# Introduction

* Software I use:
  + Google chrome
  + Visual Studio Code
* What to know before hand:
  + HTML basics
  + CSS basics
* Tutorial
  + <https://www.youtube.com/playlist?list=PLZPZq0r_RZOMRMjHB_IEBjOW_ufr00yG1>

## Notes

* Javascript isn’t like Java, it is a programming language but is really a scripting language
  + Scripting language - a programming language that employs a high-level construct to interpret and execute one command at a time

### script.js

| console.log("I like chicken!"); |
| --- |

* console.log(“”); - allows us to pass through a string inside the double quotes for the console of the website, not the web page
* //comment - a single line comment’
* /\* \*/ - multiline comment

### index.html

* !<tab> - for an empty html file if you type “!” and then tab, the document will automatically fill in the necessities for an html file.
  + The <script> and <link> were added to include the css and js files

| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <link rel="stylesheet" href="style.css">  </head>  <body>  <script src="script.js"></script>  </body>  </html> |
| --- |

# Variables and Data Types

* Variable - a container for storing data
  + Behaves as if it was the value that it contains
  + Two steps:
    - Declaration (var, let, const)
    - Assignment (= assignment operator)
* Data types
  + String - “”
  + Number - numeric value
  + Boolean - true or value

### index.html

| <body>  <p id="p1"></p>  <p id="p2"></p>  <p id="p3"></p>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| let age = 21; //numbers  let firstName = "Dakota"; //strings  let student = true; //or false  age = age + 1;  console.log("Hello ", firstName);  console.log("You are ", age," years old");  console.log("Enrolled: ", student);  document.getElementById("p1").innerHTML = "Hello " + firstName;  document.getElementById("p2").innerHTML = "You are " + age + " years old";  document.getElementById("p3").innerHTML = "Enrolled: "+ student; |
| --- |

* document.getElementById("\_") - returns an Element object representing the element whose id property matches the specified string. Since element IDs are required to be unique if specified, they're a useful way to get access to a specific element quickly.
  + innerHTML - gets or sets the HTML or XML markup contained within the element.
    - document.getElementById("\_").innerHTML = \_ - changes the inner html document based on id. In our case we change “p1”, “p2”, and “p3” from the Javascript file.

# Arithmetic expressions

* Arithmetic expression is a combination of:
  + Operands (values, variables, etc…)
  + Operators (+,-,\*,/,%,\*\*) that can be evaluated into a value

### script.js

| // students += 1; = 21  // students -= 1; = 19  // students \*= 2; = 40  // students /= 2; = 10  // let extraStudents = students % 3; = 2  // students = students \*\* 2; = 400 |
| --- |

* + adds
* - subtracts
* \* multiplies
* / divides
* % modulus (gets the remainder of the variable being divided)
* \*\* power (variable to the power to)
* Don’t forget “PEMDAS”

# User input

* There are 2 ways to get user input:
  + Window prompt
  + HTML textbox

## Window prompt

### script.js

| let username = window.prompt("What's your name? ");  console.log(username); |
| --- |

* window.prompt(“”) - prompts the webpage with a little window separate from what's actually in the page

## HTML textbox

### index.html

| <body>  <label id="myLabel">Enter your name:</label><br>  <input type = "text" id="myText"><br>  <button type = "button" id="myButton">submit</button>  <script src="script.js"></script>  </body> |
| --- |

* label - just puts a string on the page
* input type = “text” - allows the user to input a string of characters
* button type = "button" - allows the user to click a button with the word “submit” on it

### script.js

| let username  document.getElementById("myButton").onclick = function(){  username = document.getElementById("myText").value  console.log(username)  document.getElementById("myLabel").innerHTML = "Hello " + username + ":"  } |
| --- |

* Once the button with the id “myButton” is clicked the function activates and gets the value from the textbox with the id “myText”

# Type conversion

* Type conversion - change the datatype of a value to another
  + When doing inputs, the users input is automatically a string
  + typeof variable - returns the variable type

### script.js

| let x  let y  let z  x = Number("3.14")  console.log(x, typeof x)  y = String(3.14)  console.log(y, typeof y)  z = Boolean("pizza") //if it is only "" it will return false  console.log(z, typeof z) |
| --- |

* Number() - converts the variable in the parenthesis into a number (if the variable has characters that aren’t numeric it will return NaN)
* String() - converts the variable in the parenthesis into a string
* Boolean() - converts the variable in the parenthesis into true or false. If it is blank it will return false

# Const

* Const - a variable that cannot be changed
  + We assign this to any variable that doesn’t need to be changed

### script.js

| const PI = 3.14159  let radius  let circumference  radius = window.prompt("Enter the radius of a circle:")  radius = Number(radius)  circumference = 2 \* PI \* radius  console.log("The circumference is: ", circumference) |
| --- |

* PI is a constant number and will never change
  + We typically name the variable in all uppercase letters

# Math Methods

* Math - an intrinsic object that provide basic mathematics functionality and constants

### script.js

| let x = 3.14  x = Math.round(x) //rounds to the nearest 1 digit  console.log(x)  x = 3.99  x = Math.floor(x) //rounds down  console.log(x)  x = 3.14  x = Math.ceil(x) //rounds up  console.log(x)  x = 3.14  x = Math.pow(x, 2) //exponent of 2  console.log(x)  x = 3.14  x = Math.sqrt(x) //square root of x  console.log(x)  x = -3.14  x = Math.abs(x) //returns a positive number  console.log(x)  let y = 5  let z = 9  let max = Math.max(x, y, z) //returns the maximum number of the variables  let min = Math.min(x, y, z) //returns the minimum number of the variables  console.log(max)  console.log(min)  x = Math.PI //constant of PI  console.log(x)  random = Math.random() // returns a random number between 0 and 1  console.log(random \* 10) |
| --- |

# Hypotenuse Program

### script.js

| let a  let b  let c  // a = window.prompt("Enter side A")  // a = Number(a)  // b = window.prompt("Enter side B")  // b = Number(b)  // c = Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2))  // console.log("Side C: ", c)  document.getElementById("submit").onclick = function(){  a = document.getElementById("aTextBox").value  a = Number(a)  b = document.getElementById("bTextBox").value  b = Number(b)  c = Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2))    document.getElementById("cLabel").innerHTML = c  } |
| --- |

### index.html

| <body>  <label id="aLabel">Side A: </label>  <input type="text" id="aTextBox"><br>  <label id="bLabel">Side B: </label>  <input type="text" id="bTextBox"><br>  <button type="button" id="submit">Calculate</button><br>  <label>Side C: </label>  <label id="cLabel"></label>  <script src="script.js"></script>  </body> |
| --- |

# Counter Program

### script.js

| let count = 0  document.getElementById("DecreaseBtn").onclick = function(){  count -= 1  document.getElementById("countLabel").innerHTML = count  }  document.getElementById("ResetBtn").onclick = function(){  count = 0  document.getElementById("countLabel").innerHTML = count  }  document.getElementById("IncreaseBtn").onclick = function(){  count += 1  document.getElementById("countLabel").innerHTML = count  } |
| --- |

### index.html

| <body>  <label id="countLabel">0</label><br>  <button id="DecreaseBtn">Decrease</button>  <button id="ResetBtn">Reset</button>  <button id="IncreaseBtn">Increase</button>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| #countLabel {  display: block;  text-align: center;  font-size: 50px;  } |
| --- |

# Random Number Generator

## Math.random()

| let x  x = Math.random() //generates a random number between 0 and 1  console.log(x)  x = Math.random() \* 6 //generates a random number between 0 and 6  console.log(x)  x = Math.floor(Math.random() \* 6) //generates a random number between 0 and 5  console.log(x)  x = Math.floor(Math.random() \* 20) + 1 //generates a random number between 1 and 20  console.log(x) |
| --- |

## Example

### script.js

| let a  let b  let c  document.getElementById("rollBtn").onclick = function(){  a = Math.floor(Math.random() \* 20) + 1  b = Math.floor(Math.random() \* 10) + 1  c = Math.floor(Math.random() \* 6) + 1  document.getElementById("aLabel").innerHTML = a  document.getElementById("bLabel").innerHTML = b  document.getElementById("cLabel").innerHTML = c  } |
| --- |

### index.html

| <body>  <label>20d: </label>  <label id="aLabel">0</label><br>  <label>10d: </label>  <label id="bLabel">0</label><br>  <label>6d: </label>  <label id="cLabel">0</label><br>  <button type="button" id="rollBtn">Roll</button>  <script src="script.js"></script>  </body> |
| --- |

# Useful String Methods

* .length - returns the length of the string
* .charAt(#) - returns the letter at that index

### script.js

| let username = "Brogoming"  let username2 = " Dakadus "  let phoneNum = "124-243-3435"  console.log(username.length) //returns the length of the string  console.log(username.charAt(0)) //returns the letter at that index  console.log(username.indexOf("o")) //returns the first index of the char  console.log(username.lastIndexOf("o")) //returns the last index of the char  console.log(username2.trim()) //removes all unnecessary spaces before and after  console.log(username.toUpperCase()) //returns the variable with all uppercase letters  console.log(username.toLowerCase()) //returns the variable with all lowercase letters  console.log(phoneNum.replaceAll("-", "/")) //replaces all instances of the first char with the second char |
| --- |

# String slicing

* slice() - extracts a section of a string and returns it as a new string, without modifying the original string
  + inside the parentheses the first value is the starting index, the second value is the last index (doesn't include the last char)

### script.js

| let fullName = "Dakota Gullicksen"  //separated by " "  console.log(fullName.slice(fullName.indexOf(" ") + 1)) //15 to the last index  console.log(fullName.slice(0, fullName.indexOf(" "))) //0 to 5 |
| --- |

# If Statements

* If statement - a basic form of decision making if a condition is true, then do something. If not, then don’t do it
  + Order of if statements does matter

### script.js

| let age = 21  if(age >= 18){  console.log("You are older than 17")  }  age = 12  if(age >= 18){  console.log("You are older than 17")  } else {  console.log("You are a child")  }  age = -1  if(age >= 18){  console.log("You are older than 17")  } else if(age < 0){  console.log("you haven't been born yet")  } else {  console.log("You are a child")  } |
| --- |

# .checked property

### script.js

| document.getElementById("myButton").onclick = function(){  const myCheckBox = document.getElementById("myCheckBox")  const paperBtn = document.getElementById("paper")  const presentBtn = document.getElementById("present")  const practiceBtn = document.getElementById("practice")  if(myCheckBox.checked == true){  console.log("Go relax!")  } else {  console.log("Do you homework!!!")  }  if(paperBtn.checked){  console.log("You should work on your paper")  } else if(presentBtn.checked){  console.log("You should work on your presentation")  } else if(practiceBtn.checked){  console.log("You should work on your practice problems")  } else {  console.log("Do something!")  }  } |
| --- |

* .checked checks if the element was marked, if so it returns true
* You can initialize a variable using getElementById("\_")

### index.html

| <body>  <label for="myCheckBox">Do Homework</label>  <input type="checkbox" id="myCheckBox"><br>  <label for="paper">Paper</label>  <input type="radio" name="assignment" id="paper">  <label for="paper">Presentation</label>  <input type="radio" name="assignment" id="present">  <label for="paper">Practice Problems</label>  <input type="radio" name="assignment" id="practice"><br>  <button id="myButton">Submit</button>  <script src="script.js"></script>  </body> |
| --- |

# Switches

* Switch statement examines a value for a match against many case clauses. More efficient than many “else if” statements
  + Case clauses use break statements so the next case clause isn’t run after words.

