THE UNIVERSITY OF SYDNEY

School of Computer Science

COMP5347: Web Application Development

Tutorial Week 1: Introduction to JavaScript

Learning Objectives:

- Understand the core concepts of JavaScript or JS
- Understand JS variables, conditions, loop, functions and class
- Usage of arithmetic operators in JS, comments
- Hands-on practice by writing JS code

Before we start – What will we need

- We recommend using Chrome browser, but other browsers work too.
- Any choice of text editor/IDE. Visual Studio Code or VS Code is recommended. Notepad++, Eclipse, Atom, Sublime Text, Vim etc can be used as well.

How to do this tutorial:

- This tutorial is read-heavy which means there is a lot of contents for understanding the background or brushing up your memory (if you have programmed in JS before). The reading contents are within "grey" box, it's optional but highly recommended.
- Each task consists of very tiny programs, everyone learns differently. However, statistics tell us programmers learn well by doing, so instead of just copy/paste, write the program by typing the code blocks. You will see your mistakes this way almost immediately.
- We typically save JS files with the extension .js . But if you want to see the output of your JS code, you will need to save the file with .html extension and put all your code within <script></script> tags.
- Green lines are comments, you don't need to write copy/write comments. This is to help you understand.
- To see any output of a variable, write console.log(variable_name); We skipped writing the same thing to avoid repetitions. So, where it says

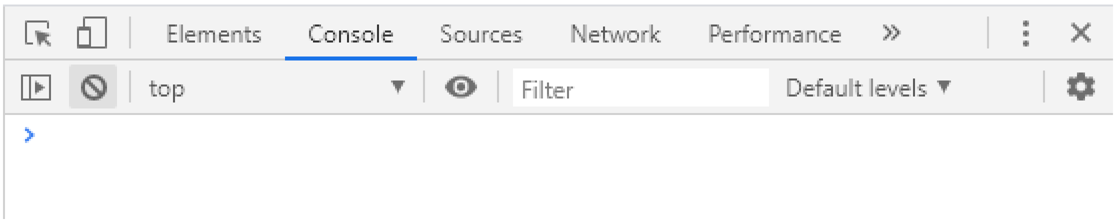
```
var x = 5+2; //output 7
```

You need an additional line of console.log(x); to see the output.

- There's a Challenge at the very end of the tasks. Please try to do the challenge, if you cannot finish it in the tutorial time, continue to do it at home.

Getting started – the quicker version:

- Open a browser window in Chrome
    - On an empty space, right click and select "Inspect"
        - Click on "Console" in the newly opened side bar, which should look like the following.



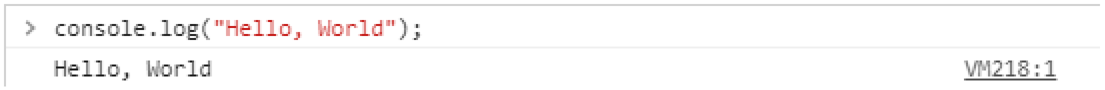
JS code can be written in this console panel. All the browsers can execute JS code on the fly. This is a good way to test a piece of code before committing to it.

Task – 1: The Infamous "Hello, World"

In the console panel, write the following code and press Enter

➢ `console.log("Hello, World");`

The output will be displayed in the next line:



```
> console.log("Hello, World");
  Hello, World                                                VM218:1
```

Don't worry about 'VM218:1'. Just for the record, Chrome uses V8 Engine, VM here is Virtual Machine, each line has its own line number.

Congratulations! You've just written your first JS code.

JavaScript is high-level interpreted type multi-paradigm programming language.

- *High-level: Programming language with strong abstractions from the details of a computer*
- *Interpreted-type: Doesn't need to compile code into machine instructions, i.e. interprets line-by-line*
- *Multi-paradigm: There are four programming paradigms – 1. functional 2. Imperative/Procedural*

3. Logical 4. Object Oriented

Good-to-know: JavaScript or JS is NOT Java. Java is Object Oriented Programming language. They have similar names because of a then marketing deal between Netscape (Originator of JS) and Sun Microsystems (Java).

