Lecture Notes: Introduction to Markdown

Week 2 – Introduction to Markdown
Monday December 2, 2024
Lecture Notes adapted from [Markdown Guide]

To start programming, you need to have tools ready on your system. This document will tell you how to use markdown in a jekyll environment, provided that you have already set up your student blog.

Basics of Markdown **2**

Basics of Markdown

Markdown is essentially a preprocessing language designed to make writing files easier for people. You type into it like normal text, and jekyll processes it to html, where it is then taken to the site. Markdown also allows you to define different formats, although we will get into this later when we discuss styling. All you have to know right now is the principles of markdown and how to edit and use it.

Markdown files typically start with a header. These headers have defined properties, and look somewhat like this:

```
---
layout: default
title: CSSE 1
---
```

They can have different fields depending on how they are set up. Each field has a purpose that can usually be determined by looking at a README, backtracking and looking at the code, experimenting, or simply asking your TA's. Finding out what does what is a skill that will come through practice.

Markdown also supports dynamic headers, denoted by a "#". The headers are styled as

```
# Header 1

## Header 2

### Header 3

#### Header 4
```

and headers continue after Header 4. We recommend you to get a markdown linter (which is a code checker) that can help you write your headers.

Other syntactical elements of markdown include:

Bold	`**bold text**`
Italic	`*italicized text*`

For a **full cheat sheet** on markdown, take a look at https://www.markdownguide.org/cheat-sheet/.

Images in Markdown

In markdown, you have to specify the image path and the alt text for the image. The alt text is something that pops up when the image cannot load, usually a 1-3 word description of what the image is supposed to be or the name of the image file. The path is determined by the location of the image. If the image is in the same folder as the file you are working on, then you can just name the image:

```
![image.jpg](alt text)
```

However, if the image is in another directory, you must name the path.

```
![path/to/image.jpg](alt text)
```

Take a look at Linux pathing conventions to understand how to write the paths for the file:

1. Absolute Path: The full path to a file or directory starting from the root (/).
    - Example: /home/user/images/image.jpg
    - This path will always point to the same file regardless of your current working directory.
2. Relative Path: A path relative to the current working directory (the directory you are working in).
    - Example: images/image.jpg
    - This depends on your current location in the file system. For example, if you are in /home/user, the relative path images/image.jpg will point to /home/user/images/image.jpg.

Rules for Writing Linux Paths

1. **Case Sensitivity**: Linux paths are case-sensitive. For example, `Image.jpg` is not the same as `image.jpg`.
2. **Directory Separator**: Use a forward slash (`/`) to separate directories.
    - Correct: `images/image.jpg`
    - Incorrect: `images\image.jpg` (this is a Windows-style path)
3. **Current Directory (`.`)**: Use `.` to reference the current directory.
    - Example: `./image.jpg` refers to `image.jpg` in the current directory.
4. **Parent Directory (`..`)**: Use `..` to reference the parent directory.
    - Example: `../images/image.jpg` refers to `image.jpg` in a directory named `images`, located one level above the current directory.