✓ If not, it will look for Helvetica Neue.
  - ✓ If not, it will look for Helvetica.
  - ✓ If not, it will use whatever the computer has designated as its generic sans-serif font.
- Choose fonts that look alike, to give the user a similar experience

## Font Stack Coding Notes

- If the font name is more than one word, you must enclose it in either double quotation marks or single quotation marks (for example: , "Helvetica Neue").
- The **generic font family** is always one word, and must **never** be enclosed in quotation marks (or it will not work).

```
body { font-family: Arial, "Helvetica Neue", Helvetica, sans-serif}
```

## Generic Font Families

- The generic font families defined in CSS are:
  - ✓ Serif
  - ✓ Sans-serif
  - ✓ Monospace
  - ✓ Fantasy
  - ✓ Cursive
- Generic font families represent a **type** of a font. They are not fonts themselves.

## Specifying Fonts with CSS

- CSS allows you to specify the font (**font-family** property) for any selector
- The font you choose must be installed on the computer of the people visiting your website for them to see it.
- Fonts installed on the computer are called "**system fonts**"
- Macs and PCs have different system fonts installed (as do Unix computers)
- The challenge:
  - ✓ **giving users a comparable visual experience no matter what type of computer they are using.**

## Which Fonts Are on Which Computers?

- The website CSSFontStack (<https://www.cssfontstack.com>) gives us data about which fonts are used on which types of computers.
- It lists fonts with their prevalence on Windows and Mac machines.
- Fonts are grouped by families that *roughly* correspond to the generic font families used in CSS

## CSS Validator & Firefox Web Developer Tool

- Firefox does not have a tool for debugging CSS codes.
- Instead, we can use an online CSS Validator provided by W3C:  
**<https://jigsaw.w3.org/css-validator/>**
- Firefox provides a web developer tool called "**Inspector**" to see the html and css codes related to each line of your page.

## Good Visual Design Promotes Usability

➤ The primary purposes of graphic design are to:

1. Create a clear **visual hierarchy of contrast**, so you can see at a glance what is important and what is peripheral
2. Define functional regions of the page
3. Group the page elements that are related, so that you can see structure in the content

— "Visual Design," *Web Style Guide*

## First, we need to define usability

Usability is “**making sure that something works well:** that a person of average or even below average ability and experience can use the thing—whether it’s a Web site, a fighter jet, or a revolving door—without getting hopelessly frustrated.”

—Steve Krug

*Don't Make Me Think*

## Elements of Visual Hierarchy

- The most important things are the most prominent
- Things that are related logically should also be related visually
- Things are “nested” visually to show what’s part of what

—Steve Krug

*Don't Make Me Think*

## Tools for Creating Visual Hierarchy

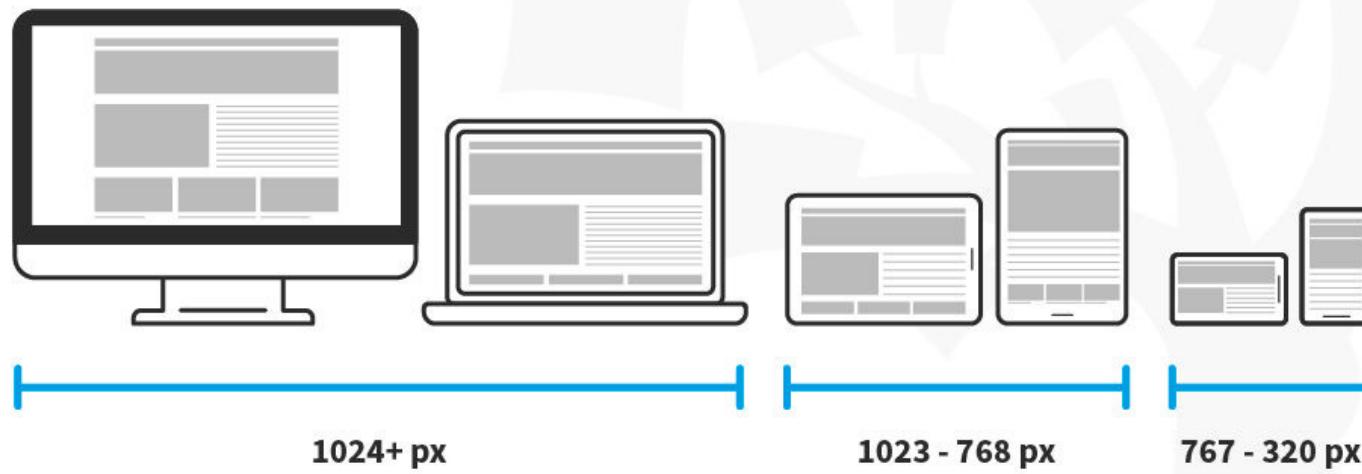
- How do we distinguish between important and unimportant, related and unrelated?
- Use **contrast** in:
  - ✓ Typeface (**font-family**)
  - ✓ Type style (**font-style, font-variant, font-weight, font-size ...**)
  - ✓ Type spacing (**letter-spacing, word-spacing ...**)
  - ✓ Color (**foreground & background colors**)
  - ✓ Whitespace (**padding/margin**)
  - ✓ **Placement on page**

## Whitespace

- One tool for creating visual hierarchy
- Better name might be “empty space” – doesn’t have to be white
- Helps user distinguish important areas
- Helps user perceive the page structure
- Do this:
  - ✓ An Event Apart: [www.aneventapart.com](http://www.aneventapart.com)
- Not this:
  - ✓ Shop in Paradise: [www.siphawaii.com](http://www.siphawaii.com)

# The Web and Mobile

## Responsive Design



## The Web and Mobile

- By some estimates, more people access the World Wide Web on mobile devices today than on desktops or laptops.
- In some parts of the world, mobile is the primary way that people access the web.
- This has affected how web designers work.

## Approaches to Mobile Web

- Do nothing
- Create an app
- Separate mobile website
- **Responsive design**

## 1. Do Nothing

➤ Advantages:

- ✓ Cheaper – no extra effort
- ✓ User doesn't have to do anything special to find it

➤ Disadvantages:

- ✓ Lots of zooming and scrolling – hard to use (usability)
- ✓ Hard for people with mobility impairments

## 2. Create an App

### ➤ Advantages

- ✓ Easier to use than a website; opportunity to gain market advantage
- ✓ Cool & fun
- ✓ Can take advantages of the phone features (GPS, camera)

### 3. Create a Separate Mobile Site

- Designed to work well on a mobile device
- Two ways to identify these:
  - ✓ Mobile sites are often called “**m.mysite.com**” or “**mobile.mysite.com**”
  - ✓ They usually have a link called "View full site," indicating that not all content is accessible

### 3. Create a Separate Mobile Site

#### ➤ Advantages

- ✓ No app required; easy & free for users
- ✓ Easier to move around the page; the buttons are bigger and optimized for use on mobile devices

#### ➤ Disadvantages

- ✓ Need to redirect users to mobile version at the right time
- ✓ Mobile site may not have everything visitors want
- ✓ May be hard for people to find what they're used to seeing; everything is in a different place
- ✓ Problem of updating two sites

## 4- Responsive Design

- Create a single website that responds to the width of the browser window
- This effect is achieved using some css techniques:
  - ✓ Specify widths using % and vw
  - ✓ Specify max-width in CSS, instead of width in HTML, for images and video
  - ✓ Use media queries to invoke different CSS rules depending on the width of the browser window
  - ✓ Using CSS files provided by third party companies such as Bootstrap (we will learn about Bootstrap next week )

## 4- Responsive Design

### ➤ Advantages

- ✓ No need to maintain two sites
- ✓ Mobile users get everything that desktop users get
- ✓ Easier to use on phones and tablets

### ➤ Disadvantages

- ✓ Takes a LOT of thought and planning to make sure that everything works no matter how wide the screen is.
- ✓ Code is harder to test and debug because multiple style sheets are in use
- ✓ Performance problem

## Technique: Max-Width for Images

- Remove width and height attributes from <img> tags
- Remove width and height properties for images from CSS
- Instead, specify max-width:100% for images in CSS
- If the window is wide enough, the image will appear at its natural size.
- Otherwise, browser will make it small enough to fit.
- To use this technique, you must make the image the right size in an image editor like Photoshop or GIMP.
- Since you can no longer use the height and width attributes in the img tag, you can no longer rescale the image in HTML.

## Technique: Media Queries

- Media queries invoke lines of CSS only if the browser meets certain criteria.  
Examples:

```
@media screen and (max-width: 768px) {  
.... /* CSS placed here will be applied only if the screen is 768 px or narrower */ }  
  
@media screen and (max-width: 520px) {  
.... /* CSS placed here will be applied only if the screen is 520 px wide or narrower */ }
```

- In the first query, **screen** is the "**media type**" **max-width** is the "**feature**" and **768px** is the "**value**"

For More Information

*Responsive Web Design*, by Ethan Marcotte

Order from A Book Apart, [www.abookapart.com](http://www.abookapart.com)

# Responsive One-column Layout Design

The screenshot displays a website layout for "CAS CS 103". The header is dark blue with white text, featuring the title "CAS CS 103" and a subtitle "Introduction to Internet Technologies and Web Development". Below the header is a dark grey navigation bar with links for "Home", "About", "News", and "Contact". The main content area has a white background and contains a section titled "Lecture 3-2" with the sub-section "Introduction to one column CSS layout". It includes a heading "What is a CSS layout?", a descriptive text about website structure, and a diagram illustrating a one-column layout. The diagram shows a vertical stack of five sections: "Header", "Navigation Menu", "Content" (divided into three columns: "Content", "Main Content", and "Content"), and "Footer". Below the diagram is a note about common layout designs. The footer is dark blue with white text, containing the copyright notice "Vahid Azadeh Ranjbar ©copy, 2018".

**CAS CS 103**

Introduction to Internet Technologies and Web Development

Home    About    News    Contact

**Lecture 3-2**

**Introduction to one column CSS layout**

**What is a CSS layout?**

A website is often divided into headers, menus, content and a footer:

The diagram illustrates a one-column layout structure. It consists of five horizontal sections stacked vertically. From top to bottom, they are labeled: "Header", "Navigation Menu", "Content", "Main Content", and "Content". The "Content" section is further divided into three equal-width columns, each labeled "Content". At the very bottom is a "Footer" section.

There are tons of different layout designs to choose from. However, the structure above, is one of the most common.

Vahid Azadeh Ranjbar ©copy, 2018

# Responsive Design

Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge a website, to make it look good on all devices (desktops, tablets, and phones):

- How can we make a responsive design?
  1. Take advantage of Relative units
    - ✓ %, vw, vh, em
  2. Some CSS properties:
    - ✓ width
    - ✓ max-width
    - ✓ calc()
    - ✓ margin:auto;
    - ✓ box-sizing



## Responsive Design

- Everything needs to be responsive to the width of the screen including:
  - ✓ Navigation Bar
  - ✓ Images
  - ✓ Font-size of texts
  - ✓ White spaces
- A demonstration of responsive vs. non-responsive design for a navigation:  
<https://youtu.be/WO-qLi4N36A>

## Responsive Design

- The most common **fixed unit** is px, meaning pixel
- To make a responsive design, **relative units** should be used:
  - ✓ %: Relative to the width of the parent's element
  - ✓ vw: Relative to 1% of the width of the browser window.
  - ✓ vh: Relative to 1% of the height of the browser window.
- No unit required if the number is 0
- For nonzero numbers, make sure there is no space between the value and the unit (e.g., **20%** not **20 %**)
- It's become very popular to use viewport units (vw and vh) for responsive typography establishing.
  - ✓ font-sizes that grow and shrink depending on the current viewport size.
  - ✓ Using fixed units (px) for font-size is not appropriate for small screens.
- Let's take a look at **<https://codepen.io/azadeh83/pen/jONgRRL>**

# Responsive Design

- **width** vs. **max-width**
  - ❖ Setting the **width** of a block-level element (like h1 element) will prevent it from stretching out to the edges of its container to the left and right.
  - ❖ Let's take a look at  
<https://codepen.io/azadeh83/pen/jONgRRL>
  - ❖ Using **max-width** property instead of **width** property will improve the browser's handling of small windows. This is important when making a site usable on mobile.

# Responsive Design

- **margin:auto;**
  - ❖ When specifying **width & max-width** properties, you can set the margin to **auto** to horizontally center that element within its container.
  - ❖ Let's take a look at  
<https://codepen.io/azadeh83/pen/JmXdVe>

## Responsive Design

- **box-sizing** to control the size inside boxes.
  - ❖ When you set the width of an element, the element can actually appear bigger than what you set.
  - ❖ The element's **border** and **padding** will stretch out of the element, beyond the specified width.
  - ❖ Let's take a look at <https://codepen.io/azadeh83/pen/jeqbwZ>

## Responsive Design

- **Calc() function** is helpful to control the size of texts.
  - ❖ Using fixed units (px) for font-size is not appropriate for small screens.
  - ❖ Direct scaling by using viewport is too dramatic as the scaling is too fast.
  - ❖ Lets take a look at <https://codepen.io/azadeh83/pen/jONgRRL>
  - ❖ We can combine a base size in more steady units (say 5px) with a smaller viewport-relative adjustment (2vw), and let the browser do the math: **calc(5px + 2vw)**.
  - ❖ Lets take a look at <https://codepen.io/azadeh83/pen/ePzQpG>

## Overview:

- Basics of JavaScript Programming
  - Data Types
  - Variables
  - Operations
  - Printing Out
  - Relational and Logical Operators
  - Conditional Statements
  - ...

# Static Web Pages vs. Dynamic Web Pages

## ➤ Static Web Pages:

- ✓ Static web pages are written entirely in HTML/CSS.
- ✓ Once a static web page is built, it looks exactly the same until a web developer updates it (static means unchanging).

## ➤ Dynamic Web Pages:

- ✓ "Dynamic" means "changing."
- ✓ To create a dynamic web page, you need to use a web programming language such as PHP, JavaScript, Python, Java ...

