

INTRO TO HTML & CSS

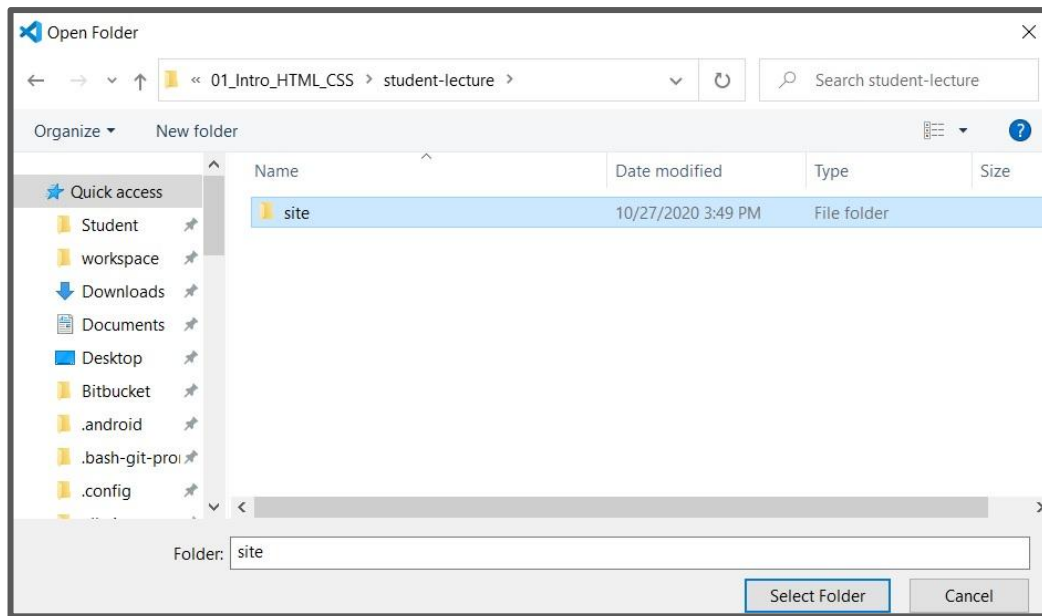
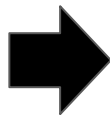
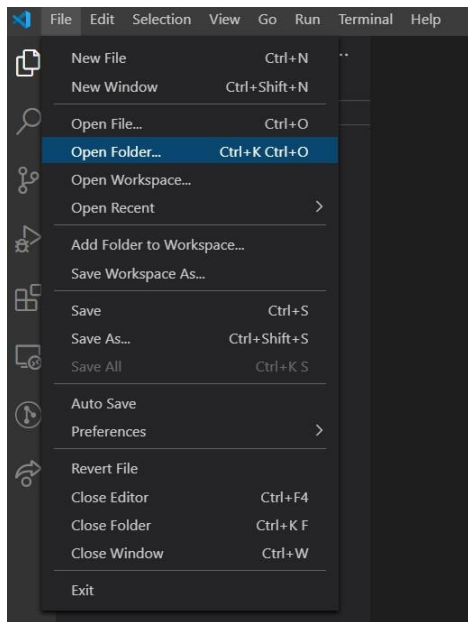
TODAY'S OBJECTIVES

- **Understand the basics of HTML**
 - HTML document structure
 - HTML tags & attributes
- **Understand the following basic HTML tags:**
 - Headings (h1, h2, etc)
 - Paragraphs
 - Links
 - Images
- **Utilize the different elements that go into building a form**
- **Understand the action attribute of the form tag**
- **Describe what Semantic HTML is**
- **Demonstrate a basic understanding of what CSS is and how to create style declarations**
- **Understand how fonts work in the browser and some of the different properties available for styling them**
- **Understand how colors work and some of the different properties for working with them**

LET'S GET
SET UP!!!

OPEN CODE

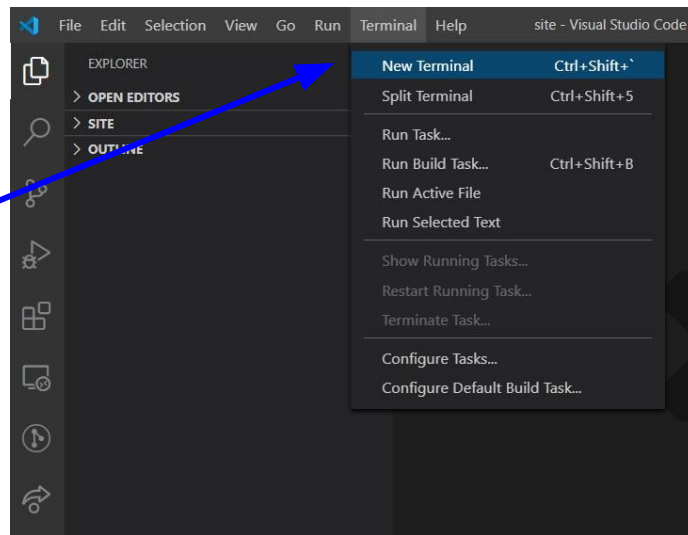
- Open Visual Studio Code and then open the `site` folder in the `student-lecture` folder for today.



SET DEFAULT TERMINAL TO BASH

- We will use a terminal here and there in VS Code but we want to make sure it is the terminal you are familiar with (Git Bash) which may not be the default for everyone.

Start by opening the Terminal

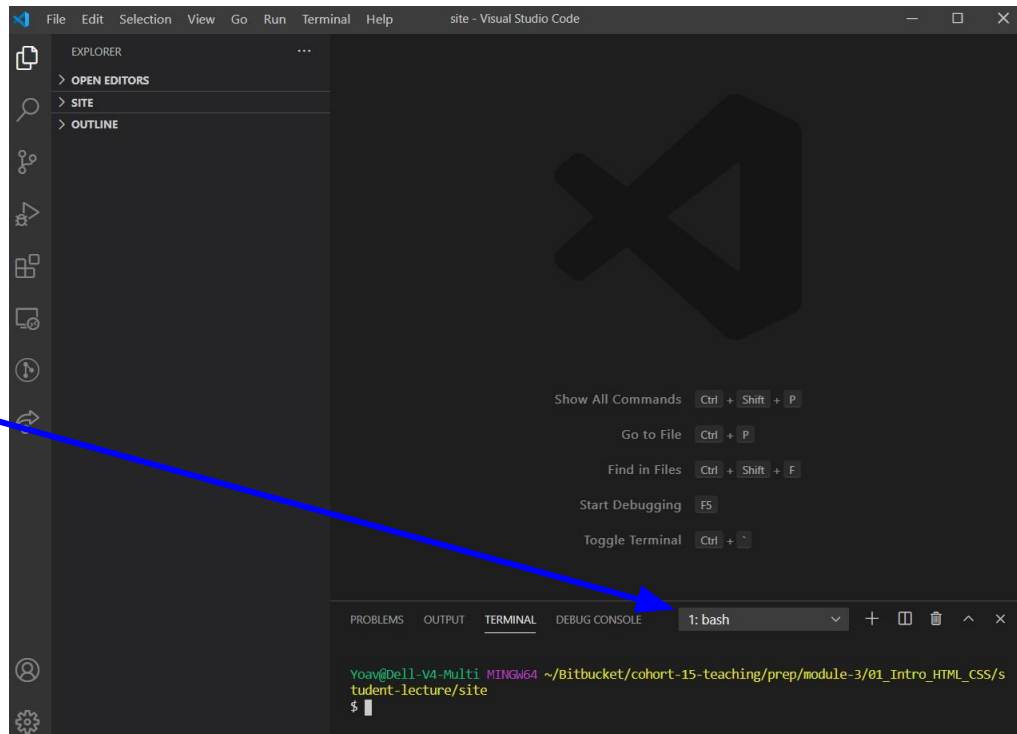


SET DEFAULT TERMINAL TO BASH

If your pulldown shows **bash** you can skip this step!

You may also have the option to select **bash** in the dropdown. If you do, do that and you can skip the rest of this part.

