Javascript 1

COSC3058 - Web Programming Bootcamp

Review Slides

Tom Huynh
School of Science, Engineering & Technology
RMIT University Vietnam

RMIT UNIVERSITY

Previously, we have learned:

- Display (inline, block, inline-block)
- Position (static, relative, absolute, fixed)
- Transition
- Transform
- Flexbox
- Media Query
- CSS Showcases
- Colour Theory [Extra Topic]
- Font Theory [Extra Topic]

Agenda Today:

- Javascript
- Run JS code directly on browser
- Attach JS to HTML document
- JS components
  - Logging and Alerting
  - Comments
  - Name Convention
  - Variables
  - Data Types
  - Operators
  - Control Structures
  - Functions
- JavaScript Style Guide

CES - Course Experience Survey

CES - Course Experience Survey

- **Please do it as soon as possible!**
- It is very important for you, me and RMIT as it is one of the best way to provide feedback for the course.
- **Please mention things you like so I will try to make it happen more often in the future courses or any of the course I will teach in the future semesters.**
- **Also, mention things you think it could be improved or changed so I will try my best to improve upon that!**
- I would like to **consider the survey is a way for us to build a long-term relationship as a lecturer and students** for many years to come!
- **Please go to this link: https://surveys.rmit.edu.au**

Javascript

HTML vs CSS vs Javascript

HTML

+

CSS

+

Javascript

=

Front-end Web Development

- The front-end of the web is based on three major technologies:

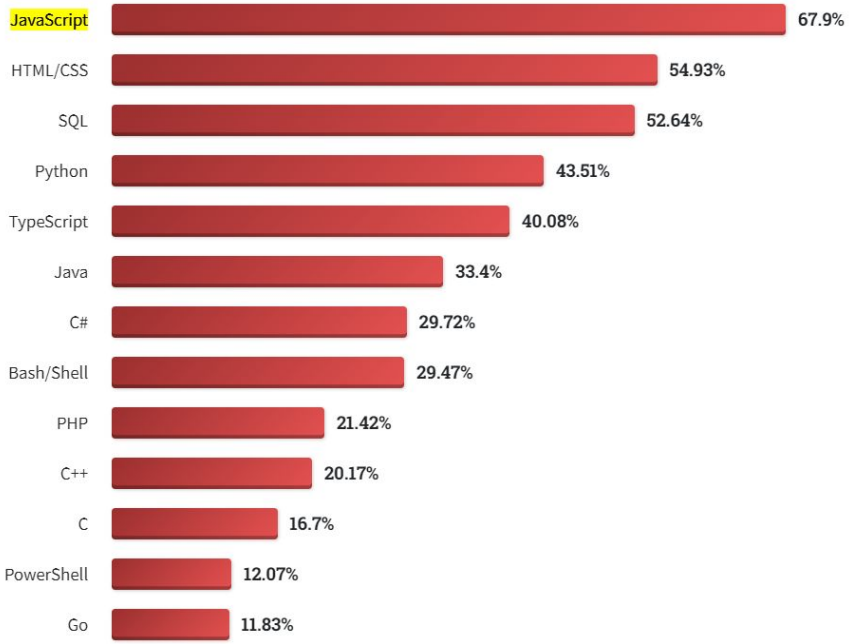
- **HTML** as "Content & Structure":
  - HyperText Markup Language (HTML) defines the structure, content and semantics of web pages on the web.

- **CSS** as "Presentation & Styling":
  - Cascading Style Sheets (CSS) sets the look and style of a web page.
  - CSS provides style to the structure provided by HTML.

- **JavaScript (JS)** as "Interaction & Behavior":
  - JavaScript allows us to define interaction in our pages.
  - What happens when a user clicks on a certain area?

Javascript is the King! 👑

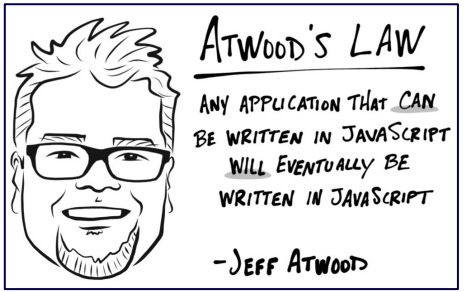
- JavaScript is the most popular programming language for professionals for 10 year in a row.



https://survey.stackoverflow.co/2022/

Atwood's Law about JavaScript

- **Atwood's Law**: "**Any application that can be written in JavaScript, will eventually be written in JavaScript.**" - Jeff Atwood, 2007
- **Jeff Atwood** is a software engineer, entrepreneur, and co-founder of the popular programming Q&A website **Stack Overflow.**
- The law is often interpreted to mean that **JavaScript has become so dominant** in the world of web development that it is now used for an ever-widening variety of applications, **not just limited to web-based tasks**. This includes **server-side applications**, **mobile app development** (using technologies like React Native and Ionic), **desktop applications** (using Electron), **Internet of Things** (IoT) devices, and even **robotics**.