### script.js

| let grade = "Pizza"  switch(grade){  case "A":  console.log("You did great!")  break  case "B":  console.log("You did good!")  break  case "C":  console.log("You did okay!")  break  case "D":  console.log("You passed ... barely!")  break  case "F":  console.log("You failed!")  break  default:  console.log(grade, " is not a letter grade!")  } |
| --- |

* Checking conditions

| grade = 95  switch(true){  case grade >= 90:  console.log("You did great!")  break  case grade >= 80:  console.log("You did good!")  break  case grade >= 70:  console.log("You did okay!")  break  case grade >= 60:  console.log("You passed ... barely!")  break  case grade < 60:  console.log("You failed!")  break  default:  console.log(grade, " is not a letter grade!")  } |
| --- |

# Operators

* = assignment operator assigns a variable a value
* == comparison operator compares the values of variables
* === strict equality operator compares to see if not only the values are the same but also the datatype are the same
* && - and - both conditions must be true
* || - or - either condition can be true
* ! - not - typically used to reverse a condition’s boolean value
* < - less than
* > - greater than
* <= - less than or equal to
* >= - greater than or equal to

# While loops

* While loop - like an if statement but the code is repeated while the condition is true
  + Be careful to not repeat it an infinite number of times

### script.js

| let username = ""  while(username == ""){  username = window.prompt("What is your username?")  }  console.log("Hello ", username) |
| --- |

# Do while loops

* Do while loops - do something, then check the condition, repeat if constitution is true

### script.js

| let username  do{  username = window.prompt("What is your username?")  }while(username == "")  console.log("Hello ", username) |
| --- |

# For Loops

* For loop - repeats some code a certain amount of times
  + The variable i is called a local variable, it doesn't exist outside of the loop

### script.js

| for(let i=10; i > 0; i-=1){  console.log(i)  } |
| --- |

# Break and Continue Statements

* Break - breaks out of a loop entirely
* Continue - skip an iteration of a loop

# Nested loops

* Nested Loop - a loop inside of another loop
  + Make sure the local variables are different like i for the outside loop and j for the inside loop

# Functions

* Function - define code once and use it many times. To perform the code, call the function name
  + Functions have access to global variables
    - Global variables - variables that any method or function can use
  + happyBirthday(username,age) - calls the function and uses arguments
  + The parameters don't have to be the same as the variables being passed in, make sure to get the order right

### scrip.js

| startProgram()  function startProgram(){  let username = "Dakota"  let age = 20  happyBirthday(username,age)  }  function happyBirthday(user, age){  console.log("Happy Birthday to you!")  console.log("Happy Birthday to you!")  console.log("Happy Birthday dear",user,"!")  console.log("Happy Birthday to you!")  console.log("You are",age,"years old!")  } |
| --- |

# Return Statements

* Return - returns a value back to the place where you invoked the function

### script.js

| let area  let width  let height  width = window.prompt("Enter width of rectangle")  height = window.prompt("Enter height of rectangle")  area = getArea(width, height)  console.log("The area is", area)  function getArea(width, height){  let result = width \* height  return result  } |
| --- |

# Ternary operator

* Ternary operator - a shortcut for an “if/else” statement. It takes 3 operands:
  + A condition with ?
  + Expression if true :
  + Expression if false
    - condition ? expression if True : expression if False

### script.js

| let adult = checkAge(21)  console.log(adult)  function checkAge(age){  return age > 18 ? true : false  } |
| --- |

# Var vs Let

* Variable scope - where a variable is accessible
* Let - variables are limited to block scope{}
* Var - variables are limited to a function(){}
* Global variables - variables that any method or function can use. It is declared outside of a function
  + Var will change the browser window’s properties

# Template literals

* Template literals - delimited with (`) instead of double or single quotes.
  + Allows embedded variables and expressions

### script.js

| let username = "Dakadus"  let items = 3  let total = 75  console.log(`Hello ${username}`)  console.log(`You have ${items} items in your cart.`)  console.log(`Your total is $${total}.`)  let text =  `Hello ${username}!<br>  You have ${items} items in your cart.<br>  Your total is $${total}.<br>`  document.getElementById("MyLabel").innerHTML = text |
| --- |

### index.html

| <body>  <label id="MyLabel" ></label>  <script src="script.js"></script>  </body> |
| --- |

# Format Currency

* toLocaleString() - returns a string with a language sensitive representation of this number
* number.toLocaleString(locale, {options})
  + Locale - specifies the language (undefined - default set in browser)
  + Options - object with formatting options

### script.js

| let myNum = 0.1  // myNum = myNum.toLocaleString("en-US") //US english  // myNum = myNum.toLocaleString("hi-IN") //Hindi  // myNum = myNum.toLocaleString("de-DE") //standard German  // myNum = myNum.toLocaleString("en-US", {style: "currency", currency: "USD"}) //USD = US Dollars  // myNum = myNum.toLocaleString("hi-IN", {style: "currency", currency: "INR"}) //INR = Indian Rupees  // myNum = myNum.toLocaleString("de-DE", {style: "currency", currency: "EUR"}) //EUR = Euros  // myNum = myNum.toLocaleString(undefined, {style: "percent"}) //percent by decimal  // myNum = myNum.toLocaleString(undefined, {style: "unit", unit: "celsius"})  console.log(myNum) |
| --- |

# Number Guessing Game

### index.html

| <body>  <h1>Number Guessing Game</h1>  <p>Pick a number between 1 - 10</p>  <label>Enter a Guess:</label>  <input id="guessField">  <input type="Submit" id="submitBtn">  <script src="script.js"></script>  </body> |
| --- |

### script.js

| const answer = Math.floor(Math.random() \* 10 + 1)  let guesses = 0  document.getElementById("submitBtn").onclick = function(){  let guess = document.getElementById("guessField").value  guesses += 1  if (guess == answer){  alert(`${answer} is the number! It took you ${guesses} guesses!`)  } else if(guess < answer){  alert("Too Small!")  } else {  alert("Too Large")  }  } |
| --- |

# Temperature Conversion Program

### index.html

| <body>  <label>Enter a temperature:</label><br>  <input type="text" id="textBox"><br>  <label>Convert to: </label><br>  <label>Celsius</label>  <input type="radio" id="cButton" name="unit"><br>  <label>Fahrenheit</label>  <input type="radio" id="fButton" name="unit"><br>  <button type="button" id="submitBtn">Submit</button><br>  <label id="tempLabel"></label>    <script src="script.js"></script>  </body> |
| --- |

### script.js

| document.getElementById("submitBtn"),onclick = function(){  let temp = document.getElementById("textBox").value  temp = Number(temp)  if(document.getElementById("cButton").checked){  temp = toCelsius(temp)  document.getElementById("tempLabel").innerHTML = temp + "°C"  } else if(document.getElementById("fButton").checked){  temp = tofahrenheit(temp)  document.getElementById("tempLabel").innerHTML = temp + "°F"  } else {  document.getElementById("tempLabel").innerHTML = "Select a unit"  }  }  function toCelsius(temp){  return (temp - 32) \* (5/9)  }  function tofahrenheit(temp){  return temp \* 9/5 + 32  } |
| --- |

# Arrays

* Array - a variable that can store multiple values

### script.js

| let fruits = ["apple", "orange", "banana"]  console.log(fruits) //displays a bunch of stuff not really needed  console.log(fruits[1]) //displays "orange"  fruits[0] = "coconut" //changes the value of the array at the index 0  console.log(fruits)  fruits.push("leamon") //adds and element to the array  console.log(fruits)  fruits.pop() //removes last element  console.log(fruits)  fruits.unshift("mango") //adds the element to the front  console.log(fruits)  fruits.shift() //removes the beginning element  console.log(fruits)  console.log(fruits.length) //returns the length of the array: 3  console.log(fruits.indexOf("coconut")) //returns the index of "apple", -1 if it doesn't exist |
| --- |

# Array strategies

## Loop through an array

### script.js

| let prices = [5,10,15,20]  for(let i = 0; i < prices.length; i+=1){  console.log(prices[i])  }  for(let i = prices.length - 1; i >= 0; i-=1){  console.log(prices[i])  }  for(let price of prices){  console.log(price)  } |
| --- |

## Sort an array of strings

### script.js

| let fruits = ["banana", "apple", "orange", "mango"]  fruits = fruits.sort() //sorted alphabetically  fruits = fruits.sort().reverse() //sorted in reverse alphabetical order  for(let fruit of fruits){  console.log(fruit)  } |
| --- |

## 2D array

* 2D array - an array of arrays

### script.js

| let fruits = ["apples", "oranges", "bananas"]  let vegs = ["carrots", "onions", "potatoes"]  let meats = ["eggs", "chicken", "fish"]  let groceryList = [fruits, vegs, meats] //place the arrays in the array  console.log(groceryList)  groceryList[0][1] = "mangoes" //first index is row, second index is column  groceryList[2][0] = "streak"  for(let list of groceryList){ //outside arrays  console.log(list)  for(let food of list){ //inside array  console.log(food)  }  } |
| --- |

