JavaScript:

- Javascript is an interpreted programming language that is used to make webpages interactive.
- It's object based.
- It runs on the client's browser.
- It uses events and actions to make webpages interactive.
- We can either have an external javascript file and link it to the html file or we can use inline javascript. Always create an external javascript file and link it. Either way, you'd put the javascript (link or code) in the html <script> tag.
  E.g.
  1. Linking to an external javascript file called index.js:
     <script src="index.js"> </script>
  2. Putting inline javascript:
     <script>
     // Some Javascript code.
     </script>
- **Comments:**
- Single line comments are denoted with //.
- Multi-line comments are denoted with /* and */.
- **Equality:**
- There are two main ways of checking for equality in Javascript. We can use either the == operator or the === operator. However, the == operator will sometimes produce odd results. The === operator checks for type equality as well as content. A strict comparison (===) is only true if the operands are of the same type and the contents match. However, (==) converts the operands to the same type before making the comparison. Therefore, it is recommended that you only use the === operator and to never use the == operator.
- E.g.

```
console.log(1 == 1); // expected output: true
console.log('1' == 1); // expected output: true
console.log(1 === 1); // expected output: true
console.log('1' === 1); // expected output: false
```

- **Operators:**
- A table of the different arithmetic operators:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Division Remainder)

++	Increment
--	Decrement

- A table of the different assignment operators:

Operator	**Example**	**Same As**
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

- A table of the different comparison operators:

Operator	**Description**
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

- A table of the different logical operators:

Operator	**Description**
&&	logical and
||	logical or

!	logical not

- A table of the different type operators:

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

- **Data Types:**
- Some Javascript data types include numbers, strings, and objects.
- JavaScript has **dynamic types**. This means that the same variable can be used to hold different data types.

1. **String:**

- A string is a series of characters.
- Strings are written with quotes. You can use single or double quotes.
- E.g.

```
var carName1 = "Volvo XC60";   // Using double quotes
var carName2 = 'Volvo XC60';   // Using single quotes
```

- You can use quotes inside a string, as long as they don't match the quotes surrounding the string.
- E.g.

```
var answer1 = "It's alright";           // Single quote inside double quotes
var answer2 = "He is called 'Johnny'";    // Single quotes inside double quotes
var answer3 = 'He is called "Johnny"';    // Double quotes inside single quotes
```

2. **Numbers:**

- JavaScript has only one type of numbers.
- Numbers can be written with, or without decimals.
- E.g.

```
var x1 = 34.00;     // Written with decimals
var x2 = 34;        // Written without decimals
```

- Extra large or extra small numbers can be written with scientific (exponential) notation.
- E.g.

```
var y = 123e5;      // 12300000
var z = 123e-5;     // 0.00123
```

3. **Boolean:**

- Booleans can only have two values: true or false.

4. **Objects:**

- JavaScript objects are written with curly braces {}.
- Object properties are written as name:value pairs, separated by commas.
- E.g.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

5. **Arrays:**

- JavaScript arrays are written with square brackets.
- Array items are separated by commas. Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

- E.g.
  var cars = ["Saab", "Volvo", "BMW"];

6. Typeof Operator:

- You can use the JavaScript **typeof** operator to find the type of a JavaScript variable. The typeof operator returns the type of a variable or an expression.
- E.g.

```
typeof ""              // Returns "string"
typeof "John"          // Returns "string"
typeof "John Doe"      // Returns "string"
typeof 0               // Returns "number"
typeof 314             // Returns "number"
typeof 3.14            // Returns "number"
typeof (3)             // Returns "number"
typeof (3 + 4)         // Returns "number"
```

7. Undefined:

- In JavaScript, a variable without a value, has the value undefined. The type is also undefined.
- E.g.
  var car; // Value is undefined, type is undefined
- Any variable can be emptied, by setting the value to undefined. The type will also be undefined.
- E.g.
  car = undefined; // Value is undefined, type is undefined
- You can empty an object by setting it to undefined.
- E.g.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = undefined;   // Now both value and type is undefined
```

8. Empty Values:

- An empty value has nothing to do with undefined.
- An empty string has both a legal value and a type.
- E.g.
  var car = ""; // The value is "", the typeof is "string"

9. Null:

- In JavaScript null is "nothing". It is supposed to be something that doesn't exist.
- In JavaScript, the data type of null is an object.
- You can empty an object by setting it to null.
- E.g.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;    // Now value is null, but type is still an object
```

- Undefined and null are equal in value but different in type.