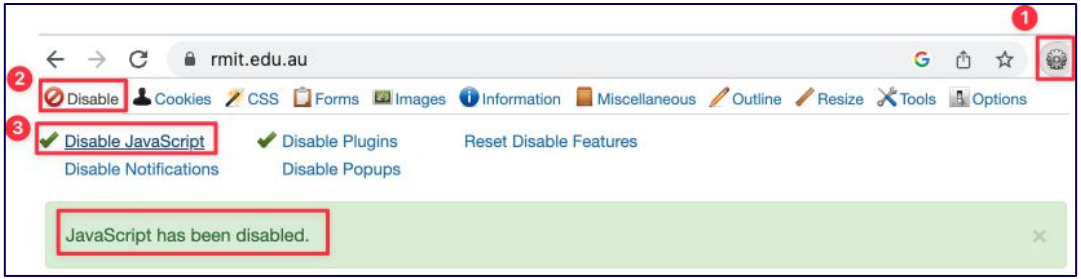


JS Library & Frameworks

- One language to rule them all!
- Anything you want and can imagine then there is a javascript library/framework for it!

Modern Websites without Javascript

[Task] Let's check out some modern websites without Javascript (JS)? Try it yourself! With “Web Developer” extension, try to disable JS like so:



- Now check all these websites without JS enabled:
  - Ecommerce Lazada
  - Ecommerce Shopee
  - Youtube
  - Tiktok
  - Cryptowatch
  - Any of your favourite websites.

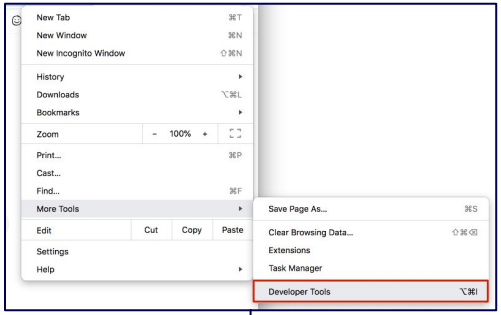
Javascript in Action

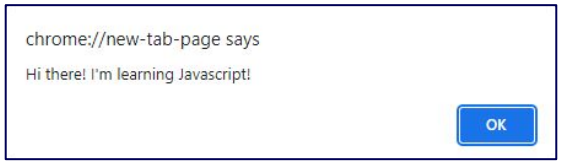
- Check out these JS projects:
  - Simple Australian Map Quiz
  - Airplane Guide
  - Snake Game
  - Pong Game
  - Tower Block
  - Frantic run of the valorous rabbit
  - 3D Rubik Cube
  - Chess Game
- All of these are possible because Javascript allows us to program every part of the web pages.

Run JS code quickly and directly on the browser

Option 1: Run JS code in the console

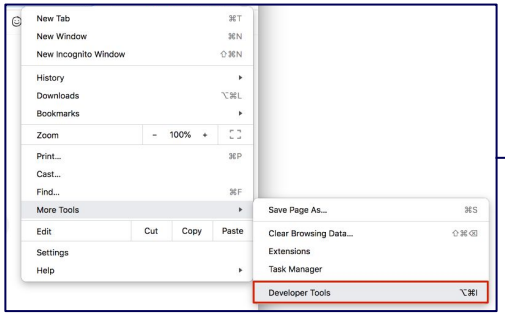
- This option allows you to run JS code directly on a website line by line and see the output for each line interactively!