## Spread operator

* Spread operator (...) - allows an iterable such as an array or string to be expanded in places where zero or more arguments are expected (unpacks the elements)

### script.js

| let numbers = [11,3,2,5,7,6,9]  console.log(numbers) //displays the details of the array  console.log(...numbers) //expands the elements of the array  let username = "Dakadus Prime"  console.log(...username) //words with strings  let max = Math.max(...numbers)  console.log(max)  let class1 = ["Kratos", "Mario", "Sack Boy"]  let class2 = ["Master Chief", "Cirby", "Sly Cooper"]  class1.push(...class2)  console.log(...class1) |
| --- |

## Rest parameters

* Rest parameters - represents an indefinite number of parameters (packs arguments into an array)

### script.js

| let a = 1  let b = 2  let c = 3  let d = 4  let e = 5  console.log(sum(a, b, c, d, e))  function sum(...numbers){  let total = 0  for(let number of numbers){  total += number  }  return total  } |
| --- |

# Callbacks

* Callback - a function passed as an argument to another function
  + Ensures that a function is not going to run before a task is completed.
  + Helps us develop asynchronous code.
    - When one function has to wait for another function
  + , that helps us avoid errors and potential problems

### index.html

| <body>  <label id="MyLabel"></label>    <script src="script.js"></script>  </body> |
| --- |

### script.js

| sum(2,3,displayConsole, displayDOM) //just put the name of the function  function sum(x,y,callback,callback2){ //the parameter names of the functions can be named anything  let result = x + y  callback(result)  callback2(result)  }  function displayConsole(output){  console.log(output)  }  function displayDOM(output){  document.getElementById("MyLabel").innerHTML = output  } |
| --- |

# Array methods

## array.forEach()

* array.forEach() - executes a provided callback function once for each array element

### script.js

| let students = ["dakota", "alex", "jayden"]  students.forEach(capitalize) //does this for each element in the array  students.forEach(print) //does this for each element in the array  function capitalize(element, index, array){  array[index] = element[0].toUpperCase() + element.substring(1) //selects the first character of every string  }  function print(element){  console.log(element)  } |
| --- |

## array.map()

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

### script.js

| let numbers = [1,2,3,4,5]  let squares = numbers.map(square)  let cubes = numbers.map(cube)  squares.forEach(print)  cubes.forEach(print)  function square(element){  return Math.pow(element,2)  }  function cube(element){  return Math.pow(element,3)  }  function print(element){  console.log(element)  } |
| --- |

## array.filter()

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

### script.js

| let ages = [18,16,21,17,19,90]  let adults = ages.filter(checkAge)  adults.forEach(print)  function checkAge(element){  return element >= 18  }  function print(element){  console.log(element)  } |
| --- |

## array.reduce()

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

### script.js

| let prices = [5,10,15,20,25,30]  let total = prices.reduce(checkOut)  console.log(total)  function checkOut(total, element){  return total + element  } |
| --- |

## Sorting an array of numbers

### script.js

| let grades = [100,50,90,60,80,70]  let grades2 = [100,50,90,60,80,70]  grades = grades.sort(descendingSort)  grades.forEach(print)  grades2 = grades2.sort(ascendingSort)  grades2.forEach(print)  function descendingSort(x,y){  return y - x  }  function ascendingSort(x,y){  return x - y  }  function print(element){  console.log(element)  } |
| --- |

# Function expression

* Function expression - function without a name (anonymous function) helps us avoid polluting the global scope with names
  + Write it then forget about it

## Example 1

### script.js

| const greeting = function(){  console.log("Hello")  }  greeting() |
| --- |

## Example 2

### index.html

| <body>  <label id="MyLabel">0</label><br>  <button id="decBtn">Decrease</button>  <button id="incBtn">Increase</button>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| let count = 0  document.getElementById("incBtn").onclick = function(){  count += 1  document.getElementById("MyLabel").innerHTML = count  }  document.getElementById("decBtn").onclick = function(){  count -= 1  document.getElementById("MyLabel").innerHTML = count  } |
| --- |

# Arrow function expressions

* Arrow function - a compact alternative to a traditional function expression (=>)
  + You need curly brackets if have pass in more than one statement

## Example 1

### script.js

| const greeting = (username) => {  console.log(`Hello ${username}`)  }  greeting("Dakota") |
| --- |

## Example 2

### script.js

| const percent = (x,y) => x/y \* 100  console.log(`${percent(45,50)}%`) |
| --- |

## Example 3

### script.js

| let grades = [100,50,90,60,80,70]  grades.sort((x,y) => y-x)  grades.forEach((element) => console.log(element)) |
| --- |

# Shuffle an array

### script.js

| let cards = ["A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]  shuffle(cards)  cards.forEach(card => console.log(card))  function shuffle(array){  let currentIndex = array.length  while(currentIndex != 0){  let randomIndex = Math.floor(Math.random() \* array.length)  currentIndex -= 1  let temp = array[currentIndex]  array[currentIndex] = array[randomIndex]  array[randomIndex] = temp  }  return array  } |
| --- |

# Nested functions

* Nested functions - functions inside other functions.
  + Outer functions have access to inner functions. Inner functions are “hidden” from outside the outer function
  + Used in closures

### script.js

| let username = "Dakadus"  let userInbox = 0  login()  function login(){  displayUser()  displayUserInbox()  function displayUser(){  console.log(`Welcome ${username}`)  }  function displayUserInbox(){  console.log(`You have ${userInbox} new messages`)  }  } |
| --- |

# Closure

* Closure - a function with preserved and private data.
  + Gives you access to an outer function’s scope, from and inner function

### index.html

| <body>  <button id="loginBtn">Login</button>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| document.getElementById("loginBtn").onclick = login()  userInbox = 6789  function login(){  let username = "Dakadus"  let userInbox = 1  function alertUser(){  alert(`${username} you have ${userInbox} new messages!`)  userInbox = 0  }  return alertUser  } |
| --- |

# Maps

* Map - object that holds key-value pairs of any data type
  + .get(key) - gets the value from the key in the parenthesis
  + .set(key, value) - adds a new key value pair
  + .delete(key) - deletes the key
  + .has(key) - returns a boolean value
  + .size - returns the number of pairs in the map

### script.js

| const store = new Map([  ["t-shirt", 20],  ["jeans", 30],  ["socks", 10],  ["underwear", 50]  ]);  let shoppingCart = 0  shoppingCart += store.get("t-shirt")  shoppingCart += store.get("underwear")  store.set("hat", 40)  store.delete("hat")  console.log(store.size)  console.log(store.has("hat"))  console.log(shoppingCart)  store.forEach((value, key) => console.log(`${key} $${value}`)) |
| --- |

# Objects

* Objects - a group of properties and methods
  + Properties - what an object has
  + Methods - what an object can do
    - Use . to access properties/methods
    - Each property and method is separated by commas

### script.js

| const car1 = {  model:"Mustang",  color:"Red",  year:2023,  drive:function(){  console.log("You drive the car")  },  brake:function(){  console.log("You stepped on the brakes")  }  }  const car2 = {  model:"Corvette",  color:"Blue",  year:2024,  drive:function(){  console.log("You drive the car")  },  brake:function(){  console.log("You stepped on the brakes")  }  }  console.log(car1.model)  console.log(car1.color)  console.log(car1.year)  car1.drive()  car1.brake()  console.log(car2.model)  console.log(car2.color)  console.log(car2.year)  car2.drive()  car2.brake() |
| --- |

# This keyword

* this. - reference to a particular object but the object depends on the immediate context
  + If you use the this keyword outside of an object it will refer to the window object

## Example 1

### script.js

| const car1 = {  model:"Mustang",  color:"Red",  year:2023,  drive:function(){  console.log(`You drive the ${this.model}`)  }  }  const car2 = {  model:"Corvette",  color:"Blue",  year:2024,  drive:function(){  console.log(`You drive the ${car2.model}`)  }  }  car1.drive()  car2.drive() |
| --- |

## Example 2

### script.js

| this.name = "MyCoolWindow"  console.log(this)  console.log(this.name) |
| --- |

# Classes

* Class - a blueprint for creating objects. They define what properties and methods they have
  + Typically contain a constructor for unique properties

### script.js

| class Player{  score = 0  pause(){  console.log("You paused the game")  }  exit(){  console.log("You exited the game")  }  }  const player1 = new Player()  player1.score += 1  console.log(player1.score)  player1.pause()  player1.exit()  const player2 = new Player()  player2.score += 4  console.log(player2.score)  player2.pause()  player2.exit() |
| --- |

# Constructor

* Constructor - a special method of a class, accepts arguments and assigns properties

### script.js

| class Student{  constructor(name, age, gpa){  this.name = name  this.age = age  this.gpa = gpa  }  study(){  console.log(`${this.name} is studying`)  }  }  const student1 = new Student("Dakota", 20, 4.0)  const student2 = new Student("Dakadus", 24, 3.9)  console.log(student1.name)  console.log(student1.age)  console.log(student1.gpa)  student1.study()  console.log(student2.name)  console.log(student2.age)  console.log(student2.gpa)  student2.study() |
| --- |

# Static

* Static - belongs to the class, not the objects
  + Properties - useful for caches, fixed configuration
  + Methods - useful for utility functions

### script.js

| class Car{  static numberOfCars = 0  constructor(model){  this.model = model  Car.numberOfCars += 1  }  static startRace(){  console.log("3...2...1...GO!")  }  }  const car1 = new Car("Mustang")  const car2 = new Car("Corvette")  const car3 = new Car("BMW")  const car4 = new Car("Ferrari")  const car5 = new Car("Lambo")  console.log(Car.numberOfCars)  Car.startRace() |
| --- |

# Inheritance

* Inheritance - a child class can inherit all the methods and properties from another class

### script.js

| class Animal{  alive = true    eat(){  console.log(`This ${this.name} is eating`)  }  sleep(){  console.log(`This ${this.name} is sleeping`)  }  }  class Rabbit extends Animal{  name = "Rabbit"  run(){  console.log(`This ${this.name} is running`)  }  }  class Fish extends Animal{  name = "Fish"  swim(){  console.log(`This ${this.name} is swimming`)  }  }  class Hawk extends Animal{  name = "Hawk"  fly(){  console.log(`This ${this.name} is flying`)  }  }  const rabbit = new Rabbit()  const hawk = new Hawk()  const fish = new Fish()  console.log(rabbit.alive)  hawk.eat()  fish.sleep()  rabbit.run() |
| --- |