NOTE: *The order/numbers in your dropdown may be different.*



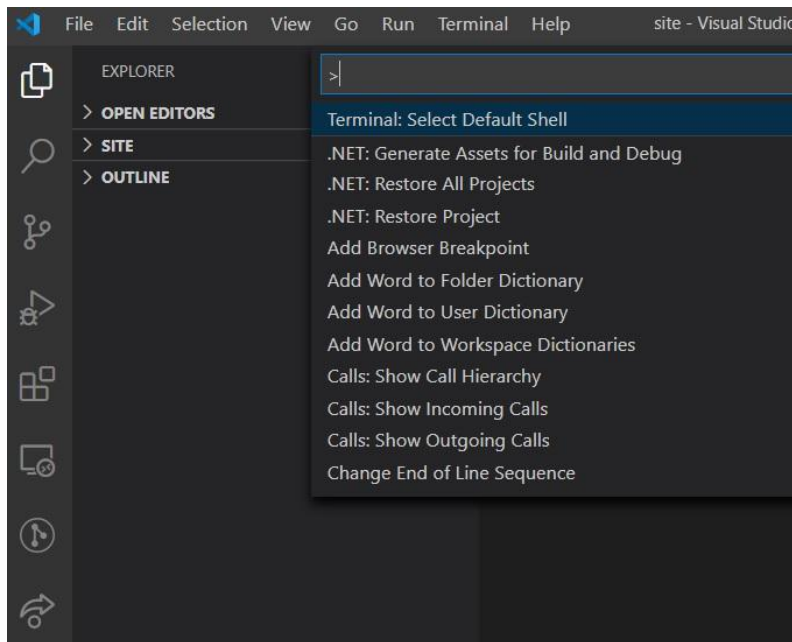
SET DEFAULT TERMINAL TO BASH

- If your dropdown does not have the bash option, you can add it

If you press

Ctrl + Shift + P

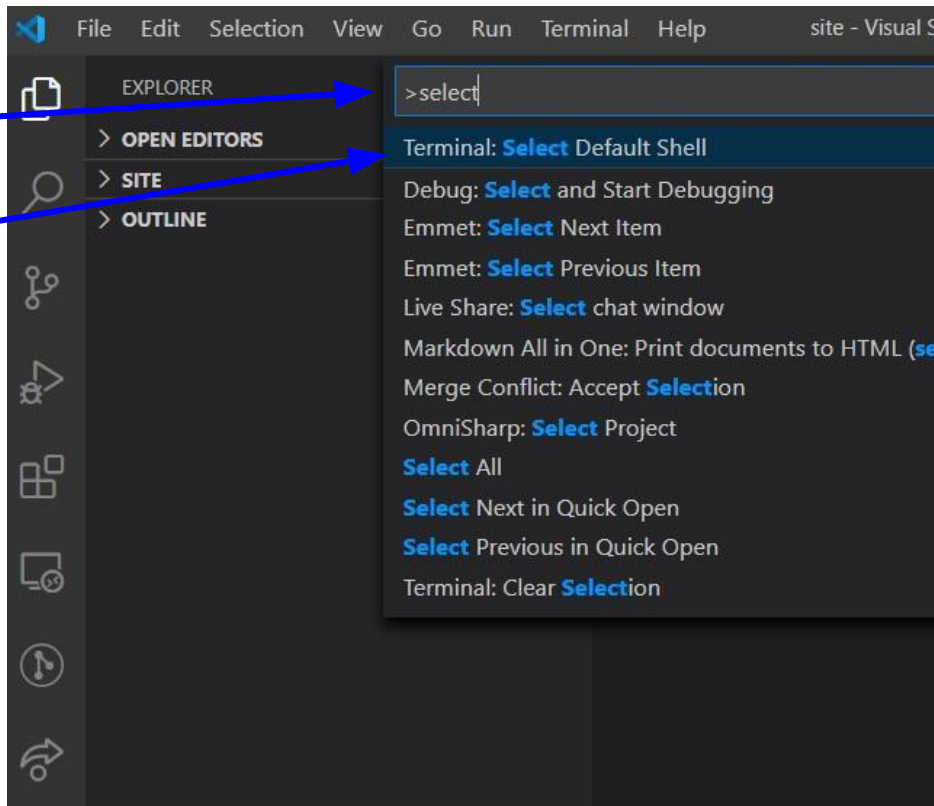
You should see this
screen pop up.



SET DEFAULT TERMINAL TO BASH

Start typing the word **select** in the search box until the **Terminal: Select Default Shell** option appears.

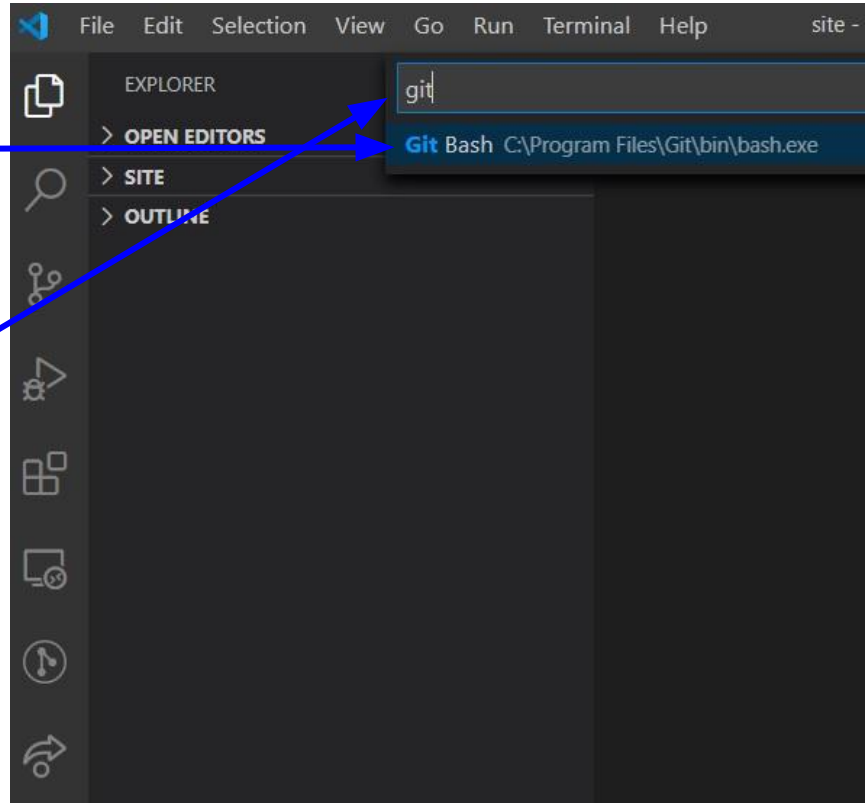
Once the option appears, click on it.



SET DEFAULT TERMINAL TO BASH

The search box will clear itself out and you will probably get a drop down list of available shells. You want to select **Git Bash**.

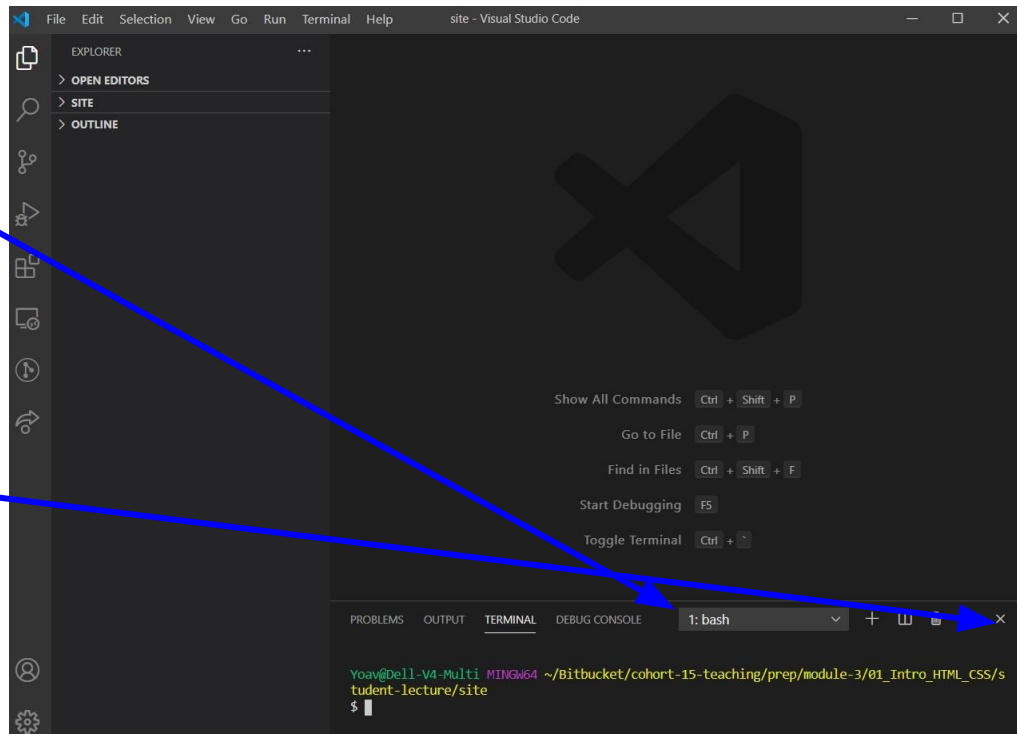
You can filter the options by typing `git` into the search box,



