Getting Started

Goals

Let’s dive into coding!

- Discover another way to view files and folders
- Basics of command line
- Variables and data types
- Getting user input

Syntax

Syntax is the set of rules for arranging phrases/sentences in a language.

Spoken/written languages like English, Spanish have their own syntax, for example

- In English, adjectives usually come *before* nouns
- In Spanish, adjectives usually come *after* nouns

Coding languages also have their own *syntax*.

- In JavaScript, you specify conditional logic in an if-statement by wrapping your conditional in `( )` parenthesis

```
if (num > 10) {
    console.log('big number');
}
```

- In Python, you specify conditional logic with a colon at the end, but no parenthesis

```
if num > 10:
    print('big number')
```

Syntax Recall

Syntax recall refers to the act of remembering the rules for a given language.

- Syntax recall is a skill on it’s own, separate from coding
- Often, the difficulty of learning to code (at first) is the difficulty of syntax recall.
- With time and continued use, you will remember how to “say” things in code.
- Be kind to yourself as you get better at this skill.

Note: Discussion

How could you practice the skill of syntax recall?

Command Line

Files and Folders

Usually people use a **GUI** to move files or view the contents of folders.

- **GUI**: Graphical User Interface. Something you can point and click.
- For example, File Explorer (Windows) and Finder (MacOS)
- We'll learn an alternative today

Command Line Interface

Controlling the operating system by typing text commands at a prompt.

Commands

To look in different files and folders, we can use the following commands:

1. `ls`: short for list, show me all the files/folders where I am
2. `cd`: "change directory", go somewhere else
3. `pwd`: "present working directory", tell me *where* I am in the filesystem
4. `mv`: "move" or rename a file
5. `touch`: create a new file
6. `cat`: short for concatenate, print out contents of file

Arguments

- `ls` and `pwd` take no **arguments**
  - Just need to type the command and press enter
  - The output is dependent on the location of where you are in the filesystem
- Many other commands need arguments, or additional info to provide with the command that changes how it works

`cd` with URLs

- `cd` needs one argument: **where** you want to “go”
- Needs to be a location in the file system, can be relative or absolute
- Location = URL (Uniform Resource Locater)

```
$ cd /Users/someuser
$ cd ..
$ cd ~/Downloads
$ cd somefolder
```

- `~` means “this user’s home directory”
- `..` means the directory above where I am

Move files

- Need two arguments for `mv`
- **What** to move, and **where** to move it to

```
$ mv somefile ~/Downloads
```

Rename files

- Need two arguments for `mv`
- **What** to rename, and the **new name**

```
$ mv somefile anothername
```

Create new files

- Need one argument for `touch`: **name** of file

```
$ touch mynewfile
```

Print file

- Need one argument for `cat`: which file to print
- Can provide name of file in current directory
- Can also provide URL

```
$ cat somefile
```

Tab Completion

- Press tab when you’re typing a long filename
- If there’s only one possible file that could match, it will autocomplete
- Also works with commands

Git

What’s git?

Git is a program

Do not confuse it with Github, which is a website. We will talk about Github later.

Why git?

- `resume.pdf`
- `resume2.pdf`
- `resume-final.pdf`
- `resume-FINAL.pdf`
- `resume-FINALFINAL.pdf`
- `resume-FINALFINALFINAL.pdf`

Imagine this, but for hundreds of files of code

- Git is a **version control** system
- Allows individuals and teams to track changes to code projects
- Prevents bugs and loss of progress
- Allows you to view the long history of a project in snapshots

Git Repository

- The most basic unit of git is a **repository**
- A repository is a code project
- It is contained in a single folder (a.k.a. directory)

- It has files inside, and perhaps subdirectories

Git Basics

- `git init`: initialize a local repository *right here in this folder*
- `git add`: add some files to be tracked (always!)
- `git commit`: create a snapshot of the files I'm tracking
- `git status`: tell me about this repo
- `git log`: show me a list of the commits, in reverse order
- `git diff`: what have I changed since last commit

Helpful to know

- Once something has been committed, it's **very difficult** to delete it
- git is **extremely complex** and there are many more commands than what we'll cover here
- Don't copy/paste git commands from the internet – you can do serious damage to your code