# Super keyword

* Super - refers to the parent class.
  + Commonly used to invoke constructor of a parent class if the child class has a constructor

### script.js

| class Animal{  constructor(name, age){  this.name = name  this.age = age  }  }  class Rabbit extends Animal{  constructor(name, age, runSpeed){  super(name, age)  this.runSpeed = runSpeed  }  }  class Fish extends Animal{  constructor(name, age, swimSpeed){  super(name, age)  this.swimSpeed = swimSpeed  }  }  class Hawk extends Animal{  constructor(name, age, flySpeed){  super(name, age)  this.flySpeed = flySpeed  }  }  const rabbit = new Rabbit("Rabbit", 1, 40)  const hawk = new Hawk("Fish", 2, 80)  const fish = new Fish("Hawk", 3, 200)  console.log(rabbit.name)  console.log(hawk.age)  console.log(fish.swimSpeed) |
| --- |

# Getters and Setters

* Get - binds an object property to a function when that property is accessed
  + If you set an \_ before the variable name it lets others know that it is protected and shouldn’t be messed with
* Set - binds an object property to a function when that property is assigned a value
  + Getters and setters can have the same name

### script.js

| class Car{  constructor(power){  this.\_gas = 25  this.\_power = power  }  get power(){  return `${this.\_power}hp`  }  get gas(){  return `${this.\_gas}L (${this.\_gas/50 \* 100}%)`  }  set gas(value){  if(value > 50){  value = 50  } else if(value < 0){  value = 0  }  this.\_gas = value  }  }  let car = new Car(400)  console.log(car.power)  console.log(car.gas)  car.gas = 100  console.log(car.gas)  car.gas = -100  console.log(car.gas) |
| --- |

# Object as arguments

### script.js

| class Car{  constructor(model, year, color){  this.model = model;  this.year = year;  this.color = color;  }  }  const car1 = new Car("Mustang", 2023, "red",);  const car2 = new Car("Corvette", 2024, "blue");  const car3 = new Car("Lambo", 2022, "yellow",);  displayInfo(car1)  displayInfo(car2)  displayInfo(car3)  changeColor(car3, "gold")  displayInfo(car3)  function displayInfo(car){  console.log(car.model)  console.log(car.year)  console.log(car.color)  }  function changeColor(car, color){  car.color = color  } |
| --- |

# Array of objects

### script.js

| class Car{  constructor(model, year, color){  this.model = model;  this.year = year;  this.color = color;  }  drive(){  console.log(`You drive the ${this.model}`);  }  }  const car1 = new Car("Mustang", 2023, "red",);  const car2 = new Car("Corvette", 2024, "blue");  const car3 = new Car("Lambo", 2022, "yellow",);  const cars = [car1, car2, car3]  presentCars(cars)  function presentCars(array){  for(let i = 0; i < array.length; i += 1){  console.log(array[i].model)  console.log(array[i].year)  console.log(array[i].color)  array[i].drive()  }  } |
| --- |

# Anonymous Objects

* Anonymous objects - objects without a name
  + Not directly referenced
  + Less syntax
  + No need for unique names

### script.js

| class Card{  constructor(value, suit){  this.value = value  this.suit = suit  }  }  let cards = [new Card("A", "Hearts"),  new Card("A", "Spades"),  new Card("A", "Diamonds"),  new Card("A", "Clubs"),  new Card("2", "Hearts"),  new Card("2", "Spades"),  new Card("2", "Diamonds"),  new Card("2", "Clubs")]  cards.forEach(card => console.log(`${card.value} ${card.suit}`)) |
| --- |

# Error Handling

* Error - object with a description of something went wrong
  + Can’t open a file
  + Lose connection
  + User types incorrect input
  + TypeError
* Catch - if something goes wrong it will do what is in the block
* Throw - executes a user-defined error
* Finally - will always executes even if there's an error

### script.js

| try{  let x = window.prompt("Enter a #")  x = Number(x)    if(isNaN(x)) throw "That wasn't a number"  if(x == "") throw "That was empty"  console.log(`${x} is a number`)  }  catch(error){  console.log(error)  }  finally{  console.log("This always executes")  } |
| --- |

# setTimeout()

* setTimeout() - invokes a function after a number of milliseconds
  + Asynchronous function (doesn’t pause the execution of the program)
* clearTimeout() - stops the timer of the variable passed in

### index.html

| <body>  <button id="MyButton">Buy</button>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| let item = "cryptocurrency"  let price = 420.69  let timer1 = setTimeout(firstMessage, 3000, item, price)  let timer2 = setTimeout(secondMessage, 6000)  let timer3 = setTimeout(thirdMessage, 9000)  function firstMessage(item, price){  alert(`Buy this ${item} fro $${price}!`)  }  function secondMessage(){  alert(`This is not a scam!!`)  }  function thirdMessage(){  alert(`DO IT!!!`)  }  document.getElementById("MyButton").onclick = function(){  clearTimeout(timer1)  clearTimeout(timer2)  clearTimeout(timer3)  alert(`Thanks for buying our fictional course`)  } |
| --- |

# setInterval

* setInterval() - invokes a function repeatedly after a number of milliseconds
  + Asynchronous function (doesn’t pause the execution of the program)

### script.js

| let count = 0  let max = window.prompt("Count up to what number?")  max = Number(max)  const myTimer = setInterval(countUp, 1000, max)  function countUp(max){  count+=1  console.log(count)  if(count >= max){  clearInterval(myTimer)  }  } |
| --- |

# Date Objects

* Date objects is used to work with dates and times

### index.html

| <body>  <label id = "MyLabel1"></label><br>  <label id = "MyLabel2"></label><br>  <label id = "MyLabel3"></label><br>  <label id = "MyLabel4"></label><br>  <label id = "MyLabel5"></label><br>  <label id = "MyLabel6"></label><br>  <label id = "MyLabel7"></label><br>  <label id = "MyLabel8"></label><br>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| let date = new Date()  let date2 = new Date(0) //if you pass in zero it will be set as 12/31/1969, 6:00:00 PM  date2 = date2.toLocaleString()  let date3 = new Date(10000000000) //if you pass in 10000000000 it will be 10000000000 milliseconds after date 0  date3 = date3.toLocaleString()  let date4 = new Date(2023, 0, 3, 4, 26, 32) //year, month, day, hour, minutes, seconds  date4 = date4.toLocaleString()  let date5 = new Date("March 2, 2024 00:00:00") //month, day, year, hour, minutes, seconds  date5 = date5.toLocaleString()  let year = date.getFullYear()  let dayOfMonth = date.getDate() //day of month  let dayOfWeek = date.getDay() //day of week: 0 = sunday, 1 = monday  let month = date.getMonth() //0 = january, 1 = february  let hour = date.getHours() //in military time  let minutes = date.getMinutes()  let seconds = date.getSeconds()  let milliseconds = date.getMilliseconds()  //you can change the date manually  let date6 = new Date()  date6.setFullYear(2024)  date6.setMonth(11)  date6.setDate(31)  date6.setHours(5)  date6.setMinutes(25)  date6.setSeconds(54)  date6.setMilliseconds(234)  date6 = date6.toLocaleString()  document.getElementById("MyLabel1").innerHTML = date  document.getElementById("MyLabel2").innerHTML = date2  document.getElementById("MyLabel3").innerHTML = date3  document.getElementById("MyLabel4").innerHTML = date4  document.getElementById("MyLabel5").innerHTML = date5  document.getElementById("MyLabel6").innerHTML = date6  document.getElementById("MyLabel7").innerHTML = formatDate(date)  document.getElementById("MyLabel8").innerHTML = formatTime(date)  function formatDate(date){  let year = date.getFullYear()  let dayOfMonth = date.getDate() //day of month  let month = date.getMonth() + 1//0 = january, 1 = february  return `${month}/${dayOfMonth}/${year}`  }  function formatTime(date){  let hours = date.getHours()  let minutes = date.getMinutes()  let seconds = date.getSeconds()  let amOrPM = hours >= 12 ? "pm" : "am"  hours = (hours % 12) || 12 //converts from military to standard  return `${hours}.${minutes}.${seconds} ${amOrPM}`  } |
| --- |

# Clock Program

### index.html

| <body>  <label id = "MyLabel"></label>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| const myLabel = document.getElementById("MyLabel")  update()  setInterval(update, 1000)  function update(){  let date = new Date()  myLabel.innerHTML = formatTime(date)  function formatTime(date){  let hours = date.getHours()  let minutes = date.getMinutes()  let seconds = date.getSeconds()  let amOrPm = hours >= 12 ? "PM" : "AM"  hours = (hours % 12) || 12  hours = formatZeroes(hours)  minutes = formatZeroes(minutes)  seconds = formatZeroes(seconds)  return `${hours}:${minutes}:${seconds} ${amOrPm}`  }  function formatZeroes(time){  time = time.toString()  return time.length < 2 ? "0" + time : time  }  } |
| --- |

# Synchronous vs Asynchronous

* synchronous code = In an ordered sequence.
  + Step-by-step linear instructions (start now, finish now)
* asynchronous code = Out of sequence.
  + (start now, finish sometime later)
  + Ex. Access a database, Fetch a file, Tasks that take time

# console.time()

* console.time() - starts a timer you can use to track how long an operation takes
  + Give each timer a unique name

### script.js

| console.time("Response Time") //start  alert("Click the Button")  console.timeEnd("Response Time") //end  console.time("Hello Time") //start  setTimeout(() => console.log("Hello!"), 3000)  console.timeEnd("Hello Time") //end |
| --- |

# Promises

* Promises - object that encapsulates the result of an asynchronous operation. Let the asynchronous methods return values like synchronous methods. “I promise to return in the future”
  + The state is “pending” then: “fulfilled” or “rejected”
  + Result is what can be returned

## Example 1

### script.js

| 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)) //if the promise is resolved we can preform a then function  .catch(error => console.log(error)) //catches any exceptions |
| --- |