10. Primitive Data:

- A primitive data value is a single simple data value with no additional properties and methods.
- The typeof operator can return one of these primitive types:
  - String
  - Number
  - Boolean
  - undefined

- E.g.
  typeof "John" // Returns "string"
  typeof 3.14 // Returns "number"
  typeof true // Returns "boolean"
  typeof false // Returns "boolean"
  typeof x // Returns "undefined" (if x has no value)

11. Complex Data:

- The typeof operator can return one of two complex types:
  - Function
  - Object
- The typeof operator returns "object" for objects, arrays, and null.
- The typeof operator does not return "object" for functions. Instead, it returns “function” for functions.
- Note: The typeof operator returns "object" for arrays because in JavaScript arrays are objects.
- E.g.
  typeof {name:'John', age:34} // Returns "object"
  typeof [1,2,3,4] // Returns "object"
  typeof null // Returns "object"
  typeof function myFunc(){} // Returns "function"
- **Variables:**
- JavaScript variables are containers for storing data values.
- Variables are defined using one of three keywords, **var**, **let** and **const**, and the = operator.
- **Note:** var is global while let is local. For this reason, it is better to use let than var.
- Javascript variables can only start with a letter, dollar sign ($) or underscore, are case sensitive and can only contain letters, numbers, underscores and dollar signs.
- A variable declared without a value will have the value undefined.
- E.g.
  let name; // name is undefined.
- **Note:** If you re-declare a JavaScript variable, it will not lose its value.
- E.g. The variable carName will still have the value "Volvo" after the execution of these statements:
  let carName = "Volvo";
  let carName;
- **Arrays:**
- JavaScript arrays are used to store multiple values in a single variable.
- E.g.
  let cars = ["Saab", "Volvo", "BMW"];
- An array is a special variable, which can hold more than one value at a time.
- Syntax for creating an array:
  let variableName = [item1, item2, … itemn];
- **Note:** Spaces and line breaks are not important. A declaration can span multiple lines.
- E.g.
  let variableName = [
  Item1,
  Item2,

```
…
Itemn
];
```

- **Note:** Another way we can create an array is using the new keyword.
  Syntax: **let variableName = new Array(item1, item2, …, itemn);**
- E.g.
  let cars = new Array("Saab", "Volvo", "BMW");
- You access an array element by referring to the index number. Index 0 refers to the first element, index 1 refers to the second element and so on.
- Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays. However, arrays always use numbers to access its elements while objects always use names to access its members.
- E.g. Consider the JS array and object below:
  let person = ["John", "Doe", 46];
  let person = {firstName:"John", lastName:"Doe", age:46};
  To access the first element of the array, you'd do person[0], but to access the first element of the object, you'd do person.firstName.
- The easiest way to add a new element to an array is using the push() method.
- E.g.
  let fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits.push("Lemon"); // adds a new element (Lemon) to fruits
- The length property of an array returns the length of an array (the number of array elements).
- E.g.
  var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits.length; // the length of fruits is 4
- **For Loops:**
- Syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

- Statement 1 is executed (one time) before the execution of the code block. Normally you will use statement 1 to initialize the variable used in the loop.
- Statement 2 defines the condition for executing the code block.
- Statement 3 is executed (every time) after the code block has been executed. Often statement 3 increments/decrements the value of the initial variable.
- E.g.

```
for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

- **For In Loops:**
- The JavaScript for/in statement loops through the properties of an object.
- E.g.

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
```

```
  text += person[x];
}
```

- **For Of Loops:**
- The JavaScript for/of statement loops through the values of an iterable objects
- for/of lets you loop over data structures that are iterable such as Arrays, Strings, etc.
- Syntax:

```
for (variable of iterable) {
  // code block to be executed
}
```

- **variable:** For every iteration the value of the next property is assigned to the variable. variable can be declared with const, let, or var.
- **iterable:** An object that has iterable properties.
- **E.g. Looping over an array:**

```
var cars = ['BMW', 'Volvo', 'Mini'];
var x;

for (x of cars) {
  document.write(x + "<br >");
}
```

