**The Odin Project**

[**Growth mindset**](https://www.theodinproject.com/lessons/foundations-motivation-and-mindset#growth-mindset)

While you’re working through the curriculum, embrace the struggles you encounter with difficult concepts and complex projects. Be sure to celebrate your persistence in overcoming those struggles!

[**Managing your study time**](https://www.theodinproject.com/lessons/foundations-motivation-and-mindset#managing-your-study-time)

It may take you longer than others to grasp concepts, or it may take you less time. This doesn’t mean you’re smarter or dumber than others, it means you’ve had differing life experiences that may or may not have prepared you for learning this stuff. Someone who grew up around an engineer may have some advantages over someone who didn’t, but it doesn’t mean you can’t learn those skills.

This course is very research based, meaning you will have to do research to complete tasks and projects. There’s no telling if you can find the article or post that helps you in the right way, quickly to meet your deadlines, but we bet you learned a TON along the way that you can use in the future. People that do this kind of research and strive to write better solutions tend to become better developers in the future.

Stress and anxiety absolutely do not help you learn either. Relax and just enjoy the ride.

Long story short: Don’t worry, just go learn!

[**Pitfalls to avoid**](https://www.theodinproject.com/lessons/foundations-motivation-and-mindset#pitfalls-to-avoid)

The following are some of the pitfalls that beginners often encounter when learning how to program. Try your best to avoid these.

Procrastination

Procrastination will be your biggest enemy when trying to make progress.

S**olution:**

 The [Pomodoro Technique](https://en.wikipedia.org/wiki/Pomodoro_Technique) is a way of managing your time in order to stay focused. The idea is to set a timer for 25 minutes and to work on a task until the timer goes off. If you get distracted or interrupted during the 25 minutes, start the 25 minutes of work over again. Once you’ve successfully focused on work for 25 minutes, take a 5 minute break. When your break is over, repeat the 25 minutes of work and 5 minute break. After you’ve completed four 25 minute blocks of work, take a longer 15-30 minute break.

What to do during your break:

* Listen to music.
* Journal
* Doodle
* Meditate
* Play a quick game.
* Go for a short walk outside.

### [GitHub](https://www.theodinproject.com/lessons/foundations-git-basics#cheatsheet)

### [Cheatsheet](https://www.theodinproject.com/lessons/foundations-git-basics#cheatsheet)

This is a reference list of the most commonly used Git commands. (You might consider bookmarking this handy page.) Try to familiarize yourself with the commands so that you can eventually remember them all:

* Commands related to a remote repository:
  + git clone git@github.com:USER-NAME/REPOSITORY-NAME.git
  + git push or git push origin main (Both accomplish the same goal in this context)
* Commands related to the workflow:
  + git add .
  + git commit -m "A message describing what you have done to make this snapshot different"
* Commands related to checking status or log history
  + git status
  + git log

The basic Git syntax is program | action | destination.

For example,

* git add . is read as git | add | ., where the period represents everything in the current directory;
* git commit -m "message" is read as git | commit -m | "message"; and
* git status is read as git | status | (no destination).

Step by step GITHub

1 - Create repository in github.com with adding a readme file (git\_test)

2 - Copy the SSH URL

3 - Create a directory on local machine using terminal (mkdir repos)

4 - Cd into the directory (cd repos)

5 - Paste the SSH URL with git clone   
( git clone git@github.com:Ishad17/git\_test.git)

6 - After cloning successful, cd into git test (cd git\_test)

7 - Now you can create your code files with touch command  
(touch myweb.html, touch Helloworld.py, touch hello\_world.txt etc.....)

8 - Add those code files to staging are (git add myweb.html)

9 - Commit new changes with a message (git commit -m “New text file”)

- Can also commit without a message. (git commit -> vs code page -> ......)

- To do that should use prompt in the terminal  
 (git config --global core.editor "code --wait")

10 - Can use git log and git status to find the status

11 - To modify file type “code .” On terminal. It will open the repository on VS Code

12 - Type your codes and make your changes and save. (on VS Code)

13 - Add README.md to the staging area with (git add README.md)

14 - Now commit (9th point)

15 - Now push your work to GitHub (git push / git push origin main)

* Refresh your repository in github.com to see the changes

**HTML: Content & Layout**

HTML is a language that determines how documents and web pages are displayed in a web browser, the language for the building blocks of any website.