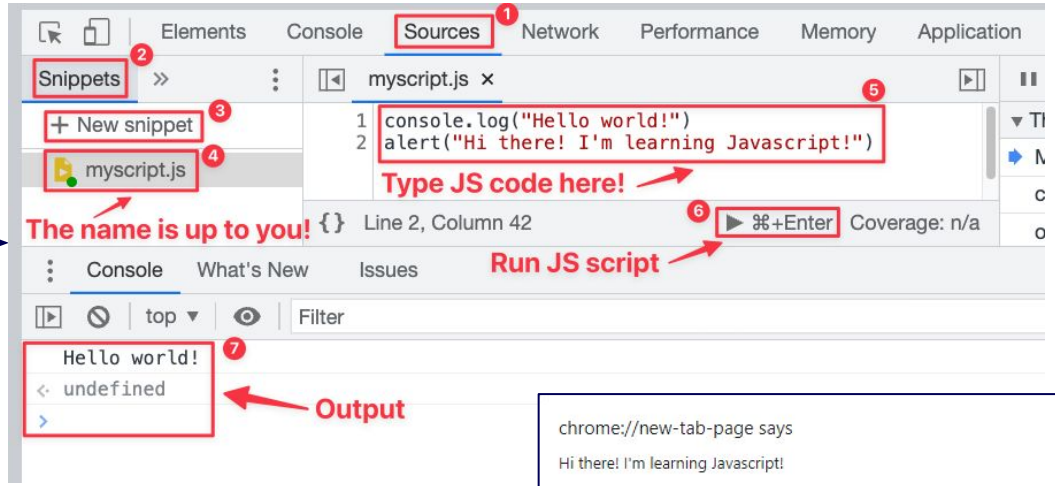




Option 2: Run JS code as a snippet file

- This option allows you to type js code in a js script file and run it on a website in one go and see the output for the script in the console!





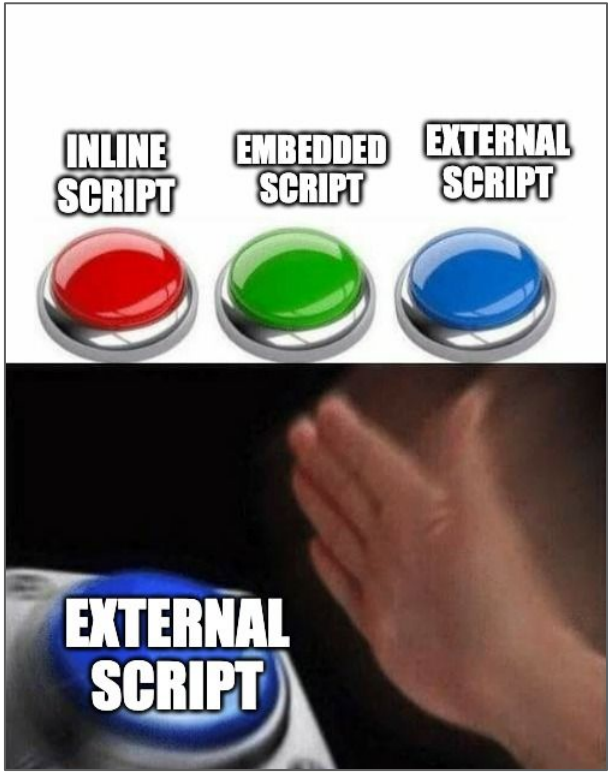
Beware! 🚩

- Both option 1 and 2 are **not permanently** so **when you refresh the website, you will lose all the work you are working on for those js code.**
- **So these options are only suitable to play around and test some js code quickly on the websites!**
- We will talk more about better ways to permanently attach js code to a HTML document.

Attach JS to HTML document

Attach CSS styles to HTML document

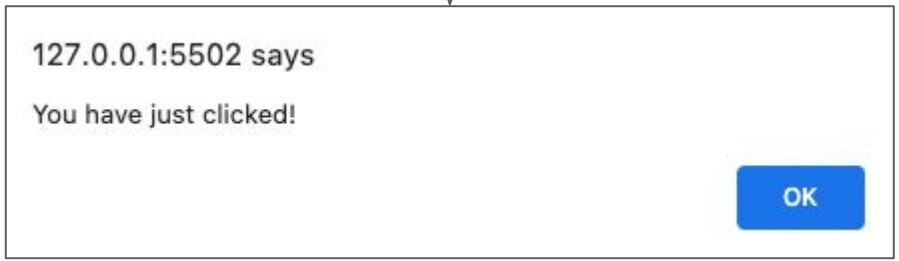
- Similar to CSS, there are three ways that JS script can be applied to an HTML document permanently:
  1. **Inline script**: applies to one HTML element. (Please, avoid it!)
  2. **Embedded script**: applies to one HTML page only.
  3. **External script** applies to many HTML pages. (The best way!)