Task – 2: Freedom of ~~speech~~ comment

Definition of Comment: Comments are used in code to make the program more readable. In enterprise-level programs, there are thousands of variables/functions. Without having appropriate comments, the code can become really hard to understand. Comments are ignore by the compiler/interpreter. So, whatever makes the program make more readable can go inside the comment.

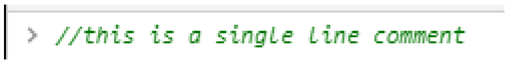
Comments can be single-line or multi-line.

In JavaScript single line comments are made using two forward slash "//". Anything between // and the end of the line will be ignored.

Multi-line comments are made using /* */. Anything between /*   */ will be ignored.

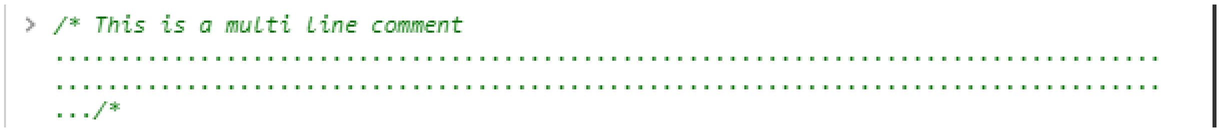
In the browser console, try to enter these two types of comments

Single-line Comment



```
> //this is a single line comment
```

Multi-line Comments



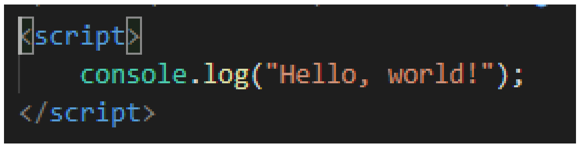
```
> /* This is a multi line comment
  ...............................................................................
  ...............................................................................
  .../*
```

Comments are not interpreted, so you can pretty much write whatever you want. Don't worry if the output is undefined. That's because the JS engine doesn't know what you're writing or simply put, we have not defined anything. We will talk understand "undefined" when we discuss variables.

Task – 3: Save the JS!

Fire up your favorite text editor, if you do not want to install one, you can just use Notepad or Emac/Vim etc whatever comes default in your OS. You might be asking, why do we need text editor if we can just run the code in the browser shell? You probably have noticed; it is not a practical way to write JS code for an application. Specially, you will be (or maybe are already) working in large applications, where the codebase will grow into millions of lines. Writing code in the browser is not maintainable.

Quick Tip: We typically save JS files with the extension .js . But if you want to see the output of your JS code, you will need to save the file with .html extension and put all your code within <script></script> tags. We will learn more on this in the next tutorial.



```
<script>
    console.log("Hello, world!");
</script>
```

If you save your code with .html tag, and open it with any browser, in the console panel (right click empty space and then inspect and then console) will display any console.log() in the console panel.

Task – 4: Variables

Before we discuss different variable types in JS, we need to understand JavaScript is essentially dynamically typed. Which means, one variable can have different kinds of values in them. For e.g. write the following in browser console. You do not need to write the part in comments.

- `var x = 12; // now the value of x is 12`
- `var x = “hello”; // now the value of x has changed from 12 to hello`
- `var x = 12 + “hello” // the output will be: 12hello`

Tip: + is used to concatenate or add two objects or things. Here, we can see a number and text has been paired up together using +. If we write

```
var x = 12 + 8;
```

The output will be 20 and NOT 128 because 12 and 8 are both numbers and NOT mixed types. JavaScript automatically understands different types of variables. Hence, it is called Dynamically Typed.

Don’t forget, you need to write console.log(x); to see the value of x.

Definition: A variable is a name storage for data. If we define x = 12; every time we call x, it will return 12. So, “x” is our variable.