## Example 2

### script.js

| const wait = new Promise(resolve => {  setTimeout(resolve, 5000)  })  wait.then(() => console.log("Thanks for waiting!")) |
| --- |

## Example 3

### script.js

| const wait = time => new Promise(resolve => {  setTimeout(resolve, time)  })  wait(3000).then(() => console.log("Thanks for waiting!")) |
| --- |

# Async

* Async - makes a function return a Promise

### script.js

| async function loadFile(){  let fileLoaded = true;    if(fileLoaded){  return "File loaded";  }  else{  throw "File NOT loaded";  }  }    loadFile().then(value => console.log(value)).catch(error => console.log(error)) |
| --- |

# Await

* Await - makes a function wait for a promise

### index.html

| <body>  <h1 id="MyH1"></h1>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| async function loadFile(){  const promise = new Promise((resolve, reject) => {  let fileLoaded = false;    if(fileLoaded){  resolve("File loaded")  }  else{  reject("File NOT loaded")  }  })    try{  document.getElementById("MyH1").innerHTML = await promise  }  catch(error){  document.getElementById("MyH1").innerHTML = error  }  }  loadFile() |
| --- |

# ES6 Modules

* The idea behind a module is that it’s a file of reusable code. We can import sections of pre-written code to use whenever
  + Great for any general utility values and functions
  + Helps to make your code more reusable and maintainable

### index.html

| <body>  <script type = "module" src="script.js"></script>  </body> |
| --- |

### math\_util.js

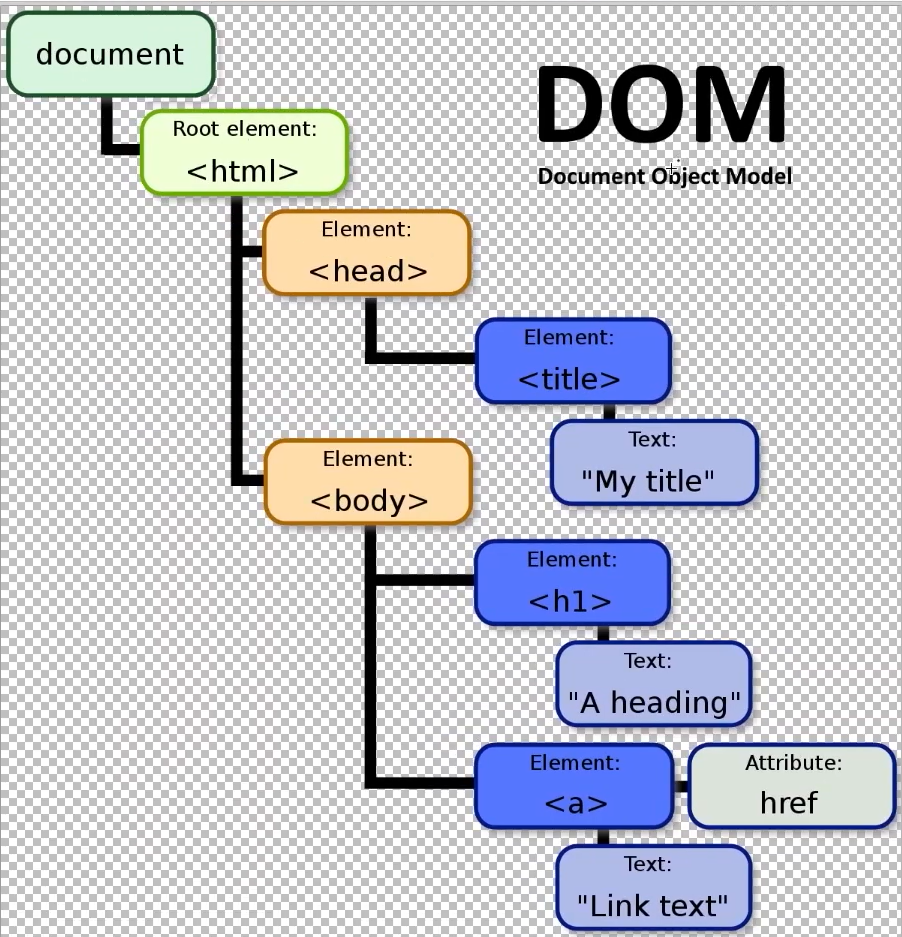
| export const PI = 3.14159  export function getCircumference(radius){  return 2 \* PI \* radius  }  export function getArea(radius){  return PI \* radius \* radius  } |
| --- |

### script.js

| // import {PI, getCircumference, getArea} from "./math\_util.js"  //if there is a lot to import use \*  import \* as MathUtil from "./math\_util.js"  console.log(MathUtil.PI)  let circumference = MathUtil.getCircumference(10)  console.log(circumference)  let area = MathUtil.getArea(10)  console.log(area) |
| --- |

# DOM intro

* DOM - Document Object Model (API)
  + An interface for changing the content of a page



* Document is the entry point of the DOM

### index.html

| <body>  <div id = "MyDiv"></div>  <script type = "module" src="script.js"></script>  </body> |
| --- |

### script.js

| console.log(document) //displays everything of the DOM  console.dir(document) //Lists all of the properties of the DOM  console.log(document.title) //presents the document of the DOM  console.log(document.URL) //presents the URL of the site  document.title = "New Title" //changes the title of the document  //document.location = "http://www.google.com" //changes the location we go when we run this  document.body.style.backgroundColor = "red" //changes the body color  document.getElementById("MyDiv").innerHTML = "Hello" |
| --- |

# Element Selectors

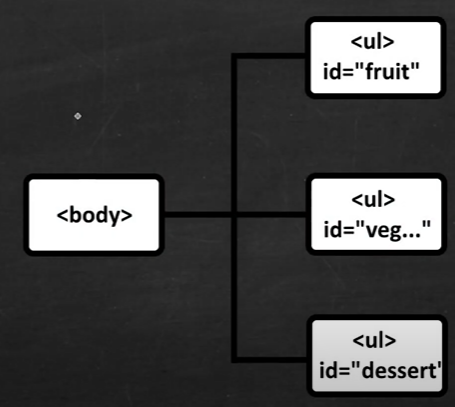
### index.html

| <body>  <h1 id="MyTitle">This is the Menu</h1>  <input type="radio" name="fruits" value="apple" checked="checked"> <!--this will already be checked-->  <label for="apple">Apple</label><br>  <input type="radio" name="fruits" value="orange">  <label for="orange">Orange</label><br>  <input type="radio" name="fruits" value="banana">  <label for="banana">Banana</label><br>  <ul>  <li>Carrots</li>  <li>Potatoes</li>  <li>Onions</li>  </ul>  <div class="desserts">Ice Cream</div>  <div class="desserts">Cake</div>  <div class="desserts">Pie</div>    <script type = "module" src="script.js"></script>  </body> |
| --- |

### script.js

| let element = document.getElementById("MyTitle")  element.style.backgroundColor = "lightgreen"  let fruits = document.getElementsByName("fruits")  // console.log(fruits)  // console.log(fruits[0]) //apple  // console.log(fruits[1]) //orange  // console.log(fruits[2]) //banana  fruits.forEach(fruit => {  if(fruit.checked){  console.log(fruit.value)  }  })  let vegetables = document.getElementsByTagName("li")  vegetables[0].style.backgroundColor = "orange" //carrots  vegetables[1].style.backgroundColor = "tan" //potatoes  vegetables[2].style.backgroundColor = "yellow" //onions  let desserts = document.getElementsByClassName("desserts")  desserts[0].style.backgroundColor = "gray" //ice cream  let element2 = document.querySelector("#MyTitle") //you can use id, classname, tagname  element2.style.color = "white"  let elements = document.querySelectorAll(".desserts")  elements.forEach(element => {  element.style.backgroundColor = "pink"  }) |
| --- |

# DOM traversal



* <body> - parent
  + id=”fruit” - the firstChild
  + id=”vegetable” - the second child
    - Fruit and dessert would be considered as siblings to vegetable
    - Fruit would be previous sibling
    - Dessert would be next sibling
  + id=”dessert” - the lastChild

### index.html

| <body>  <ul id="fruit">  <li>apple</li>  <li>orange</li>  <li>banana</li>  </ul>  <ul id="vegetables">  <li>carrots</li>  <li>potatoes</li>  <li>onions</li>  </ul>  <ul id="dessert">  <li>ice cream</li>  <li>cake</li>  <li>pie</li>  </ul>  </body> |
| --- |

### script.js

| let element = document.querySelector("#fruit");  let child = element.firstElementChild;  child.style.backgroundColor = "lightgreen";  // .firstElementChild  // .lastElementChild  // .parentElement  // .nextElementSibling  // .previousElementSibling  // .children[]  // Array.from(.children) |
| --- |

# Add/Change HTML elements

* .innerHTML (vulnerable to XSS attacks)
* .textContent (more secure)

## Example 1

### script.js

| const nameTag = document.createElement("h1") //creates a h1 header tag  nameTag.textContent = window.prompt("Enter you name")  document.body.append(nameTag) |
| --- |

## Example 2

### index.html

| <body>  <ul id="fruit">  <li>apple</li>  <li>orange</li>  <li>banana</li>  </ul>  <script type = "module" src="script.js"></script>  </body> |
| --- |

### script.js

| const myList = document.querySelector("#fruit")  const listItem = document.createElement("li")  const listItem2 = document.createElement("li")  const listItem3 = document.createElement("li")  listItem.textContent = "mango" //last index  myList.append(listItem)  listItem2.textContent = "kiwi"  myList.prepend(listItem2)  listItem3.textContent = "pear"  myList.insertBefore(listItem3, myList.getElementsByTagName("li")[4]) //placed before index 4  console.log(myList) |
| --- |

# Add/Change CSS Properties

### index.html

| <body>  <h1 id="MyTitle">This is my Title</h1>  <script type = "module" src="script.js"></script>  </body> |
| --- |

### script.js

| const title = document.getElementById("MyTitle")  title.style.backgroundColor = "black" //or "rgb(#, #, #)"" red, green, blue or Hex values  title.style.color = "rgb(50,200,250)"  title.style.fontFamily = "consolas"  title.style.textAlign = "center"  title.style.border = "2px solid"  title.style.display = "none" //hides the element  title.style.display = "block" //shows the element |
| --- |