**CSS: Styling and “Look & Feel”**

CSS (Cascading Style Sheets) is a popular style sheet language that determines how a document created in HTML is styled (colors, font styles, layout and responsive features).

**JavaScript: Interactive Elements**

JavaScript allows you to change CSS and HTML elements on your website after the site has been loaded, which gives you the ability to make your site more interactive and engaging for users with animations, embedded [video media](https://brandefy.com/austin-texas-video-production/), and more.

**HTML**

**HTML Boilerplate**

***Elements and Tags***

A full paragraph element looks like this:

<p>some text content</p>

**Let’s break this down:**

**<p>  - is the opening tag.**

**some text content  - represents content wrapped within the opening and closing tags.**

**</p>  - is the closing tag.**

[**Creating an HTML file**](https://www.theodinproject.com/lessons/foundations-html-boilerplate#creating-an-html-file)

* Create a folder and an HTML file inside it on VS Code.
* We should name our homepage as index.html

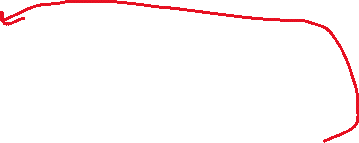
It is worth noting that we named our HTML file index. We should always name the HTML file that will contain the homepage of our websites index.html. This is because web servers will by default look for an index.html page when users land on our websites - and not having one will cause big problems.

[**The DOCTYPE**](https://www.theodinproject.com/lessons/foundations-html-boilerplate#the-doctype)

* Every HTML page starts with a doctype declaration.
* The doctype’s purpose is to tell the browser what version of HTML it should use to render the document.
* The latest version of HTML is HTML5, and the doctype for that version is simply:



<!DOCTYPE html>



- The doctypes for older versions of HTML were a bit more complicated. For example, this is the doctype declaration for HTML4:



<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">



[**HTML element**](https://www.theodinproject.com/lessons/foundations-html-boilerplate#html-element)

<!DOCTYPE html>

<html lang="en">

</html>

Noticed the word **lang** here? It represents an HTML attribute which is associated with the given HTML tag i.e. **<html>**in this case. These attributes provide additional information about HTML elements.

What is the lang attribute?

* lang specifies the language of the text content in that element.
* This attribute is primarily used for improving accessibility of the webpage.
* It allows assistive technologies, for example screen readers, to adapt according to the language and invoke correct pronunciation.

[**Head element**](https://www.theodinproject.com/lessons/foundations-html-boilerplate#head-element)

**Meta element**

We should always have the <meta> tag with the charset encoding of the webpage in the <head> element: <meta charset="utf-8">.

**Title element**

Another element we should always include in the head of an HTML document is the <title> element:

<title>My First Webpage</title>

[**Body element**](https://www.theodinproject.com/lessons/foundations-html-boilerplate#body-element)

The final element needed to complete the HTML boilerplate is the <body> element. This is where all the content that will be displayed to users will go - the text, images, lists, links, and so on.

### [VSCode shortcut](https://www.theodinproject.com/lessons/foundations-html-boilerplate#vscode-shortcut)

Generating all the boilerplate in one go, only works is .html files  
press ! and select one of the two options.

**Working with Text**

[Paragraphs](https://www.theodinproject.com/lessons/foundations-working-with-text#paragraphs)

If we want to create paragraphs in HTML, we need to use the paragraph element, which will add a new line after each of our paragraphs. A paragraph element is defined by wrapping text content with a <p> tag.

[**Headings**](https://www.theodinproject.com/lessons/foundations-working-with-text#headings)

There are 6 different levels of headings starting from <h1> to <h6>. The number within a heading tag represents that heading’s level.

[Strong element](https://www.theodinproject.com/lessons/foundations-working-with-text#strong-element)

The <strong> element makes text bold. It also semantically marks text as important;

[Em element](https://www.theodinproject.com/lessons/foundations-working-with-text#em-element)

The <em> element makes text italic.

