
Embrace Opportunity

Software Engineering Day 2

Deeper Dive Into JavaScript, Git & GitHub

Today’s Overview

- JavaScript Deep Dive
- Version Control Best Practices with GitHub & Git

- Cheat Sheet:
  - https://byrondev121.github.io/ix-vue-press/
  - https://byrondev121.github.io/ix-vue-press/day-2

JavaScript

Core Web Technology

JavaScript - Introduction

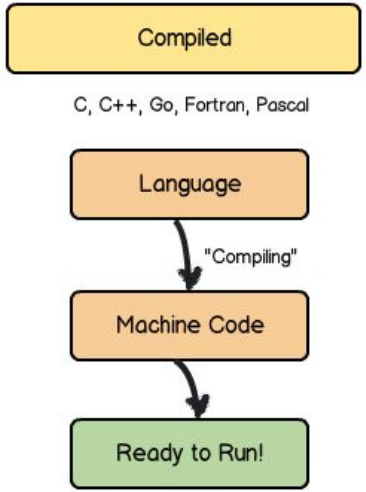
- Single threaded, non-blocking, asynchronous, high level object oriented programming language
- Interpreted (just-in-time compiling)
- ECMAScript specification (ES14)
- Runs in the browser (e.g. google V8 Engine, Mozilla Firefox: SpiderMonkey, Safari: JavaScriptCore)
- We can also run as a server thanks to run-time environments like node JS
- Used in web frameworks like Angular, React, VueJs
- Used to build very fast full stack applications with tools like express JS
- Used to build mobile frameworks like React Native and Ionic
- Used in desktop application development with frameworks like Electron

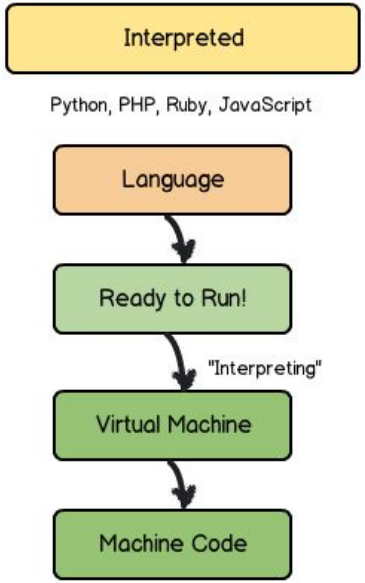
JavaScript - Introduction

- **Interpreted** (just-in-time compiling)
  - Typically Slower
  - Can lead to run-time errors (no compile step)

vs.

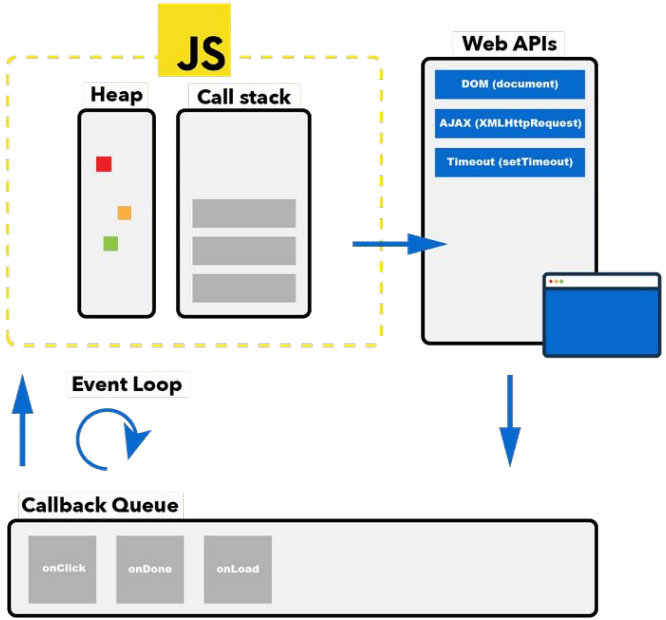
- **Compiled**
  - Typically Faster
  - Robust (Pick up syntax errors in compile step)





JavaScript - Introduction

- JavaScript is composed of:
  - Call stack
  - Memory Heap
  - Callback queue
  - and a few other API's, i.e. DOM API, Timeout, etc.



- JS using an Event Loop, model which makes it non-blocking and asynchronous.