# Events

* Event - some action the user or the browser does

### index.html

| <body>  <button id="MyButton">button</button>  <input id="MyText">  <div id="MyDiv"></div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| #MyDiv{  background-color: lightgreen;  width: 100px;  height: 100px;  } |
| --- |

### script.js examples of event functions

| function doSomething(){  alert("You did something")  }  function changeSomething(){  element.style.backgroundColor = "red"  }  function changeSomethingElse(){  element.style.backgroundColor = "lightgreen"  }  const element = document.getElementById("MyButton")  element.onclick = doSomething //onclick is an event attribute  const element = document.body  element.onload = doSomething //when the document loads the function runs  const element = document.getElementById("MyText")  element.onchange = doSomething //when we change the text box then leave the text box the function will run  const element = document.getElementById("MyDiv")  element.onmouseover = changeSomething //when we move the mouse over the element  element.onmouseout = changeSomethingElse //when mouse is outside of the element  element.onmousedown = changeSomething //when we click over the element  element.onmouseup = changeSomethingElse //when we don't click the element |
| --- |

# addEventListener()

* addEventListener(event, function, useCapture) - we can link an element and a function to an HTML element
  + You can add many event handlers to one element
  + Even the same event that invokes different functions
  + If two elements are listening for the same event the useCapture specifies which one goes first
  + Element runs the function based on the event listener

### index.html

| <body>  <div id="OuterDiv">  <div id="InnerDiv"></div>  </div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| #InnerDiv{  background-color: lightgreen;  width: 100px;  height: 100px;  border: 1px solid;  }  #OuterDiv{  background-color: lightgreen;  width: 200px;  height: 200px;  } |
| --- |

### script.js

| const InnerDiv = document.getElementById("InnerDiv")  const OuterDiv = document.getElementById("OuterDiv")  // InnerDiv.addEventListener("mouseover", changeRed)  // InnerDiv.addEventListener("mouseout", changeGreen)  InnerDiv.addEventListener("click", changeBlue)  OuterDiv.addEventListener("click", changeBlue, true) //this handles the outer element first  // function changeRed(){  // InnerDiv.style.backgroundColor = "red"  // }  // function changeGreen(){  // InnerDiv.style.backgroundColor = "lightgreen"  // }  function changeBlue(){  alert(`You selected ${this.id}`)  this.style.backgroundColor = "lightblue"  } |
| --- |

# Show/Hide HTML Elements

### index.html

| <body>  <button id="MyButton">toggle</button><br>  <image id="MyImage" src="santa.jpg" style="display: none;"></image>  <p>Press the Button...</p>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| #MyImage{  width: 300px;  } |
| --- |

### script.js

| const MyButton = document.querySelector("#MyButton")  const MyImage = document.querySelector("#MyImage")  MyButton.addEventListener("click", () => {  if(MyImage.style.display == "none"){  MyImage.style.display = "block"  }  else {  MyImage.style.display = "none"  }  }) |
| --- |

# Detect Key Presses

### style.css

| #MyDiv{  background-color: blue;  width: 100px;  height: 100px;  border: 1px solid;  position: relative;  } |
| --- |

### index.html

| <body>  <div id="MyDiv"></div>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| const MyDiv = document.getElementById("MyDiv")  window.addEventListener("keydown", move)  let x = 0  let y = 0  function move(event){  switch(event.key){  case "ArrowDown":  y+=5  MyDiv.style.top = y + "px"  break  case "ArrowUp":  y-=5  MyDiv.style.top = y + "px"  break  case "ArrowRight":  x+=5  MyDiv.style.left = x + "px"  break  case "ArrowLeft":  x-=5  MyDiv.style.left = x + "px"  break  default:  break  }  } |
| --- |

# Animations

### index.html

| <body>  <button id="MyButton">Begin</button>  <div id="MyDiv"></div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| #MyDiv{  background-color: red;  width: 100px;  height: 100px;  border: 1px solid;  position: relative;  } |
| --- |

## Example 1 sliding

### script.js

| const MyButton = document.getElementById("MyButton")  const MyAnimation = document.getElementById("MyDiv")  MyButton.addEventListener("click", begin)  function begin(){  let timerId = null  let x = 0  let y = 0  timerId = setInterval(frame, 5)  function frame(){  if(x >= 200 || y >= 200){  clearInterval(timerId)  } else {  x+=1  y+=1  MyAnimation.style.left = x + "px"  MyAnimation.style.top = y + "px"  }  }  } |
| --- |

## Example 2 turning

### script.js

| const MyButton = document.getElementById("MyButton")  const MyAnimation = document.getElementById("MyDiv")  MyButton.addEventListener("click", begin)  function begin(){  let timerId = null  let degrees = 0  let x = 0  let y = 0  timerId = setInterval(frame, 5)  function frame(){  if(x >= 200||y >= 200){  clearInterval(timerId)  } else {  degrees+=5  x+=1  y+=1  MyAnimation.style.left = x + "px"  MyAnimation.style.top = y + "px"  MyAnimation.style.transform = "rotateZ("+degrees+"deg)"  }  }  } |
| --- |

## Example 3 Scaling up and down

### script.js

| const MyAnimation = document.getElementById("MyDiv")  MyButton.addEventListener("click", begin)  function begin(){  let timerId = null  let scaleX = 1 //1 means 100%  let scaleY = 1  timerId = setInterval(frame, 5)  function frame(){  if(scaleX >= 2 || scaleY >= 2){ //scales up to 200%, you can scale up as well  clearInterval(timerId)  } else {  scaleX+=0.01  scaleY+=0.01  MyAnimation.style.transform = "scale("+scaleX+","+scaleY+")"  }  }  } |
| --- |

# Canvas API

* Canvas API - a means for drawing graphics. Used for animations, games, and data visualization

### index.html

| <body>  <canvas id="MyCanvas" width="500" height="500"></canvas>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| #MyCanvas{  /\* background-color: red; \*/  border: 1px solid black;  position: relative;  } |
| --- |

## Example 1 Draw lines

### script.js

| let canvas = document.getElementById("MyCanvas")  let context = canvas.getContext("2d") //to draw in the canvas we use the context  context.strokeStyle = "purple"  context.lineWidth = 10  context.beginPath() //starts the drawing  context.moveTo(0,0)  context.lineTo(250,250)  context.lineTo(250,500)  context.moveTo(500,0)  context.lineTo(250,250)  context.stroke() //ends the drawing |
| --- |

## Example 2 Triangles

### script.js

| let canvas = document.getElementById("MyCanvas")  let context = canvas.getContext("2d") //to draw in the canvas we use the context  context.strokeStyle = "green"  context.fillStyle = "yellow" //changes the fill color  context.lineWidth = 10 //changes the border  context.beginPath()  context.moveTo(250,0) //moves the position  context.lineTo(0,250) //draws the line  context.lineTo(500,250) //draws the line  context.lineTo(250,0) //draws the line  context.stroke()  context.fill() //fills in the drawing |
| --- |

## Example 3 Rectangle

### script.js

| let canvas = document.getElementById("MyCanvas")  let context = canvas.getContext("2d") //to draw in the canvas we use the context  context.fillStyle = "red"  context.lineWidth = 10  context.strokeStyle = "blue"  context.fillRect(0,0, 250, 300) //starting coordinates, width, height  context.strokeRect(0,0, 250, 300) //starting coordinates, width, height  context.fillStyle = "green"  context.lineWidth = 5  context.strokeStyle = "yellow"  context.fillRect(0,300, 250, 200) //starting coordinates, width, height  context.strokeRect(0,300, 250, 200) //starting coordinates, width, height  context.fillStyle = "purple"  context.lineWidth = 1  context.strokeStyle = "orange"  context.fillRect(250,200, 250, 300) //starting coordinates, width, height  context.strokeRect(250,200, 250, 300) //starting coordinates, width, height  context.fillStyle = "gray"  context.lineWidth = 15  context.strokeStyle = "pink"  context.fillRect(250,0, 250, 200) //starting coordinates, width, height  context.strokeRect(250,0, 250, 200) //starting coordinates, width, height |
| --- |

## Example 3 Circle

### script.js

| let canvas = document.getElementById("MyCanvas")  let context = canvas.getContext("2d") //to draw in the canvas we use the context  context.fillStyle = "lightblue"  context.lineWidth = 10  context.strokeStyle = "darkblue"  context.beginPath()  context.arc(250, 250, 200, 0, 2\*Math.PI, true) //center coordinates, radius, starting angle, ending angle,  context.stroke()  context.fill() |
| --- |

## Example 4 Text

### script.js

| let canvas = document.getElementById("MyCanvas")  let context = canvas.getContext("2d") //to draw in the canvas we use the context  context.font = "50px MV Boli"  context.fillStyle = "grey"  // context.fillText("You Win!", 100,100) //text, coordinates  context.textAlign = "center"  context.fillText("You Win!", canvas.width/2,canvas.height/2) //centers the text |
| --- |

# Window

* Window - interface used to talk to the web browser
  + The DOM is a property of the window

## Example 1

### script.js

| console.dir(window)  console.log(window.innerWidth) //inside the window  console.log(window.innerHeight)  console.log(window.outerWidth) //outside the window  console.log(window.outerHeight)  console.log(window.scrollX) //returns the scroll position  console.log(window.scrollY)  console.log(window.location.href) //returns the URL  //window.location.href = "https://google.com" //changes the location of the page  console.log(window.location.hostname) //returns the hostname (local Ip address)  console.log(window.location.pathname) //returns the landing page |
| --- |

## Example 2

### index.html

| <body>  <button id="MyButton">Button</button>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| const myButton = document.querySelector("#MyButton")  //myButton.addEventListener("click", () => window.open("https://google.com")) //when the button is clicked a new window opens  //myButton.addEventListener("click", () => window.close()) //closes the current window  //myButton.addEventListener("click", () => window.print()) //initiates the process of printing the page  //pop-ups  //window.alert("Hello!")  //window.confirm("Press OK to continue")  let age = window.prompt("Enter your age")  if(age < 18){  window.alert("You must be 18+ to visit this site")  window.close()  } |
| --- |

# Cookies

* Cookie - a small text file stored on your computer used to remember information about the user saved in name=value pairs (map)