## Dynamic Web Pages

- Web programming languages come in two types:
  - ✓ client-side which runs in the web browser (individual's computers)
    - Example: JS (JavaScript)
  - ✓ server-side which runs on the web server (web host)
    - Example: PHP (PHP Hypertext Preprocessor)

# Server-Side Web Programming Languages

➤ Top five as of March 2021, as measured by W3Techs.com

1. PHP	78.8%	-0.2%
2. ASP.NET	10.5%	-0.1%
3. Java	3.6%	+0.2%
4. Ruby	3.3%	+0.2%
5. static files	1.7%	-0.1%

## Server-Side Web Programming Languages

- Server-side programming languages run in the server not in a computer.
- Advantages:
  - ✓ They work with all browsers and screen readers
  - ✓ User can't turn them off
  - ✓ They can access resources (like databases) that are accessible from the web server, but not open to the public

# Server-Side Programming Languages

## ➤ Disadvantages

- ✓ Most are expensive
  - Except for PHP, which is open source.
- ✓ All processing occurs on the web server
  - The work of processing is not shared by the user's computer
  - You can not see the actual programming by using "Source" tab in the menu of web browsers. Instead, you can only see the resultant html codes.
  - **Loading the web page may take longer because of the distance between the web browser and the web server**

# Client-side Web Programming Languages

- The other type of web programming language is client-side.
- "Client" – an old word for a personal computer
- "Client-side" means "installed on the user's computer."
- The only major client-side web programming language is JavaScript.
- JavaScript is installed inside every web browser.
- Users can turn them off

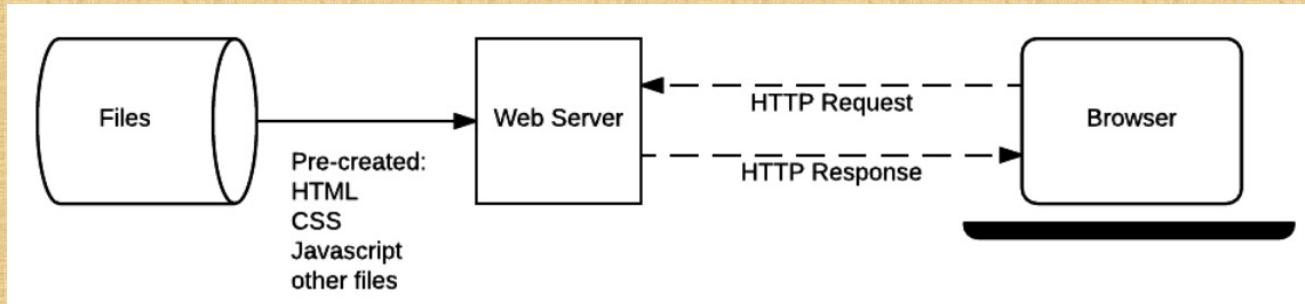
1. <a href="#">JavaScript</a>	95.0%
2. <a href="#">Flash</a>	2.6%

-0.1%

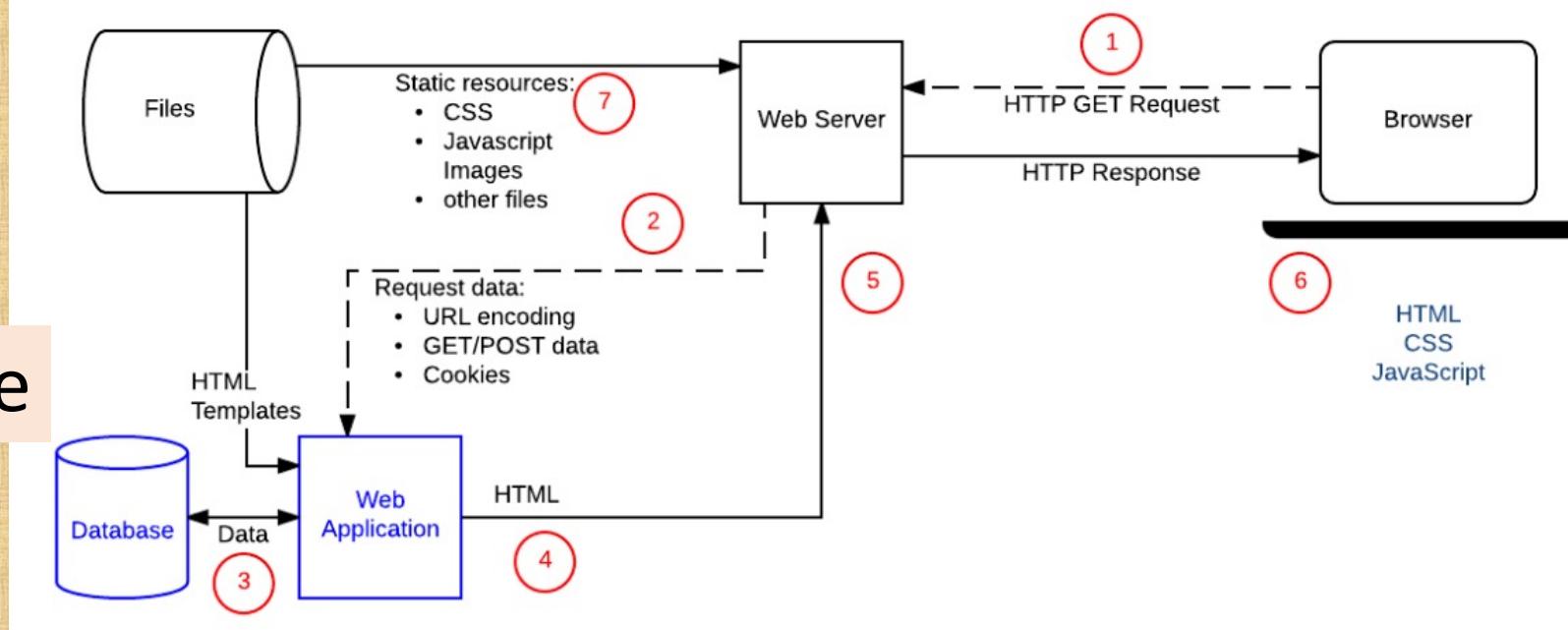
measured by W3Techs.com at March 2021

# Server-side vs. Client-side

## Client-side



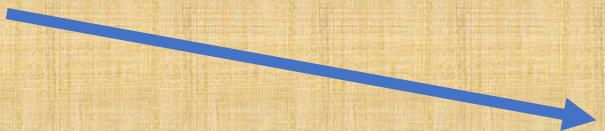
## Server-side



- JavaScript can be written in a separate file which end in .js (**External JS coding**)
  - ✓ We need to link a JS file to a HTML file by using the **script tag**
- JS codes can also be incorporated into HTML pages (**Internal JS coding**)
  - ✓ For internal JS coding, JavaScript code must be inserted in the **script tag**
- The script tag is a paired tag and must have a closing tag
- The script tag can appear in the **head** or the **body** of an HTML page

**<script>** Do **Internal JS Coding** here!!! **</script>**

**<script src="main.js"></script>**



**main.js**

Do **External JS Coding** here!!!

# Basics of JS Coding: Different types of information (Data Type)

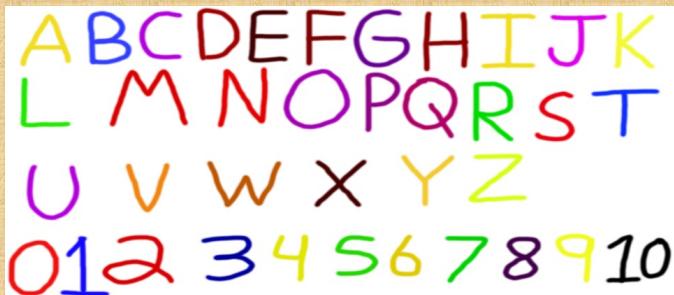
- In programming, information (data) is categorized by type:
  - ✓ numerical data for **computations**
  - ❖ example: 451

A B C D E F G H I J K  
L M N O P Q R S T  
U V W X Y Z  
0 1 2 3 4 5 6 7 8 9 10



# Basics of JS Coding: Different types of information (Data Type)

- In programming, information (data) is categorized by type:
  - ✓ numerical data for **computations**
    - ❖ example: 451
  - ✓ Alphanumeric for **text**
    - ❖ examples: "hello" 'Picobot' '451'



A decorative graphic featuring a grid of letters and numbers in various colors (blue, red, green, yellow, purple). The letters include A through Z, and the numbers include 0 through 10.



## Basics of JS Coding: Different types of information (Data Type)

- There are many different **data types** In JS.
- Two of them are **numbers** and **texts** (also called **strings**)

## Basics of JS Coding:

### Different types of information (Data Type)

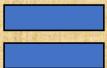
- There are many different **data types** In JS.
  - Two of them are **numbers** and **texts** (also called **strings**)
  - **Strings** must be enclosed in single or double quotation marks.

# Basics of JS Coding: Variables

- Variables are names that refer to a value.

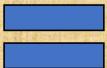
# Basics of JS Coding: Variables

- Variables are names that refer to a value.
- Values are assigned to variables using the assignment operator



# Basics of JS Coding: Variables

- Variables are names that refer to a value.
- Values are assigned to variables using the assignment operator



- They allow us to store a value for later use in our program:

```
temp = 70  
date = 'September 13, 2016'  
name = 'Jane Doe'
```

# Basics of JS Coding: Variables

- Variables are names that refer to a value.
- Values are assigned to variables using the assignment operator



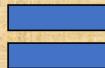
- They allow us to store a value for later use in our program:

```
temp = 70  
date = 'September 13, 2016'  
name = 'Jane Doe'
```

- Key words **let** and **const** are used to create a variable.

# Basics of JS Coding: Variables

- Variables are names that refer to a value.
- Values are assigned to variables using the assignment operator



- They allow us to store a value for later use in our program:

```
temp = 70  
date = 'September 13, 2016'  
name = 'Jane Doe'
```

- Key words **let**, **let**, and **const** are used to create a variable.

```
let a;  
const name;  
a = 100;  
name = 'Jack';
```

# Basics of JS Coding: Some Syntax

- Every line of code in JavaScript coding must end in a semicolon

```
let a;  
const name;  
a = 100;  
name = 'Jack';
```

## Basics of JS Coding: Some Syntax

- Every line of code in JavaScript coding must end in a semicolon

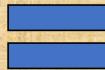
```
let a;  
let name;  
a = 100;  
name = 'Taymaz';
```

- It is possible to assign a value to a variable in one line.

```
let a = 100;  
let name = 'Taymaz';
```

# Basics of JS Coding: Variables

- Variables are names that refer to a value.
- Values are assigned to variables using the assignment operator



- They allow us to store a value for later use in our program:

```
temp = 70  
date = 'September 13, 2016'  
name = 'Jane Doe'
```

- Key words **let** and **const** are used to create a variable.

```
let a;  
const name;  
a = 100;  
name = 'Taynaz';
```

## Basics of JS Coding: Some Syntax

- Every line of code in JavaScript coding must end in a semicolon

```
let a;  
let name;  
a = 100;  
name = 'Taymaz';
```

- It is possible to assign a value to a variable in one line.

```
let a = 100;  
let name = 'Taymaz';
```

# Basics of JS Coding: Comment Tags

- Single Line Comments in JavaScript start with two slashes: //
- Multi-line comments start with /\* and end with \*/ the same as CSS.

```
let a = 5;
```

```
// This is the world's best piece of JavaScript coding.
```

```
/*
```

```
We aren't quite sure yet what it does but it is bound to  
be awesome.
```

```
Set up some variables first.
```

```
*/
```

```
let b = 6;
```

# Basics of JS Coding: Print out

- We can print things to the console by using **console.log**

```
let a = 12;  
let b = 16;  
console.log(a);  
console.log(b);
```

# Basics of JS Coding: Operations

- We can do basic math functions such as addition, subtraction, and so on

```
let a = 12;  
let b = 16;  
let e;  
let f;  
e = a - b;  
f = a + b;
```

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Remainder)

- Now, **e** equals **-4** and **f** equals **28**.

# Basics of JS Coding: Operators and Data Types

- To be able to operate on variables, it is important to know something about their data type.

```
let name = "Taymaz " + "Davoodi";
```

Question: Does it make any sense to add two strings(texts)?

Answer: Yes! It is called "**Concatenation**".

- The strings will be attached and make a longer string.

```
name = "Taymaz Davoodi"
```

# Basics of JS Coding: Operators and Data Types

- How about the following code? Does it make any sense to add number sixteen and string "Volvo"?

```
let x = 16 + "Volvo";
```

Answer: When adding a number and a string, JavaScript treats the number as a string. So the answer of the above example is:

```
x = "16Volvo"
```

# Basics of JS Coding: Operators and Data Types

- Some other examples:

```
let x = "Volvo" + 16;  
let x = "Volvo" + 16 + 4;  
let x = 16 + 4 + "Volvo";
```

answer: x = ?  
answer: x = ?  
answer: x = ?

# Basics of JS Coding: Operators and Data Types

- Some other examples:

```
let x = "Volvo" + 16;  
let x = "Volvo" + 16 + 4;  
let x = 16 + 4 + "Volvo";
```

answer: x = "Volvo16"  
answer: x = ?  
answer: x = ?

# Basics of JS Coding: Operators and Data Types

- Some other examples:

```
let x = "Volvo" + 16;  
let x = "Volvo" + 16 + 4;  
let x = 16 + 4 + "Volvo";
```

answer: x = "Volvo16"  
answer: x = "Volvo164"  
answer: x = ?

# Basics of JS Coding: Operators and Data Types

- Some other examples:

```
let x = "Volvo" + 16;  
let x = "Volvo" + 16 + 4;  
let x = 16 + 4 + "Volvo";
```

answer: x = "Volvo16"  
answer: x = "Volvo164"  
answer: x = ?

Note that order of  
operations matters in JS!

"Volvo" + 16 → "Volvo16" + 4 → "Volvo164"

# Basics of JS Coding: Operators and Data Types

- Some other examples:

```
let x = "Volvo" + 16;  
let x = "Volvo" + 16 + 4;  
let x = 16 + 4 + "Volvo";
```

answer: x = "Volvo16"  
answer: x = "Volvo164"  
answer: x = ?

Note that order of  
operations matters in JS!

"Volvo" + 16 → "Volvo16" + 4 → "Volvo164"

# Basics of JS Coding: Operators and Data Types

- Some other examples:

```
let x = "Volvo" + 16;  
let x = "Volvo" + 16 + 4;  
let x = 16 + 4 + "Volvo";
```

answer: x = "Volvo16"  
answer: x = "Volvo164"  
answer: x = "20Volvo"

- In the last example,
  - **16** and **4** are summed up first. The answer is number **20**.
  - Then, number **20** and string "**Volvo**" should be added.
  - JS treats number **20** a string and concatenate "**20**" and "**Volvo**"
  - So, the final answer is "**20Volvo**"

# Basics of JS Coding: Change the Data Type

- We can change the data type of values:
  - To change the data type of a number to string, you need to use **String()**.
  - To change the data type of a string to number, you need to use **Number()**.

# Basics of JS Coding: Change the Data Type

- We can change the data type of values:
  - To change the data type of a number to string, you need to use **String()**.
  - To change the data type of a string to number, you need to use **Number()**.
- What will be printed into the console?

```
let x1 = "5";
let x2 = 6;
console.log(Number(x1) + x2);
```

# Basics of JS Coding: Change the Data Type

- We can change the data type of values:
  - To change the data type of a number to string, you need to use **String()**.
  - To change the data type of a string to number, you need to use **Number()**.

- What will be printed into the console?

```
let x1 = "5";
let x2 = 6;
console.log(Number(x1) + x2);
```

- Answer:

**Number("5") → 5 + 6 → 11**

# Basics of JS Coding:

## Change the Data Type

- We can change the data type of values:
  - To change the data type of a number to string, you need to use **String()**.
  - To change the data type of a string to number, you need to use **Number()**.