JavaScript - Crash Course

- Variables & Data Types
- Arrays
- Objects Literals
- Methods for string, arrays objects, etc
- Loops - for, while .. of ForEach, map
- Conditions
- Functions

JavaScript - Data Structure & Data Types

- Declaration:
  - let
    - Local declaration
    - New method
  - var
    - Global declaration
    - Old method
  - const
    - Declares a constant/unchanging variable
    - Immutable declaration
    - New method

JavaScript - Data Structure & Data Types

```js
// VAR - old
var firstName1 = "Byron";

// CONST - new - Immutable
const lastName1 = "de Villiers";
lastName = "Smith";
// const lastName; // This will result in an error

// LET - new - Mutable
let age1;
age1 = 42;
```

JavaScript - Data Structure & Data Types

- Data structures for data types:
  - Primitive values
    - Immutable values
  - Arrays
    - Mutable values
  - Objects
    - Mutable values

JavaScript - Primitives

- Primitive characteristics:
  - Immutable - they cannot be altered
  - Not an object
  - Has no methods or properties
- The variable may be reassigned to a new value, but the existing value cannot be changed.

JavaScript - Primitives

- String

```js
// String
let firstName = "Byron";
let lastName = "de Villiers";
let address = "11 Zinnia";
```

- Number

```js
// Numbers
let age = 29;
let rating = 4.5;
```

- BigInt
  - Safely represents large numbers that Number can't precisely store.

```js
let x = 22222222222222222222222222n;
```

JavaScript - Primitives

- Symbol
  - Unique
  - Can be used as keys for object properties

```js
let occupation = Symbol('engineer');
```

- Boolean

```js
// Boolean
let isCool = true;
let isVeryCool = false;
```

JavaScript - Primitives

- Undefined
  - Value that is not assigned

```js
let hairColor;
```

- Null
  - Empty / unknown value

```js
// null
address = null;
```

- Documentation

JavaScript - Strings

String instance methods

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

```js
let s = `My name is ${firstName}`;

// string properties
console.log(s.length);

// string methods
console.log(s.toLocaleUpperCase());

s = s.toLocaleUpperCase();

s.indexOf("Y", 2);

s.substring(0, 7);

s1 = s[0];

s2 = s[8];

console.log(s1, s2);
```

JavaScript - Object

- The other type of data structure in JavaScript are Objects.
  - Mutable
  - Complete data structure to store collection of data

```js
// Object
let person1 = {
  firstNameKey: "firstNameValue",
  lastNameKey: "lastNameValue",
  ageKey: "ageValue",
};
```

JavaScript - Objects

Key value pairs

- Values can be multiples types
- Values can be functions

```js
let person = {
  // Object properties:
  firstName: "John",
  lastName: "Doe",
  age: 29,
  hobbies: ["golf", "surfing", "sport", "music"],
  address: {
    street: "Zinnia",
    city: "Potchefstroom",
    province: "North West",
  },
  // Object methods:
  getFullName: () => {
    return this.firstName + " " + this.lastName;
  },
  getAddress: () => {
    return (
      this.address.street +
      " " +
      this.address.city +
      " " +
      this.address.province
    );
  },
};

console.log(person);

// Calling object methods
console.log(person.getFullName());
console.log(person.getAddress());

// Adding properties:
person.email = "byron@mail.com";
person["email"] = "byron@mail.com";
console.log(person);
```

JavaScript - Array

- An array is an object data type that contains a list of data types or combinations of them.
- Syntax:

```js
// Array
let arrayNumber1 = [1, 2, 3, "a", "b", "c"];
```

- Array manipulation:
  - pop() - removes the last element of the array
  - push() - add item to end of the array
  - .length - finds the size of the array
    - Array property

```js
console.log(alphaNumeric.length);
```

JavaScript - Arrays

```js
// Initializing an array - old
let nums1 = new Array(1, 2, 3, 4, 5, 6);
console.log(nums1);

// Initializing an array - new
let alphaNumeric = [1, 2, 3, 4, 5, "a", "b"];
console.log(alphaNumeric);

let fruits = ["orange", "pear", "apple"];

console.log(fruits);

// get element in array (remember it's zero indexed):
console.log(fruits[2]);

// array properties
console.log(fruits.length);

// array methods
fruits.push("strawberries");
console.log(fruits);
fruits.unshift("grape");
console.log(fruits);
const lastFruit = fruits.pop();
console.log(lastFruit, fruits);
const index = fruits.indexOf("orange");
console.log(index, fruits);
```