Inline script

- This applies the JS code directly to the HTML element, in this case, the paragraph element.
- 🚩 **Inline script should be avoided.** It is generally discouraged because it can make the HTML code harder to read and maintain.
- The demo link is here.

```
<button onclick="alert('You have just clicked!')">Click me</button>
```



Embedded script

- This places the JS code in the end of body section of the HTML page, making it accessible to all elements on the page.
- It only applies to the current HTML page and not to other pages on the website.
- 🚩 The script could be in the body or in the head tag. But **we recommend placing it at the end of body of the document to when the script is loaded, it can access any HTML element.**
- The demo link is here.

```
<body>
  <button onclick="clickTrigger()">Click me</button>

  <script>
    function clickTrigger() {
      alert('You have just clicked!');
    }
  </script>
</body>
```

External script

- This links the JS code in an external file called **script.js** and applies it to the current HTML page.
- The same external script file can be used on multiple pages throughout the website, making it more efficient and easier to maintain.
- 🚩 The script could be in the end of the body or in the head tag. But **we recommend placing it at the end of the body of the document to when the script is loaded, it can access any HTML element.**
- The demo link is here.

```
<body>
  <h1>External script example</h1>
  <button onclick="clickTrigger()">Click me</button>

  <script src="script.js"></script>
</body>
```

script.js

```
function clickTrigger() {
    alert('You have just clicked!');
}
```

JS components

Learning Javascript

- Since we are all familiar with other languages like Python and Java so learning Javascript (JS) is very easy.
- There are a few basic concepts you need to remember. They act like building blocks. To build a tall tower, you first put a block on top of another. Here are some of the essential programming building blocks of Javascript:

- **Variables** to store data (aka state) during your program's execution.
- **Operators** to perform actions on.
- **Conditionals** like if statements to make decisions.
- **Loops** to repeat tasks until a condition stops being true.
- **Functions** to organize your code into logical and reusable chunks.

Logging and Alerting

Logging and Alerting

- **console.log()** and **alert()** are two commonly used functions for displaying information to the user. However, they differ in their functionality and usage.
- **console.log():** a method used to **output messages** to the **browser's console,** which is **not visible to the end-user.**
  - This makes it useful for **debugging purposes**, as it **allows developers to see what's happening behind the scenes and track down issues with their code.**
- **alert()**: a method used to **display a message box to the user**. When called, an **alert message pops up** in the user's browser window, and **the user must click on the OK button to close it.**
  - This method is useful for providing **immediate feedback to the user** and can be used to **display important information or error messages to the user.**
- The demo links are: logging example, and alerting example.

```
let x = 5;
let y = 10;
let sum = x + y;
console.log("The sum of " + x +
" and " + y + " is " + sum);
```

```
let age = prompt("What is your age?");
if (age >= 18) {
    alert("You are eligible to
    vote!");
} else {
    alert("You are not eligible to
    vote yet. Please come back when
    you are 18 or older.");
}
```

Comments

Comments

- Use Comments, they make your programs readable.
- They can appear anywhere in a program.
- Single line comment:

```
// This is a comment on one line or the end of a line
```

- Multiple line comment:

```
/* This is a comment
   on more than
    one line */
```

- Example:

```
// Output hello world string
console.log('Hello World');

/* a function
to add
two numbers */
function add(a, b){
    return a + b;
}
```

Name Convention

Identifiers

- Identifiers are the names of your variables, functions, arrays and objects. They must start with a letter or underscore and can contain letters and digits 0–9.
- Identifiers should be meaningful.
- This is a bad example:

```
let a1 = 5;
let array4Something = ['Morning','Afternoon','Evening'];
function x3(a) {
    return a*3;
}
```

- This is a good example:

```
let numAttempts = 5;
let daySegments = ['Morning','Afternoon','Evening'];
function multipleByThree(a) {
    return a*3;
}
```

Name Convention Options

snake_case

Pros: Concise when it consists of a few words.

Cons: Redundant as hell when it gets longer.

```
push_something_to_first_queue, pop_what, get_whatever…
```

PascalCase

Pros: Seems neat.

```
GetItem, SetItem, Convert, ...
```

Cons: Barely used. (why?)

camelCase

Pros: Widely used in the programmer community.

