Terms

Shallow copy of an object is a copy whose properties share the same reference as those of the source object that the copy was made, resulting in when you change the source or copy, you change the other object as well, contrasting to **deep copy** where the source and copy are completely independant

Variables

Container for storing data
Declaration for (var, let, const)

```
let name = “Fred” //strings
let age = 21; //numbers
let boolOp = true; //bools
console.log(name);
const x = 1; //cannot change the value of x since it is considered a const
```

Empty string is considered false;
Anything in string is considered true;

Arithmetic expressions

Operators for arithmetic (* + - / %)

```
let x = 4 + 6; //10
let y = 2 - 1; //1
let z = 1 / 1; //1
let total = x * y * z; //10
let firstMod = 11 % 2; // 1 is the remainder
```

User Input

Easy way to accept user input with a window prompt
When accepting user input it is a **STRING**

```
let username = document.getElementById(“myText”); //use a button onclick func to call
let username = window.promt(“Please enter your name”);
```

Type Conversion

Can use typeof to check type of variable, can change with type conversion

```
let x = 1;
console.log(x, typeof x); //will output “1 string”
```

- Number()
- String(21)
- Boolean(“”) //is false since is empty string
- Boolean(“Hello”)// is true since is not empty string

Math

Intrinsic object that provides basic mathmatic functions and constants

- Math.round()
- Math.floor()
- Math.ceil()
- Math.pow(base, exponent) //Math.pow(2,3) is 8, 2 to power 3
- Math.sqrt()
- Math.abs();
- Math.max(x, y, z) //returns max between values
- Math.min()
- Math.PI;

String Methods

Useful string properties and methods, strings are immutable

- .length
- .charAt() //returns char at what string position, string.charAt(2)
- .indexOf() //returns first index of char, string.indexOf("H")
- .lastIndexOf() //last occurrence of
- .trim() //gets rid of empty spaces before and after the string
- .toUpperCase() //string.toUpperCase makes all char capital
- .toLowerCase()
- .replaceAll(x, y) // takes two arguments, first is what to replace, second is what is replacing it
- .slice(x, y)//extracts a section of the string and returns it as a new string without modifying original, with 1 or 2 values, starting index and ending, if one value it will copy everything until end of string
- .subString(x) // rest of the string at and after index x

Method Chaining

Calling one method after another in one continuous line of code
let letter = userName.charAt(0).toUppeCase().trim();

If Statements

Basic form of decision making, if true then do, else
if (let age = 10 > 5) {console.log("Age is greater than 5")}

Checked Property

document.getElementByID("test").checked// tests if it is checked

Switches

Statement that examines a value against many case clauses

```
let grade = “a”;
switch(grade) {
        case(“a”): console.log(“Great”);
        case(“b”): console.log(“Good enough”);
        default: console.log(“You sucked or grade was not a letter grade”);
}
```

AND/OR/NOT logic

Gives ability to check more than one condition concurrently (&& ||)

```
if (grade === “a” && name === “Austin”) {console.log(“Man I’m great!”);}
if (!true) {doThis();}
```

While Loops

Repeat some code while some condition is true, can repeat infinite times

```
while(let score = 0 < 10) {score += 1;}
```

Do While Loops

Do something, then check condition of while loop, then repeats if true, so always runs once

```
let sum = 0;
do {sum += 1} while (sum < 10)
```

For Loops

Does code a certain amount of time, better for something that you want to end after certain amount of time

```
for (let i = 0; i < 10; i += 1){console.log(`The count is ${i}`);}
```

Can use a **for of** loop. for (let x of y) to iterate over multiple variables, every x of y in this case

```
for (let numberItems of cart) {console.log(`You have ${numberItems}) in cart`}
```

Break and Continue Statements

Break breaks out of loop entirely, continue will skip that iteration of the loop

```
for (let i = 0; i < 10; i+=1){if (i != 7) {continue;} break} //will break when i === 7
```

Nested Loops

Loop inside of another loop, once the inside loop is done once, does another iteration of outer loop

Functions

Define some code once, use it many times

```
sayWelcome(Austin);
function sayWelcome(name) {console.log(`Welcome ${name}`)}
```

Can use nested functions for security, nested functions are hidden from outside the function that is used to call them

```
function login(){
        greet();
        function greet() {
                console.log("hello");
        }
}
```

Return Statement

returns a value back to the place where you invoked a function

```
function sum(a, b) {return (a + b)}
```

