**Intro to JavaScript**

Full Course Notes

Samuel Chau

A Brief Introduction to JavaScript

* JS is a high level, object oriented, multi-paradigm programming language
  + High level means no memory management
  + OO based on object
  + Multi paradigm means we can use multiple styles of programming
* JS defines the behaviour of HTML and CSS elements
* JS can also be used in addition with Node JS for back-end development

Linking a JS file

* Create a script.js file
* Create script element in body and link the js file as source

  <body>

    <h1>JavaScript Fundamentals – Part 1</h1>

    <script src="script.js"></script>

  </body>

Values and Variables

* We store values (numbers, strings, Booleans, etc) inside variables
* We can then manipulate these variables with other variables or methods
* Variable names cannot start with a number

Data Types

* JS is dynamically typed, meaning you do not need to manually define the type of variable you are assigning. It is done automatically
* Variables can change their type! They can be a number say, then be reassigned to be a string
  + Only use let keyword the first time you assign a variable, not needed when reassigning

let a = true;

console.log(typeof a);

a = "SAM";

console.log(typeof a);

* We can use the *typeof* operator to return the type of a variable

let a = true;

console.log(typeof a);

* Primitive type:
  + Number: always floating point, used for decimals and integers
  + String: sequence of characters, used for text, in quotes, single or double quotes
  + Boolean: true / false
  + Undefined: declaring a variable without a value. Both the variable and the value are undefined
  + Null: empty value, similar to undefined but used in different scenarios
  + Symbol: Unique and cannot be changed
  + BigInt: numbers too large for the number type

let age = 23; // number

let firstName = "Jonas"; // string

let fullAge = true; // Boolean

let children; // undefined

Let, Const and Var

* Use let to declare variables that can be changed later
* Use const to declare variables that cannot be changed later
  + Can’t have undefined const variables
* Never use var, works like let in legacy JS

Basic Operators

* Arithmetic operators
  + Addition, subtraction, division, multiplication, powers, addition can be used to concatenate string
  + Power (2 \*\* 3 is 2 to the power 3)
* Typeof operator
  + Returns the type of variable
  + Let a = true, typeof a *boolean*
* Assignment operators
  + Letting a variable equal a value
  + X=10
  + X +=10 is X=X+10, can be used with \*=, -= and /= too
  + X++ is X=X+1, X—is X=X-1

* Comparison operators
  + Greater than / equal to, less than / equal to
    - >, >=, <, <=
  + Entirely equal to
    - ===

Operator Precedence

* JS will always compute mathematical operators before comparison operators
* Some operators occur left to right or right to left
  + Assignment occurs right to left to allow for mathematical and assignment operators to occur beforehand.
  + This allows dynamic typing; JS will always know what the variable will be because it reads it from right to left before assigning it
  + This means you can use the assignment operator more than once in a single line
    - X = Y = 10
      * Y = 10, then X=Y
* Follows BODMAS, Brackets over Division, Multiplication, Addition and Subtraction

Strings and Template Literals

* It can be a pain to concatenate strings and variables together

const sam =

  "I'm " + firstName + ", a " + (year - birthYear) + " year old " + job;

* Instead, we should use template literals

const samNew = `I'm ${firstName}, a ${year - birthYear} year old ${job}`;

// dollar sign + curly braces contain variables to be concatenated into a string

* We can use back ticks in place of quotation marks for regular strings too, therefore, we should use backticks for all strings as it can be used for every scenario
* Use \n for new line, or just create a new line in the code with backticks
  + The following code will output this on multiple lines, this is useful when creating HTML code

console.log(`yo

wassup

my

name

is

sam`);

If / Else Statements

* Check whether Boolean is true or false, if so, execute following code in curly braces
* Else, do something else, or do nothing if otherwise stated
* Can also include another if statement in else statements  *if… else if …. else*
* Do not declare new variables inside if blocks, instead declare it outside and then reassign its value in the if statement
  + This is because the variable will only exist only within the scope of the if statement and can therefore not be seen outside of it

const age = 1;

if (age === 19) {

  console.log("yeah im 19");