In JavaScript, there are several variable types and there are three common ways to define variables.

How to define variables in JavaScript?

- var x = 12;
  - This is globally scoped or function locally scoped (more on this later)
- let x = 12;
  - Block scoped. Gained popularity after the ES6 standard (more on this later)
- const x = 12;

- Block scoped. Cannot be updated/re-defined.

Task – 5: Data Types

Data Types, sometimes referred to as Variable Types are the kind of values the variable holds.

There are 7 Basic data types in JavaScript.

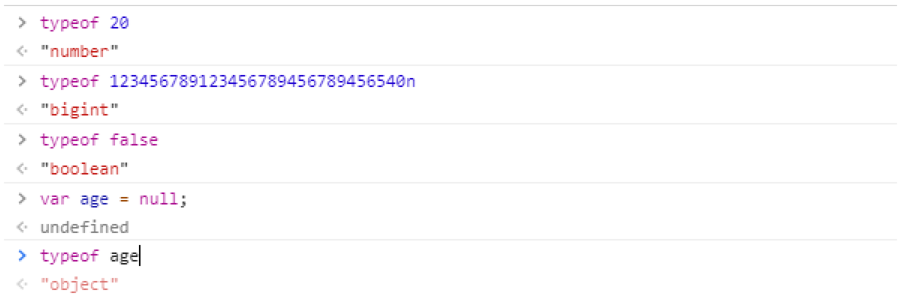
- number: For any integers or floating-point (decimals). Limited by $\pm 2^{53}$
- bigint: for integer greater than number. Usually used in crypto/multi-second precise timestamps
- string: for texts or characters. Text are defined using single/double quotes ‘’ or “”. e.g. “hello”
- boolean: true/false. true means correct, false means incorrect
- null: For value not known
- undefined: For unassigned values. We almost never assign “undefined” to variables. For empty values use “null” instead.
- object: For complex data structures, can be number/text/boolean or anything (more on this later)

There is also “symbol” type. But not very commonly used.

Type the followings to get a feel for different data types in JavaScript

```
var x = 100; //number

var x = 50.4; //number

var x = 123456789123456789123456789154687945 60n; // bigint

var x = 55 > 5; //boolean true

var age = null; // null type object i.e. age is empty.

var age; // undefined
```

Tip: Because JavaScript is dynamically typed, it’s not always clear to understand the data type. We can use “**typeof**” to get the type of data.



```
> typeof 20
<· "number"
> typeof 123456789123456789456789456540n
<· "bigint"
> typeof false
<· "boolean"
> var age = null;
<· undefined
> typeof age
<· "object"
```

Task – 6: Arithmetic and Logical Operations

Type the following to see how we perform arithmetic operations.

```
Addition: var x = 2 + 5; //output: 7

Subtraction: var x = 5 – 2; //output: 3

Multiplication: var x = 5 * 2; //output 10

Division: var x = 1 / 2; //output 0.5

Remainder: var x = 12 % 5; //output 2

Exponential: var x = 2**2; //output 4
```

There are other types Increment (++), Decrement (--), Unary Plus (+x), Unary Negation (-x). Read more here: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators

We use '=' to assign a value to a variable. For e.g. var x = 2; 2 is assigned to x.

We use '==' to check if the value of left hand side is equal to the value of right hand side. For e.g. 2 == 2; This outputs 2, because both of them are equal.

We use '===' to check for both value and the data type. For e.g. 2 === '2'; This outputs false because left hand side is a number and right hand side is a text.

We can also use '>' for greater than, '<' for less than, '<=' for less than or equal and '>=' for greater than or equal. e.g. x >= 30; It means x can be greater than or equal to 30.

In addition, we can use logical operators to perform logical operations. E.g.