Ternary Operator

Shortcut for if/else statements, condition ? ifTrue : ifFalse

```
(x > 7 ? console.log("x is greater than 7") : console.log("x is less than 7"));
```

Var vs Let

let is limited to block scope, var is limited to a function, use let over var usually

Template Literals

delimited with backticks ``, and can use ${} for variables

```
let x = 1;
console.log(`You have ${x} items in cart`);
```

Format Currency

toLocaleString() returns a string with a language sensitive representation of this number

number.toLocaleString(locale, {options})

- locale specifies that language, undefined is default set in browser
- options is the object with formatting options

```
let x = 123456;
x.toLocaleString("en-US") //if consoled, will show with comma 123,456
x.toLocaleString("en-US", {style: "currency", currency : "USD"}) // $123,456
```

Arrays

a variable that can store multiple values

```
let myNums = [1, 2, 3, “element”];
```

- .push(4) //adds element to last index
- .pop() //removes last element
- .unshift(0) //adds element to beginning
- .shift() //removes element from beginning
- .indexOf(“element”) //returns index that was found at that index like a string, -1 if not found
- .sort() //used for strings
- .sort().reverse() //reverse the sorting
- .forEach(func) //executes a provided callback function, upto 3 arguments forEach func

```
let arr1 = [“todd”, “fred”];
arr1.forEach(capitalize)
arr1.forEach(print)
function capitalize (element, index, array) {
        array[index] = element[0].toUpperCase() + element.subString(1)
}
function print(element){
        console.log(element);
}
```

- .map(func) // executes a provided callback function once and for each array element and creates new array

```
let numbers = [1, 2, 3, 4];
let squares = numbers.forEach(square)
function square(element) {return Math.pow(element, 2)}
```

- .filter() //creates a new array with all elements that pass the test provided by callback function

```
let ages = [5, 8, 22, 18, 14];
let adults = ages.filter(isAnAdult)
function isAnAdult(element){return element >= 18}
```

- .reduce(func) //reduces an array to a single value

```
let prices = [5, 10, 15, 17, 25];
let total = prices.reduce(checkout)
function checkout(total, element) {
        return total + element
}
```

- .findLast(func) // finds last value that satisfies provided testing function (returns true)
- .concat(array) //concats 2 arrays, array3 = arr1.concat(arr2)
- .includes(variable) //checks if array includes variable
- .splice(start, deleteCount, variable) //can insert or update value in array, start is index, delete is 0 to insert, delete 1 to remove at start index, and variable is what to be changed

- .slice(start, end) //slices array at **start** index to **end** index
- .sort(func) //does not require callback function, then compares first element of array element with rest and modifies array in place

2D Arrays

An array of arrays, useful for grids of information

```
let fruits = [“apple”, “orange”];
let vegetables = [“potatoes”, “onions”]
let groceryList = [fruits, vegetables];
```

Can access with either groceryList[0][0], or iterate through nested for/forof loops

Spread Operator …

Allows an iterable such as an array or string to be expanded in places where zero or more arguments are expected (basically unpacks elements)

```
let numbers = [1, 2, 3, 4, 5]
console.log(...numbers) // will now display every element in numbers array
```

Rest Parameters

Represents an indefinite number of parameters, packs arguments into an array

```
let arr1 = [1, 2, 3, 4, 5]
function sum(...numbers){ //make sure rest parameter, or …numbers in this case, is last element
        let total = 0;
        for (let number of numbers){
                total += number;
        }
}
```

Callback Functions

a function passed as an argument to another function

```
let total = sum(2, 3, DisplayResult)
function sum(x, y, doSomething) {return x + y; doSomething(result)}
function DisplayResult(output) {console.log(result)}
```

Function Expressions

A function without a name, or is an anonymous function

```
const greeting = function() {console.log(“Hello”)}
greeting()
```

Arrow Functions

compact function
const greeting = () => {console.log("Hello")}

Maps

An object that holds key-value pairs of any data type

- .get(key) //gets the associated value based on key
- .set(x, y) //adds new key-value of [x, y]
- .delete(key) //removes key-value pair based on key
- .has(key)//checks if map contains Kv pair based on key
- .size(mapName) //shows how many kv pairs in map based on its name

```
const store = new Map([
        ["t-shirt", 21],
        ["underwear", 6],
        ["pants", 77],
]);
store.forEach((key, value) => console.log(`${key}, $${value}`));
let shoppingCart = 0;
shoppingCart += store.get("t-shirt"); //is now 21
```