- What will be printed into the console?

```
let x1 = "5";
let x2 = 6;
console.log(Number(x1) + x2);
```

- Answer:

**Number("5") → 5 + 6 → 11**

## Basics of JS Coding:

**What is it for? Where is it written? How is it executed?**

- JavaScript can be written in a separate file which end in .js (**External JS coding**)
  - ✓ We need to link a JS file to a HTML file by using the **script tag**
- JS codes can also be incorporated into HTML pages (**Internal JS coding**)
  - ✓ For internal JS coding, JavaScript code must be inserted in the **script tag**
- The script tag is a paired tag and must have a closing tag
- The script tag can appear in the **head** or the **body** of an HTML page

# What is it for?

- JavaScript and HTML web pages are intertwined
  - ✓ HTML is great for pages where things don't change
  - ✓ JavaScript was designed to let coders add interactivity to their pages
- All the 'fancy' things you see on web pages are in almost every case JavaScript programs running inside the page

## Where do you write JS codes?

- JavaScript can be written in a separate file which end in .js (**External JS coding**)
  - ✓ We need to link a JS file to a HTML file by using the **script tag**
- JS codes can also be incorporated into HTML pages (**Internal JS coding**)
  - ✓ For internal JS coding, JavaScript code must be inserted in the **script tag**
- The script tag is a paired tag and must have a closing tag
- The script tag can appear in the **head** or the **body** of an HTML page

# Where does it run?

- where does the program run?
  - ✓ In the browser
- Browsers include a JavaScript ‘engine’ that reads and executes the JavaScript code in each web page
- You can turn the engine off in Settings or Preferences for each browser

# Browsers and JS!

- How a browser recognize a JavaScript file?
  - ✓ External JavaScript files are ordinary text files (like HTML and CSS) which end in .js
- How a browser recognize a JavaScript piece of code?
  - ✓ Internal JavaScript codes are places between `<script>...</script>` tags

# Where do you write JS codes?

- JavaScript can be written in a separate file which end in .js (External JS coding)
  - ✓ We need to link a JS file to a HTML file by using the **script tag**
- JS codes can also be incorporated into HTML pages (Internal JS coding)
  - ✓ For internal JS coding, JavaScript code must be inserted in the **script tag**
- The script tag is a paired tag and must have a closing tag
- The script tag can appear in the head or the body of an HTML page

- JavaScript and HTML web pages are intertwined
  - ✓ HTML is great for pages where things don't change
  - ✓ JavaScript was designed to let coders add interactivity to their pages
- All the "**fancy**" things you see on web pages are coming from JS
- JavaScript can be written inside an HTML page or in a separate file.
  - ✓ In both cases, we need to use the **script** tag.
  - ✓ The script tag is a paired tag and must have a closing tag
  - ✓ The script tag can appear in the **head** or the **body** of an HTML page.
- Browsers include a JavaScript engine that reads and executes JS programs.
  - ✓ How a browser recognize a JavaScript file?
    - External JavaScript files are text files (like HTML and CSS) which end in .js
  - ✓ How does a browser recognize a JavaScript piece of code?
    - Internal JavaScript codes are lines of code between **<script>...</script>** tags

# Basics of JS Coding: Using parentheses

We learned that:

```
let x = "Volvo" + 16;
```

answer: x = "Volvo16"

```
let x = "Volvo" + 16 + 4;
```

answer: x = "Volvo164"

```
let x = 16 + 4 + "Volvo";
```

answer: x = "20Volvo"

How about:

```
let x = "Volvo" + (16 + 4);
```

answer: x = "Volvo20"

```
let x = 16 + (4 + "Volvo");
```

answer: x = "164Volvo"

"Volvo" + (16 + 4) → "Volvo" + 20 → "Volvo20"

16 + (4 + "Volvo") → 16 + "4Volvo" → "164Volvo"

- Note that operations inside **parentheses** will be computed first.

Fill in the blank to print text "Oct. 31 Halloween 2019" in the console?

```
let x1 = "8";
let x2 = 20;
let x3 = 7;
let x4 = " Halloween ";
let x5 = 19;
let x6 = "Oct. ";
let x7 = 3;
console.log_____);
```

- A. x6 + x7 + (Number(x1) - x3) + x4 + String(x2 + x5);
- B. (x6 + x7 + Number(x1) - x3) + x4 + x2 + x5;
- C. x6 + (x7 + (Number(x1) - x3)) + x4 + x2 + x5;
- D. (x6 + x7) + (Number(x1 - x3)) + x4 + x2 + x5;
- E. (x6 + x7) + (Number(x1) - x3) + x4 + x2 + x5;
- F. None of the above!

Fill in the blank to print text "Oct. 31 Halloween 2019" in the console?

```
let x1 = "8";
let x2 = 20;
let x3 = 7;
let x4 = " Halloween ";
let x5 = 19;
let x6 = "Oct. ";
let x7 = 3;
```

"Oct. " + 3 + 1 + " Halloween " + "39" = "Oct. 31 Halloween 39"

$$8 - 7 = 1$$

$$20 + 19 = 39$$

A. **x6 + x7 + (Number(x1) - x3) + x4 + String(x2 + x5);**

Fill in the blank to print text "Oct. 31 Halloween 2019" in the console?

```
let x1 = "8";
let x2 = 20;
let x3 = 7;
let x4 = " Halloween ";
let x5 = 19;
let x6 = "Oct. ";
let x7 = 3;
```

$$(x6 + x7 + \text{Number}(x1) - x3) + x4 + x2 + x5 = \text{NAN}$$

$$\text{"Oct. 38"} - 7 = \text{NAN}$$

$$\text{"Oct. "} + 3 + 8 - 7$$

B. **(x6 + x7 + Number(x1) - x3) + x4 + x2 + x5;**

Fill in the blank to print text "Oct. 31 Halloween 2019" in the console?

```
let x1 = "8";
let x2 = 20;
let x3 = 7;
let x4 = " Halloween ";
let x5 = 19;
let x6 = "Oct. ";
let x7 = 3;
```

"Oct. " + 4 + " Halloween " + 20 + 19 = "Oct. 4 Halloween 2019"

$$3 + 1 = 4$$

$$8 - 7 = 1$$

C. x6 + (x7 + (Number(x1) - x3)) + x4 + x2 + x5;

Fill in the blank to print text "Oct. 31 Halloween 2019" in the console?

```
let x1 = "8";
let x2 = 20;
let x3 = 7;
let x4 = " Halloween ";
let x5 = 19;
let x6 = "Oct. ";
let x7 = 3;
```

"Oct. 3" + 1 + " Halloween " + 20 + 19 = "Oct. 31 Halloween 2019"

"Oct. " + 3 = "Oct. 3"      "8" - 7 = 8 - 7 = 1

D.  $(x6 + x7) + (\text{Number}(x1 - x3)) + x4 + x2 + x5;$

Fill in the blank to print text "Oct. 31 Halloween 2019" in the console?

```
let x1 = "8";
let x2 = 20;
let x3 = 7;
let x4 = " Halloween ";
let x5 = 19;
let x6 = "Oct. ";
let x7 = 3;
```

"Oct. 3" + 1 + " Halloween " + 20 + 19 = "Oct. 31 Halloween 2019"

$$\text{"Oct. "} + 3 = \text{"Oct. 3"}$$

$$8 - 7 = 1$$

E.  $(x6 + x7) + (\text{Number}(x1) - x3) + x4 + x2 + x5;$

Fill in the blank to print text "Oct. 31 Halloween 2019" in the console?

```
let x1 = "8";
let x2 = 20;
let x3 = 7;
let x4 = " Halloween ";
let x5 = 19;
let x6 = "Oct. ";
let x7 = 3;
console.log_____);
```

➤ Note that operations inside **parentheses** will be computed first.

- A. x6 + x7 + (Number(x1) - x3) + x4 + String(x2 + x5);
- B. (x6 + x7 + Number(x1) - x3) + x4 + x2 + x5;
- C. x6 + (x7 + (Number(x1) - x3)) + x4 + x2 + x5;
- D. (x6 + x7) + (Number(x1 - x3)) + x4 + x2 + x5;
- E. (x6 + x7) + (Number(x1) - x3) + x4 + x2 + x5;
- F. None of the above!



# Basics of JS Coding: Output

- We can print to the *console* by using **console.log**
  - ✓ `Console.log("This will be printed");`
- We can print to the *HTML* by using **innerHTML**
  - ✓ `<p id="output"></p>`
  - ✓ `document.getElementById("output").innerHTML="This will be printed"`

`document.getElementById("output").innerHTML = "You have reached the maximum login attempts.;"`

# Lets make an age-restricted website!!!

1. This website is supposed to show users a warning message about age.
2. Then, there has to be another message to ask user's age.
3. Based on the answer, another message will be displayed to show that if the user is or is not allowed to enter the website.
4. Also, inside the document (the webpage), we would like to see a paragraph as following
  - A warning message asking to leave the page if user is less than 18
  - A welcome message if user is greater than or equal 18.
5. Also, if the user put a non-numeric answer, ask the user to put an acceptable numerical answer.

# Basics of JS Coding: Operations

- We can do basic math functions such as addition, subtraction, and so on

```
let a = 12;  
let b = 16;  
let e;  
let f;  
e = a - b;  
f = a + b;
```

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Remainder)

- Now, **e** equals **-4** and **f** equals **28**.

# Basics of JS Coding: Relational Operators

- JS provides a set of **relational operators** for making comparisons:

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal value to
====	equal value & equal type to
!=	not equal to

don't confuse with =

# Basics of JS Coding: Relational Operators

- We often want to know whether one thing is less than, greater than, or equal to another thing.
- So, we can use relational operators.
  - ✓  $a < b$  (is a less than b?)
  - ✓  $a > b$  (is a greater than b?)
  - ✓  $a == b$  (is a equal to b?)
- All of these expressions result in either **true** or **false**
- **true** and **false** values are called **Boolean values**

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: 12 > 16? false  
answer: 12 == "12"? true  
answer: 12 === "12"?false  
answer: 16 != 16? false  
answer: 16 >= 16? true  
answer: -4 <= 28? true

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "12";
let d = 16;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d !== b);
console.log(d = b);
```

answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "12";
let d = 16;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d !== b);
console.log(d = b);
```

answer: 12 > "16"? false  
answer: 12 == "12"? true  
answer: 12 === "12"? false  
answer: 16 != "16"? false  
answer: 16 !== "16"? true  
answer: "16"

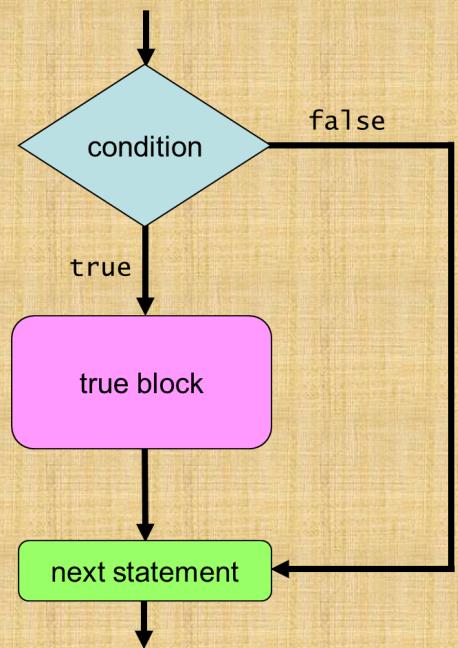
# Basics of JS Coding: Conditional Statements

- Sometimes, we want something to happen based on a comparison
  - ✓ For example: "**if a is less than b, then give c the value 22**"
- Syntax:

```
if (condition){  
    true block  
}
```

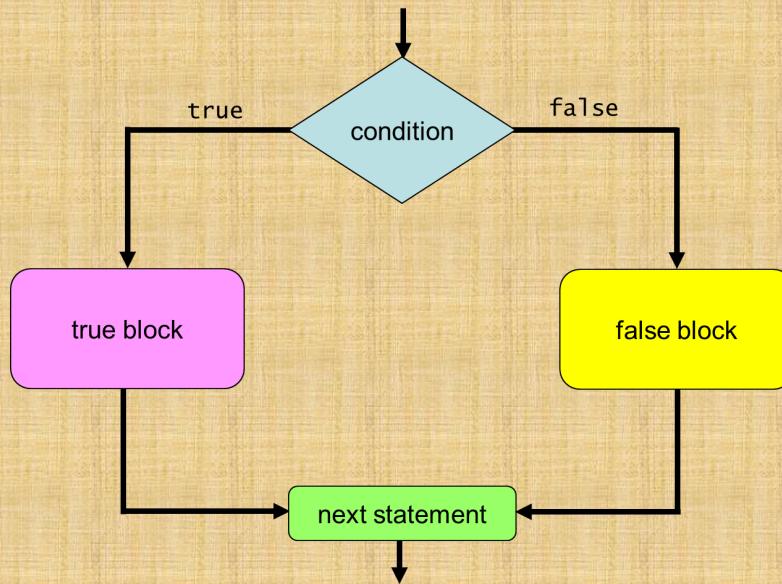
- Example: **if a is less than b, then give c the value 22**

```
if (a < b) {  
    c = 22;  
}
```



# Basics of JS Coding: Conditional Statements

- This is called simple decisions: if statements
- We have also two-way decisions: if-else statements



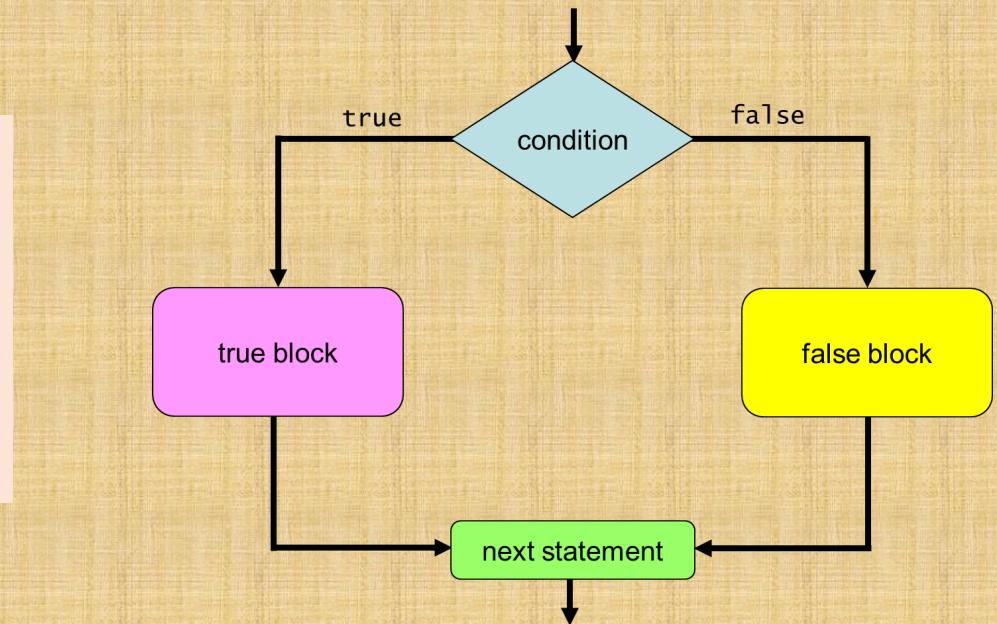
```
if (condition){  
    true block  
}
```

```
if (condition){  
    true block  
}  
else{  
    false block  
}
```

# Basics of JS Coding: Conditional Statements

For example: "if a is less than b, then give c the value 22, otherwise give c the value 9."

```
if (a < b) {  
    c = 22;  
}  
else {  
    c = 9;  
}
```



# Lets make an age-restricted website!!!

```
<input id="input"></input>
```

➤(Output)We can print to the *HTML* by using **innerHTML**

✓ <p id="output"></p>

✓ document.getElementById("output").innerHTML="This will be printed"

*document.getElementById("output").innerHTML = "You are not old enough";*

# Lets make an age-restricted website!!!

```
<input id="input">
```

➤(Output)We can print to the *HTML* by using **innerHTML**

- ✓ <p id="output"></p>
- ✓ document.getElementById("output").innerHTML="This will be printed"

```
document.getElementById("output").innerHTML = "You are not old enough";
```

```
<h3>to enter this website, you need to be at least 18 years old.</h3>
```

```
<label for="input">How old Are you: </label><input id="input">
```

```
let age = document.getElementById("input").value;
```

## Let's make an age-restricted website!!!

1. This website is supposed to show users a warning message about age.
2. Then, there has to be another message to ask user's age.
3. Based on the answer, another message will be displayed to show that if the user is or is not allowed to enter the website.
4. Also, inside the document (the webpage), we would like to see a paragraph as following
  - A warning message asking to leave the page if user is less than 18
  - A welcome message if user is greater than or equal 18.
5. Also, if the user put a non-numeric answer, ask the user to put an acceptable numerical answer.

# Basics of JS Coding: Operations

- We can do basic math functions such as addition, subtraction, and so on

```
let a = 12;  
let b = 16;  
let e;  
let f;  
e = a - b;  
f = a + b;
```

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Remainder)

- Now, **e** equals **-4** and **f** equals **28**.