SET DEFAULT TERMINAL TO BASH

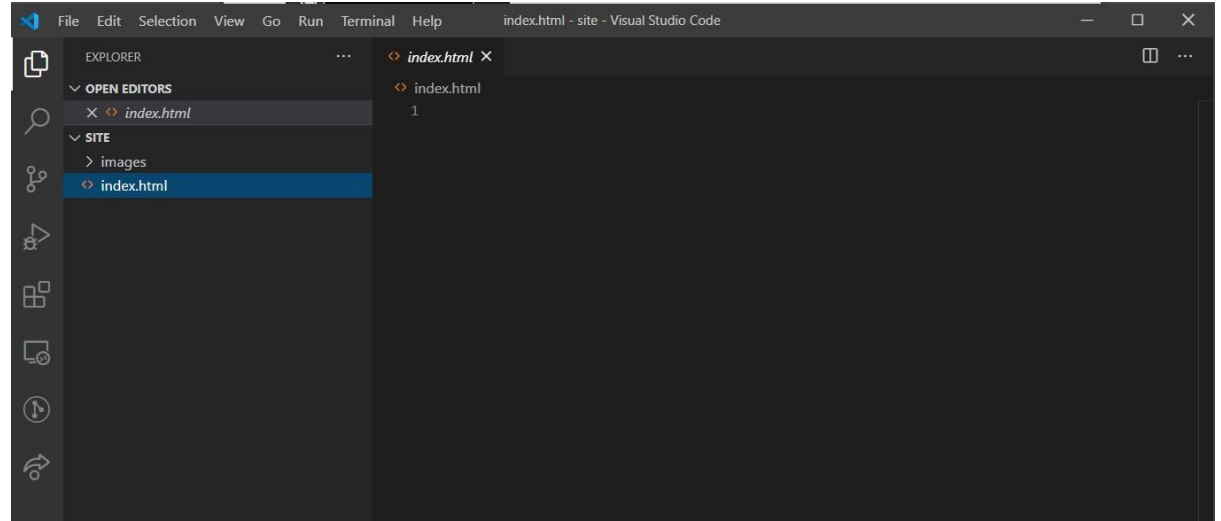
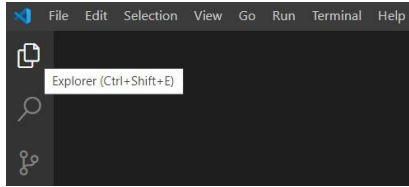
Your terminal should now have the option to select **bash** from the dropdown.

You may need to close the Terminal and reopen it to see this option. You can close it by clicking the **x** on the right and then reopening it from the **Terminal** menu.



OPEN INDEX.HTML

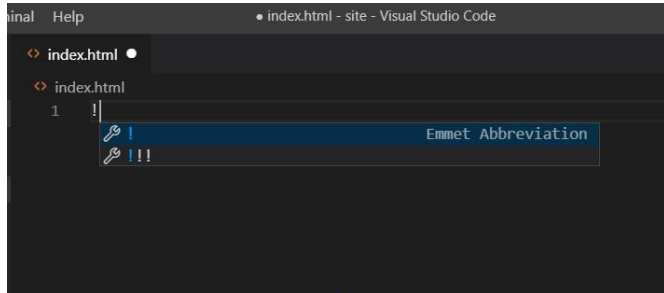
- In the **SITE** portion of the **Explorer**, double click on `index.html`



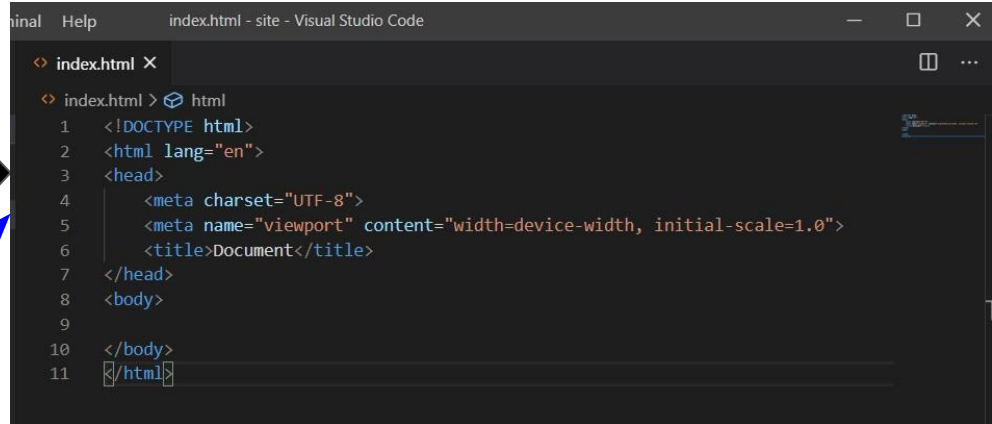
If you don't see the explorer tab, click the files icon

POPULATE HTML SKELETON INFO

- You can use the **Emmet** extension to populate the basic **HTML skeleton** information.



A screenshot of the Visual Studio Code editor. The file explorer on the left shows 'index.html'. The editor window shows the first line of 'index.html' with the text '!' followed by a cursor. A dropdown menu titled 'Emmet Abbreviation' is open, showing options like '!' and '!!!'.



A screenshot of the Visual Studio Code editor showing the completed HTML skeleton. The file explorer on the left shows 'index.html'. The editor window shows the following code:

```
index.html X
<index.html> > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

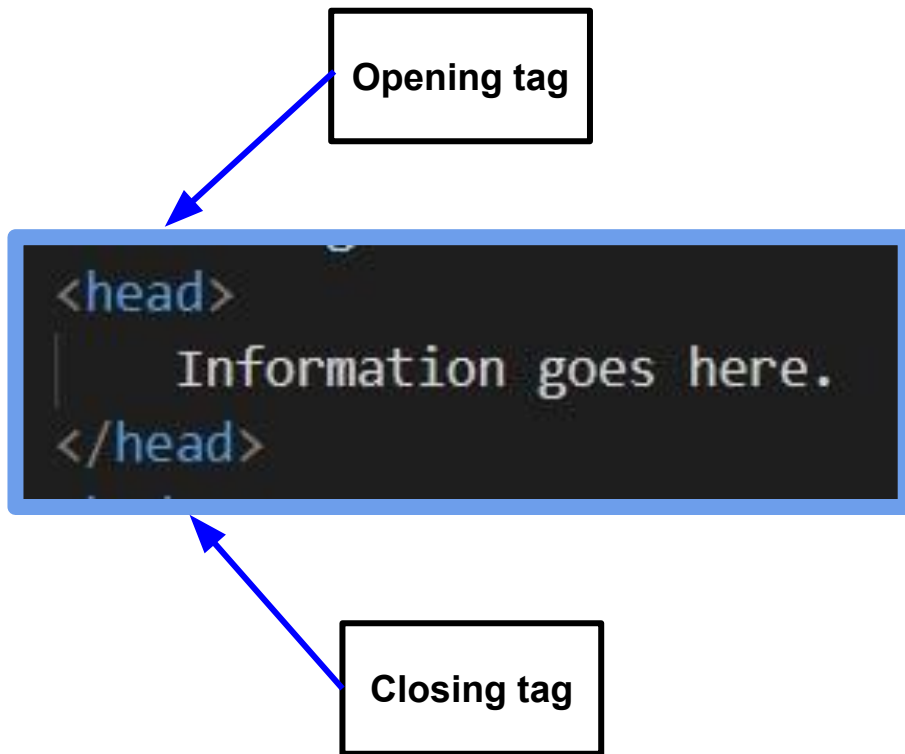
Type an ! and when you see the Emmet bar come up, press the Tab key.

Emmet will fill in a basic HTML skeleton for you automatically.

WHAT IS HTML?

HTML (Hypertext Markup Language) is a **markup** language that is used to **describe** how a **document** should be **rendered** by a web **browser**.

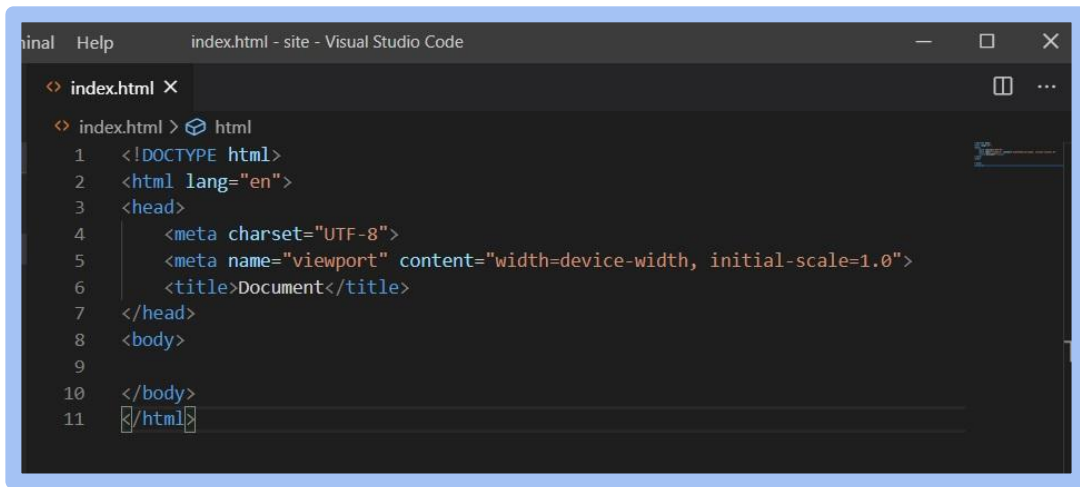
- The markup is done using HTML **element tags**.
- Most of the time, element **tags** **enclose** information.
- The **opening tag** name is **bracketed** by the **<** and **>** symbols.
- The **closing tag** is the same as the opening tag but the **<** is **followed** by a **/** to indicate it is a closing tag.



LET'S DIG INTO SOME ELEMENT TAGS...

<head>

- Defines any metadata of the document.
- Includes any links to resources the document may need.

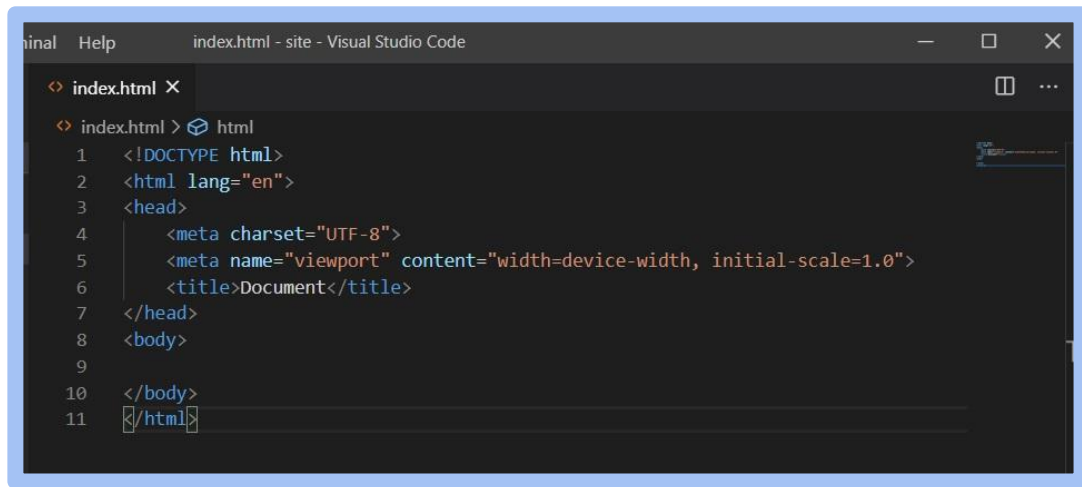