Variables & Datatypes

Variables

Variables are used to store values in JavaScript. There are three ways to create, or declare, a variable:

```
var name = "Thundercat";

let name = "Thundercat";

const name = "Thundercat";
```

Variable Declaration

- `var` : create a global variable (value might change)
- `let` : create a local variable (value might change)
- `const` : create a constant (value will not change)

Datatypes

- Number `let newNumber = 7`
- String `let newString = "hello"`
- Boolean `let newBoolean = true`
- Undefined `let newUndefined = undefined`
- Null `let newNull = null`
- Array `let newArray = []` (later)
- Object `let newObject = {}` (later)

Numbers

- Numbers refer to any integer or “floating point “number.
- NaN (which stands for Not a Number) is, ironically, a number datatype.
- Math operations can be performed on Number data types.

```
let num = 2;
let numTwo = 2;

let sum = 2 + 2; //=> 4

let summedNums = num + numTwo //=> 4
```

Above, both the ***sum*** and ***summedNums*** variables have a value of 4.

Strings

- Strings are groups of characters (letters, numbers, or special characters).
- A string can be created using double, single, or back-tick quotes.

```
let name = "Thundercat";

let name = 'Thundercat';

let name = `Thundercat`;
```

- Strings can also be added together, an operation called **concatenation**.
- Using back-ticks, variables can also be placed directly into a string. This is known as a template string, or template literal.

```
let name = "Thundercat";
let greeting = "Hello, "

let nameGreet = greeting + name;
let tempString = `Hello, ${name}`;
```

Booleans

The boolean data type has only two values: true and false.

```
const isThundercatAwesome = true;

const isItPartyTime = false;
```

Null/Undefined

- Null means the absence of a value, or that the value represents is nothing.
- Undefined is a data type itself, and occurs when a variable is declared but not given a value;

```
let nothing = null;

let notDefined;
```

Mathematical Operators

Intro to Operators

While there are many operators, a lot of which we will cover in this course, the most basic and commonly used operators are the mathematical operators. They are as you would expect:

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division

There is one additional operator that you might not be familiar with:

- `%` modulo (remainder)

Operators in Action

Let’s take a look at some of these mathematical operations in action.

```
let a = 6 + 2 // Addition operator: a will be equal to 8
let b = 6 - 2 // Subtraction operator: b will be equal to 4
let c = 6 * 2 // Multiplication operator: c will be equal to 12
let d = 6 / 2 // Division operator: c will be equal to 3

let e = 6 % 2 // Modulo operator: e will be equal to 0 because 6 / 2 has no remainder
let f = 6 % 4 // Modulo operator: f will be equal to 2 because 6 / 4 has a remainder of 2
```

Operators with Variables

Just like in our previous example, mathematical operators can be used with variables that hold numerical values, for example:

```
let x = 10
let y = 5
let z = 2

let a = x + 7 // a will be equal 17
let b = x * y // b will be equal to 50
let c = b / z // c will be equal to 25
```

Running JavaScript

JavaScript on the front end

- Most typically, you'll see JavaScript running in a web browser (which we'll see later in the course)
- Today, we'll run JavaScript locally, on the command line, using Node.js

Steps to run JavaScript code

1. Make sure Node is installed
2. Create a file, use a `.js` file ending
3. Run the following command (**use your file's name!**)

```
$ node myfile.js
```

Celebrate!

Showing Output

- One of the most common things you'll do in JavaScript is create output
- This gives you feedback about whether your code is working!
- Use `console.log()`

```
console.log("Hello world");

let message = "Hello again, world";
console.log(message);
```

Basic If-statements

Check for equality

```
let temperature = 75;

if (temperature === 75) {
    console.log("It's perfect!");
} else {
    console.log("It's fine.");
}
```

Compare value

```
let age = 27;

let age2 = 28;

if (age > age2) {
  console.log("first age is bigger");
} else {
  console.log("second age bigger");
}
```

Summary

The objectives of this lecture are listed below. As you review this material, make sure you are comfortable with them.

- Student can navigate the filesystem within their command line environment
- Student can execute basic commands within their command line environment
- Student can create variables of all basic data types, including integers, strings, floats, and booleans
- Student can utilize basic mathematical operators
- Student can use basic git commands
- Student can console.log values
- Student can run JavaScript files using Node
- Student understands basic conditional statements

The End