Cons: Looks ugly when a few methods are n-worded.

```
push, reserve, beginBuilding, ...
```

skewer-case

Pros: Easy to type.

```
easier-than-capitals, easier-than-underscore, …
```

Cons: Any sane language freaks out when you try it.

SCREAMING_SNAKE_CASE

Pros: Can demonstrate your anger with text.

Cons: Makes your eyes deaf.

```
LOOK_AT_THIS, LOOK_AT_THAT, LOOK_HERE_YOU_MORON, …
```

nocase

Pros: Looks professional.

Cons: Misleading af.

```
supersexyhippothalamus, bool penisbig, …
```

fUcKtHeCaSe

Pros: Can live outside of the law.

Cons: Can be out of a job.

Name Convention in Javascript

NAMING CONVENTIONS	APPLICATION	EXAMPLES
Lower Camel Case	variables and methods (functions)	firstName timeToFirstLoad indexNumber
Upper Camel Case	classes, interfaces, annotations, enums, records	TomcatServer RestController WriteOperation
Screaming Snake Case	constants	INTEREST_RATE MINIMUM_SALARY EXTRA_SAUCE

Variables

Variables

- A variable is a place to store values.
- Javascript is a **weakly typed language**. When declaring a new variable or constant, it will pick what type it thinks best. (unlike Java which is a strong typed language)
- You should use the following keywords:
  - **var** to create a new function scope variable (scope limit within the function).
  - **let** to create a new local scope variable (scope exists just for the current code block).
  - **const** to create an unchangeable global/function constant (scope limit within the function).

```
var aNumber = 5;
var aString = '5';
let anotherString = "5";
const goldenRatioNumber =  1.61803399;
var aBoolean = true;
let anObject = [1,2,3,4,5];
var aFunction = function sayHello() { ... };
let aStudentObject = {id:'s1234567', type:'Ok I guess', ... };
```

```
let amount = 9;
amount = amount * 2;
console.log(amount); // 18

// Convert `amount` to a string
amount = "$" + amount;
console.log(amount); // "$18"
```