## Example 1

### script.js

| console.log(navigator.cookieEnabled) //checks to see if cookies are enabled  document.cookie = "firstName=Dakadus; expires=Sun, 1 January 2030 12:00 UTC; path=/" //name=value pair, expiration date, path  document.cookie = "lastName=Prime; expires=Sun, 1 January 2030 12:00 UTC; path=/" //if we change the date to one that has already passed the cookie will be deleted  console.log(document.cookie) //the cookie property can hold more than one cookie |
| --- |

## Example 2

### index.html

| <body>  <label for="firstText">First Name:</label>  <input id="firstText"><br>  <label for="lastText">Last Name:</label>  <input id="lastText"><br>  <button id="submit">Submit</button>  <button id="cookieBtn">Get Cookies</button>  <script src="script.js"></script>  </body> |
| --- |

### script.js

| // setCookie("email", "dakpri@outlook.com", 365) //name, value, days  // setCookie("firstName", "Dakadus", 365) //name, value, days  // setCookie("lastName", "Prime", 365) //name, value, days  // console.log(document.cookie)  // // deleteCookie("firstName")  // // deleteCookie("lastName")  // // deleteCookie("email")  // // console.log(document.cookie)  // console.log(getCookie("email"))  // console.log(getCookie("firstName"))  // console.log(getCookie("lastName"))  const firstText = document.querySelector("#firstText")  const lastText = document.querySelector("#lastText")  const submitBtn = document.querySelector("#submit")  const cookieBtn = document.querySelector("#cookieBtn")  submitBtn.addEventListener("click", ()=>{  setCookie("firstName", firstText.value, 365)  setCookie("lastName", lastText.value, 365)  })  cookieBtn.addEventListener("click", ()=>{  firstText.value = getCookie("firstName")  lastText.value = getCookie("lastName")  })  function setCookie(name, value, daysToLive){ //name, value, number of days to save  const date = new Date()  date.setTime(date.getTime + daysToLive \*24 \*60 \*60 \*1000)  let expires = "expires=" + date.toUTCString  document.cookie = `${name}=${value}; ${expires}; path=/`  }  function deleteCookie(name){  setCookie(name, null, null) //set the value and date to null so it doesn't exist  }  function getCookie(name){  const cDecoded = decodeURIComponent(document.cookie)  const cArray = cDecoded.split("; ")  let result = null  cArray.forEach(element => {  if(element.indexOf(name) == 0){  result = element.substring(name.length + 1)  }  })  return result  } |
| --- |

# Stopwatch Project

### index.html

| <body>  <div id="timeContainer">  <div id="timeDisplay">00:00:00</div>  <button id="startBtn" class="timerBtn">Start</button>  <button id="pauseBtn" class="timerBtn">Pause</button>  <button id="resetBtn" class="timerBtn">Reset</button>  </div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| .timerBtn{  width: 80px;  height: 30px;  border: 3px solid;  border-radius: 12px;  background-color: #333333;  color: white;  cursor: pointer;  font-family: consolas, monospace;  }  #timeDisplay{  font-size: 75px;  color: #40c437;  font-family: consolas, monospace;  }  #timeContainer{  text-align: center;  border: 3px solid;  border-radius: 25px;  background-color: #222222;  } |
| --- |

### script.js

| const timeDisplay = document.querySelector("#timeDisplay")  const startBtn = document.querySelector("#startBtn")  const pauseBtn = document.querySelector("#pauseBtn")  const resetBtn = document.querySelector("#resetBtn")  let startTime = 0  let elapsedTime = 0  let currentTime = 0  let paused = true  let intervalId  let hrs = 0  let mins = 0  let secs = 0  startBtn.addEventListener("click", () => {  if(paused){  paused = false  startTime = Date.now() - elapsedTime //will give you the current date and time in milliseconds  intervalId = setInterval(updateTime, 1000)  }  })  pauseBtn.addEventListener("click", () => {  if(!paused){  paused = true  elapsedTime = Date.now() - startTime  clearInterval(intervalId)  }  })  resetBtn.addEventListener("click", () => {  paused = true  clearInterval(intervalId)  startTime = 0  elapsedTime = 0  currentTime = 0  hrs = 0  mins = 0  secs = 0  timeDisplay.textContent = "00:00:00"  })  function updateTime(){  elapsedTime = Date.now() - startTime  secs = Math.floor((elapsedTime/1000)%60)  mins = Math.floor((elapsedTime/(1000\*60))%60)  hrs = Math.floor((elapsedTime/(1000\*60\*60))%60)  secs = pad(secs)  mins = pad(mins)  hrs = pad(hrs)  timeDisplay.textContent = `${hrs}:${mins}:${secs}`  function pad(unit){  return(("0")+unit).length > 2 ? unit : "0" + unit  }  } |
| --- |

# A game of Rock Paper Scissors

### index.html

| <body>  <div id="gameDiv">  <h1 class="gameText" id="playerText">Player: </h1>  <h1 class="gameText" id="computerText">Computer: </h1>  <h1 class="gameText" id="resultText">Result: </h1>  <button class="choiceBtn">Rock</button>  <button class="choiceBtn">Paper</button>  <button class="choiceBtn">Scissors</button>  </div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| .choiceBtn{  line-height: 30px;  width: 150px;  }  #gameDiv{  font-family: 'Brush Script MT', cursive;  border: 3px solid;  border-radius: 25px;  padding: 10px;  background-color: lightgray;  text-align: center;  }  #playerText{  color: blue;  }  #computerText{  color: red;  } |
| --- |

### script.js

| const playerText = document.querySelector("#playerText")  const computerText = document.querySelector("#computerText")  const resultText = document.querySelector("#resultText")  const choiceBtns = document.querySelectorAll(".choiceBtn")  let player  let computer  let result  choiceBtns.forEach(button => button.addEventListener("click", () => {  player = button.textContent //returns either rock, paper, or scissors  computerTurn()  playerText.textContent = `Player: ${player}`  computerText.textContent = `Computer: ${computer}`  resultText.textContent = checkWinner()  }))  function computerTurn(){  const randNum = Math.floor(Math.random() \* 3) + 1 //gives us a random number between 1 and 3  switch(randNum){  case 1:  computer = "Rock"  break  case 2:  computer = "Paper"  break  case 3:  computer = "Scissors"  break  }  }  function checkWinner(){  if(player == computer){  return "Draw!"  }  else if(computer == "Rock"){  return (player == "Paper") ? "You Win!" : "You Lose!"  }  else if(computer == "Paper"){  return (player == "Scissors") ? "You Win!" : "You Lose!"  }  else if(computer == "Scissors"){  return (player == "Rock") ? "You Win!" : "You Lose!"  }  } |
| --- |

# A game of TicTacToe

### index.html

| <body>  <div id="gameContainer">  <h1>Tic Tac Toe</h1>  <div id="cellContainer">  <div cellIndex="0" class="cell"></div>  <div cellIndex="1" class="cell"></div>  <div cellIndex="2" class="cell"></div>  <div cellIndex="3" class="cell"></div>  <div cellIndex="4" class="cell"></div>  <div cellIndex="5" class="cell"></div>  <div cellIndex="6" class="cell"></div>  <div cellIndex="7" class="cell"></div>  <div cellIndex="8" class="cell"></div>  </div>  <h2 id="statusText"></h2>  <button id="restartBtn">Restart</button>  </div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| .cell{  width: 75px;  height: 75px;  border: 2px solid;  box-shadow: 0 0 0 2px;  line-height: 75px;  font-size: 50px;  cursor: pointer;  }  #gameContainer{  font-family: "Permanent Marker", cursive;  text-align: center;  }  #cellContainer{  display: grid;  grid-template-columns: repeat(3, auto);  width: 225px;  margin: auto;  } |
| --- |

### script.js

| const cells = document.querySelectorAll(".cell")  const statusText = document.querySelector("#statusText")  const restartBtn = document.querySelector("#restartBtn")  const winConditions = [ //all the different ways to win  [0,1,2],  [3,4,5],  [6,7,8],  [0,3,6],  [1,4,7],  [2,5,8],  [0,4,8],  [2,4,6]  ]  let options = ["","","","","","","","",""]  let currentPlayer = "X"  let running = true  initializeGame()  function initializeGame(){  cells.forEach(cell => cell.addEventListener("click", cellClicked))  restartBtn.addEventListener("click",restartGame)  statusText.textContent = `${currentPlayer}'s turn`  }  function cellClicked(){  const cellIndex = this.getAttribute("cellIndex") //refers to what ever cell we click on  if(options[cellIndex] != "" || !running){ //checks to see if the box can be clicked  return  } else {  updateCell(this, cellIndex)  checkWinner()  }  }  function updateCell(cell, index){  options[index] = currentPlayer //updates the cell index with an X or O  cell.textContent = currentPlayer  }  function changePlayer(){  currentPlayer = (currentPlayer == "X") ? "O" : "X"  statusText.textContent = `${currentPlayer}'s turn`  }  function checkWinner(){  let roundWon = false  for(let i = 0; i < winConditions.length; i++){ //interate the inner arrays  const condition = winConditions[i]  const cellA = options[condition[0]]  const cellB = options[condition[1]]  const cellC = options[condition[2]]  if(cellA == "" || cellB == "" || cellC == ""){  continue  }  if (cellA == cellB && cellB == cellC){  roundWon = true  break  }  }  if(roundWon){  statusText.textContent = `${currentPlayer} Wins!`  running = false  } else if(!options.includes("")){  statusText.textContent = `Draw`  } else {  changePlayer()  }  }  function restartGame(){  currentPlayer = "X"  options = ["","","","","","","","",""]  statusText.textContent = `${currentPlayer}'s turn`  cells.forEach(cell => cell.textContent = "")  running = true  } |
| --- |

# A game of Snake

### index.html

| <body>  <div id="gameContainer">  <canvas id="gameBoard" width="500" height="500"></canvas>  <div id="scoreText">0</div>  <button id="resetBtn">Reset</button>  </div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| #gameBoard{  border: 3px solid;  }  #gameContainer{  text-align: center;  }  #scoreText{  font-family: "Permanent Marker", cursive;  font-size: 100px;  }  #resetBtn{  font-family: "Permanent Marker", cursive;  font-size: 22px;  width: 100px;  height: 50px;  border: 4px solid;  border-radius: 15px;  cursor: pointer;  } |
| --- |