} else {

  console.log("no im not 19");

} // console will output “no im not 19”

Type Conversion and Coercion

* Convert strings to numbers using the Number(*string*) function, numbers to strings using String(*number*)
  + Functions start with a capital letter

const inputYear = "1991";

Number(inputYear);

* Type coercion occurs when combining different variables in outputs without converting them manually
  + Occurs during template literals
  + Adding different variables in template literals will concatenate them as strings, whereas subtracting them will concatenate them as numbers
  + Read from left to right, depending on the last operation, can be either string or number

Truthy and Falsy Values

* Falsy values are not entirely false, but will be converted to false when converted into a Boolean type
* 0, undefined, NaN, null and empty strings ‘’ are falsy values
* The following are all converted to false

console.log(Boolean(0));

console.log(Boolean(undefined));

console.log(Boolean(NaN));

console.log(Boolean(null));

console.log(Boolean(""));

* Any number or string that is not 0 / empty, is a truthy value and will be converted to true when converted into a Boolean type
* Empty objects are also truthy ({})
* The following are converted to true

console.log(Boolean(1));

console.log(Boolean("a"));

* Though, this is never necessary to do explicitly, creating booleans is only ever done through type coercion with logical operators >,< and =

const money = 0;

if (!money) {

  console.log(`ya broke`);

}

* Type coercion is used to convert *money* into a boolean to be checked in the if statement
* ! is used to check *if NOT*
  + if (!money) is the same is saying *if (money != true)*
  + if (money) is the same is saying *if (money === true)*

Equality Operators

* we can check if two values are equal by using triple ===
  + can also use == for type coercion, comparing string 18 to number 18 will return true
  + === does not have type coercion

const age = 1;

if (age === 18) {

  console.log("is 18");

} else {

  console.log("not 18");

}

* we can check if two values are not equal by using !== or !=

const age = 1;

if (age != 18) {

  console.log("is 18");

} else {

  console.log("not 18");

}

Boolean Logic

* Combining boolean values and checking true / false collectively
* A is true and B is true, or A is true and B is not true, or A is true OR B is true.

Logical Operators

* && means AND
* || means OR
* ! means NOT

const hasLicense = false;

const hasVis = true;

console.log(hasLicense && hasVis); // checks if both are true, returns false

console.log(hasLicense || hasVis); // checks if either is true, returns true

Switch Statements

* Easier way to write long if else statement
* Can be used with EVERY type, not just numbers and Booleans
* Make sure to add breaks to every case so that the switch statement does not iterate through the subsequent cases too
* Can combine cases by not including break and writing the code in the last case
* If all cases fail, the default case executes

switch (day) {

  case 1:

    console.log(`monday`);

    break;

  case 2:

    console.log(`tuesday`);

    break;

  default:

    console.log(`a day`);

}

Statements and Expression

* An expression is a piece of code that produces a value
* A statement is a piece of code that does NOT produce a value, they verbalise code behaviour
  + If / else, for, while, etc

Conditional (Ternary) Operators

* Can use ? and : in place of if / else statement

const age = 18;

const drink = age >= 18 ? `Wine` : `Water`;

* Write a condition, then use ? in front of behavior for if true. **:** in front of behaviour for else / if
* This allows you to declare variables in the if / else block and keep the boolean within scope later
* Allows to write if / else statements within other statements such as template literals

console.log(`I like to drink ${age >= 18 ? `Wine` : `Water`}`);

Ternary Example

function tip(price) {

  return price >= 50 && price <= 300 ? price \* 0.15 : price \* 0.2;

}

function total(price) {

  console.log(

    `The bill was ${price}, the tip was ${tip(price)}, and the total value ${

      tip(price) + price

    }`

  );

}

const p1 = 275;

const p2 = 40;

const p3 = 430;

total(p1);

* The tip function says:
  + Return price\*0.15 IF 50 <= price <= 300 ELSE, return price\*0.2