JavaScript - Higher Order Array Methods

- Higher order functions
  - Function that accepts one or more functions as arguments and/or returns a function as their result
- Callback function
  - Function that is passed into another function as an argument
- Array functions:
  - Documentation

JavaScript - Higher Order Array Methods

- Array functions:
  - .filter()
    - Executes a callback function on each element in an array
      - The callback function for each element must return either true or false
      - Returns new array with any element which the callback function was true
  - .map()
    - Executes a callback function on each element in an array
      - Returns a new array made up of the returned values

JavaScript - Higher Order Array Functions

```js
const array1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
// filter by even values:
const filteredArray = array1.filter((x) => {
  return x % 2 == 0;
});

console.log(array1);
console.log(filteredArray);

const array2 = [10, 2, 5, 4, 8, 6];
// sort smallest to largest
const sortedArray = array2.sort((a, b) => {
  return a - b;
});

console.log(array2);
console.log(sortedArray);
```

JavaScript - Higher Order - Extra Example

- Filter the blogPosts based on the categoryId

```js
blogPosts = blogPosts.filter((x) =>
  categoryId !== undefined
    ? x.categories.find((y) => y.id.toString() === categoryId)
    : true
);
```

JavaScript - Variable Naming

- Variable naming convention:
  - camelCase:
    - userName,
    - blogTitle
  - snake_case:
    - my_awesome_variable,
    - blog_description
- In a project always stick to a single naming convention.
  - JavaScript camelCase is most commonly used.
    - Linters (code analysis tool) will give errors if the wrong naming convention is used for that certain language.

JavaScript - Operators

- Operators that perform operations on your data.
- Types of operators:
  - Arithmetic
    - Mathematical operations
  - Assignment
    - Assign data or value to a variable
  - Comparison
    - Compare two values, which returns a boolean value
  - Logical
    - Check whether one or more expression is true or false
  - “typeof”
    - Checks the data type

JavaScript - Arithmetic Operators

- Addition
  - x + y
- Subtraction
  - x - y
- Multiplication
  - x * y
- Exponentiation
  - x ** y
- Division
  - x / y
- Remainder
  - Also called Modulo
  - x % y
- Increment
  - x++
- Decrement
  - x - -

JavaScript - Assignment Operators

- Assignment:
  - x = y
- Addition assignment
  - x += y
    - x = x + y
- Subtraction assignment
  - x -= y
    - x = x - y
- Multiplication assignment
  - x *= y
    - x = x * y
- Division assignment
  - x /= y
    - x = x / y
- Remainder assignment
  - x %= y
    - x = x % y

JavaScript - Comparison Operators

- Equal:
  - x == y
  - Returns true if variables are equal
- Not equal
  - x != y
  - Returns true if variables are not equal
- Strict equal
  - x === y
  - Returns true if variables are equal and of the same type
- Strict not equal
  - x !== y
  - Returns true if variables are not equal or have different types

JavaScript - Comparison Operators

- Greater than
  - x > y
  - Returns true if x is greater than y
- Great than or equal
  - x >= y
  - Returns true if x is greater than or equal to y
- Less than
  - x < y
  - Returns true if x is less than y
- Less than or equal
  - x <= y
  - Returns true if x is less than y or equal to y

JavaScript - Logical Operators

- Logical:
  - AND
    - x && y
    - Returns true if all variables are true
  - OR
    - ||
    - Returns true if either variable is true
  - NOT
    - !
    - Reverses the result

JavaScript - typeof Operators

- typeof
  - Returns the type of data as a string
  - E.g.
    - typeof(5)
      - Returns number
    - typeof(“John”)
      - Returns string
    - typeof(true)
      - Returns boolean

JavaScript - Control Flows

- Control flows allows specific code to be run bases on defined criteria
- Control flow incorporates:
  - Conditional statements
  - Loops
- Conditional statements:
  - if…else statement
  - switch…case statement
- Loops
  - for statement
  - while

JavaScript - Control Flow - Conditionals

- if…else statement
- Run only when a specific condition is met
  - if statements can be nested

```js
const num = 10;

// If statement:
if (num == 10) {
  console.log("num = 10");
} else if (num > 5) {
  console.log("num > 5");
} else {
  console.log("num < 5");
}
```

JavaScript - Control Flow - Conditionals

- switch…case statement
- Alternative to if…else statements, for readability when handling multiple conditions

```js
const num = 10;
// Switch statement:
switch (num) {
  case 10:
    console.log("num = 10");
  case 5:
    console.log("num = 5");
  default:
    console.log("num not 5 or 10");
}
```