```
index.html - site - Visual Studio Code
index.html X
index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

LET'S DIG INTO SOME ELEMENT TAGS...

`<title>`

- Sets the title of the page, which is displayed in the browser's title bar.
- Used when bookmarking the page.

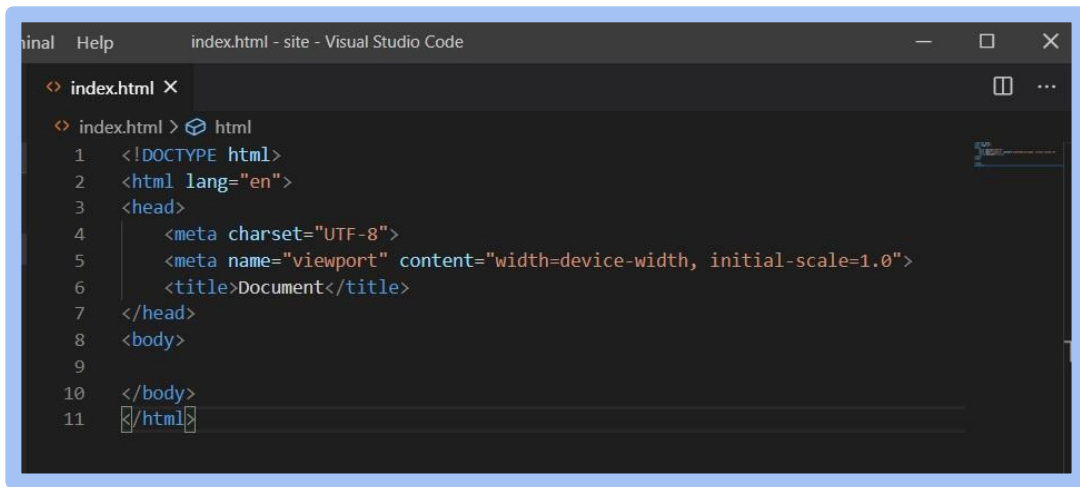


```
index.html - site - Visual Studio Code
index.html X
index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

LET'S DIG INTO SOME ELEMENT TAGS...

`<body>`

- Actual content of the page
- Includes:
 - Text
 - Images
 - Media Files

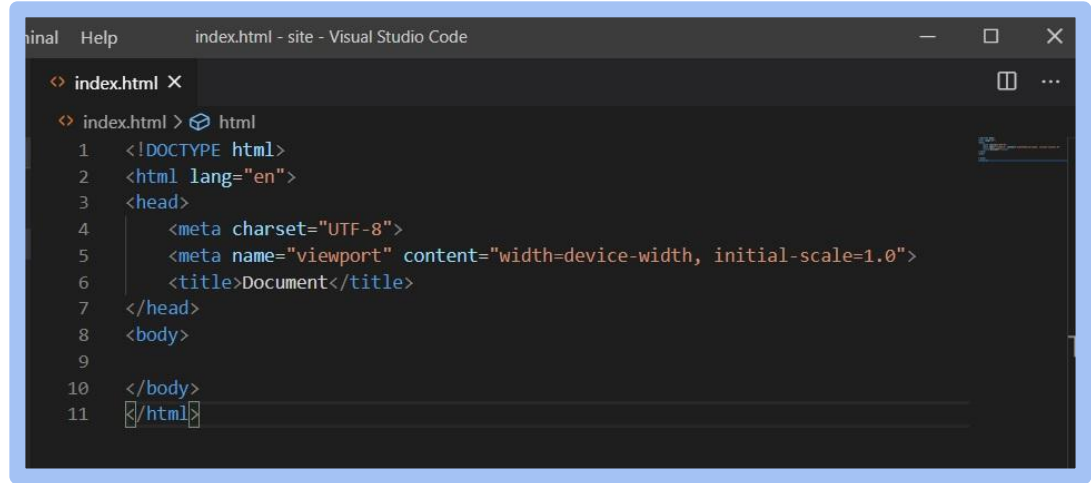


```
index.html - site - Visual Studio Code
index.html X
index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```


HTML DOCUMENT STRUCTURE

`<!DOCTYPE html>`

- Tells the browser this is an HTML file and to treat it as such.
- Most browsers render HTML without it, but for better interoperability, it should always be included.

A screenshot of a Visual Studio Code editor window titled 'index.html - site - Visual Studio Code'. The editor shows a file named 'index.html' with the following HTML code:

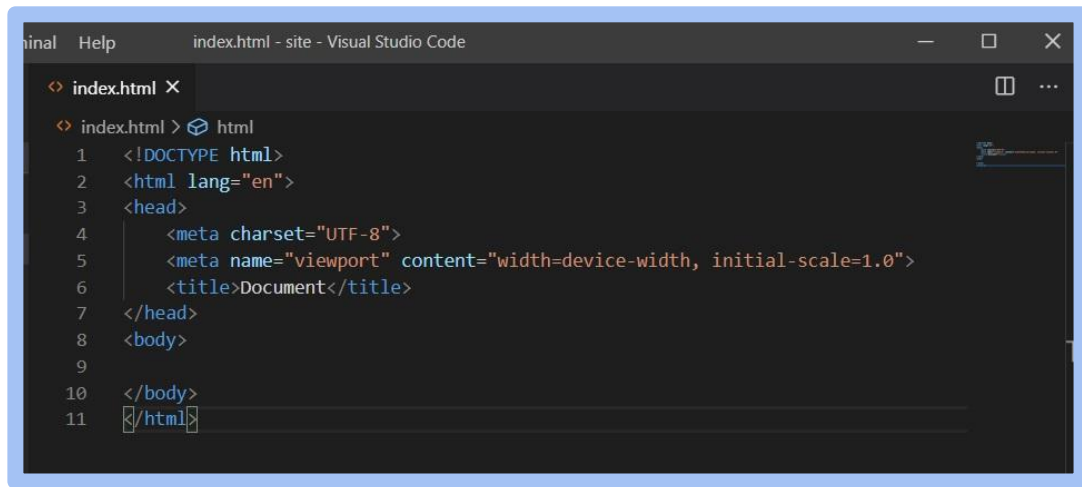
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
</body>
</html>
```

The code is syntax-highlighted, with the DOCTYPE declaration in blue, the html tag in blue, and the meta and title tags in blue. The body tag is in blue. The closing tags are in blue. The code is displayed on a dark background with a light blue border around the editor window.

HTML DOCUMENT STRUCTURE

`<html>`

- The "root" element.
- Every other element on the page should be a "descendant" of this element, meaning it's nested underneath, like a folder inside another folder.

A screenshot of a Visual Studio Code editor window titled 'index.html - site - Visual Studio Code'. The editor shows a file named 'index.html' with the following HTML code:

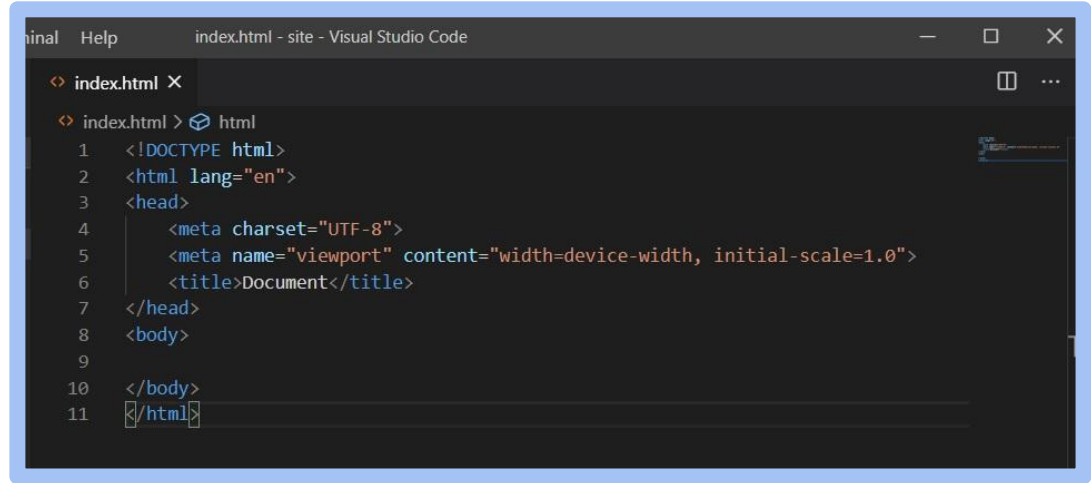
```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Document</title>
7   </head>
8   <body>
9
10  </body>
11 </html>
```

The code is color-coded: blue for tags, red for attributes, and black for text. The editor interface includes a sidebar on the left with a file explorer, a top menu bar with 'File', 'Edit', and 'Help', and window control buttons on the right.

HTML DOCUMENT STRUCTURE

<html>

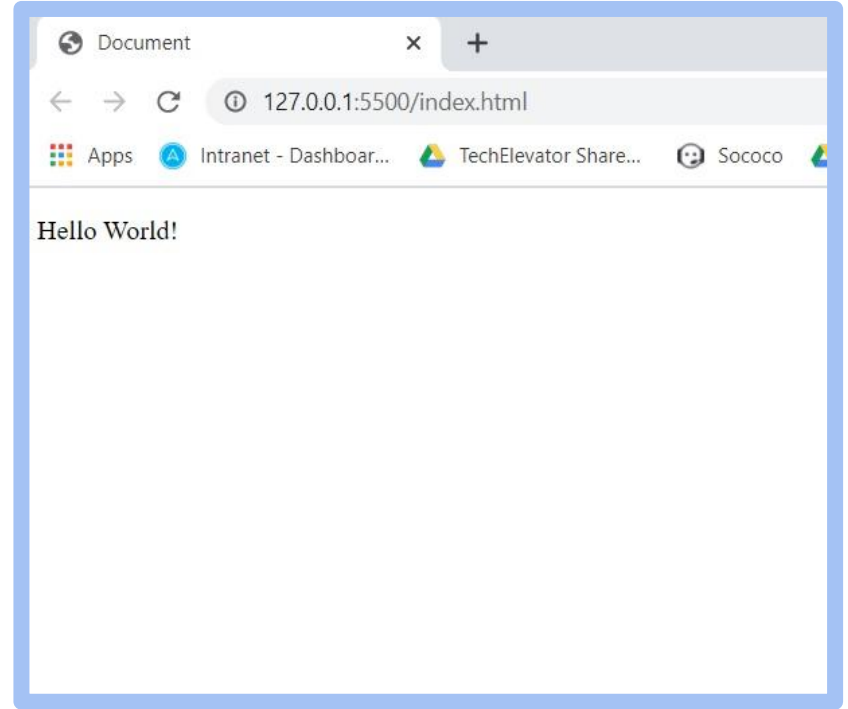
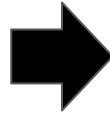
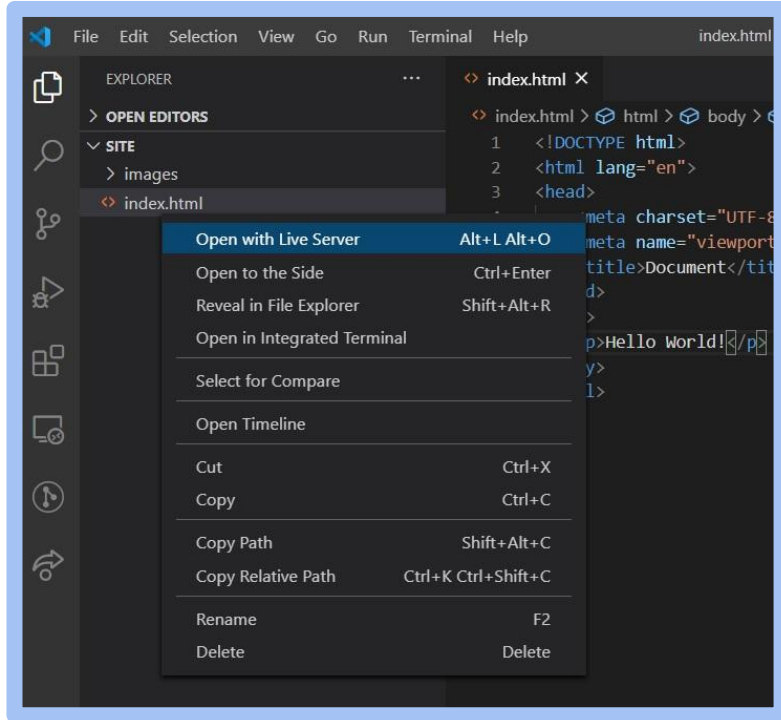
- <head> and <body> are child elements of the <html> element.
- <head> has three child elements: two <meta> elements and a <title> element.

A screenshot of the Visual Studio Code editor interface. The title bar shows 'index.html - site - Visual Studio Code'. The editor window has a tab for 'index.html'. The code is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
  </body>
</html>
```

USING LIVE SERVER
TO RENDER YOUR
HTML DOCUMENTS

OPENING A DOCUMENT WITH LIVE SERVER



INTRO TO CREATING CONTENT USING HTML ELEMENT TAGS

WHY USE `<h1>`, `<h2>`, `<h3>` RATHER THAN A SIZE?

- **Structure**

- You should make sure your content is organized in a way that makes sense:
 - `<h2>`'s follow an `<h1>`, and `<h3>`'s follow an `<h2>`.

- **Semantics**

- Refers to the "role" or "meaning" of a piece of code.
- HTML Elements you use can be meaningful - just like good variable names.
 - `<h1>` means "a top level heading on your page."
 - We could style other elements to look like `<h1>`, but the semantic meaning wouldn't be there.

WHY ARE SEMANTICS IMPORTANT?

- **Semantics are important**
 - Screen readers help visually impaired users read a page.
 - They can use elements as "sign posts" to help navigate around it.
- **HTML should be coded to represent the data, and not based on its default presentation styling.**
 - Presentation— meaning how a page should look—is the sole responsibility of the stylesheet (Stay tuned for that soon when we introduce CSS).

WHY ARE SEMANTICS IMPORTANT?