# Basics of JS Coding: Relational Operators

- JS provides a set of **relational operators** for making comparisons:

Operator	Description
<	less than
>	greater than

# Basics of JS Coding: Relational Operators

- JS provides a set of **relational operators** for making comparisons:

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

# Basics of JS Coding: Relational Operators

- JS provides a set of **relational operators** for making comparisons:

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal value to
====	equal value & equal type to

don't confuse with =

# Basics of JS Coding: Relational Operators

- JS provides a set of **relational operators** for making comparisons:

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal value to
====	equal value & equal type to
!=	not equal to

don't confuse with =

# Basics of JS Coding: Relational Operators

- We often want to know whether one thing is less than, greater than, or equal to another thing.
- So, we can use relational operators.
  - ✓  $a < b$  (is a less than b?)
  - ✓  $a > b$  (is a greater than b?)
  - ✓  $a == b$  (is a equal to b?)
- All of these expressions result in either **true** or **false**
- **true** and **false** values are called **Boolean values**

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: 12 > 16? false  
answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: 12 > 16? false  
answer: 12 == "12"? true  
answer: ?  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: 12 > 16? false  
answer: 12 == "12"? true  
answer: 12 === "12"?false  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: 12 > 16? false  
answer: 12 == "12"? true  
answer: 12 === "12"?false  
answer: 16 != 16? false  
answer: ?  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: 12 > 16? false  
answer: 12 == "12"? true  
answer: 12 === "12"?false  
answer: 16 != 16 ? false  
answer: 16 >= 16? true  
answer: ?

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
let e = a - b;
let f = a + b;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d >= b);
console.log(e <= f);
```

answer: 12 > 16? false  
answer: 12 == "12"? true  
answer: 12 === "12"?false  
answer: 16 != 16? false  
answer: 16 >= 16? true  
answer: -4 <= 28? true

# Basics of JS Coding: Relational Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "12";
let d = 16;
console.log(a > b);
console.log(a == c);
console.log(a === c);
console.log(d != b);
console.log(d !== b);
console.log(d = b);
```

answer: 12 > "16"? false  
answer: 12 == "12"? true  
answer: 12 === "12"? false  
answer: 16 != "16"? false  
answer: 16 !== "16"? true  
answer: "16"

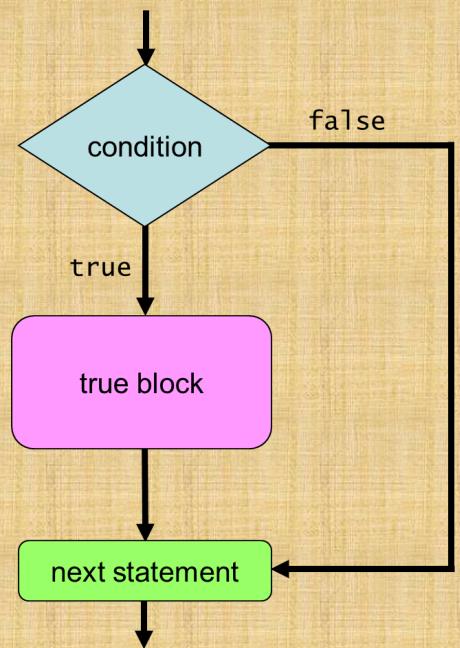
# Basics of JS Coding: Conditional Statements

- Sometimes, we want something to happen based on a comparison
  - ✓ For example: "**if a is less than b, then give c the value 22**"
- Syntax:

```
if (condition){  
    true block  
}
```

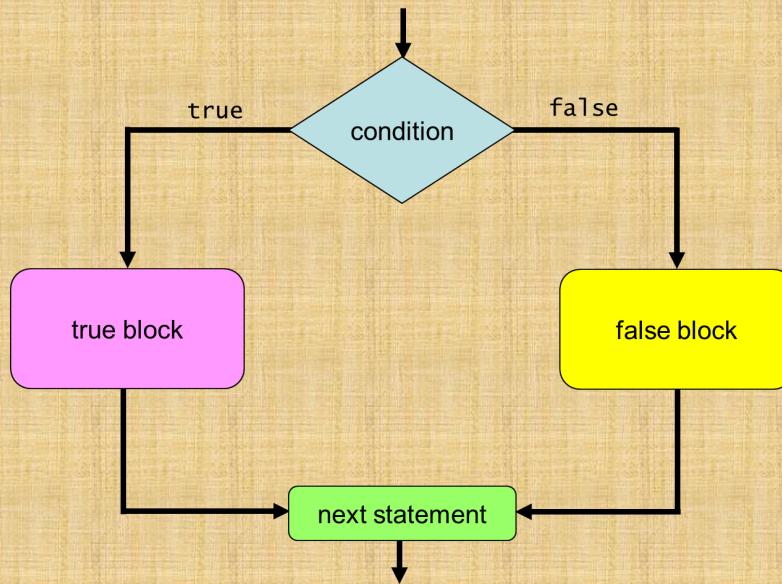
- Example: **if a is less than b, then give c the value 22**

```
if (a < b) {  
    c = 22;  
}
```



# Basics of JS Coding: Conditional Statements

- This is called simple decisions: if statements
- We have also two-way decisions: if-else statements



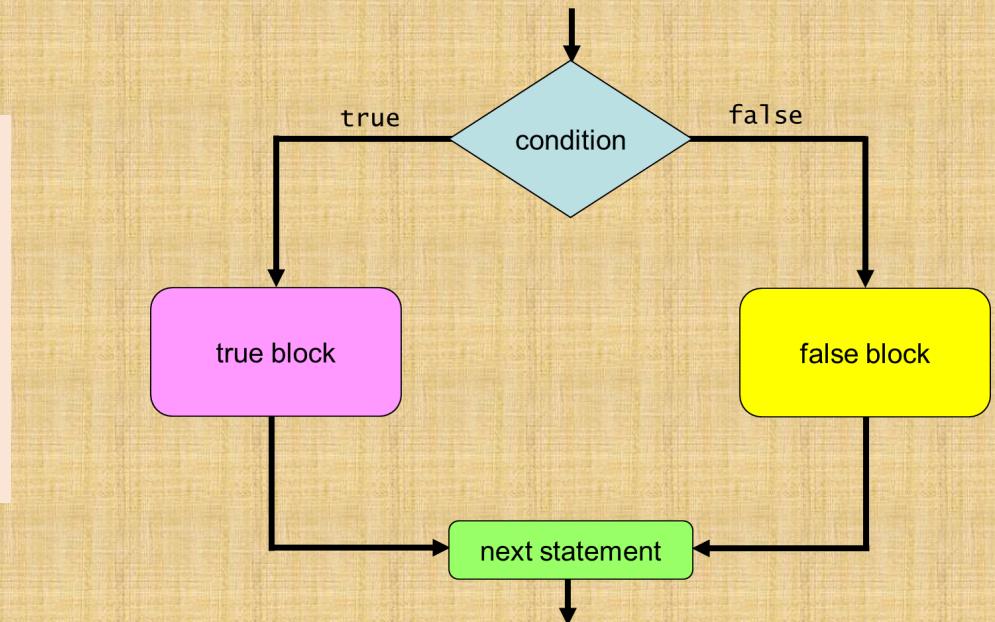
```
if (condition){  
    true block  
}
```

```
if (condition){  
    true block  
}  
else{  
    false block  
}
```

# Basics of JS Coding: Conditional Statements

For example: "if a is less than b, then give c the value 22, otherwise give c the value 9."

```
if (a < b) {  
    c = 22;  
}  
else {  
    c = 9;  
}
```



# Let's make an age-restricted website!!!

The screenshot shows a code editor with two tabs: `quiz-7-a.html` and `quiz-7-a.js`. The `quiz-7-a.html` file contains an HTML form to ask the user for their age, and a `script` tag linking to `quiz-7-a.js`. The `quiz-7-a.js` file contains a JavaScript function `checkAge()` that checks if the user is 18 or older, and displays a welcome message or restricted content accordingly.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>

    <!--Ask user for their age:-->
    <label for="input">How old Are you: </label><input id="input">

    <!--Use the checkAge() in JavaScript to see if user is above 18 or not.-->
    <button onclick="checkAge()">Check Age</button>

    <!--If the user is above 18 "Welcome" otherwise "You are not allowed to enter"-->
    <h3 id="output"></h3>
    

    <script src="quiz-7-a.js"></script>
  </body>
</html>
```

```
function checkAge() : void {
  let input=document.getElementById( elementId: "input").value;
  let output : HTMLElement =document.getElementById( elementId: "output");
  if(Number(input)>=18){
    output.innerHTML="Welcome";
    document.getElementById( elementId: "profile-pic").style.visibility="visible";
  }
  else {
    output.innerHTML="You are not allowed to see the content";
    document.getElementById( elementId: "profile-pic").style.visibility="hidden";
  }
}
```

Recall from Last Class:  
How operators work when values are different types?

```
let x1 = "Volvo" + 2;  
let x2 = 2 - "Volvo";  
let x3 = "Volvo" * 2;  
let x4 = 2 / "Volvo";
```

Recall from Last Class:

How operators work when values are different types?

```
let x1 = "Volvo" + 2;    answer:      x1 = "Volvo2"  
let x2 = 2 - "Volvo";  
let x3 = "Volvo" * 2;  
let x4 = 2 / "Volvo";
```

Recall from Last Class:  
How operators work when values are different types?

<code>let x1 = "Volvo" + 2;</code>	answer:	<code>x1 = "Volvo2"</code>
<code>let x2 = 2 - "Volvo";</code>	answer:	<code>x2 = NaN</code>
<code>let x3 = "Volvo" * 2;</code>	answer:	<code>x3 = NaN</code>
<code>let x4 = 2 / "Volvo";</code>	answer:	<code>x4 = NaN</code>

## Recall from Last Class: How operators work when values are different types?

<code>let x1 = "Volvo" + 2;</code>	answer:	<code>x1 = "Volvo2"</code>
<code>let x2 = 2 - "Volvo";</code>	answer:	<code>x2 = NaN</code>
<code>let x3 = "Volvo" * 2;</code>	answer:	<code>x3 = NaN</code>
<code>let x4 = 2 / "Volvo";</code>	answer:	<code>x4 = NaN</code>

<code>let x5 = 2 + "10";</code>
<code>let x6 = "10" - 2;</code>
<code>let x7 = 2 * "10";</code>
<code>let x8 = "10" / 2;</code>
<code>let x9 = "10" / "2";</code>

## Recall from Last Class: How operators work when values are different types?

```
let x1 = "Volvo" + 2;  
let x2 = 2 - "Volvo";  
let x3 = "Volvo" * 2;  
let x4 = 2 / "Volvo";
```

answer:      x1 = "Volvo2"  
                x2 = NaN  
                x3 = NaN  
                x4 = NaN

```
let x5 = 2 + "10";  
let x6 = "10" - 2;  
let x7 = 2 * "10";  
let x8 = "10" / 2;  
let x9 = "10" / "2";
```

answer:      x5 = "210"

## Recall from Last Class: How operators work when values are different types?

```
let x1 = "Volvo" + 2;  
let x2 = 2 - "Volvo";  
let x3 = "Volvo" * 2;  
let x4 = 2 / "Volvo";
```

answer: x1 = "Volvo2"  
answer: x2 = NaN  
answer: x3 = NaN  
answer: x4 = NaN

```
let x5 = 2 + "10";  
let x6 = "10" - 2;  
let x7 = 2 * "10";  
let x8 = "10" / 2;  
let x9 = "10" / "2";
```

answer: x5 = "210"  
answer: x6 = 8

Recall from Last Class:

How operators work when values are different types?

```
let x1 = "Volvo" + 2;  
let x2 = 2 - "Volvo";  
let x3 = "Volvo" * 2;  
let x4 = 2 / "Volvo";
```

answer:	x1 = "Volvo2"
answer:	x2 = NaN
answer:	x3 = NaN
answer:	x4 = NaN

```
let x5 = 2 + "10";  
let x6 = "10" - 2;  
let x7 = 2 * "10";  
let x8 = "10" / 2;  
let x9 = "10" / "2";
```

answer:	x5 = "210"
answer:	x6 = 8
answer:	x7 = 20

Recall from Last Class:

How operators work when values are different types?

```
let x1 = "Volvo" + 2;  
let x2 = 2 - "Volvo";  
let x3 = "Volvo" * 2;  
let x4 = 2 / "Volvo";
```

answer:	x1 = "Volvo2"
answer:	x2 = NaN
answer:	x3 = NaN
answer:	x4 = NaN

```
let x5 = 2 + "10";  
let x6 = "10" - 2;  
let x7 = 2 * "10";  
let x8 = "10" / 2;  
let x9 = "10" / "2";
```

answer:	x5 = "210"
answer:	x6 = 8
answer:	x7 = 20
answer:	x8 = 5

Recall from Last Class:

How operators work when values are different types?

```
let x1 = "Volvo" + 2;  
let x2 = 2 - "Volvo";  
let x3 = "Volvo" * 2;  
let x4 = 2 / "Volvo";
```

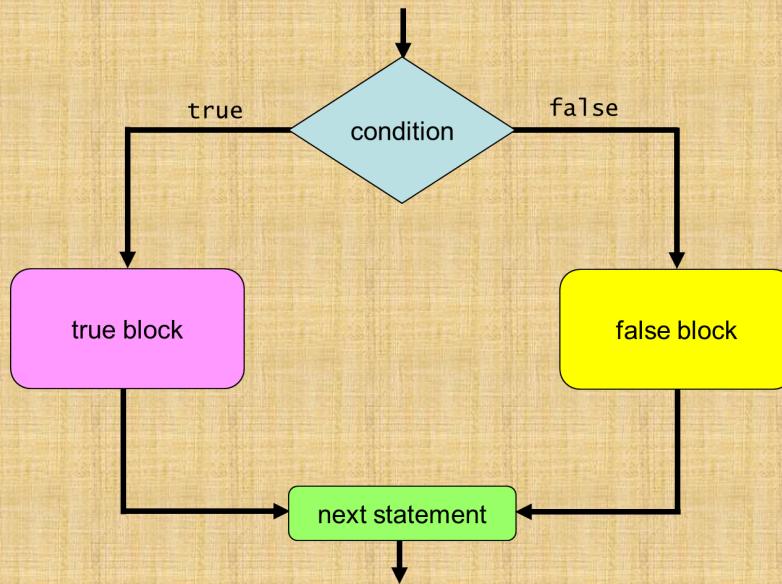
answer:	x1 = "Volvo2"
answer:	x2 = NaN
answer:	x3 = NaN
answer:	x4 = NaN

```
let x5 = 2 + "10";  
let x6 = "10" - 2;  
let x7 = 2 * "10";  
let x8 = "10" / 2;  
let x9 = "10" / "2";
```

answer:	x5 = "210"
answer:	x6 = 8
answer:	x7 = 20
answer:	x8 = 5
answer:	x9 = 5

# Basics of JS Coding: Conditional Statements

- This is called simple decisions: if statements
- We have also two-way decisions: if-else statements



```
if (condition){  
    true block  
}
```

```
if (condition){  
    true block  
}  
else{  
    false block  
}
```

How many lines does this print in the console?