Objects

A group of properties and methods, or what an object has and what an object can use to access properties/methods

```
const person1 = {
        name: "fred",
        age: 21,
        fav-color: blue,
        food: pizza,
        foodie : function(){console.log(`My fav food is ${this.food}!`)}
}
console.log(person1.fav-color)
```

This keyword

A reference to a particular object, object depends on immediate context

Classes

A class is a blueprint for creating objects, defines what properties and methods they have, use a constructor for unique properties

class Player { score; pause() {console.log("You paused the game")}}
let player1 = new Player(); player1.pause(); //creates a new player object, then calls pause function in the player object

Constructors

A special method of a class, accepts arguments and assigns properties

```
class Student {
        constructor(name, age, gpa) {
                this.name = name;
                this.age = age;
                this.gpa = gpa;
        }
        study() {console.log(`Student ${this.name} is studying.`)}
}
```

Static Keyword

A member that is static belongs to a class, not any object that is created from the class

```
class Car {
        static numCars = 0;
        constructor(model){
                this.model = model;
                Car.numCars += 1; //everytime an object created, this is updated for Car class
        }
        static startRace() {console.log("The race is starting!")}
}
console.log(Car.numCars);
Car.startRace()
```

Inheritance

A child class can inherit all the methods and properties from another class

```
class Parent {
        constructor(firstName, lastName, age, walkSpeed){
                this.firstName = firstName;
                this.lastName = lastName;
                this.age = age;
                this.walkSpeed = walkSpeed;
        }
}

class Baby extends Parent {
        constructor(firstName, lastName, age, toddleSpeed) {
                super(firstName, lastName, age); //what both child and parent share
                this.toddleSpeed = toddleSpeed;
        }
}
const baby1 = new Baby("Tom", "Franklin", 0, 21)
```

Super Keyword

Refers to parent class, commonly used to invoke the constructor of a parent class

Getters and Setters

Get binds an object property to a function when that property is accessed
Set binds an object property to a function when that property is assigned a value

```
class Car {
        constructor(power){
                this._gas = gas;
                this._power = power; //use _ to precede word to show it is a protected property
                                     //is readOnly
        }
        get power() {
                return this._power;
        }
        get gas() {
                return {`${this._gas} Liters`}
        }
        set gas(amount) {
                this._gas = amount;
        }
}
const car1 = new Car(400);
console.log(car1.power);//does not need to invoke method()
car1.gas = 12;
```

Anonymous Objects

Objects without a name, Not directly referenced, less syntax so no need for unique names
let cars = [new Car(25, 200), new Car(14, 300)];

Error Handling

Object with a description of something went wrong
try {x = 1; if (x != 2) throw “x does not equal 2”;} catch(error){console.log(error)} finally {console.log(“This finally runs no matter if an error occurred or not”)}

SetTimeout

Invokes a function after a number of milliseconds, it’s an asynchronous function, so it does pause execution
setTimeout(greetings, 1000)
function greetings(){console.log(“Hello”)}

SetInterval

Similar to setTimeout(), but continuously calls the function every number of milliseconds

```
myTimer = setInterval(greetings, 1000);
clearInterval(myTimer); //used to clear the setInterval so it no longer calls the function
```

Date Objects

Date object is used to work with dates and times

```
let date = new Date()//current date if no parameters given, can give year, month, day, hour etc
let year = date.getFullYear();
let dayOfMonth = date.getDate();
let dayOfWeek = date.getDay();
let month = date.getMonth();
let hour = date.getHours(); //military time
let minutes = date.getMinutes();
let seconds = date.getSeconds();
date.setFullYear(2024);
console.log(date);
date = date.toLocaleString();// much more readable
```

Promises

object that encapsulates the result of an asynchronous operation, let asynchronous methods return values like synchronous methods, “I promise to return something in the future”

State is ‘pending’ then is either ‘fulfilled’ or ‘rejected’, RESULT is what can be returned

```
const promise = new Promise((resolve, reject) => {
        let fileLoaded = false;
        if (fileLoaded) {
                resolve(“file Loaded”);
        } else {
                reject(“file not loaded”);
        }
});

promise.then((value) => console.log(value)).catch((error) => console.log(error))
const wait = (time) => new Promise(resolve => {
        setTimeout(resolve, time);
});
wait(3000).then(() => console.log(“Thanks for waiting”));
```

Async and Await

ES6 Modules

DOM and DOM traversal

Child/Parent elements

- .firstElementChild
- .lastElementChild
- .parentElement
- .nextElementSibling
- .previousElementSibling
- .children
- Array.from(.children)

Events

Animations

Canvas API

Window

Cookies