JavaScript - Control Flow - Conditionals

- Ternary Operator:
  - The only JavaScript operator that takes three operands:
    - Condition
    - Expression to execute for true condition
    - Expression to execute for false condition
  - Alternative to an if…else statement

```js
// ternary operator:
const lessThanFive = num < 5 ? true : false;
console.log(lessThanFive);
```

JavaScript - Control Flow - Loops

- for statement:
- Expressions:
  - Initialization
    - declaring a variable used as the source of the loop condition
    - x = 0;
  - Condition
    - logic block will only run when the condition is true
    - x < 10;
  - Arithmetic
    - variable is incremented or decremented after each loop

JavaScript - Control Flow - Loops

```js
// For loop example

console.log("starting for loop");

for (let i = 0; i <= 10; i = i + 1) {
  console.log(i);
  console.log("execute this line if code in the for loop");
  // i = i + 1
}

console.log("for loop finished");
```

JavaScript - Control Flow - Loops

- while statement
- Runs as long as condition is true
- while loops are used over for loops when the amount of iterations you want to happen are unknown

```js
// While loop example
let i = 0;
while (i <= 10) {
  console.log(i);
  console.log("execute this line if code in the while loop");
  i++;
}
```

JavaScript - Functions

- A function is a block of code that performs a specific task.
- Fundamental characteristic of functions is the ability to call a single function multiple times where similar actions are performed throughout a project.
- JavaScript built-in functions
  - alert(message), prompt(message, default), confirm(question)
- Function declaration:

```js
// function definition:
function addTwoNumbers(num1, num2) {
  return num1 + num2;
}

// calling/executing the function
const result = addTwoNumbers(5, 4);

//logging result:
console.log(result);
```

JavaScript - Functions

- Functions can have a return
  - This will have an output where the function is executed from

```js
// Function definition
function isPerfectSquare(x) {
  const sqrRoot = Math.sqrt(x);
  return sqrRoot * sqrRoot == x;
}

// Calling/executing the function:
const resPrfSqr = isPerfectSquare(16);

// Logging result:
console.log(resPrfSqr);
```

JavaScript - Function Extra Example

```js
function areTasksDone(todos) {
  for (let todo of todos) {
    if (!todo.completed) {
      return false;
    }
  }

  return true;
}
let todos = [
    {
        id: 1,
        title: "Learn HTML, CSS and JS",
        completed: false
    },{
        id: 2,
        title: "Write code",
        completed: false
    },{
        id: 3,
        title: "Get a SDE job",
        completed: false
    }]
console.log(areTasksDone(todos));
todos[1].completed = true;
console.log(areTasksDone(todos));
```

JavaScript - Arrow Functions

- Another syntax to create function expressions.
  - Benefits over function syntax:
    - Writing short, more straightforward functions
    - Single-line functions, with implicit return
  - When not to use:
    - When you define an object method

```js
const greeting = (name) => console.log(‘Hello, $(name)!’);
```

JavaScript - Constructor Function

- Special function that create and initializes an object instance of a class
- Constructor is called when using the keyword: “new”
- Operation:
  - New empty object is created
  - “this” refers to the new object and becomes the current instance
  - New object is returned

```js
// function definition:
function logDate() {
  console.log(new Date());
}

// call/executing the function:
logDate();
```

JavaScript - Constructor & Prototype

- Properties and methods can be added to a constructor using a prototype

```js
function User(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

User.prototype.getFullName = function() {
  return this.firstName + " " + this.lastName;
}

const user = new User('John', 'Doe');
console.log(user.getFullName());
```

JavaScript - ES6 Classes

- ES6(2015) introduced classes, which allowed cleaner writing of classes with methods (Syntactic sugar).

```js
class User {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}

function getFullName() {
  return this.firstName + " " + this.lastName;
}

const user = new User('John', 'Doe');
console.log(user.getFullName());
```

JavaScript - Additional

- Local variables
  - Variables declared inside a function are only accessible inside the function
- Global variables
  - Variables declared outside of any function/block can be accessed through the project and from within a function
- Official documentation:
  - Documentation
- Additional resource:
  - JavaScript Info

Version Control

Git & GitHub

Git & GitHub - Introduction

- What is version control?
  - Definition: A system to record changes to files or sets of files over time.
  - Purpose: Allows for recalling specific versions of files, facilitates collaboration among multiple contributors, and helps track the history of a project.
- Git
  - Description: A DevOps tool used for version control.
  - Benefits:
    - Track history of a project.
    - Work collaboratively with multiple contributors.
    - Revert back to previous versions if needed.