```
let x = 5;
if(x == 8){console.log("how");}
if(x > 1){console.log("now");}
if(x < 20){console.log("wow");}
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

## How many lines does this print in the console?

```
let x = 5;  
if(x == 8){console.log("how");}  
if(x > 1){console.log("now");}  
if(x < 20){console.log("wow");}  
console.log("cow");
```

- A. 0
- B. 1
- C. 2
-  D. 3
- E. 4

How many lines does this print in the console?  
(note the changes)

```
let x = 5;
if(x == 5){console.log("how");}
else{console.log("now");}
if(x < 20){console.log("wow");}
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

How many lines does this print in the console?  
(note the changes)

```
let x = 5;  
if(x == 5){console.log("how");}  
else{console.log("now");}  
if(x < 20){console.log("wow");}  
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

How many lines does this print in the console?  
(note the changes)

```
let x = 5;  
if(x == 5){console.log("how");}  
else{console.log("now");}  
if(x < 20){console.log("wow");}  
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4



# Let's make an age-restricted website!!!

The fourth requirement:

1. This website should show a pop-up warning message about age.
2. Then, there has to be another pop-up message to ask user's age.
3. Based on the answer, another message will be asked to show that if the user is or is not allowed to enter the website.
4. **Also, if the user put an unacceptable answer, ask the user to put an acceptable answer.**

➤ What kind of answers are not acceptable?

- ✓ Alphabets
- ✓ Negative numbers

## isNaN()

To do the fourth requirement, we need to use a new command:

- isNaN(x) returns **true** if x is not a number, otherwise returns **false**

Examples:

- isNaN(10) returns False
- isNaN("10") returns True
- isNaN("Vahid") returns True
- isNaN("S10") returns True

## isNaN()

To do the fourth requirement, we need to use a new command:

- isNaN(x) returns **true** if x is not a number, otherwise returns **false**

Examples:

- isNaN(10) returns False
- isNaN("10") returns True
- isNaN("Vahid") returns True
- isNaN("S10") returns True

**The data type of the answer is text (string) even if the user's answer is a number.**

## isNaN()

To do the fourth requirement, we need to use a new command:

- isNaN(x) returns **true** if x is not a number, otherwise returns **false**

Examples:

- isNaN(10) returns False
- isNaN("10") returns True
- isNaN("Vahid") returns True
- isNaN("S10") returns True

**The data type of the answer is text (string) even if the user's answer is a number.**

- We must apply command Number() to change the data type.

## isNaN()

To do the fourth requirement, we need to use a new command:

- isNaN(x) returns **true** if x is not a number, otherwise returns **false**

Examples:

- isNaN(10) returns False
- isNaN("10") returns True
- isNaN("Vahid") returns True
- isNaN("S10") returns True

**The data type of the answer is text (string) even if the user's answer is a number.**

- We must apply command Number() to change the data type.

Examples:

- isNaN(Number("10")) returns ???
- isNaN(Number("S10")) returns ???

## isNaN()

To do the fourth requirement, we need to use a new command:

- isNaN(x) returns **true** if x is not a number, otherwise returns **false**

Examples:

- isNaN(10) returns False
- isNaN("10") returns True
- isNaN("Vahid") returns True
- isNaN("S10") returns True

The data type of the answer is text (string) even if the user's answer is a number.

- We must apply command Number() to change the data type.

Examples:

- isNaN(Number("10")) returns False
- isNaN(Number("S10")) returns True

## isNaN()

To do the fourth requirement, we need to use a new command:

- isNaN(x) returns **true** if x is not a number, otherwise returns **false**

Examples:

- isNaN(10) returns False
- isNaN("10") returns True
- isNaN("Vahid") returns True
- isNaN("S10") returns True

The data type of the answer is text (string) even if the user's answer is a number.

- We must apply command Number() to change the data type.

Examples:

- isNaN(Number("10")) returns False      Number("10") = 10    isNaN(10) = False
- isNaN(Number("S10")) returns True

## isNaN()

To do the fourth requirement, we need to use a new command:

- isNaN(x) returns **true** if x is not a number, otherwise returns **false**

Examples:

- isNaN(10) returns False
- isNaN("10") returns True
- isNaN("Vahid") returns True
- isNaN("S10") returns True

The data type of the answer is text (string) even if the user's answer is a number.

- We must apply command Number() to change the data type.

Examples:

- isNaN(Number("10")) returns False      Number("10") = 10      isNaN(10) = False
- isNaN(Number("S10")) returns True      Number("S10") = NaN      isNaN(NaN) = True

# Basics of JS Coding: Logical Operators

- JS provides a set of **logical operators** for combining/modifying Boolean expressions:

Operator	Description	Example and meaning
&&	and	<code>age &gt; 18 &amp;&amp; age &lt; 35</code> <code>true</code> if both conditions are <code>true</code> , and <code>false</code> otherwise
	or	<code>age &lt; 3    age &gt; 65</code> <code>true</code> if one or both of the conditions are <code>true</code> ; <code>false</code> if both conditions are <code>false</code>
!	not	<code>! (grade &gt; 80)</code> <code>true</code> if the condition is <code>false</code> , and <code>false</code> if it is <code>true</code>

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```

answer: ?

answer: ?

answer: ?

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```

false  
a > b || b === d

answer: ?  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;  
let b = 16;  
let c = "12";  
let d = 16;  
console.log(a > b || b === d);  
console.log(!(a == c));  
console.log(!(a === c));  
console.log(!(a === c) && a < d);
```

false      or  
 a > b || b === d

answer: ?

answer: ?

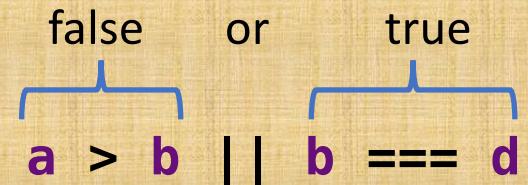
answer: ?

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;  
let b = 16;  
let c = "12";  
let d = 16;  
console.log(a > b || b === d);  
console.log(!(a == c));  
console.log(!(a === c));  
console.log(!(a === c) && a < d);
```



answer: ?

answer: ?

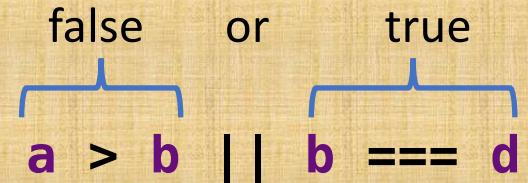
answer: ?

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;  
let b = 16;  
let c = "12";  
let d = 16;  
console.log(a > b || b === d);  
console.log(!(a == c));  
console.log(!(a === c));  
console.log(!(a === c) && a < d);
```



answer: true

answer: ?

answer: ?

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```

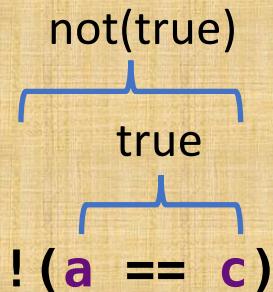
true  
  !(a == c)

answer: true  
answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```



answer: true

answer: ?

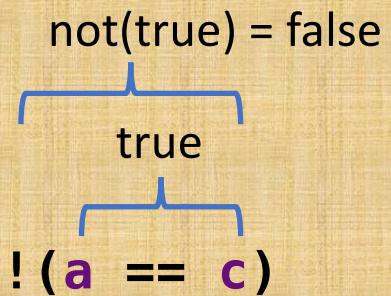
answer: ?

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```

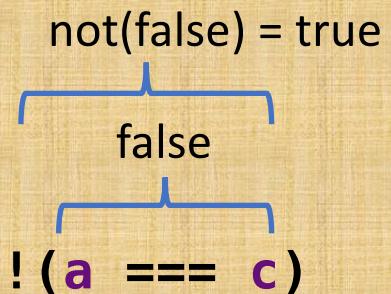


answer: true  
answer: false  
answer: ?  
answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```



answer: **true**  
answer: **false**  
answer: **true**  
answer: **?**

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```

`!(a === c) && a < d`

answer: `true`

answer: `false`

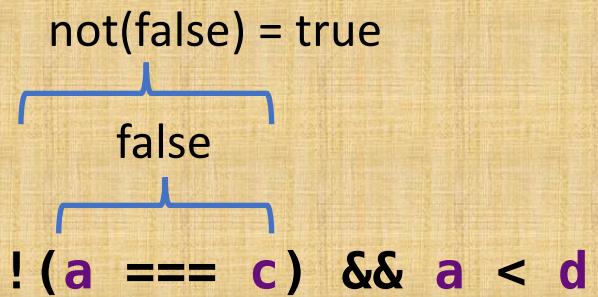
answer: `true`

answer: `?`

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```



answer: true

answer: false

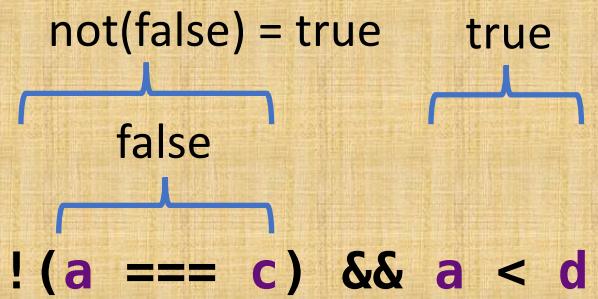
answer: true

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```



answer: true

answer: false

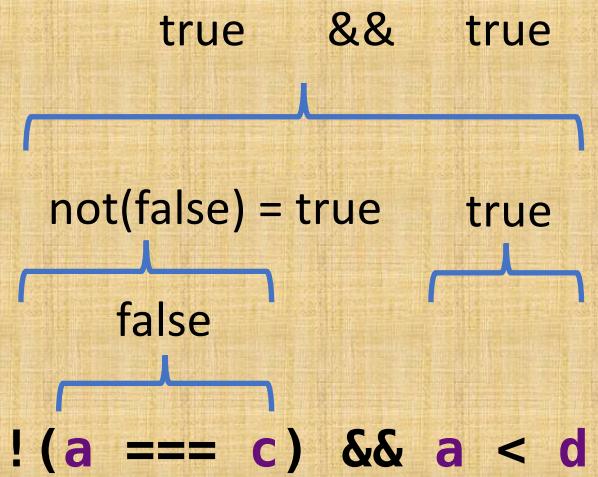
answer: true

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```



answer: true

answer: false

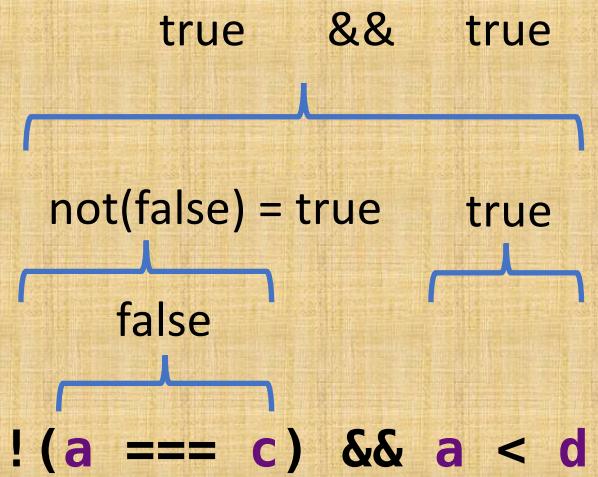
answer: true

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```



answer: true

answer: false

answer: true

answer: true

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = 16;
let c = "12";
let d = 16;
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(a === c));
console.log(!(a === c) && a < d);
```

answer: true  
answer: false  
answer: true  
answer: true

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```

answer: ?

answer: ?

answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```

12 > "16"      "16" === "g"  
false      or      false  


answer: ?  
answer: ?  
answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```

12 > "16"      "16" === "g"  
false      or      false  


answer: **false**  
answer: ?  
answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```

`!(a == c)` `12 == "a"`

answer: `false`

answer: `?`

answer: `?`

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```

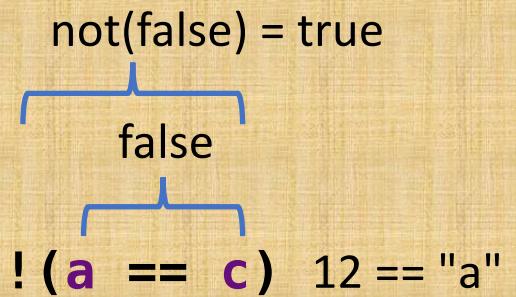
false  
! (a == c) 12 == "a"

answer: false  
answer: ?  
answer: ?

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```



answer: **false**

answer: **true**

answer: **?**

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```

`!(b === c) && a < d`  
`"16" === "a"`

answer: `false`

answer: `true`

answer: `?`

# Basics of JS Coding: Logical Operators

- Comparison will be done based on ASCII code of the characters.

- An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```

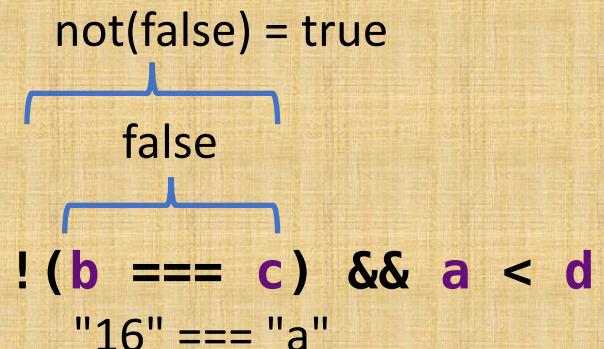
false  
! (b === c) && a < d  
"16" === "a"

answer: **false**  
answer: **true**  
answer: **?**

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```



answer: **false**

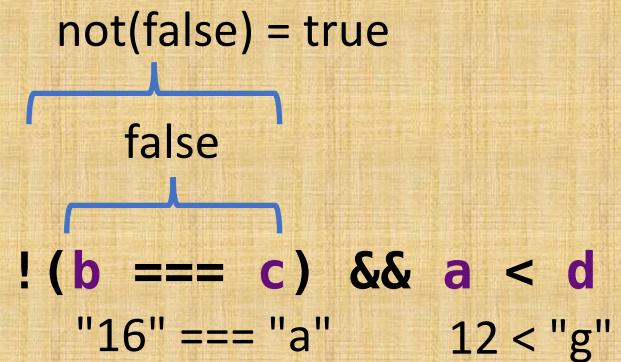
answer: **true**

answer: **?**

# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```



answer: **false**

answer: **true**

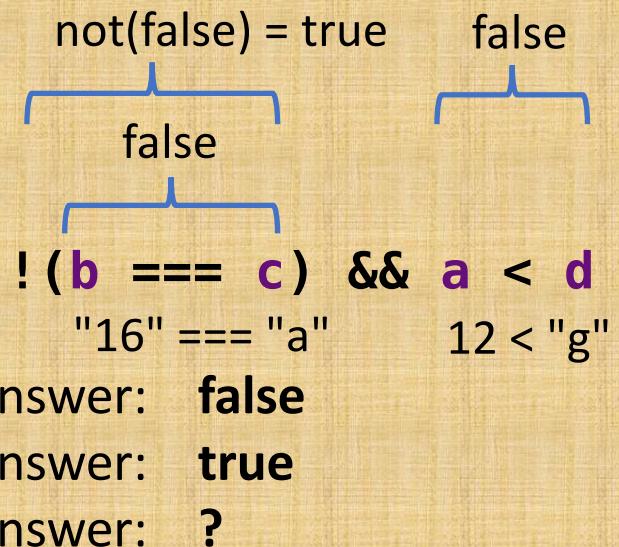
answer: **?**

# Basics of JS Coding: Logical Operators

- when an expression does not make sense, JS gives **false**.

- An example:

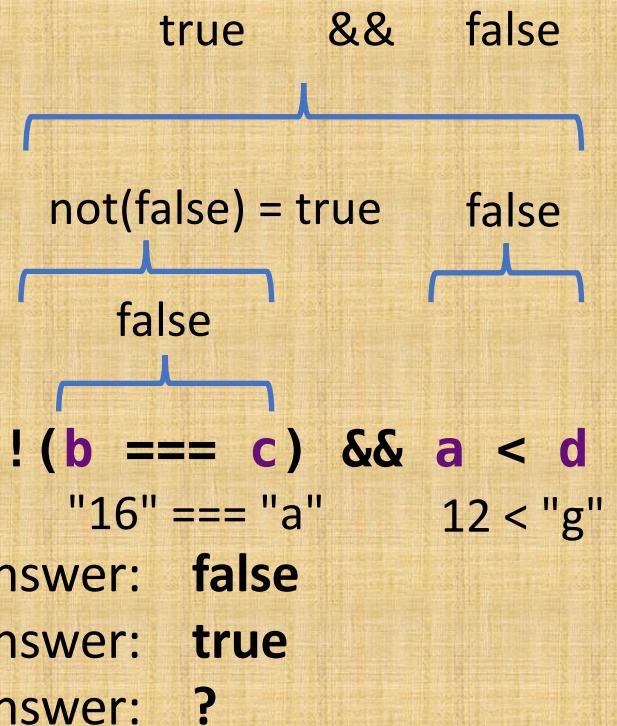
```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```



