JavaScript-Essentials

Why JS?

Making cool things!!!
taking a static website, make it dynamic

- **We started Building a house**
  - we did `HTML` (structure)
  - we did design (`css`)
  - what about the actual functioning of the house (heat, electricity, etc..)
  - functionality = JS!
- bonus : material icon them :)

what is a server ?

- 127.0.0.1/localhost

Link JS with html

- where do we place JS?
- `<script></script>`
- preferred to place is at the bottom of body tag
- What do I do if have a really long script or I want to use it other pages ?
- `<script src="./script.js"> </script>`
- `;` marking an end of a statement, like full stop in english (code will still work, not required)

Chrome dev tools

- console to test JS code
- accept direct JS code
- we can also log values into our console.

variables

- store values and use the values later on
- variables in js
  - `var coolName = "Kwel David"`
  - `let coolCar = "some cool car"`
  - `const coolAge = 120`
- let/var can be assigned different values later on const does not allow that
- old key word `var coolScore = 25`
- constraints
  - no space
  - Camel Case
  - cant start with number
  - some words are reserved cant name a var const for example.

Comments in JS

- single line
- multiline

Variable types

Variable Type	example
Number	1,2,3..100, 3.14 etc
String	"1,2" "anything"
Boolean	true / false
null	variable has no value yet
undefined	variables that have not been defined yet
Object	Complex structure, arrays, arrays, literals etc
Symbol	used with object

- variable can hold any data type, can also change the data type (loosely typed language)

strings

- name emails and so on letter number and chars in quote
- can use string
- string string as variable
- join two strings l.e concat xsx
- position starts with 0

sting method and properties

- `let name = "Qasim"` length = `name.length` = 5
- method and function
  - `toUpperCase()` // because its a method
  - `toLowerCase()`
  - method do not alter the variable, some do
  - `indexOf` find position of a char
  - `lastIndexOf` last position/occurrence of a char
  - `slice` take in two args wehre we want to slice from and to say 0 to 10
  - `substr` `qasim.substr(0,3) // qasi`
  - `replace` replace a char with another char
    - replace the first string
    - `name.replace("Q", "A") // Aasim`

numbers

- can do all the math operations with numbers

```
let num = 15;
num = num + 5;
num = num - 5;
num = num * 5;
num = num / 5;
num = num % 5;
num = num ** 5;
num = num ++;
num = num --;
num = num + "5";
```

- order of operations `B() I** D/ M* A+ S-`
- short hand notation `+= -= *= /= %= **=`
- NaN - calculation that do not result in number

Template Literals (string)

- Template Literals are string that can contain expressions
  - `let [name, age] = [qasim, 12]`
- for example `let ts = ${name} is ${age} years old`
- use case HTML Template :)

Arrays

- Arrays are a list of values, collection of number, strings and other obects

```
let myArry = ['q', 'a', 's', 'i', 'm']
```

- can use index to access the value
  - `myArry[0]` // `q`
- overwrite values
  - `myArry[0] = 'A'` // `A`
- we can store diffrent types in an array
  -
```
let myArry = ['q', 'a', 's', 'i', 'm', 1, 2, 3, 4, 5, true, false, null, undefined, {}, [], function(){}]
```

array methods

- `length` get the length of the array `myArry.legnth`
- `join` join the array into a string `myArry.join("-")`
- `indexOf` find the position of a value `myArry.indexOf("s")`
- `concat` join two arrays `myArry.concat(['a', 'b', 'c'])`
- some methods alter array value, called the `indistinctive method`
- `push` add a value to the end of the array `myArry.push("a")`
- `pop` remove the last value of the array `myArry.pop()`

null and undefined

- null is an intentional lack of value
- undefined is a value that is not assigned a value
- `let a = null` null is an intentional lack of value 2. null is not the same as 0

Boolean

- boolean is either true or false
- not same as "true" and "false"
- `let name = "Qasim"`
- `name.includes("Q")` //returns true

Comparison Operators

- `let age = 25;`
- `==` equal to `age == 25`
- `!=` not equal to `age != 1`
- `<=` less than or equal to `age <= 25`
- `>=` greater than or equal to `age >= 25`
- `>` greater than `age > 25`
- `<` less than `age < 25`
- lower case letter are greater than upper case letter
- Strict equality `===` value and type are checked for equality no type conversion

Type conversion

- change one data type to another
- `let age = "25"` type conversion `age = Number(age)`
- check type of using `typeof`
- change to a string `age = String(age)`
- change to a boolean `age = Boolean(age)`
  - truthy values are positive number and string that is not empty
- Implicit conversion is what js does behind the scenes

(control)

- conditional statement
  - `if(age < 18) {`
  - `} else if(age < 21) {`
  - `} else {`
  - `}`

(flow) control flow

- a loop to itirate 10 time and log I
  - `for(let I = 0; I < 10; I++) {`
  - `console.log(in loop, I);`
  - `}`
- for array we done know how many elements are there now we can cycle through its length
  - `for(let I = 0; I < myArry.length; I++) {`
  - `console.log(in loop, myArry[i]);`
  - `}`
- Bonus create HTML, for in(key), for of(value)

Functions

- code gets bigger and bigger gets messy
  group section of code together
- `function myFunction() {`
- `}`
- args can be empty or pass values to the function
  - `function myFunction(name, age) {`
  - `}`
- code block
  - `function myFunction() {`
  - `let name = "Qasim";`
  - `let age = 25;`
  - `}`

function delaration vs function expression

- function delaration is when we declare a function and then use it
- function expression is when we declare a function and then assign it to a variable
  - `let myFunction = function() {`
  - `}`
- function declaration are hoisted that is they are loaded before the code is run
- function expression are not hoisted that is they are loaded after the code is run

Scope

- Global scope - variables that are defined outside of a function
- Local Scope - variables that are defined inside a function

this keyword

- who owns the current space (context)
- why would I care about this I am fine with writing same things a million times (I am not I love `this` keyword)

```
let myCar = {
  make: "Ford",
  model: "Civic",
  year: "1964BC",
  color: "red",
  getDescription: function() {
    return `Tyler has Honda  and Michael has American Honda called  ${this.make}  made in ${this.year} model ${this.model}`;
  }
};
```