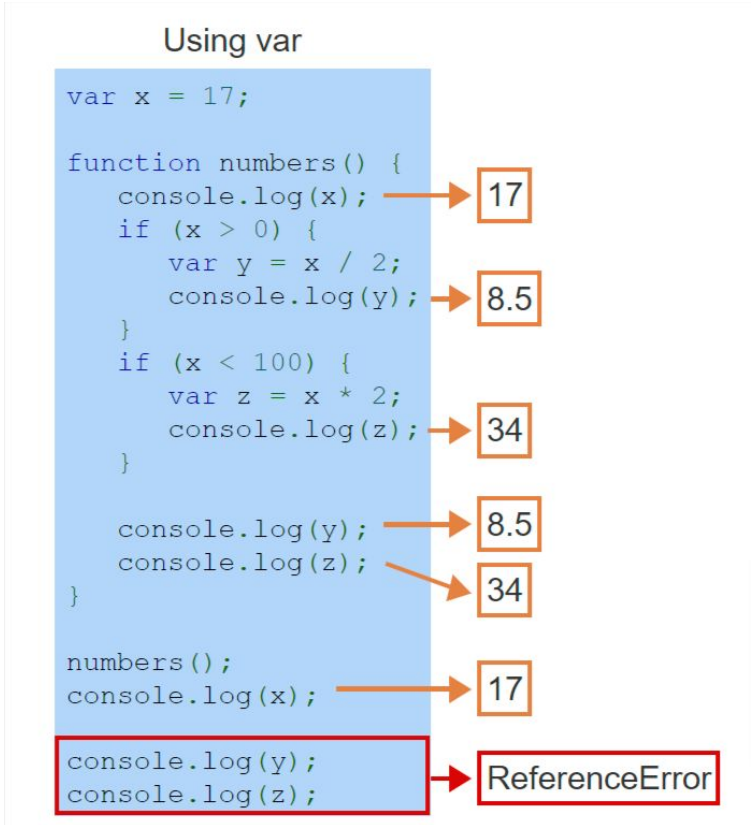
var 🆚 let

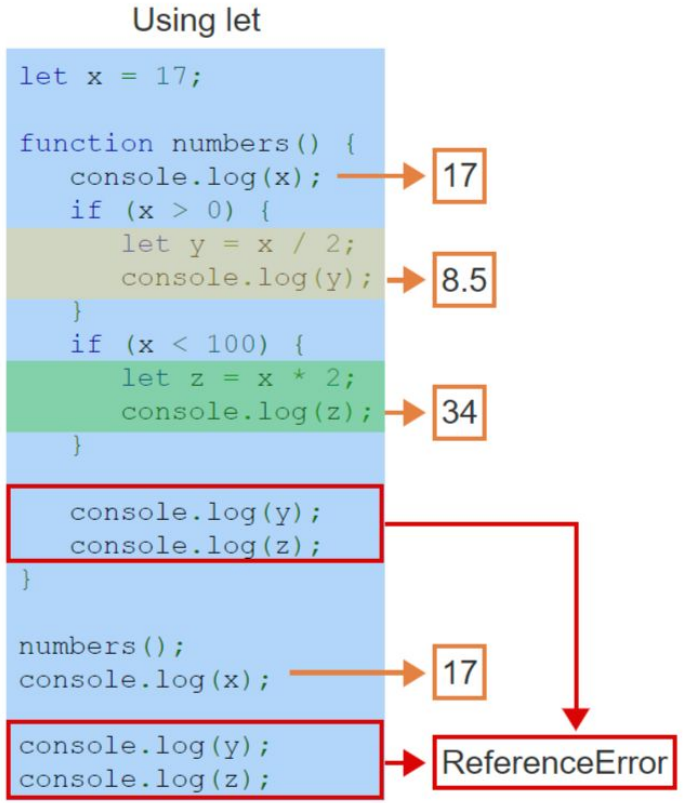
- In modern JavaScript, **it is recommended to use let instead of var for declaring variables.**
- **let** was introduced in (ES6) and provides **block-scoped variable** declarations, which means that the **variable is only accessible within the block of code where it is defined.**
  - ✅ This can help prevent certain types of bugs and makes code easier to reason about.
- **var** has **function scope**, which means that the **variable is accessible throughout the function where it is defined.**
  - ❌ This can lead to potential issues with variable hoisting and unintentional global variables.

```
function exampleFunction() {
    var x = 10;
    if (true) {
      var x = 20;
      console.log(x); // output: 20
    }
    console.log(x); // output: 20
}

exampleFunction();
```

```
function exampleFunction() {
    let x = 10;
    if (true) {
      let x = 20;
      console.log(x); // output: 20
    }
    console.log(x); // output: 10
  }

exampleFunction();
```

var vs let


Using var
var x = 17;
function numbers() {
console.log(x);
17
if (x > 0) {
var y = x / 2;
console.log(y);
8.5
}
if (x < 100) {
var z = x * 2;
console.log(z);
34
}
console.log(y);
8.5
console.log(z);
34
}
numbers();
console.log(x);
17
console.log(y);
console.log(z);
ReferenceError


Using let
let x = 17;
function numbers() {
console.log(x);
17
if (x > 0) {
let y = x / 2;
console.log(y);
8.5
}
if (x < 100) {
let z = x * 2;
console.log(z);
34
}
console.log(y);
console.log(z);
}
numbers();
console.log(x);
17
console.log(y);
console.log(z);
ReferenceError

var vs let question?

[Question] Based on what you have learned, what is the output of each example?

Example 1

```
function exampleFunction() {
    if (true) {
      var x = 20;
      console.log(x);
    }
    console.log(x);
}

exampleFunction();
```

Example 2

```
function exampleFunction() {
    if (true) {
      let x = 20;
      console.log(x);
    }
    console.log(x);
}

exampleFunction();
```

Example 3

```
function exampleFunction() {
  let x = 20;
  console.log(x);
}

exampleFunction();
console.log(x);
```

Example 4

```
let x = 10;

function exampleFunction() {
  let x = 20;
  console.log(x);
}

exampleFunction();
console.log(x);
```

Data Types

Data Types

1. **Number**: integer, float, etc
2. **String**: a group of 0, 1 or more text characters
3. **Boolean**: true or false
4. **Null**: a value that represents the intentional absence of any object value.
5. **Undefined**: a value that is automatically assigned to a variable or object property when no value is explicitly assigned to it. (it doesn't exist or is not set)
6. **Function**: a variable can refer to a function, that is a block of code
7. **Object**: an array, object or unfortunately null:
    - **Array**: a data structure that can store a collection of elements of any data type.

```
var aNumber = 5;
var aString = '5';
let anotherString = "5";
const goldenRatioNumber =  1.61803399;
var aBoolean = true;
let anObject = [1,2,3,4,5];
var aFunction = function sayHello() { ... };
let aStudentObject = {id:'s1234567', type:'Ok I guess', ... };
```