- **The tagline doesn't need any special meaning applied to it.**
 - Using `<p>` is sufficient here
 - We can still use whatever styling we think makes sense regardless of the tag used.

```
<body>  
  <h1>Bits & Bytes</h1>  
  <p>Welcome to the internet's best restaurant.</p>  
</body>
```

AN EXAMPLE OF SEMANTIC MEANING

- You've decided that you want to show the menu on the homepage.
- You've also decided an `<h2>` is acceptable for the header of the menu content. It's a subheading under the restaurant's name.
- The restaurant serves lunch and dinner, so `<h3>` makes sense for the headers of those sections.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  <title>Document</title>
</head>

<body>
  <h1>Bits & Bytes</h1>
  <p>Welcome to the internet's best restaurant</p>

  <h2>Menu</h2>
  <h3>Lunch</h3>

  <h3>Dinner</h3>
</body>
</html>
```

SEMANTIC ELEMENTS AND ORGANIZATION

- You know that the restaurant offers several dishes and can see this might become an organizational headache.
- It's always a good idea to start with a good organization of your code. That's why you use folders and namespaces.
- Before you add any more content, maybe it's a good time to organize it.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  <title>Document</title>
</head>

<body>
  <h1>Bits & Bytes</h1>
  <p>Welcome to the internet's best restaurant</p>

  <h2>Menu</h2>
  <h3>Lunch</h3>

  <h3>Dinner</h3>
</body>
</html>
```

SEMANTIC ELEMENTS AND ORGANIZATION

- There are elements that exist to group similar content together. These are called **semantic elements**.
- This is different from using an `<h1>` to define the heading—though that's still valid—as these elements have no visual display or indicator. Of course, you can always add styles if you wish.

SEMANTIC ELEMENTS IN ACTION

- The `<h1>` and `<p>` elements make up the **header** of the **page** but their meaning is just “**a top level heading**” and “**a paragraph.**”

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  <title>Document</title>
</head>

<body>
  <h1>Bits & Bytes</h1>
  <p>Welcome to the internet's best restaurant</p>

  <h2>Menu</h2>
  <h3>Lunch</h3>

  <h3>Dinner</h3>

</body>

</html>
```

SEMANTIC ELEMENTS IN ACTION

- The `<h1>` and `<p>` elements make up the **header** of the **page** but their meaning is just “**a top level heading**” and “**a paragraph.**”
- We can use the `<header>` element, which is a **semantic element** tag, to indicate that the `<h1>` and `<p>` elements make up the **page's header**.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, ini
  <title>Document</title>
</head>

<body>
  <header>
    <h1>Bits & Bytes</h1>
    <p>Welcome to the internet's best restaurant</p>
  </header>

  <h2>Menu</h2>
  <h3>Lunch</h3>

  <h3>Dinner</h3>
</body>

</html>
```

OTHER SEMANTIC ELEMENTS

- We can indicate the main part of the content by using the `<main>` element.
- The `<section>` element is intended to represent a standalone section which doesn't have a more specific semantic element to represent it.

LET'S ADD
SOME CONTENT!!!

INTRODUCING ATTRIBUTES

- Attributes are pieces of code that are included within the tag definition (within the tag's angle brackets).
- Often used to “extend” a tag by changing its behavior or providing metadata.
- Usually in the format **name** = **value** but there are a few instances where you can use shorthand and omit the value (it's a good idea to include it anyway though).

MORE ABOUT THE TAG

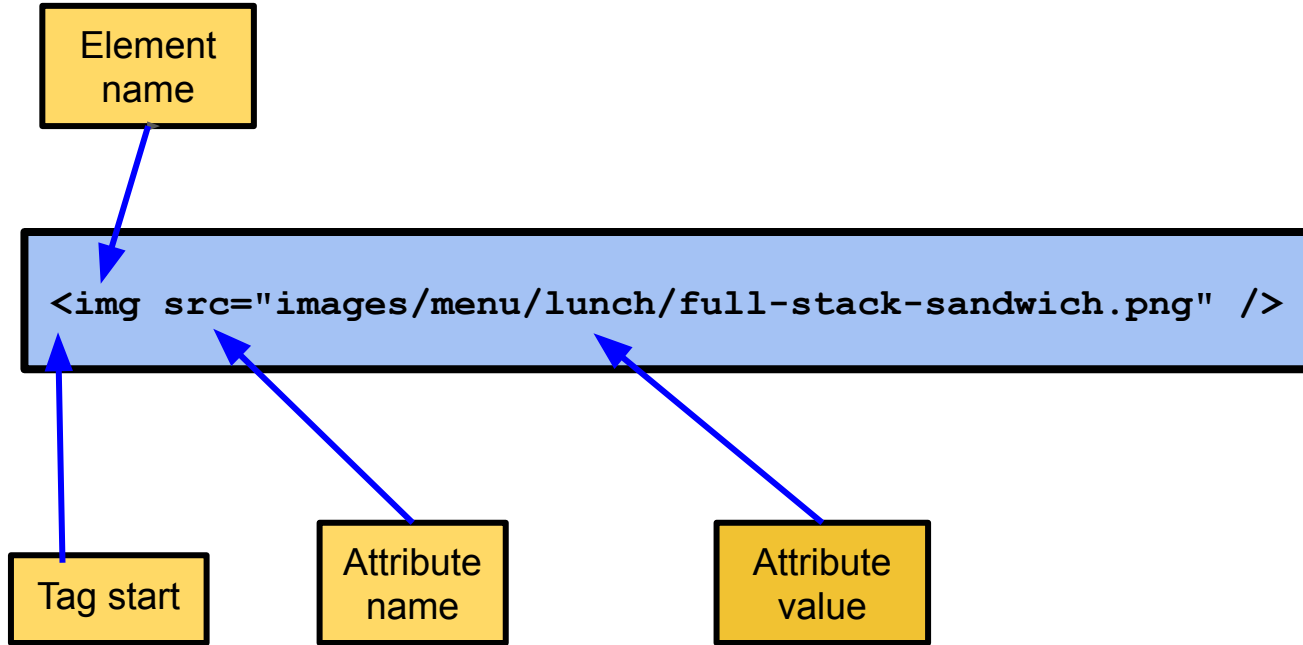
Element
name

``

Tag start

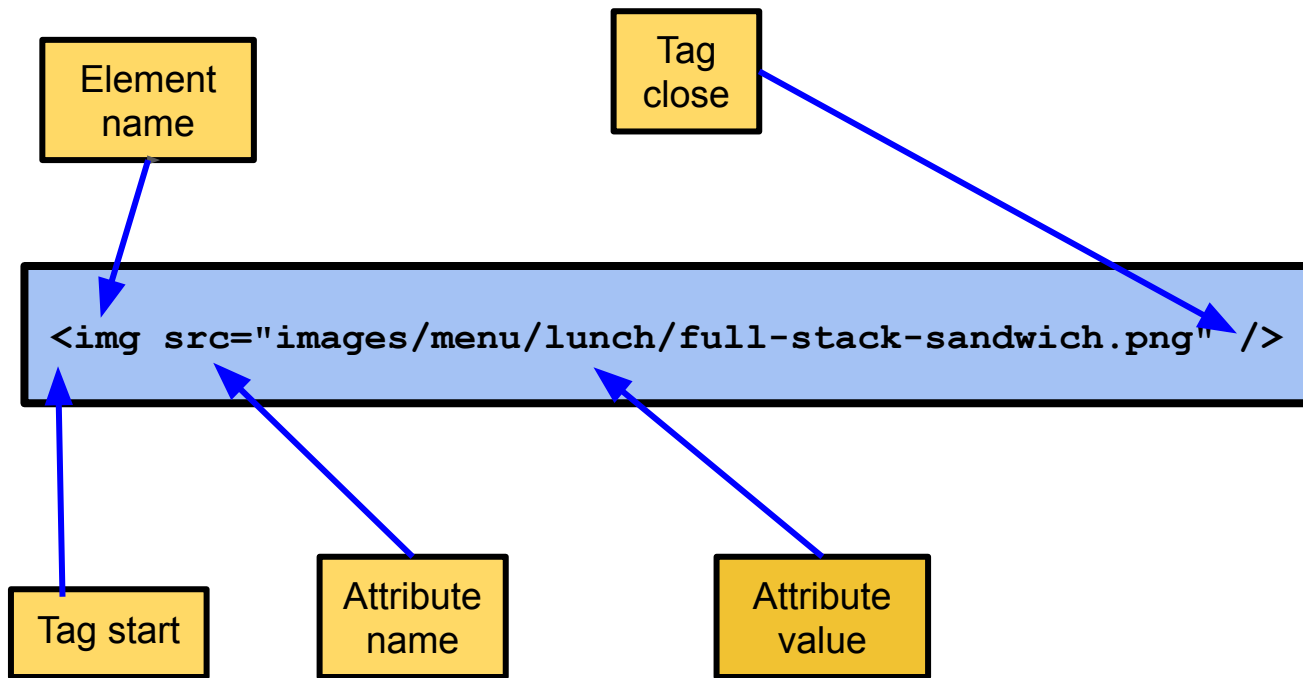
- The `` element is used to display an image on an HTML page.

MORE ABOUT THE TAG



- The `` element is used to display an image on an HTML page.
- The `src` attribute is used to specify the “source” image by providing the **relative file path** to the image file.