[**HTML comments**](https://www.theodinproject.com/lessons/foundations-working-with-text#html-comments)

<!-- I am a html comment -->

Keyboard Shortcut: Select the line and press ctrl + /

[Lists](https://www.theodinproject.com/lessons/foundations-lists#unordered-lists)

Learn endless contents on list - [Creating Lists - Learn to Code HTML & CSS (shayhowe.com)](https://learn.shayhowe.com/html-css/creating-lists/)

**Links and Images**

[Anchor elements](https://www.theodinproject.com/lessons/foundations-links-and-images#anchor-elements)

To create a link in HTML, we use the anchor element. An anchor element is defined by wrapping the text or another HTML element we want to be a link with an <a> tag.

<a>click me</a>

* You may have noticed that clicking this link doesn’t do anything.
* This is because an anchor tag on its own won’t know where we want to link to.
* We have to tell it a destination to go to. We do this by using an HTML **attribute**.
* An HTML attribute gives additional information to an HTML element and always goes in the element’s opening tag.

<a href="https://www.theodinproject.com/about">click me</a>

Attribute has 2 parts: 1. Name - href  
2. Value - <https://www.theodinproject.com/about>

[**Opening links in a new tab**](https://www.theodinproject.com/lessons/foundations-links-and-images#opening-links-in-a-new-tab)

* While href specifies the destination link, target specifies where the linked resource will be opened.
* To open the link in a new tab or window (depends on browser settings) you can set it to \_blank as follows:

  <a href="https://www.theodinproject.com/about" target="\_blank" rel="noopener noreferrer">click me</a>

* You may have noticed that we snuck in the rel attribute above. This attribute is used to describe the relation between the current page and the linked document.
* The noopener value prevents the opened link from gaining access to the webpage from which it was opened.
* The noreferrer value prevents the opened link from knowing which webpage or resource has a link (or ‘reference’) to it.
* The prevention of access that is caused by noopener prevents [phishing attacks](https://www.ibm.com/topics/phishing) where the opened link may change the original webpage to a different one to trick users. This is referred to as [tabnabbing](https://owasp.org/www-community/attacks/Reverse_Tabnabbing" \t "_blank).

[**Absolute and relative links**](https://www.theodinproject.com/lessons/foundations-links-and-images#absolute-and-relative-links)

**Absolute links**

* Links to pages on other websites on the internet are called absolute links.
* An absolute link will be made up of the following parts:  protocol://domain/path

<https://www.theodinproject.com>

Relative links

* Links to other pages within our own website are called relative links.
* Relative links do not include the domain name, since it is another page on the same site, it assumes the domain name will be the same as the page we created the link on.
* Relative links only include the file path to the other page, relative to the page you are creating the link on.
* We will usually want to organize our website directories a little better. Normally we would only have the index.html at the root directory and all other HTML files in their own directory.

<body>

    <h1>Homepage</h1>

    <a href="pages/about.html">About</a>

  </body>

* Creating a new directory named pages inside odin-links-and-images and adding the about.html file inside of the directory.
* Some class it will run into unexpected errors. To fix it we use prepending “./”

<body>

    <h1>Homepage</h1>

    <a href="./pages/about.html">About</a>

  </body>

A metaphor

* Absolute and relative links are a tricky concept to build a good mental model of, a metaphor may help:

Think of your domain name (town.com) as a town,

the directory in which your website is located (/museum) as a museum,

and each page on your website as a room in the museum (/museum/movie\_room.html and /museum/shops/coffee\_shop.html).

Relative links like ./shops/coffee\_shop.html are directions from

the current room (the museum movie room /museum/movie\_room.html) to another room (the museum shop).

Absolute links, on the other hand, are full directions including the protocol (https), domain name (town.com) and the path from that domain name  
(/museum/shops/coffee\_shop.html): https://town.com/museum/shops/coffee\_shop.html.

[Images](https://www.theodinproject.com/lessons/foundations-links-and-images#images)

* To display an image in HTML we use the <img> element.
* Unlike the other elements we have encountered, the <img> element is self-closing.
* Empty, self-closing HTML elements do not need a closing tag.