Strict Mode

* Catches errors before run time
* Should ALWAYS use strict mode

"use strict";

* Goes at the start of every .JS file
* Catches wrong variable names, undeclared variables, etc

Functions

* Functions are segments of code that can be reused over and over to repeat a process
* You pass variables into the function, in which they can be updated and changed, then return a variable from the function to be used within the main scope of the program
* Call a function by using it’s name with parenthesis and any required input parameters
* Can be called over and over with different parameters
* When a function returns, any following code will NOT execute. This also happens when ANY statement breaks

function juicer(a, o) {

  console.log(a, o);

  const juice = `Juice with ${a} apples, and ${o} oranges`;

  return juice;

}

console.log(juicer(1, 2));

console.log(juicer(5, 6));

Function Declarations vs Expressions

* Declaration

function ageCalc(bYear) {

  return 2022 - bYear;

}

console.log(`I was born in ${ageCalc(21)}`)

* Expression

const ageCalc2 = function (bYear) {

  return 2022 - bYear;

};

console.log(`I was born in ${ageCalc2(21)}`);

* They work the same way, are just written differently
* Both useful in different situations

Arrow Functions

* Quicker way of writing functions

const calcAge = (birthYear) => 2022 - birthYear;

// parentheses are parameters, RHS is return value

console.log(calcAge(21));

const yearsUnitlRetirement = (birthYear, name) => {

  const age = 2022 - birthYear;

  return `${name} will retire in ${65 - age} years`;

};

// if multiple lines of code, curly braces and return statement needed

console.log(yearsUnitlRetirement(2001, "Sam"));

Functions calling other Functions

* Functions can call other functions and use their data within their own scope

function cut(fruit) {

  switch (fruit) {

    case "apple":

      return 3;

    case "banana":

      return 1;

    case "orange":

      return 6;

    default:

      return 4;

  }

}

function juicer(fruit1, fruit2) {

  const a = cut(fruit1);

// call the cut function with the parameter fruit to be used as another parameter

  const b = cut(fruit2);

  if (fruit1 === fruit2) {

    console.log(`You made a smoothie with ${a + b} pieces of ${fruit1}`);

  } else {

    console.log(

      `You made a smoothie with ${a} pieces of ${fruit1} and ${b} pieces of ${fruit2}`

    );

  }

}

juicer("apple", "kiwi");

Function Final Example

function calcAge(birthYear) {

  return 2022 - birthYear;

}

function retirement(age) {

  return 65 - age;

}

function yearsUnitlRetirement(birthYear, firstName) {

  return retirement(calcAge(birthYear)) > 0

    ? `${firstName} retires in ${retirement(calcAge(birthYear))} years`

    : `You're old`;

}

console.log(yearsUnitlRetirement(2001, "sam"));

Arrays

* Arrays are data structures that contain a list of variables
* Arrays are NOT constricted to hold a single type of variable and can hold any variable in any position
* Can be declared using [], or by using new Array(*elements*)

const friends = [a, "b", NaN, undefined, null];

const years = new Array(1, 2, 3);

* Access an element inside an array using array[x], where x is the index of the element you are trying to access
  + Array indexing starts at 0
  + You can also update these access elements

console.log(years[1]); // will print the element in the SECOND position of the array

years[2] = 5; // update the 3rd element in the array to 5

* You can find the length of an array using array.length
  + Note, length is not the final index, but rather how many literal variables are inside the array, n+1
  + Knowing the length of the array is useful in finding the last indexed variable = length -1
    - This allows backwards iteration in for loops, etc

console.log(friends.length);

* Arrays cannot be ENTIRELY replaced by a different array, only specific elements one at a time
* Cannot pass entire arrays into functions, rather only a single element from the array
  + To fix this, use a for loop to iterate through the array and call the function for each of the elements by iterating through it 1 element at a time

function calcAge(age) {

  return 2022 - age;

}

const years = [1997, 1998, 1999, 2000, 2001, 2006, 2015, 2022];

for (let x = 0; x < years.length; x++) {

  console.log(calcAge(years[x]));

}