# Basics of JS Coding: Logical Operators

➤ An example:

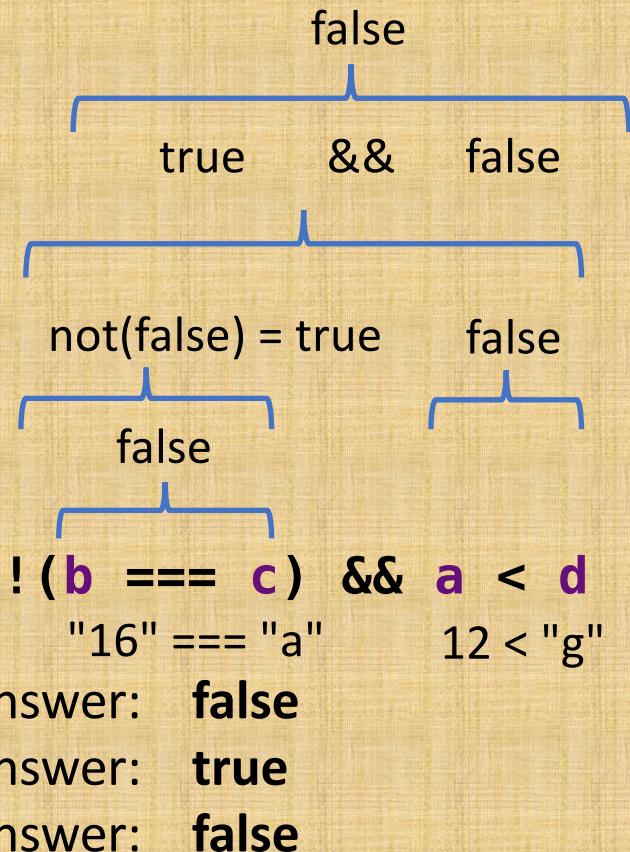
```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```



# Basics of JS Coding: Logical Operators

➤ An example:

```
let a = 12;
let b = "16";
let c = "a";
let d = "g";
console.log(a > b || b === d);
console.log(!(a == c));
console.log(!(b === c) && a < d);
```



## Basics of JS Coding:

### Making more complex conditional statements

➤example: "if a is less than b and both of them are greater than 20, then give c the value 22"

```
if () {  
    c = 22;  
}
```

# Basics of JS Coding:

## Making more complex conditional statements

➤example: "if a is less than b and both of them are greater than 20, then give c the value 22"

```
if (a < b) {  
    c = 22;  
}
```

## Basics of JS Coding:

### Making more complex conditional statements

➤example: "if a is less than b and both of them are greater than 20, then give c the value 22"

```
if (a < b && b > 20) {  
    c = 22;  
}
```

## Basics of JS Coding:

### Making more complex conditional statements

➤example: "if a is less than b and both of them are greater than 20, then give c the value 22"

```
if (a < b && b > 20 && a > 20) {  
    c = 22;  
}
```

## Basics of JS Coding: Making more complex conditional statements

- example: "if a is less than b and both of them are greater than 20, then give c the value 22"

```
if (a < b && b > 20 && a > 20) {  
    c = 22;  
}
```

- Is there any better way to do that (using one && operator instead of two)?

## Basics of JS Coding: Making more complex conditional statements

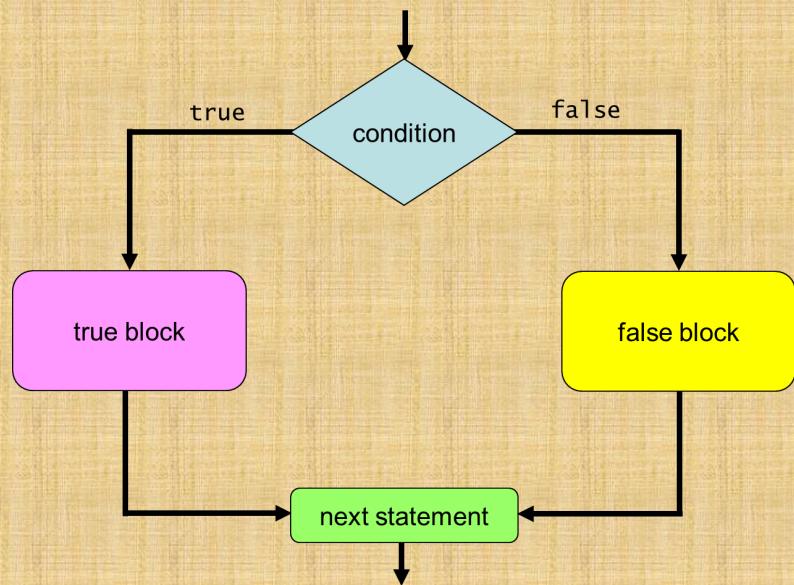
➤example: "if a is less than b and both of them are greater than 20, then give c the value 22"

```
if (a < b && b > 20 && a > 20) {  
    c = 22;  
}
```

➤Is there any better way to do that (using one && operator instead of two)?

```
if (a > 20 && a < b ) {  
    c = 22;  
}
```

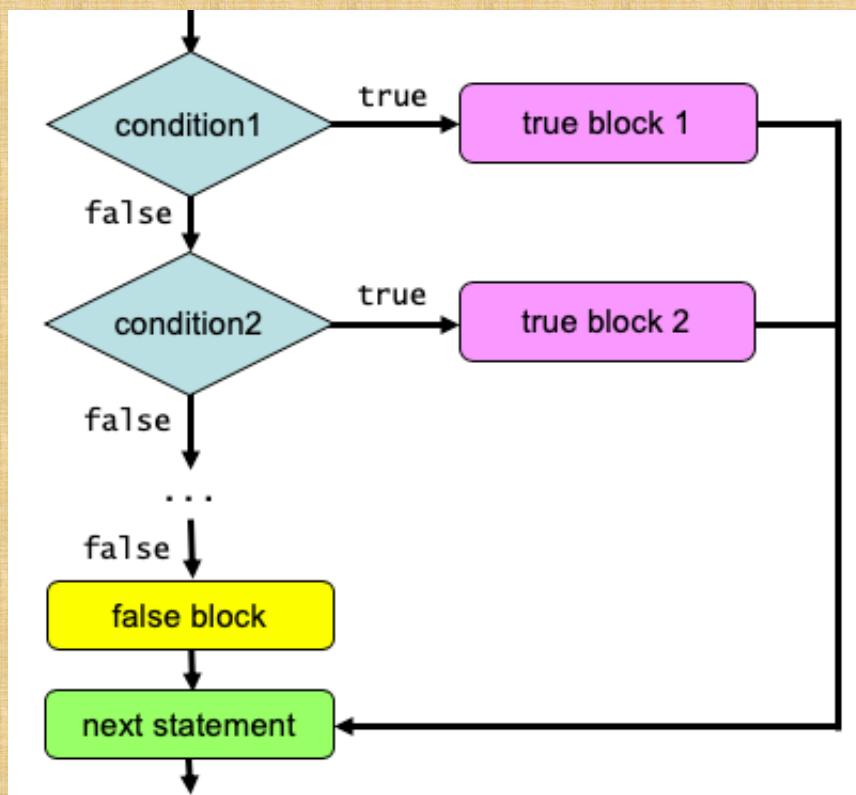
# Basics of JS Coding: One-way & Two-way Decisions



```
if (condition) {  
    true block  
}
```

```
if (condition) {  
    true block  
}  
else {  
    false block  
}
```

# Basics of JS Coding: Multi-way Decisions: if - else if - else Statements



```
if (condition1){  
    true block for condition 1  
}  
else if (condition2){  
    true block for condition 2  
}  
else if (condition3){  
    true block for condition 3  
}  
...  
else{  
    false block  
}
```

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;  
let grade;  
if (ave >= 94) {grade = "A";}  
if (ave >= 90) {grade = "A-";}  
if (ave >= 87) {grade = "B+";}  
if (ave >= 83) {grade = "B";}  
if (ave >= 80) {grade = "B-";}  
if (ave >= 77) {grade = "C+";}  
if (ave >= 73) {grade = "C";}  
if (ave >= 70) {grade = "C-";}  
if (ave >= 65) {grade = "D";}  
else {grade = "F";}  
console.log(grade)
```

grade  
"A"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"
"B"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"
"B"
"B-"
"C+"
"C"
"C-"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"
"B"
"B-"
"C+"
"C"
"C-"
"D" else is not executed

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

Answer: "D"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
else if (ave >= 90) {grade = "A-";}
else if (ave >= 87) {grade = "B+";}
else if (ave >= 83) {grade = "B";}
else if (ave >= 80) {grade = "B-";}
else if (ave >= 77) {grade = "C+";}
else if (ave >= 73) {grade = "C";}
else if (ave >= 70) {grade = "C-";}
else if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;  
let grade;  
if (ave >= 94) {grade = "A";}  
else if (ave >= 90) {grade = "A-";}  
else if (ave >= 87) {grade = "B+";}  
else if (ave >= 83) {grade = "B";}  
else if (ave >= 80) {grade = "B-";}  
else if (ave >= 77) {grade = "C+";}  
else if (ave >= 73) {grade = "C";}  
else if (ave >= 70) {grade = "C-";}  
else if (ave >= 65) {grade = "D";}  
else {grade = "F";}  
console.log(grade)
```

grade

"A"

exit the multiway  
decision

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example: `let ave = 95;`

```
let grade;
if (ave >= 94) {grade = "A";}
else if (ave >= 90) {grade = "A-";}
else if (ave >= 87) {grade = "B+";}
else if (ave >= 83) {grade = "B";}
else if (ave >= 80) {grade = "B-";}
else if (ave >= 77) {grade = "C+";}
else if (ave >= 73) {grade = "C";}
else if (ave >= 70) {grade = "C-";}
else if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

Answer: "A"

How many lines does this print in the console?

```
let x = 5;
if(x == 8){console.log("how");}
else if(x > 1){console.log("now");}
else if(x < 20){console.log("wow");}
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

## How many lines does this print in the console?

```
let x = 5;  
if(x == 8){console.log("how");}  
else if(x > 1){console.log("now");}  
else if(x < 20){console.log("wow");}  
console.log("cow");
```

- A. 0
- B. 1
-  C. 2
- D. 3
- E. 4

How many lines does this print in the console?  
(note the changes)

```
let x = 5;
if(x == 8){console.log("how");}
if(x > 1){console.log("now");}
if(x < 20){console.log("wow");}
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

How many lines does this print in the console?  
(note the changes)

```
let x = 5;  
if(x == 8){console.log("how");}  
if(x > 1){console.log("now");}  
if(x < 20){console.log("wow");}  
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

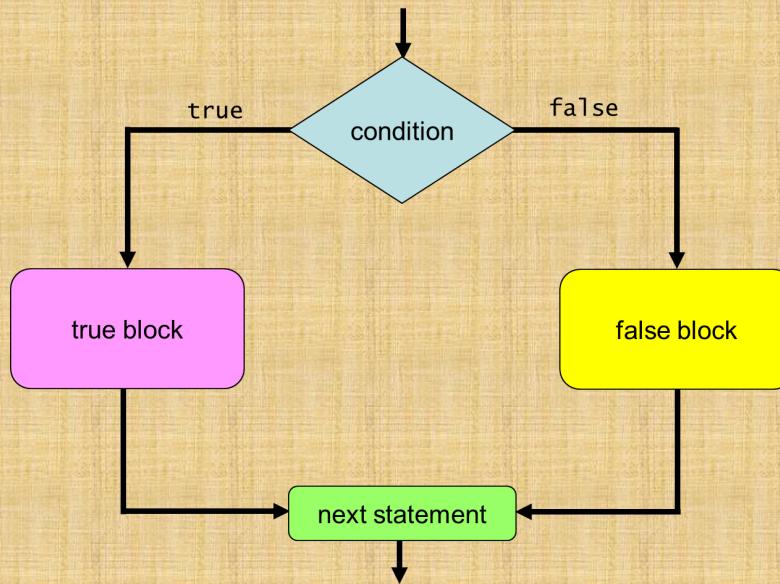


# Let's make an age-restricted website!!!

```
function checkAge(): void {
    let input=document.getElementById(elementId: "input").value;
    let output : HTMLElement =document.getElementById(elementId: "output");
    if (isNaN(input)) {
        output.innerHTML="Your answer is not acceptable";
        console.log("user entered a none-integer input")
    }
    if (input < 0) {
        output.innerHTML="Your answer is not acceptable";
        console.log("user entered a negative integer")
    }
    if (input > 17 && input < 150) {
        output.innerHTML="Welcome";
        console.log("user entered an acceptable input")
    }
    else {
        output.innerHTML="You are not allowed to see the content";
    }
}
```

# Basics of JS Coding: Conditional Statements

- This is called simple decisions: if statements
- We have also two-way decisions: if-else statements



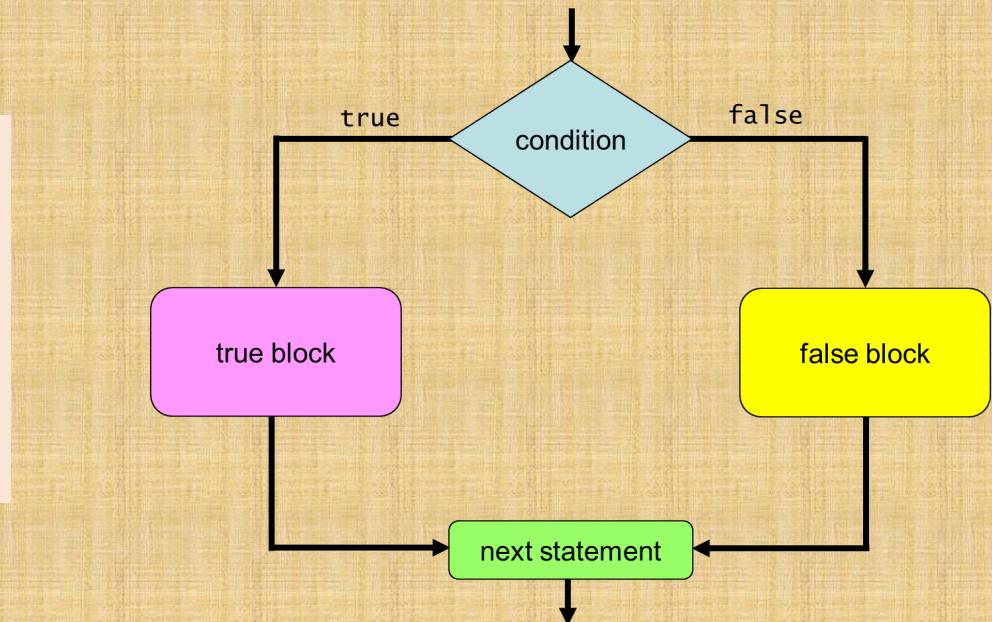
```
if (condition){  
    true block  
}
```

```
if (condition){  
    true block  
}  
else{  
    false block  
}
```

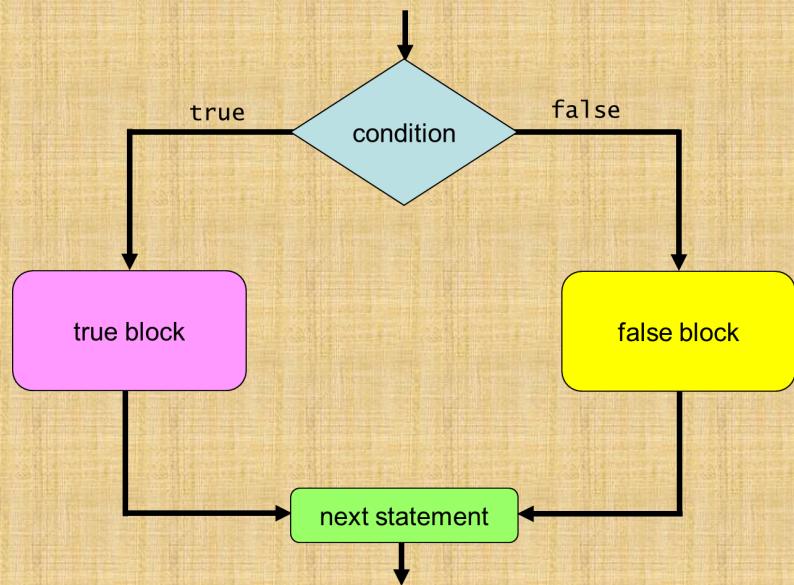
# Basics of JS Coding: Conditional Statements

For example: "if a is less than b, then give c the value 22, otherwise give c the value 9."