MORE ABOUT THE TAG



- The `` element is used to display an image on an HTML page.
- The `src` attribute is used to specify the “source” image by providing the **relative file path** to the image file.
- The `` element is used to specify a resource rather than contain child data, so it is “self-closing” - we close the single tag with a space and `/>`

LET'S ADD SOME
STYLE TO OUR PAGE...
WITH CSS

INTRODUCING CSS

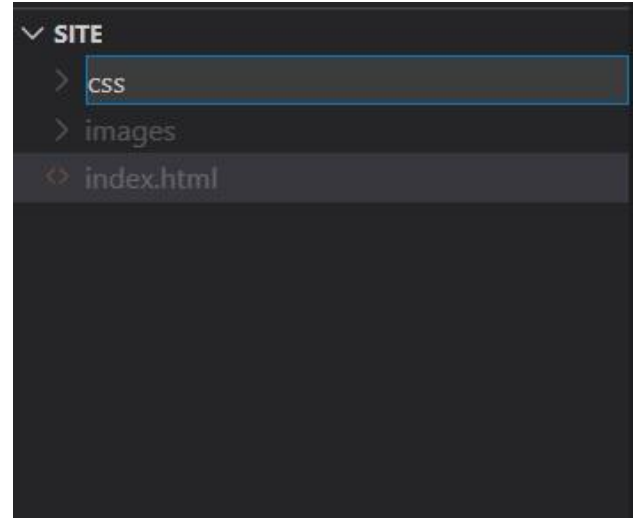
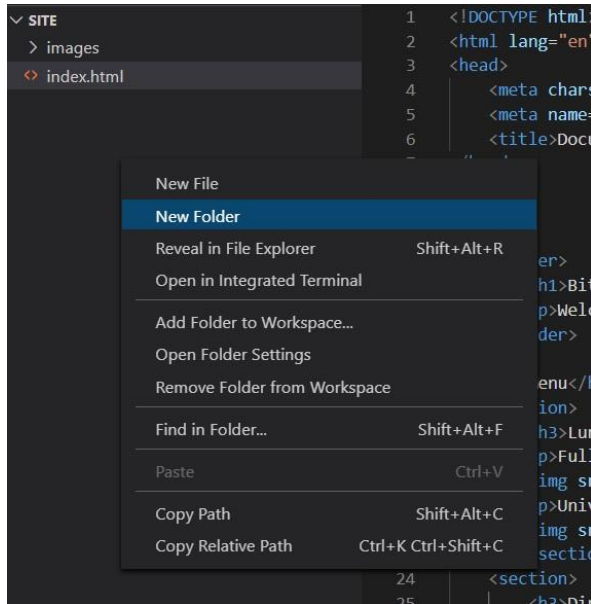
We know that HTML is used to represent data and that styling is done separately.

We use **Cascading Style Sheets** (CSS) to style HTML.

Let's see how this works...

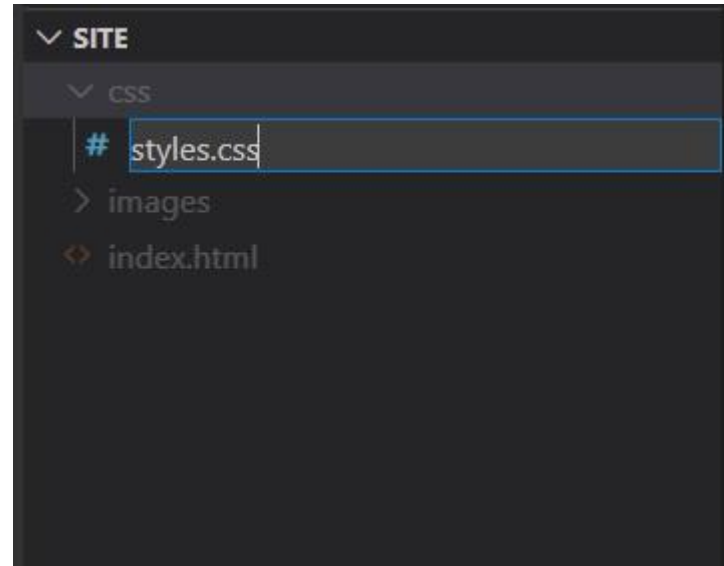
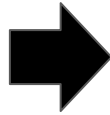
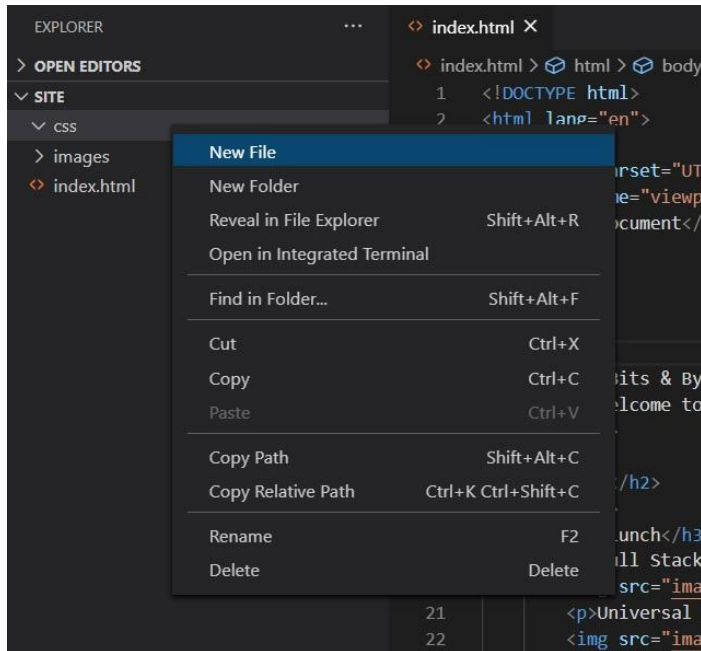
CREATING A CSS DOCUMENT

First, create a folder to hold your css documents.



CREATING A CSS DOCUMENT

Now create a styles.css file in the css folder.



WORKING WITH CSS

CSS contains **only styling** information.

We tell CSS which elements we want to affect using selectors. Anything that matches the selector of a CSS rule will be affected by it.

There are several different types of selectors which we will look at more tomorrow. Some examples are:

- element
- id
- class

The example on the right selects all `` elements.

```
img {  
  
}
```

WORKING WITH CSS

CSS rules contain styling properties.

Properties are specified as the **property name**, followed by a **colon**, followed by a **space**, the property **value**, and a **semicolon**.

```
img {  
    height: 200px;  
}
```

You can specify multiple properties by adding additional lines in this format.

ADDING CSS TO AN HTML DOCUMENT

<link> tag

In order for a CSS document to be used by an HTML document, we need to add a **link to the CSS** document in the **HTML**.

To do this, we place a **<link>** tag in our **<head>** element (**where** in the **<head>** element doesn't matter).

○

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/styles.css" />
  <title>Document</title>
</head>

<body>
  <header>
    <h1>Bits & Bytes</h1>
    <p>Welcome to the internet's best restaurant</p>
  </header>

  <h2>Menu</h2>
  <section>
    <h3>Lunch</h3>
    <p>Full Stack Sandwich</p>
    
    <p>Universal Serial Burger</p>
    
  </section>
</body>
```

ADDING CSS TO AN HTML DOCUMENT

<link> tag

rel
attribute

In order for a CSS document to be used by an HTML document, we need to add a **link to the CSS** document in the **HTML**.

To do this, we place a **<link>** tag in our **<head>** element (**where** in the **<head>** element doesn't matter).

<link> has two attributes:

- **rel**
 - Stands for “**relationship**” to the HTML file. For **CSS** files, we use the value “**stylesheet**”

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/styles.css" />
  <title>Document</title>
</head>

<body>
  <header>
    <h1>Bits & Bytes</h1>
    <p>Welcome to the internet's best restaurant</p>
  </header>

  <h2>Menu</h2>
  <section>
    <h3>Lunch</h3>
    <p>Full Stack Sandwich</p>
    
    <p>Universal Serial Burger</p>
    
  </section>
```

ADDING CSS TO AN HTML DOCUMENT

<link> tag

rel
attribute

href
attribute

In order for a CSS document to be used by an HTML document, we need to add a **link to the CSS** document in the **HTML**.

To do this, we place a **<link>** tag in our **<head>** element (**where** in the **<head>** element doesn't matter).