- && for logical AND. e.g. `3 && false; //output: false`
- || for logical OR e.g. `3 || false; //output: true`
- ! for logical NOT e.g. `3 && !3; //output: false`

Task – 7: Conditions

Conditions are very similar to a real life scenario. If it rains, carry an umbrella. We can write similar things in JS like so:

```
if (rains) {
        carry umbrella;
}
```

Of course, the example doesn't mean anything for computers. But if conveys an interesting message. "Rains" here is the check, and anything within {} will be executed if that is true. The structure is

```
if ( condition ) {
         // follow the instructions
}
var age = 30;
if ( age > 30 ) {
      var y = 4;
}
```

Explanation: Here we're writing a silly program, where we define an age, and assign a value 30. Next, we check if age is greater than 30, if it is, then we define y and assign 4 in to y.

We can extend the same logic and use "else if" or "else". We can replace if-else if-else block with switch when the condition gets very big.

Write the following and save it in a file. Remember, if you would like to see the output, put it within <script></script> tag and save as .html.

```
var grade = 70;
if ( grade >= 85 ) {
    console.log("You got HD");
}
else if ( grade >= 75 ) {
    console.log("You got D");
}
else if ( grade >= 65 ) {
    console.log("You got C");
}
else {
    console.log("You have failed :-( ");
}
```

Task – 8: Switch

Switch is similar to if-else, the format is:

```
var expression = “something”;
switch ( expression ) {
        case “Condition 1”:
              // Do something
              break;
         case “Condition 2”:
             // Do another thing
             break;
         default: //default is similar to else
              // Do something different
}
```

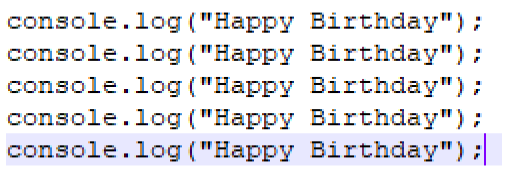
Take note of “break” here. If break is not used, all the cases will be executed regardless of the conditions.

Like before, save it in a file and run it

```
var grade = 70;
switch ( true ) {
    case grade >= 85:
        console.log(“You got HD”);
         break; // Take note of break!
    case grade >= 75:
        console.log(“You got D”);
         break;
     case grade >= 65:
        console.log(“You got C”);
        break;
     default:
        console.log(“You have failed :-( “);
}
```

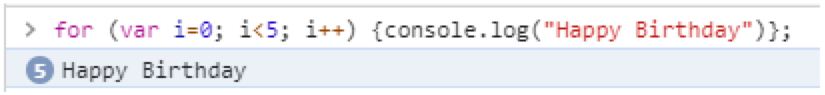
Task – 9: Loops

There are many times in programming we need to do the same tasks over and over again. Say, I need to say “Happy Birthday” to you 5 times. The most obvious (and silly) way to accomplish that is:



```
console.log("Happy Birthday");
console.log("Happy Birthday");
console.log("Happy Birthday");
console.log("Happy Birthday");
console.log("Happy Birthday");
```

But we’re programmers, a programmer would simply do it this way:



```
> for (var i=0; i<5; i++) {console.log("Happy Birthday")};
5 Happy Birthday
```

The number 5 means the same thing has been repeated 5 times.

Loops are the most common way to iterate over something. The most common type of loop is for loop. In a for loop, we need a variable with an initial value, we need a condition to define when the loop will terminate and an increment to define how many steps to take in each iterations.

E.g.

```
for ( initial_value, condition, increment ) { do something; }

for ( var i = 0; i < 5; i++) {

        console.log(i); //outputs the value of i

}
```

Here, ++ is used (remember arithmetic operator section?) to increase the value of i by 1 each time. It is the short hand version of writing i = i + 1; We can use “--” in a similar fashion.

There are other types of loops, while, do..while etc. For loop can be used to accomplish the same tasks and vice versa. “for” is the most commonly used.