- E.g. Looping over a string:

```
var txt = 'JavaScript';
var x;

for (x of txt) {
  document.write(x + "<br >");
}
```

- The difference between for in and for of loops is that for in loops return a list of keys on the object being iterated, whereas for of loops return a list of values of the numeric properties of the object being iterated.
- E.g. Consider the below 2 pieces of code:

1.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var txt = "";
var person = [1, 2, 3];
var x;
for (x in person) {
  txt += x + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
```

```
</html>
```

This returns 0 1 2.

2.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var txt = "";
var person = [1, 2, 3];
var x;
for (x of person) {
  txt += x + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

This returns 1 2 3.

- **For Each Loop:**
- Used to loop over arrays.
- The forEach() method calls a function once for each element in an array, in order.
- **Note:** The function is not executed for array elements without values.
- Syntax: **array.forEach(function(currentValue, index, arr), thisValue)**

Parameter	Description
function	Required. A function to be run for each element in the array.
currentValue	Required. The value of the current element
index	Optional. The array index of the current element
arr	Optional. The array object the current element belongs to
thisValue	Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

- E.g.

```
var sum = 0;
var numbers = [65, 44, 12, 4];
numbers.forEach(myFunction);
```

```
function myFunction(item) {
  sum += item;
  document.getElementById("demo").innerHTML = sum;
}
```

- **While Loops:**
- The while loop loops through a block of code as long as a specified condition is true.
- Syntax:

```
while (condition) {
  // code block to be executed
}
```

- E.g.

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

- **Do While Loop:**
- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax:

```
do {
  // code block to be executed
}
while (condition);
```

- E.g.

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

- **If Statements:**
- Syntax:

```
if (condition 1) {
...
}
else if (condition 2){
…
}
else{
…
}
```

- **Note:** You must have the if block and only 1 if block. You can have as many else if blocks as you want, and you can have at most 1 else block.
- E.g.

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
```

```
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

- **Switch Statement:**
- The switch statement is used to perform different actions based on different conditions.
- Use the switch statement to select one of many code blocks to be executed.
- Syntax:

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

- This is how it works:
  - The switch expression is evaluated once.
  - The value of the expression is compared with the values of each case.
  - If there is a match, the associated block of code is executed.
- E.g.

  The getDay() method returns the weekday as a number between 0 and 6.
  (Sunday=0, Monday=1, Tuesday=2 ..)
  This example uses the weekday number to calculate the weekday name:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
```

```
}
```

- When JavaScript reaches a **break** keyword, it breaks out of the switch block. This will stop the execution of inside the block.
- The **default** keyword specifies the code to run if there is no case match.
- **Note:** The default case does not have to be the last case in a switch block, but if default is not the last case in the switch block, remember to end the default case with a break.
- **Functions:**
- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when something invokes it. There are 3 ways we can invoke a function:
    1. When an event occurs (when a user clicks a button)
    2. When it is invoked (called) from JavaScript code
    3. Automatically (self invoked)
- You declare a function by prefixing it with the function keyword.
- Syntax:

```
function function_name(...){
…
}
```

- E.g.

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

- Variables declared within a JavaScript function, become local to the function. **Local variables** can only be accessed from within the function. Since local variables are only recognized inside their functions, variables with the same name can be used in different functions. Local variables are created when a function starts, and deleted when the function is completed.
- **Objects:**
- An object in Javascript is similar to a dictionary in Python. Unlike in Python, the keys of the object must be String but the value can be any valid type. This type of structure is called a **JSON or Javascript Object Notation**.
- E.g.

```
myObject = {
  "firstName": "kevin",
  "lastName": "zhang",
  "age": 69,
  "cool": true
}
```

- The name:values pairs in JavaScript objects are called properties.
- You can access object properties in two ways:
    1. **objectName.propertyName**
    2. **objectName["propertyName"]**
- Objects can also have methods.
- Methods are actions that can be performed on objects.
- Methods are stored in properties as function definitions.
- E.g.

```
var person = {
  firstName: "John",
```

```
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

- You access an object method with the following syntax:
  objectName.methodName()
- E.g.
  name = person.fullName();
- In a function definition, **this** refers to the "owner" of the function. It is equivalent to self in Python.
- In the example above, **this** is the person object that "owns" the fullName function.
  I.e. this.firstName means the firstName property of this object.
- When a JavaScript variable is declared with the keyword "new", the variable is created as an object. Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.
- E.g. Creating an object using the keyword “new”:
  let apple = new Object();
  apple.color = “red”;
  apple.shape = “round”;
  This is equivalent to **let var = {“color”: “red”, “shape”: “round”};**
- However, the above way is tedious and won’t work well if you want to create many similar objects. You can use a constructor pattern instead.
- E.g.