<link> has two attributes:

- **rel**
 - Stands for “**relationship**” to the HTML file. For **CSS** files, we use the value “**stylesheet**”
- **href**
 - Stands for “**hyperlink reference**” to the HTML file. This is where we put the path to the CSS file.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/styles.css" />
  <title>Document</title>
</head>

<body>
  <header>
    <h1>Bits & Bytes</h1>
    <p>Welcome to the internet's best restaurant</p>
  </header>

  <h2>Menu</h2>
  <section>
    <h3>Lunch</h3>
    <p>Full Stack Sandwich</p>
    
    <p>Universal Serial Burger</p>
    
  </section>
</body>
```

FONT STYLING

We can change font styling using the `font-family` property.

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

- It's typically a good idea to **use "web safe" fonts**.
- Can specify fonts that can be downloaded from web font providers.
- Example above shows **font name** as well as some **"fallback" options**.
- **Always include a fallback web safe font.**

FONT STYLING

We can change font size using the **font-size** property.

```
h1 {  
    font-size: 40px;  
}
```

- Font size is usually specified in **px** (pixels) but there are **other units** as well... we'll talk about those later this week.

FONT STYLING

We can change font color using the `color` property.

```
h1 {  
    font-size: 40px;  
    color: darkred;  
}
```

- Color can be specified in several ways
 - **keyword** (currently about 140 options such as `blue`, `darkred`, etc)
 - Six digit **hexadecimal** value (i.e. `#0000FF`, `#8B0000`, etc.)
 - Red, green, blue (**rgb**) value (i.e. `rgb(0, 0, 255)`, `rgb(139, 0, 0)`, etc)

FONT STYLING

To explore colors more, take a look at this site:

<https://www.w3schools.com/colors/>

- You'll find lots of info there including
 - Color name keywords available along with the colors they represent
 - Color converter (convert to/from name, hex value, rgb value, others)
 - Color Picker (choose a color from a palette and get all its relevant info)
 - LOTS more

SOME OTHER CSS PROPERTY EXAMPLES

There are many CSS properties you can use. We'll take a look at a couple of examples:

- border-style (required)
- border-width
- border-color
- background-color

You can find a full list of CSS properties at:

<https://www.w3schools.com/cssref/>

INTRODUCING
FORMS...

FORMS

Forms are one way to **submit data** from the client to the server.

You've likely encountered forms on the web when logging in to websites, posting on social media, or completing an e-commerce purchase.

Forms are **enclosed** within a `<form>` element.

- `<form>` element has at least two attributes:
 - **method**
 - GET
 - POST
 - **action**
 - URL the form will send its data to (an API endpoint for example)

NOTES ON USING GET AS METHOD

More about using **GET** as the **method**:

- Form data is appended into the URL in name/value pairs.
- **Never use GET to send sensitive data.** It will be visible in the URL.
- Useful for form submissions where a user wants to bookmark the result.
- **GET** is better for non-secure data, like query strings for a search engine.

NOTES ON USING POST AS METHOD

More about using **POST** as the **method**:

- Form data is send inside the body of the HTTP request (data is not shown in URL).
- Form submissions with **POST** cannot be bookmarked.
- Just like **POST** methods on the server, it should be reserved for "creating" things like reservations.

NOTES ON USING ACTION ATTRIBUTE

More about the **action** attribute:

- The **action** attribute is the URL that data is sent to (such as an API endpoint)
- **action** often uses an absolute path.
 - An absolute path differs from a relative path by the leading `/` in the URL. This is useful for when the page may be at a different path than the resource being referenced.
 - Example:
 - `<form method="POST" action="/api/reservation">`
 - Absolute paths can also be a full URL, like <https://example.com/api/reservation>.

WHAT CAN WE
PUT ON A FORM?

INPUT ELEMENTS

```
<input type="text" />
```

Some examples of the type of <input> elements we can use:


- **text**
- **number**
 - Forces the input data to be numeric
- **date**
 - Provides a date picker
- **time**
 - Provides a time picker
- **checkbox**
 - allows you to check if box is checked or not

FORM LABELS

Form elements typically require a label of some sort to indicate what the input field is asking for.

HTML provides a `<label>` element which is associated with an input element.

Label text
wrapped in
`<label>`
element.



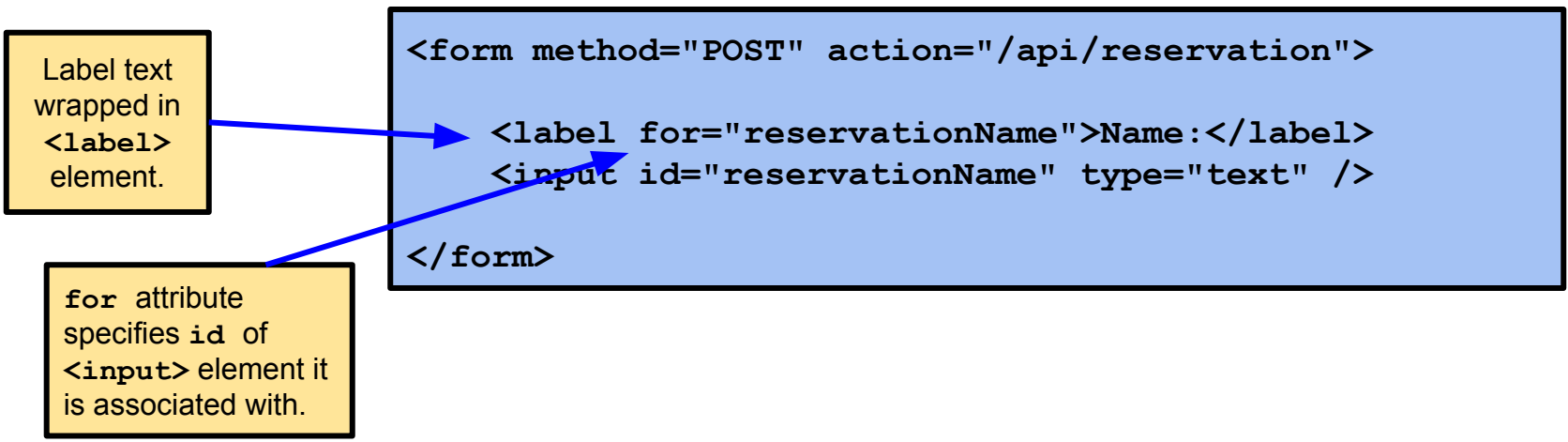
```
<form method="POST" action="/api/reservation">  
  <label for="reservationName">Name:</label>  
  <input id="reservationName" type="text" />  
</form>
```

FORM LABELS

Form elements typically require a label of some sort to indicate what the input field is asking for.

HTML provides a `<label>` element which is associated with an input element.

Label text
wrapped in
`<label>`
element.



`for` attribute
specifies id of
`<input>` element it
is associated with.

```
<form method="POST" action="/api/reservation">  
  <label for="reservationName">Name:</label>  
  <input id="reservationName" type="text" />  
</form>
```

FORM LABELS

Form elements typically require a label of some sort to indicate what the input field is asking for.

HTML provides a `<label>` element which is associated with an input elements.

Label text
wrapped in
`<label>`
element.

```
<form method="POST" action="/api/reservation">  
  <label for="reservationName">Name:</label>  
  <input id="reservationName" type="text" />  
</form>
```

`for` attribute
specifies id of
`<input>` element it
is associated with.

In order to use `<label>`, `<input>` element
must have a unique id attribute

FORM ELEMENT NAMES

Form elements should also have a **name** attribute.

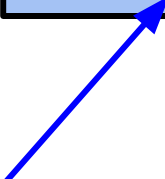
```
<input id="firstName" name="firstName" type="text" />
```

The element **id** is often used by CSS/Javascript but when a form is submitted to backend code as a set of form fields, the fields will be identified by their **name** attribute.

USING NAV AND HYPERLINKS

- You can use the `<nav>` element to give semantic meaning to the navigation portion of an HTML page.
- Navigation is often accomplished by using hyperlinks. Hyperlinks are all over the web and they are easy to add to your code.

```
<a href="https://google.com">Go to Google!</a>
```



Links are created using the `<a>` element tag.

USING NAV AND HYPERLINKS

- You can use the `<nav>` element to give semantic meaning to the navigation portion of an HTML page.
- Navigation is often accomplished by using hyperlinks. Hyperlinks are all over the web and they are easy to add to your code.

```
<a href="https://google.com">Go to Google!</a>
```

Links are created using the `<a>` element tag.

The `href` attribute contains the URL to link to (remember stylesheet `<link>` element?)

USING NAV AND HYPERLINKS

- You can use the `<nav>` element to give semantic meaning to the navigation portion of an HTML page.
- Navigation is often accomplished by using hyperlinks. Hyperlinks are all over the web and they are easy to add to your code.

```
<a href="https://google.com">Go to Google!</a>
```

Links are created using the `<a>` element tag.

The `href` attribute contains the URL to link to (remember stylesheet `<link>` element?)

The text contained within the `<a>` element is what will be displayed on the page for the link.

FONT STYLING

HTML Elements Links:

<https://www.w3schools.com/html/default.asp>