### script.js

| const gameBoard = document.querySelector("#gameBoard")  const ctx = gameBoard.getContext("2d")  const scoreText = document.querySelector("#scoreText")  const resetBtn = document.querySelector("#resetBtn")  const gameWidth = gameBoard.width  const gameHeight = gameBoard.height  const boardBackground = "white"  const snakeColor = "lightgreen"  const snakeBorder = "black"  const foodColor = "red"  const unitSize = 25  let running = false  let xVelocity = unitSize  let yVelocity = 0  let foodX  let foodY  let score = 0  let snake = [ //an array of body parts at different coordinates  {x:unitSize \*4, y:0},  {x:unitSize \*3, y:0},  {x:unitSize \*2, y:0},  {x:unitSize \*1, y:0},  {x:0, y:0}  ]  window.addEventListener("keydown", changeDirection)  resetBtn.addEventListener("click", resetGame)  gameStart()  function gameStart(){  running = true  scoreText.textContent = score  createFood()  drawFood()  nextTick()  }  function nextTick(){  if(running){  setTimeout(() => {  clearBoard()  drawFood()  moveSnake()  drawSnake()  checkGameOver()  nextTick()  }, 75) //75 milliseconds  } else {  displayGameOver()  }  }  function clearBoard(){  ctx.fillStyle = boardBackground  ctx.fillRect(0,0,gameWidth, gameHeight)  }  function changeDirection(event){ //we invoke this function every time we press a key  const keyPressed = event.keyCode  // numeric values of the arrow keys  const left = 37  const up = 38  const right = 39  const down = 40  //boolean valuables  const goingUp = (yVelocity == -unitSize)  const goingDown = (yVelocity == unitSize)  const goingRight = (xVelocity == unitSize)  const goingLeft = (xVelocity == -unitSize)  switch(true){  case(keyPressed == left && !goingRight):  xVelocity = -unitSize  yVelocity = 0  break  case(keyPressed == up && !goingDown):  xVelocity = 0  yVelocity = -unitSize  break  case(keyPressed == right && !goingLeft):  xVelocity = unitSize  yVelocity = 0  break  case(keyPressed == down && !goingUp):  xVelocity = 0  yVelocity = unitSize  break  }  }  function resetGame(){  score = 0  xVelocity = unitSize  yVelocity = 0  snake = [ //an array of body parts at different coordinates  {x:unitSize \*4, y:0},  {x:unitSize \*3, y:0},  {x:unitSize \*2, y:0},  {x:unitSize \*1, y:0},  {x:0, y:0}  ]  gameStart()  }  function createFood(){ //creates the food  function randomFood(min, max){  const randNum = Math.round((Math.random() \* (max - min) + min) / unitSize) \* unitSize  return randNum  }  foodX = randomFood(0, gameWidth - unitSize)  foodY = randomFood(0, gameHeight - unitSize)  }  function drawFood(){ //places said "food" in the board  ctx.fillStyle = foodColor  ctx.fillRect(foodX, foodY, unitSize, unitSize)  }  function moveSnake(){  const head = {x: snake[0].x + xVelocity, y: snake[0].y + yVelocity}  snake.unshift(head)  if(snake[0].x == foodX && snake[0].y == foodY){ //if food is eaten  score+=1  scoreText.textContent = score  createFood()  } else {  snake.pop()  }  }  function drawSnake(){  ctx.fillStyle = snakeColor  ctx.strokeStyle = snakeBorder  snake.forEach(snakePart => {  ctx.fillRect(snakePart.x, snakePart.y, unitSize, unitSize)  ctx.strokeRect(snakePart.x, snakePart.y, unitSize, unitSize)  })  }  function checkGameOver(){  switch(true){  case (snake[0].x < 0):  running = false  break  case (snake[0].x >= gameWidth):  running = false  break  case (snake[0].y < 0):  running = false  break  case (snake[0].y >= gameHeight):  running = false  break  }  for(let i = 1; i < snake.length; i+=1){  if(snake[i].x == snake[0].x && snake[i].y == snake[0].y){ //if it hits itself  running = false  }  }  }  function displayGameOver(){  ctx.font = "50px MV Boli"  ctx.fillStyle = "black"  ctx.textAlign = "center"  ctx.fillText("Game Over!", gameWidth/2, gameHeight/2)  running = false  } |
| --- |

# A game of Pong

### index.html

| <body>  <div id="gameContainer">  <canvas id="gameBoard" width="500" height="500"></canvas>  <div id="scoreText">0:0</div>  <button id="resetBtn">Reset</button>  </div>  <script src="script.js"></script>  </body> |
| --- |

### style.css

| body{  background-color: rgb(61, 61, 61);  }  #gameBoard{  border: 3px solid;  border-color: darkgreen;  }  #gameContainer{  text-align: center;  }  #scoreText{  font-family: "consolas", monospace;  font-size: 100px;  color: darkgreen;  }  #resetBtn{  font-family: "Permanent Marker", cursive;  font-size: 22px;  width: 100px;  height: 50px;  border: 4px solid;  border-radius: 15px;  cursor: pointer;  } |
| --- |

### script.js

| const gameBoard = document.querySelector("#gameBoard")  const ctx = gameBoard.getContext("2d")  const scoreText = document.querySelector("#scoreText")  const resetBtn = document.querySelector("#resetBtn")  const gameWidth = gameBoard.width  const gameHeight = gameBoard.height  const boardBackground = "black"  const paddle1Color = "darkgreen"  const paddle2Color = "darkgreen"  const paddleBorder = "lightgreen"  const ballColor = "darkgreen"  const ballBorderColor = "lightgreen"  const ballRadius = 12.5 //full diameter is 25  const paddleSpeed = 50  let intervalID  let ballSpeed = 1  let ballX = gameWidth / 2  let ballY = gameHeight /2  let ballXDirection = 0  let ballYDirection = 0  let player1Score = 0  let player2Score = 0  let paddle1 = {  width: 25,  height: 100,  x:0, //all the way to the left corner  y:0 //top  }  let paddle2 = {  width: 25,  height: 100,  x:gameWidth-25, //all the way to the right corner  y:gameWidth-100 //bottom  }  window.addEventListener("keydown", changeDirection)  resetBtn.addEventListener("click", resetGame)  gameStart()  function gameStart(){  createBall()  nextTick()  }  function nextTick(){  intervalID = setTimeout(() => {  clearBoard()  drawPaddles()  moveBall()  drawBall(ballX, ballY)  checkCollision()  nextTick()  }, 10) //repeats every 10 milliseconds  }  function clearBoard(){  ctx.fillStyle = boardBackground  ctx.fillRect(0,0,gameWidth, gameHeight)  }  function changeDirection(event){ //we invoke this function every time we press a key  const keyPressed = event.keyCode  const paddle1Up = 87  const paddle1Down = 83  const paddle2Up = 38  const paddle2Down = 40  switch(keyPressed){  case(paddle1Up):  if(paddle1.y >0){  paddle1.y -= paddleSpeed  }  break  case(paddle1Down):  if(paddle1.y < gameHeight-paddle1.height){  paddle1.y += paddleSpeed  }  break  case(paddle2Up):  if(paddle2.y > 0){  paddle2.y -= paddleSpeed  }  break  case(paddle2Down):  if(paddle2.y < gameHeight-paddle1.height){  paddle2.y += paddleSpeed  }  break  }  }  function resetGame(){  player1Score = 0  player2Score = 0  //reset paddles  paddle1 = {  width: 25,  height: 100,  x:0, //all the way to the left corner  y:0 //top  }  paddle2 = {  width: 25,  height: 100,  x:gameWidth-25, //all the way to the right corner  y:gameWidth-100 //bottom  }  //reset ball  ballSpeed = 1  ballX = 0  ballY = 0  ballXDirection = 0  ballYDirection = 0  updateScore()  clearInterval(intervalID)  gameStart()  }  function drawPaddles(){  ctx.strokeStyle = paddleBorder  ctx.fillStyle = paddle1Color  ctx.fillRect(paddle1.x, paddle1.y, paddle1.width, paddle1.height) //fills in the paddle  ctx.strokeRect(paddle1.x, paddle1.y, paddle1.width, paddle1.height) //draws the border  ctx.fillStyle = paddle2Color  ctx.fillRect(paddle2.x, paddle2.y, paddle2.width, paddle2.height) //fills in the paddle  ctx.strokeRect(paddle2.x, paddle2.y, paddle2.width, paddle2.height) //draws the border  }  function createBall(){  ballSpeed = 1  if(Math.round(Math.random()) == 1){  ballXDirection = 1  } else {  ballXDirection = -1  }  if(Math.round(Math.random()) == 1){  ballYDirection = 1  } else {  ballYDirection = -1  }  ballX = gameWidth / 2  ballY = gameHeight /2  drawBall(ballX, ballY)  }  function moveBall(){  ballX += (ballSpeed \* ballXDirection)  ballY += (ballSpeed \* ballYDirection)  }  function drawBall(ballX, ballY){  ctx.fillStyle = ballColor  ctx.strokeStyle = ballBorderColor  ctx.lineWidth = 2  ctx.beginPath()  ctx.arc(ballX, ballY, ballRadius, 0, 2\*Math.PI)  ctx.stroke()  ctx.fill()  }  function checkCollision(){  if(ballY <= 0 + ballRadius){  ballYDirection \*= -1  }  if(ballY >= gameHeight - ballRadius){  ballYDirection \*= -1  }  if(ballX <= 0){  player2Score += 1  updateScore()  createBall()  return  }  if(ballX >= gameWidth){  player1Score += 1  updateScore()  createBall()  return  }  if(ballX <= (paddle1.x + paddle1.width + ballRadius)){ //paddle1  if(ballY > paddle1.y && ballY < paddle1.y + paddle1.height){  ballX = (paddle1.x + paddle1.width) + ballRadius //if ball gets stuck  ballXDirection \*= -1  ballSpeed += 1  }  }  if(ballX >= (paddle2.x - ballRadius)){ //paddle2  if(ballY > paddle2.y && ballY < paddle2.y + paddle1.height){  ballX = paddle2.x - ballRadius //if ball gets stuck  ballXDirection \*= -1  ballSpeed += 1  }  }  }  function updateScore(){  scoreText.textContent = `${player1Score}:${player2Score}`  } |
| --- |