```
function Fruit(name, color, shape){
        this.name = name;
        this.color = color;
        this.shape = shape;
}

let apple = new Fruit(‘apple’, ‘red’, ‘round’);
let melon = new Fruit(‘melon’, ‘green’, ‘round’);
```

- **Print Statements:**
- JavaScript can output data in different ways:
  - Writing into an HTML element, using **innerHTML**.
  - Writing into the HTML output using **document.write()**.
  - Writing into an alert box, using **window.alert()**.
  - Writing into the browser console, using **console.log()**.
    Note: You can view console output in Chrome by pressing CMD+OPTION+J on a Mac or CTRL+SHIFT+J on Windows.
- Using innerHTML:
  - To access an HTML element, JavaScript can use the **document.getElementById(id)** method. This is the most common way of obtaining an element to modify. Since IDs in HTML are guaranteed to be unique, we can access a pre-existing element in an HTML form using this strategy.
  - The id attribute defines the HTML element. The innerHTML property defines the HTML content.

- E.g.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

- E.g.

HTML (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My Website</title>
</head>
<body>
  <!-- we will not see the text here because it will be overwritten
   by the Javascript -->
  <p id="my-content">This is the original content</p>
  <script src="script.js"></script>
</body>
</html>
```

Javascript (script.js):

```
document.getElementById('my-content').innerHTML = 'keviniscool';
```

- **Note:** There is a similar command called **document.getElementsByClassName()** however since classes are not unique, this command will always return the result in an array.

- Using document.write():
  - For testing purposes, it is convenient to use document.write().
  - **Note:** Using document.write() after an HTML document is loaded, will delete all existing HTML.
  - E.g.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>
```

```
<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

- Using window.alert():
  - You can use an alert box to display data.
  - E.g.

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

- Using console.log()
  - For debugging purposes, you can use the console.log() method to display data.
  - E.g.

```
<!DOCTYPE html>
<html>
<body>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

- **Events:**
- An **HTML event** can be something the browser does or something a user does.
- Here are some examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- Often, when events happen, you may want to do something. JavaScript lets you execute code when events are detected.
- HTML allows event handler attributes, along with JavaScript code, to be added to HTML elements.
- A table of HTML event:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element

onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

- Event handlers can be used to handle, and verify, user input, user actions, and browser actions:
    - Things that should be done every time a page loads.
    - Things that should be done when the page is closed.
    - Action that should be performed when a user clicks a button.
    - Content that should be verified when a user inputs data.
- Many different methods can be used to let JavaScript work with events:
    - HTML event attributes can execute JavaScript code directly.
    - HTML event attributes can call JavaScript functions.
    - You can assign your own event handler functions to HTML elements.
    - You can prevent events from being sent or being handled.
- E.g. In the past labs, we've made forms that don't do anything. Using Javascript, we can retrieve input from forms and perform interactive behaviour with the input we've collected. The most common way of doing this is by accessing the .value attribute of an element and reacting to an event such as onclick().

HTML (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My Website</title>
</head>
<body>
   <label>What is your name?</label>
   <input type="text" id="name-form" placeholder="Enter your name">
   <button type="submit" onclick="nameResult()">Submit</button>
   <!-- we put a placeholder <p> here so that it can be written into
   later -->
   <!-- using document.write() would overwrite the entire body -->
   <p id="result"></p>
   <script src="script.js"></script>
</body>
</html>
```

Javascript (script.js):

```
// this function is called whenever the button is clicked
function nameResult() {
  // get the value inside the form
  name = document.getElementById('name-form').value;
  // add the value
  document.getElementById('result').innerHTML = 'your name is: ' + name;
}
```