Git & GitHub - Introduction

- GitHub
  - Description: A web-based platform that hosts Git repositories.
  - Features:
    - Collaboration through pull requests and code reviews.
    - Issue tracking and project management.
    - Continuous Integration/Continuous Deployment (CI/CD) workflows.

Git & GitHub - Creation

- Initializing git:
  - This created the .git folder inside your project which will contain the information needed for version control.

```
git init
```

Git & GitHub - Files

- Inside your project, create a new file:
  - tutorial-page.txt
- This file will be “untracked” from the git history.
- Check the status of the project

```
git status
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        tutorial-page.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Git & GitHub - Files

- Adding the file to be tracked (AKA Staged)
  - This is used to add all unstaged files

```
git add .
```

```
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   tutorial-page.txt
```

Git & GitHub - Committing

- Once the files have been staged, they can be committed.
    - Always keep the commit message descriptive, to make it easier to track the history.

```
git commit -m "My cool new version"
```

- If there is an issue, or the file needs to be worked on more after staging a file:
    - It will unstage the file but keep any changes locally
    - This must happen prior to git commit

```
git reset <filename>
```

Git & GitHub - Push To Repository

- Once the file/s have been committed, the last step is to push the changes to the remote repository. (Aka GitHub)
  - First add to the project the repository url

```
git remote add origin  <REMOTE_URL>
```

  - Then the changes can be pushed to that repository
    - master would be the branch you are pushing your changes to
    - main/master is by default the main branch

```
git push <remote> <branch>

git push origin master
```

Git & GitHub - Branches and Feature Branches

- Branches are snapshots of your current changes
- An extremely important aspect in Software Engineering.
  - For fixes or features, working on a new branch is always recommended.
    - This helps for clean, effective and stable code.
- Git Terms:
  - Head
    - Reference to the current commit of the working directory.
    - This updates with the next commit made on that branch.
  - Tag
    - Reference pointing to a commit that never moves.
    - Used to define an important history portion of a project (Release)

Git & GitHub - Branches and Feature Branches

- Creating branch:
- Making new feature branch working directory:

```
git branch <branch-name>

get checkout <branch-name>
```

- Shortcut to create and check out

```
git checkout -b <branch-name>
```

Git & GitHub - Branches and Feature Branches

- Merging branches:

```
git merge <branch-name>

git merge master
```

- Show local branches:

```
git branch -l
```

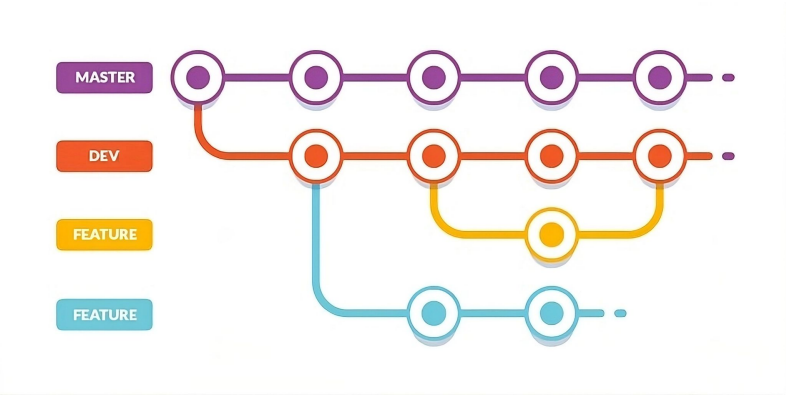
Git & GitHub - Branches and Feature Branches

- Show remote branches:

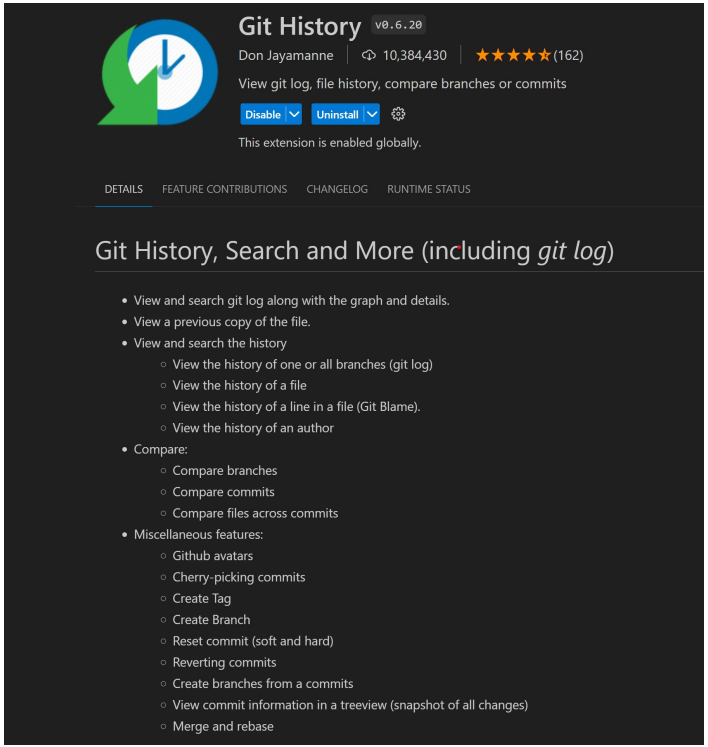
```
git remote show origin
```

Git & GitHub - Branching and Feature Branches

- Illustrated:



- Install the VSC Git History extension



Git & GitHub - Revert & Reset & Delete

- Resetting
  - Undo changes in your working directory to a specific commit, removing all commits until that point.
  - git reset
    - -- soft
      - Aims to change the HEAD to a specific commit
    - -- hard
      - Forces HEAD to a specific commit

Git & GitHub - Revert & Reset & Delete

- Reverting
  - Undoes a single commit by reverting back to the staged files before the commit
  - git revert
    - git revert <commit hash>
      - Removes the changes from the single commit without impacting other commits.
- Delete
  - git branch -d delete-this-branch
    - Only affects the local branch
- git push origin --delete delete-this-branch
  - Affects the remote repository

Git & GitHub - Commonly Used Commands

- Creating
  - git init
  - git branch -m master
    - renames main branch to master
  - git add .
    - Stages all files
    - Or git add <file name.ext> to stage just a file
  - git commit -m "Message"
  - git remote add origin <repo url>
  - git push origin master
- Merge
  - git checkout “branch” (branch you want to merge into)
  - git merge “branch” (branch you want to merge with)

Git & GitHub - Commonly Used Commands

- Cloning:
  - git clone <'repo address'>
- Other helpful commands
  - git reset <file>
    - unstaged commits but keeps changes locally
  - git restore <file>
    - restores file from the repo
  - git stash
    - Stashes changes that can be reapplied later
- Feature branching
  - git checkout -b myFeature dev
    - Both creates and checkout into a new feature branch

Git & GitHub - Documentation

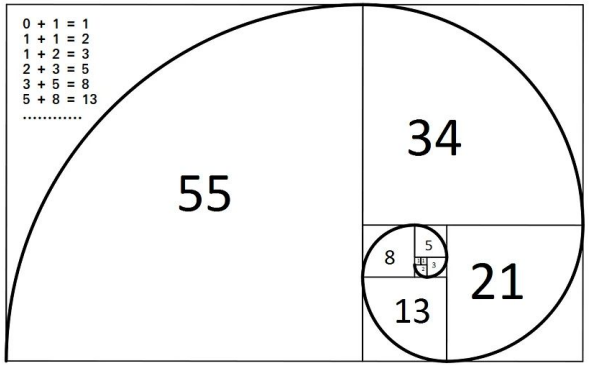
- Github
  - Official Documentation
- Git
  - Official Documentation
  - Git Cheat Sheet

Exercise

Exercise: JavaScript

Exercises: JavaScript

- Write a function to print out the first 10 digits of the Fibonacci sequence.
- Fibonacci sequence:
  - Each number is the sum of the two preceding ones
  - Equation: $x_n = x_{n-1} + x_{n-2}$



- Expected result: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Homework

Apply what we have learned

Homework

- Write a JavaScript function that takes in an array of length == 3 which represents the 3 lengths of the three sides of a triangle.
  - Calculate the area of the triangle, and return the area as a number.
- GitHub and Git
  - Create a GitHub repo for the iX course
  - Create 2 new directories in the workspace for your personal portfolio web page from the day 1 and the JavaScript function from the homework.

Next Class

Introduction to React JS