```
if (a < b) {  
    c = 22;  
}  
else {  
    c = 9;  
}
```



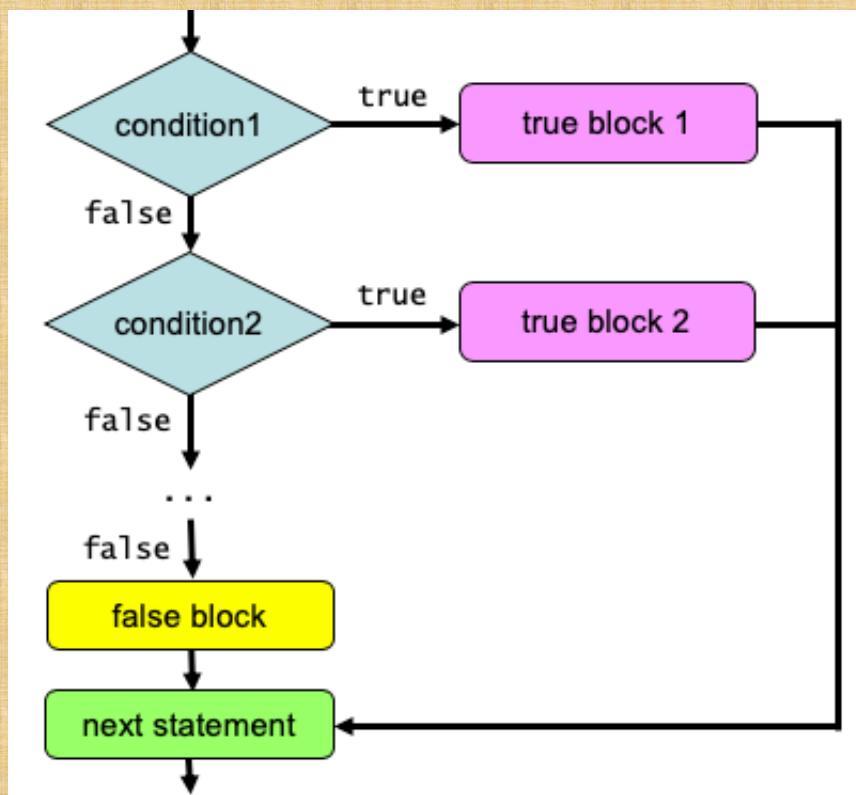
# Basics of JS Coding: One-way & Two-way Decisions



```
if (condition) {  
    true block  
}
```

```
if (condition) {  
    true block  
}  
else {  
    false block  
}
```

# Basics of JS Coding: Multi-way Decisions: if - else if - else Statements



```
if (condition1){  
    true block for condition 1  
}  
else if (condition2){  
    true block for condition 2  
}  
else if (condition3){  
    true block for condition 3  
}  
...  
else{  
    false block  
}
```

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade  
"A"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"
"B"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"
"B"
"B-"
"C+"
"C"
"C-"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

grade
"A"
"A-"
"B+"
"B"
"B-"
"C+"
"C"
"C-"
"D" else is not executed

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
if (ave >= 90) {grade = "A-";}
if (ave >= 87) {grade = "B+";}
if (ave >= 83) {grade = "B";}
if (ave >= 80) {grade = "B-";}
if (ave >= 77) {grade = "C+";}
if (ave >= 73) {grade = "C";}
if (ave >= 70) {grade = "C-";}
if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

Answer: "D"

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
else if (ave >= 90) {grade = "A-";}
else if (ave >= 87) {grade = "B+";}
else if (ave >= 83) {grade = "B";}
else if (ave >= 80) {grade = "B-";}
else if (ave >= 77) {grade = "C+";}
else if (ave >= 73) {grade = "C";}
else if (ave >= 70) {grade = "C-";}
else if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;  
let grade;  
if (ave >= 94) {grade = "A";}  
else if (ave >= 90) {grade = "A-";}  
else if (ave >= 87) {grade = "B+";}  
else if (ave >= 83) {grade = "B";}  
else if (ave >= 80) {grade = "B-";}  
else if (ave >= 77) {grade = "C+";}  
else if (ave >= 73) {grade = "C";}  
else if (ave >= 70) {grade = "C-";}  
else if (ave >= 65) {grade = "D";}  
else {grade = "F";}  
console.log(grade)
```

grade

"A"

exit the multiway  
decision

## Basics of JS Coding:

### Multi-way Decisions: if - else if - else Statements

➤ Example:

```
let ave = 95;
let grade;
if (ave >= 94) {grade = "A";}
else if (ave >= 90) {grade = "A-";}
else if (ave >= 87) {grade = "B+";}
else if (ave >= 83) {grade = "B";}
else if (ave >= 80) {grade = "B-";}
else if (ave >= 77) {grade = "C+";}
else if (ave >= 73) {grade = "C";}
else if (ave >= 70) {grade = "C-";}
else if (ave >= 65) {grade = "D";}
else {grade = "F";}
console.log(grade)
```

Answer: "A"

How many lines does this print in the console?

```
let x = 5;
if(x == 8){console.log("how");}
else if(x > 1){console.log("now");}
else if(x < 20){console.log("wow");}
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

## How many lines does this print in the console?

```
let x = 5;  
if(x == 8){console.log("how");}  
else if(x > 1){console.log("now");}  
else if(x < 20){console.log("wow");}  
console.log("cow");
```

- A. 0
- B. 1
-  C. 2
- D. 3
- E. 4

How many lines does this print in the console?  
(note the changes)

```
let x = 5;
if(x == 8){console.log("how");}
if(x > 1){console.log("now");}
if(x < 20){console.log("wow");}
console.log("cow");
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

How many lines does this print in the console?  
(note the changes)

```
let x = 5;
if(x == 8){console.log("how");}
if(x > 1){console.log("now");}
if(x < 20){console.log("wow");}
console.log("cow");
```

- A. 0
- B. 1
- C. 2
-  D. 3
- E. 4

## What will be printed in the console?

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
else{
    if(x > 2){console.log("Three");}
}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
else{if(x > 2){console.log("Three");}}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
else{if(x > 2){console.log("Three");}}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
else{[if(x > 2){console.log("Three");}]}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){ true
    [
        if(x > 8){console.log("One");}
        else{console.log("Two");}
    ]
else[if(x > 2){console.log("Three");}]}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What will be printed in the console?

```
let x = 5;
if(x < 15){ true
  [
    if(x > 8){console.log("One");}
    else{console.log("Two");}
  ]
else{[if(x > 2){console.log("Three");}]}  
_____
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){ true
    [
        if(x > 8){console.log("One");}
        else{console.log("Two");}
    ]
else{[if(x > 2){console.log("Three");}]} }
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){ true
  [
    if(x > 8){ [console.log("One");]
      [
        else{ [console.log("Two");]
          ]
        }
      ]
    else{ [if(x > 2){console.log("Three");}]}
  ]
}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){ true
  [ false if(x > 8){ [console.log("One"); ]
    [   else{ [console.log("Two"); ]
      }
  ]
else{ [if(x > 2){console.log("Three"); } ] }
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){ true
  [ false if(x > 8){ [ console.log("One"); }
    [ else{ [ console.log("Two"); ]
      ]
  else{ [ if(x > 2){ console.log("Three"); } ] }
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){ true
  [ false if(x > 8){ [ console.log("One"); }
    [   else{ [ console.log("Two"); ]
      ]
  else{ [ if(x > 2){ console.log("Three"); } ] }
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){ true
    [ false if(x > 8){ [ console.log("One"); ]
        [ else{ [ console.log("Two"); ]
            }
    ]
else{ [ if(x > 2){ console.log("Three"); } ]
// program would go here next...
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

## What will be printed in the console?

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
else{if(x > 2){console.log("Three");}}
```

- A. One
-  B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");}
}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");
}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
    else{console.log("Two");
}
if(x > 2){console.log("Three");
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){ console.log("One"); }
    else{ console.log("Two"); }
}
if(x > 2){ console.log("Three"); }
```

- A. One
- B. Two
- C. Three
-  D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the new changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
}
else{console.log("Two");}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the new changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
}
else{console.log("Two");}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
- C. Three
- D. More than one of the above
- E. Nothing is printed

What does this print in the console?  
(note the new changes!)

```
let x = 5;
if(x < 15){
    if(x > 8){console.log("One");}
}
else{console.log("Two");}
if(x > 2){console.log("Three");}
```

- A. One
- B. Two
-  C. Three
- D. More than one of the above
- E. Nothing is printed

# Lets make an age restricted website!!!

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>

    <!--Ask user for their age:-->
    <label for="input">How old Are you: </label><input id="input">

    <!--Use the checkAge() in JavaScript to see if user is above 18 or not.-->
    <button onclick="checkAge()">Check Age</button>

    <!--If the user is above 18 "Welcome" otherwise "You are not allowed to enter"-->
    <h3 id="output"></h3>
    

    <script src="quiz-7-a.js"></script>
  </body>
</html>
```

# Lets make an age restricted website!!!

The image shows a code editor on the left and a browser window on the right. The code editor displays a file named `quiz-7-a.js` with the following content:

```
1 usage
function checkAge() : void {
    let age = document.getElementById("input").value;
    let output : HTMLElement = document.getElementById( elementId: "output");

    if (Number(age) >= 18 && Number(age) < 150) {
        output.innerHTML = "Welcome";
        document.getElementById( elementId: "profile-pic").style.visibility = "visible";
    }

    if (Number(age) < 0) {
        output.innerHTML = "You gave me a negative number, stop being silly";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }

    if (isNaN(age)) {
        output.innerHTML = "You didn't give me a number, you silly goose";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }

    if (Number(age) % 1 !== 0) {
        output.innerHTML = "You gave me a decimal, you so and so";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }
    else{
        output.innerHTML = "You are not allowed to see the content";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }
}
```

The browser window on the right shows a simple form with the text "How old Are you:" followed by an input field and a "Check Age" button. The URL in the address bar is `localhost:63342/cs-103-quiz-7/c`.

# Lets make an age restricted website!!!

The image shows a code editor on the left and a web browser on the right. The code editor displays the file `quiz-7-a.js` with the following content:

```
1 usage
function checkAge() : void {
    let age = document.getElementById( elementId: "input").value;
    let output : HTMLElement = document.getElementById( elementId: "output");

    if (Number(age) >= 18 && Number(age) < 150) {
        output.innerHTML = "Welcome";
        document.getElementById( elementId: "profile-pic").style.visibility = "visible";
    }

    else if (Number(age) < 0) {
        output.innerHTML = "You gave me a negative number, stop being silly";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }

    else if (isNaN(age)) {
        output.innerHTML = "You didn't give me a number, you silly goose";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }

    else if (Number(age) % 1 !== 0) {
        output.innerHTML = "You gave me a decimal, you so and so";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }
    else{
        output.innerHTML = "You are not allowed to see the content";
        document.getElementById( elementId: "profile-pic").style.visibility = "hidden";
    }
}
```

The browser window on the right shows a simple form with the title "Title". The form has a label "How old Are you:" followed by an input field and a button labeled "Check Age".

# Lets make an age restricted website!!!

The image displays a development environment with two windows side-by-side. On the left is a code editor showing a JavaScript file named `quiz-7-a.js`. The code contains a function `checkAge()` that performs several checks on the input age and updates an output element accordingly. On the right is a web browser window displaying a simple form with the title "How old Are you:" and a text input field followed by a "Check Age" button.

```
1 usage
27 function checkAge() : void {
28     let age = document.getElementById("input").value;
29     let output : HTMLElement = document.getElementById("output");
30
31     if (Number(age) >= 18 && Number(age) < 150 && Number(age) % 1 === 0) {
32         output.innerHTML = "Welcome";
33         document.getElementById("profile-pic").style.visibility = "visible";
34     }
35
36     else if (Number(age) < 0) {
37         output.innerHTML = "You gave me a negative number, stop being silly";
38         document.getElementById("profile-pic").style.visibility = "hidden";
39     }
40
41     else if (isNaN(age)) {
42         output.innerHTML = "You didn't give me a number, you silly goose";
43         document.getElementById("profile-pic").style.visibility = "hidden";
44     }
45
46     else if (Number(age) % 1 !== 0) {
47         output.innerHTML = "You gave me a decimal, you so and so";
48         document.getElementById("profile-pic").style.visibility = "hidden";
49     }
50     else{
51         output.innerHTML = "You are not allowed to see the content";
52         document.getElementById("profile-pic").style.visibility = "hidden";
53     }
54 }
55
56
57 }
```

# Lets make a password protected website!!!

The image shows a development environment with two main windows. On the left is a code editor window titled 'test' containing the file 'index.html'. The code in 'index.html' is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link type="text/css" rel="stylesheet" href="index.css">
    <title>child | parent</title>
  </head>
  <body>

    <div id="page-wrapper">

      <div id="main-content">
        <h2>Checking:</h2>
        <h3>Balance:</h3>
        <h1>$5000</h1>
      </div>

      <div id="authentication">
        <Label for="username">User id: </label><input id="username">
        <Label for="password">Password: </label><input id="password">
        <button id="check" onclick="check()">Login</button>
        <p id="message"> </p>
      </div>

    </div>

    <script src="index.js"></script>

  </body>
</html>
```

On the right is a browser window titled 'child | parent' showing the URL 'localhost:63342/test/class-test/'. The page displays a login form with fields for 'User id:' and 'Password:', and a 'Login' button.

# Lets make a password protected website!!!

The screenshot shows a developer's workspace with a code editor, browser, and terminal.

**Code Editor:** The left panel displays the `index.js` file:

```
1 let userName;
2 let password;
3 let limit : number = 0;
...
1 usage
2 function check(): void {
3     userName = document.getElementById("username").value;
4     password = document.getElementById("password").value;
5
6     if (userName === "Jack" && password === "jack-123") {
7         document.getElementById("main-content").style.visibility = "visible";
8     } else {
9         limit++;
10        document.getElementById("message").innerHTML =
11            `The username "${userName}" and the password "${password}" are not in our records`;
12
13        document.getElementById("username").value = "";
14        document.getElementById("password").value = "";
15        console.log(`The Limit: ${limit}`);
16
17        if (limit === 3) {
18            document.getElementById("check").disabled = true;
19            document.getElementById("message").innerHTML = "You have reached the maximum login attempts.";
20        }
21    }
22
23
24
25 }
```

**Browser:** The right panel shows a browser window at `localhost:63342/test/class-test/index.htm`. It contains a login form with fields for User id and Password, and a "Login" button. Below the form, a message states: "The username "blah" and the password "blah" are not in our records".

**Terminal:** The bottom right panel is a terminal window titled "Console". It shows the output of the `console.log` statement from the code: "The Limit:1". The timestamp "index.js:19:17" is also visible.

# Lets make a password protected website!!!

The screenshot shows a developer's workspace with several windows open:

- Left Panel:** A code editor window titled "index.js" showing the following JavaScript code:

```
1 let userName;
2 let password;
3 let limit : number = 0;
...
11
12 function check(): void {
13     userName = document.getElementById("username").value;
14     password = document.getElementById("password").value;
15
16     if (userName === "Jack" && password === "jack-123") {
17         document.getElementById("main-content").style.visibility = "visible";
18     } else {
19         limit++;
20         document.getElementById("message").innerHTML =
21             `The username "${userName}" and the password "${password}" are not in our records`;
22
23         document.getElementById("username").value = "";
24         document.getElementById("password").value = "";
25         console.log(`The Limit: ${limit}`);
26     }
27
28
29 }
30
```

- Middle Panel:** A browser window titled "Debug Application" showing a login form with fields for "User id:" and "Password:", and a "Login" button. Below the form, a message states: "The username "yada" and the password "yada" are not in our records".
- Bottom Panel:** A "Console" tab showing the output of the JavaScript code. It displays two log entries:

```
The Limit:1  
index.js:19:17
The Limit:2  
index.js:19:17
>
```

# Lets make a password protected website!!!

The screenshot shows a development environment with a code editor and a browser window.

**Code Editor (left):** The file `index.js` contains the following JavaScript code:

```
1 let userName;
2 let password;
3 let limit : number = 0;
...
4
5 usage
6 function check(): void {
7     userName = document.getElementById("username").value;
8     password = document.getElementById("password").value;
9
10    if (userName === "Jack" && password === "jack-123") {
11        document.getElementById("main-content").style.visibility = "visible";
12    } else {
13        limit=limit+1;
14        document.getElementById("message").innerHTML =
15            `The username "${userName}" and the password "${password}" are not in our records`;
16
17        document.getElementById("username").value = "";
18        document.getElementById("password").value = "";
19        console.log("The Limit:" + limit);
20
21        if (limit === 3) {
22            document.getElementById("check").disabled = true;
23            document.getElementById("message").innerHTML = "You have reached the maximum login attempts.";
24        }
25    }
26}
27
28
29
30
```

**Browser (right):** The browser window shows a simple login form with fields for User id and Password, and a Login button. Below the form, a message states: "You have reached the maximum login attempts." The browser's developer tools are open, showing the Console tab with the following logs:

- The Limit:1
- The Limit:2
- The Limit:3
- >

The logs are timestamped at index.js:19:17.

# Lets make a password protected website!!!

The screenshot shows a development environment with a code editor and a browser window.

**Code Editor (left):** The file `index.js` contains the following JavaScript code:

```
1 let userName;
2 let password;
3 let limit : number = 0;
...
4
5 usage
6 function check(): void {
7     userName = document.getElementById("username").value;
8     password = document.getElementById("password").value;
9
10    if (userName === "Jack" && password === "jack-123") {
11        document.getElementById("main-content").style.visibility = "visible";
12    } else {
13        limit=limit+1;
14        document.getElementById("message").innerHTML =
15            `The username "${userName}" and the password "${password}" are not in our records`;
16
17        document.getElementById("username").value = "";
18        document.getElementById("password").value = "";
19        console.log("The Limit:" + limit);
20
21        if (limit === 3) {
22            document.getElementById("check").disabled = true;
23            document.getElementById("message").innerHTML = "You have reached the maximum login attempts.";
24        }
25    }
26}
27
28
29
30
```

**Browser (right):** The browser window shows a simple login form with fields for User id and Password, and a Login button. Below the form, a message states: "You have reached the maximum login attempts." The browser's developer tools are open, showing the Console tab with the following logs:

- The Limit:1
- The Limit:2
- The Limit:3
- >

The logs are timestamped at index.js:19:17.

# Debugging your code!

- Pay attention to data types
  - Use proper commands like `Number()` and `String()` to change the data types if needed!
- Add print statements using `console.log()`
  - Everything will be printed in **console**
  - Keep track of the values of letiables in your code!

# Basics of JS Coding: Repetitions

- It's often the case that we need to do a set of steps several times.
- JavaScript provides us a way with a **loop**

# Basics of JS Coding: Repetitions

- It's often the case that we need to do a set of steps several times.
- JavaScript provides us a way with a **loop**
  - Example: Write a JS code to sum up numbers between 0 and n where n can be any number like 5, or 1000.

# Basics of JS Coding: Repetitions

- It's often the case that we need to do a set of steps several times.
- JavaScript provides us a way with a **loop**
  - Example: Write a JS code to sum up numbers between 0 and n where n can be any number like 5, or 1000.
  - It means that we should add  $0 + 1 + 2 + 3 + \dots + (n - 1) + n$

# Basics of JS Coding: Repetitions

- It's often the case that we need to do a set of steps several times.
- JavaScript provides us a way with a **loop**
  - Example: Write a JS code to sum up numbers between 0 and n where n can be any number like 5, or 1000.
  - It means that we should add  $0 + 1 + 2 + 3 + \dots + (n - 1) + n$

# Basics of JS Coding: Repetitions

- It's often the case that we need to do a set of steps several times.
- JavaScript provides us a way with a **loop**
  - Example: Write a JS code to sum up numbers between 0 and n where n can be any number like 5, or 1000.
  - It means that we should add  $0 + 1 + 2 + 3 + \dots + (n - 1) + n$
  - We can solve this problem by using the following plan

make a variable `i = 0` as a counter

make another variable `sum = 0` to hold the result

As long as `i` does not equal `n + 1` do the following:

`i = i + 1`

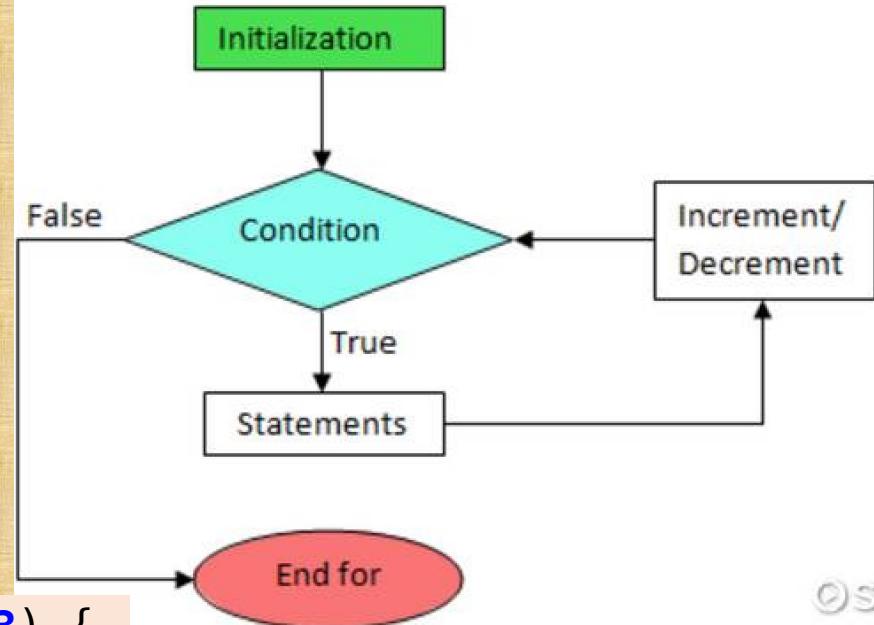
`sum = sum + i`

<code>i</code>	<code>sum</code>
0	0
1	1
2	3
3	6
4	10
...	...

# Basics of JS Coding: Repetitions (For Loop)

- It's often the case that we need to do a set of steps several times.
- JavaScript provides us a way with a **loop**
- Syntax for For Loops:

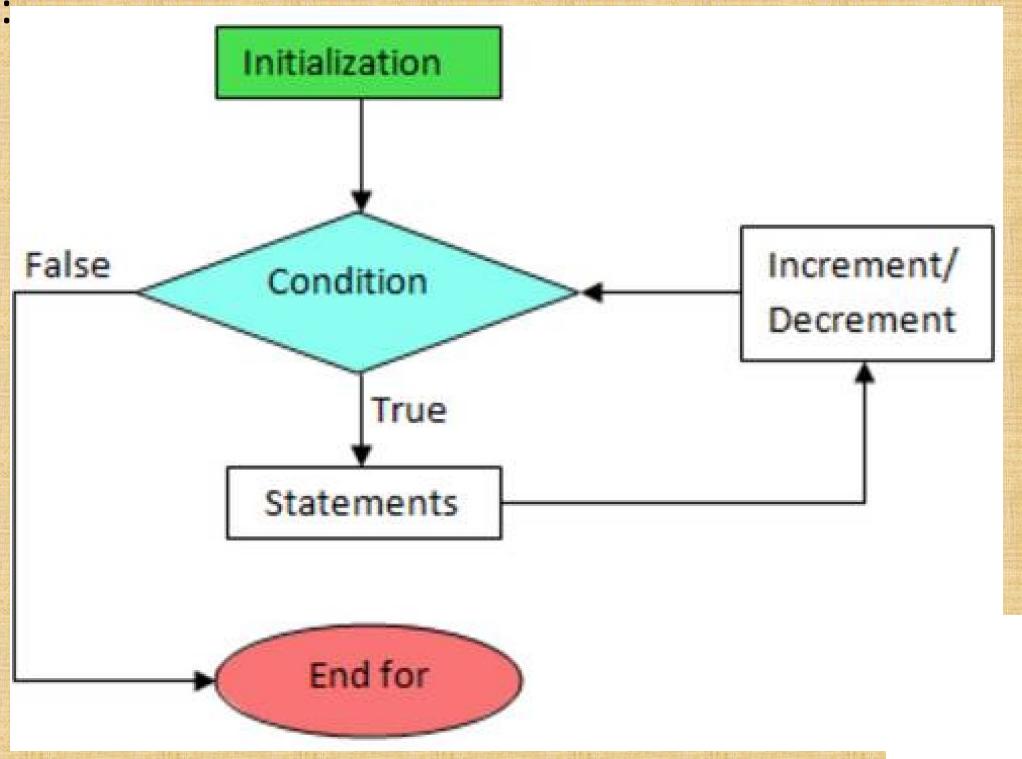
```
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}
```



- **Statement 1** initialize a variable (counter) before the loop starts.
- **Statement 2** defines a condition showing when to stop the loop.
- **Statement 3** how to increase the counter.

# Basics of JS Coding: For Loop

➤ Logical Diagram:



# Basics of JS Coding: For Loop

```
if (condition) {  
    true block  
}
```

- Syntax: 

```
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}
```

  - **Statement 1** initialize a variable (counter) before the loop starts.
  - **Statement 2** defines a condition showing when to stop the loop.
  - **Statement 3** how to increase the counter.
- Example: Create a power button for your online calculator (lab-8-a)

# Basics of JS Coding: For Loop

The image shows a development environment with two main windows. On the left is a code editor window titled "test" containing the file "index.js". The code in index.js is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>
    <div>
      <label for="first-number">First Number: </label><input id="first-number">
      <label for="second-number">Second Number: </label><input id="second-number">
      <button onclick="addition()">+</button>
      <button onclick="subtraction()">-</button>
      <button onclick="multiplication()">*</button>
      <button onclick="division()">/</button>
      <button onclick="power()">**</button>
      <h3 id="output"></h3>
    </div>
    <script src="index.js"></script>
  </body>
</html>
```

On the right is a browser window titled "Title" showing the URL "localhost:63342/test/lab-8-a/index.html". The page displays a form with two input fields labeled "First Number:" and "Second Number:". Below the inputs are five buttons with operators: "+", "-", "\*", "/", and "\*\*".

# Basics of JS Coding: For Loop

The image shows a split-screen view. On the left is a code editor window for 'index.js' in a 'test' folder. The code contains three functions: multiplication, division, and power. The multiplication function uses a simple assignment operator (=). The division function uses a standard assignment operator (=). The power function uses a for loop with a step of 1 (i=0; i<second; i=i+1) to calculate the power of a number.

```
1 usage
16 function multiplication() : void {
17     let first=document.getElementById( elementId: "first-number").value;
18     let second=document.getElementById( elementId: "second-number").value;
19     let result : number =Number(first)*Number(second);
20     document.getElementById( elementId: "output").innerHTML=String(result);
21 }
22
23 usage
24 function division() : void {
25     let first=document.getElementById( elementId: "first-number").value;
26     let second=document.getElementById( elementId: "second-number").value;
27     let result : number =Number(first)/Number(second);
28     document.getElementById( elementId: "output").innerHTML=String(result);
29 }
30
31 usage
32 function power() : void {
33     let first=document.getElementById( elementId: "first-number").value;
34     let second=document.getElementById( elementId: "second-number").value;
35
36     let result : number =1;
37     for(let i:number =0; i<second; i=i+1){
38         result=first*result;
39     }
40
41     document.getElementById( elementId: "output").innerHTML=String(result);
42
43
44 }
```

On the right is a browser window showing the output of the 'power' function. It has input fields for 'First Number' (2) and 'Second Number' (3), and a row of buttons for '+', '-', '\*', '/', and '\*\*'. The result is displayed as 8.

First Number:  
2  
Second Number:  
3  
+ - \* / \*\*  
8

## Using command write()

- So far, we have learned that there are two methods to print something out.
  - ✓ Printing in the console
  - ✓ Printing in a pop-up window using alert() and prompt() commands.
- But ...
  - ✓ how about print something inside the web page not in the console or pop-up windows.
- You can print everywhere inside your web page using write()
- Syntax:

```
document.write("a text");
```
- Note the **dot notation** to connect **document** and **write**.

# Let's make an age restricted website!!!

- A question: Is it possible to make a content including elements and tags?
  - ✓ For example, “the welcome” warning message is printed in the web page as h1 element

# Let's make an age restricted website!!!

- A question: Is it possible to make a content including elements and tags?
  - ✓ For example, “the welcome” warning message is printed in the web page as h1 element
  - ✓ The answer is yes! We can do html coding between the quotation marks:

# Let's make an age restricted website!!!

- A question: Is it possible to make a content including elements and tags?
  - ✓ For example, “the welcome” warning message is printed in the web page as h1 element
  - ✓ The answer is yes! We can do html coding between the quotation marks:

```
document.write("<h1>welcome to my ...</h1>");
```

# Let's make an age restricted website!!!

- A question: Is it possible to make a content including elements and tags?
  - ✓ For example, “the welcome” warning message is printed in the web page as h1 element
  - ✓ The answer is yes! We can do html coding between the quotation marks:

```
document.write("<h1>welcome to my ...</h1>");
```

- ✓ Also we can do inline styling:

```
document.write("<h1 style='color:lightblue;'>welcome ...</h1>");
```

## Using command getElementById()

- If you need to find an element with a specific id, you should use the following syntax
- Syntax: `document.getElementById("name of id");`

## Using command getElementById()

- If you need to find an element with a specific id, you should use the following syntax
- Syntax: `document.getElementById("name of id");`
- We should use name **document** because the id exists in the current document.
- Note the **dot notation** to connect **document** & **getElementById**.
- Note the upper (E, B, I) and lower case letters in **getElementById**.

## Using command removeAttribute()

- If you need to remove an attribute from an element, you should use the following syntax
- Syntax:

```
document.getElementById("name of id").removeAttribute("name of attribute");
```

## Using command removeAttribute()

- If you need to remove an attribute from an element, you should use the following syntax
- Syntax:

```
document.getElementById("name of id").removeAttribute("name of attribute");
```
- We should use name **getElementById** to find the element that it's attribute should be removed
- Note the **dot notation** to connect **document**, **getElementById** & **removeAttribute**.
- At the end, we should specify name of the attribute that we want to remove.

## Using command setAttribute()

- If you need to set an attribute for an element, you should use the following syntax
- Syntax:

```
document.getElementById("name of id").setAttribute("name of attribute", "value");
```

## Using command setAttribute()

➤ If you need to set an attribute for an element, you should use the following syntax

➤ Syntax:

```
document.getElementById("name of id").setAttribute("name of attribute", "value");
```

➤ **setAttribute** needs to inputs:

- ✓ Which attribute you want to set
- ✓ Which value you want to give to the attribute.

➤ Let's go back to our real web page and try these commands.

# Introduction to React