To know more about loops, read more here: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

Task – 10: Break and Continue

Break is used inside a loop to “break” the loop if a condition is met, while continue is the opposite, the loop will skip over and continue if certain conditions are met. Write the following -

```
for ( var i = 0; i < 10; i++ ){
        if ( i = 5 ) {
             continue;
             console.log(“Hello”); //this line won’t be executed because we skipped
        }
        if ( i == 8 ) {
             break;// loop terminates, so “i” will never reach the value 9.
        }
    }
```

Task - 11: Data Structure (Array)

Last section when we discussed data types, we discussed mostly the primitive data types. There are two other data types in JavaScript known as “Array” and “Object”. Primitive data types are useful to contain only single type of data. For e.g. var x = 2; When we write x = 4, then x is not 2 anymore. What if we want to contain both 2 and 4 in x?

Array is a container to hold multiple values. We define array in JavaScript using []. For e.g.

```
    var x = [2, 4] //we now use x to hold both values
```

Here, x has two elements 2 and 4. 2 is in the 0th index, and 4 is in the 1st.

Tip: In programming, we iterate from 0.

Task – 12: Data Structure (Objects)

We can define an arrays as:

```
var x = [25, ‘Michael’, 90251, ‘Student’];
```

We don’t have any clue what these values really mean. To fix that problem JavaScript gave us another type called “Object”. Object has key/value pair to hold multiple elements. If we convert the above array into an object, it makes much more sense:

```
    var x = {
               “age”  : 25,
               “name” : “Michael”,
               “id”   : 90251,
               “type” : “Student”
            };
```

For Objects, we use {}

Task – 13: Functions

Functions are subprograms used to perform a certain task.

Functions can take values known as parameters.

In JavaScript, function always returns “something” if no return is defined, it returns “undefined”.

For example, if we a design a calculator, it can have the basic addition, subtraction, multiplication and division function.

Functions are defined using the “function” keyword.

Function returns values can be determined by using “return” keyword.

Function calls are usually made outside of that corresponding function known as invoking function. Recursive functions can call themselves.

```
function addition( val1,  val2 ){

    return val1 + val2;

}

function subtraction( val1, val2) {

    return val1 - val2;

}

addition(5, 10);

//Invoking addition function but not subtraction. Output 15
```

Task 14: Scope

Scope in JavaScript is defined using {} - curly braces. Anything outside of {} is outside of the scope/global scope. Anything gets defined in the global scope is accessible by all the child elements.

Example-1:

```
var x = 5; //global scope

function eg(){
```

```
            console.log(x); //output: 5
        }
```

Example-2:

```
        function eg(){
            let x = 5; //block scoped, not accessible outside of this function
        }
        console.log(x); //output: undefined.
```

Challenge

Find the mean, median, mode and range from the series of following numbers

23,9,14,2,28,19,3,15,9,25,2,4,9

Tips: You can use the following format as a guide

```
var inputArr = [ 23, 9, 14, 2, 28, 19, 3, 15, 9, 25, 2, 4, 9 ]
function mean( inputArr ){
        // your code
}
function median( inputArr ){
        // your code
}
function mode( inputArr ){
        // your code
}
mean( inputArr ); //invoking mean function
median( inputArr ); //invoking median function
mode( inputArr ); //invoking mean function
```

Answer:

Mean: **12.461538461538**

Median: **9**

`Mode:` **9**

Reading 1:

If function perform a certain task, Class on the other hand performs a set of tasks or invokes a set of functions. Classes are the blueprint of a design. For the previous example, we can have a calculator class which has all of the addition, subtraction, division and multiplication etc functions.

We will not dive too deep in to classes in this lecture. To understand more about class see here: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

Reading 2:

In JavaScript, variable, function and class names starts with string, names can contain number and underscore "_"

It is quite common to use camelCase in JavaScript names. e.g. var circleRadius = 5;

Class name usually starts with uppercase letters. e.g. class Rectangle()

Common naming conventions:

- var person_age
- var personAge
- var person1Age
- var PersonAge
- var Person_Age

JavaScript resources:

https://developer.mozilla.org/en-US/docs/Web/JavaScript

https://www.w3schools.com/js/