Operators

Relational Operators

a = 5, b = 2;

Operator	Description	Example	Output
== (equal to)	Check if the values are equal.	a == b	false
!= (not equal to)	Check if the values are not equal.	a != b	true
> (greater than)	Check if the left value is greater than the right value.	a > b	true
< (less than)	Check if the left value is less than the right value.	a < b	false
>= (greater than or equal to)	Check if the left value is more than or equal to the right value	a >= b	true
<= (less than or equal to)	Check if the left value is less than or equal to the right value	a <= b	false
typeof	Return a type of a object	typeof('A')	‘string’

typeof operator

- **typeof** operator that checks a variable's type and returns a string describing its type: `var whatTypeAreYou = typeof aVariable;`
- Here are some examples:

```
let a;
typeof a; // "undefined"

a = "hello world";
typeof a; // "string"

a = 42;
typeof a; // "number"

a = true;
typeof a; // "boolean"
```

```
a = null;
typeof a; // "object"

a = undefined;
typeof a; // "undefined"

a = [3, "black", "dog"];
typeof a; // "object"

a = { b: "c" };
typeof a; // "object"
```

Quiz of typeof

[Question] What will be returned when testing the following variables?

```
typeof "";
typeof 5;
typeof 5.0;
typeof function () {} ;
typeof wtv;
```

```
var wtv
typeof wtv;
```

```
var wtv = null;
typeof wtv;
```

```
typeof [3, 1, 4];
```

```
typeof typeof [3, 1, 4];
```

Equality Operators

a = 1, b = “1”;

Operator	Description	Example	Output
== (loose-equals)	compare values regardless of the types	a==b	true
=== (strict-equals)	compare values only with the same types	a===b	false
!= (loose not-equals)	the opposite of loose-equals	a!=b	false
!== (strict not-equals)	the opposite of strict-equals	a!==b	true

```
console.log(1 == '1');   // true
console.log(1 === '1');  // false
console.log(null == undefined);  // true
console.log(null === undefined); // false
```

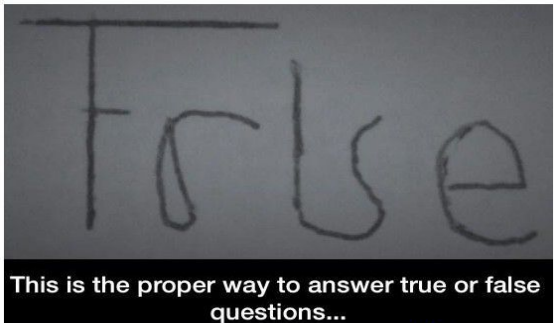
[Explanation] null == undefined would be true because they are both "falsy" values.

Logical Operators

a = true, b = false;

Operator	Description	Example	Output
&& (logical and)	If both inputs are true then only that the condition is true.	a && b	false
|| (logical or)	If either of any input is true then the condition is true.	a || b	true
! (logical not)	Use to reverse the logical state of the input.	!(a && b)	true

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False



Assignment Operator

a = 6, b = 3;

Operator	Description	Example	Output
= (assignment operator)	Assign values from the right side to the left side.	b = a + b	9.0
+= (Add AND assignment operator)	Is equivalent to b = b + a	b += a	9.0
-= (Subtract AND assignment operator)	Is equivalent to b = b - a	b -= a	-3.0
*= (Multiply AND assignment operator)	Is equivalent to b = b * a	b *= a	18.0
/= (Divide AND assignment operator)	Is equivalent to b = b / a	b /= a	0.5

Ternary Operator

- A special ternary (three-way) operator that can replace simple statement of if- then-else statements to have it in one line.
- expression1 ? expression2 : expression3
- For example,

```
val1 = 10;
val2 = 20;
max = val1 >= val2 ? val1 : val2;
```

[**Question**] Can you guess what is the value of max?

Type Conversion Functions

- At some point, you will need to convert a variable from one type to another, for example:

```
> "5" + "3"
<· '53'
```

- You can use the following built-in javascript functions:

```
let aNumber = Number(aString);
let aString = String(aNumber);
let aFloat  = parseFloat(anInt);
let anInt   = parseInt(aFloat);
```

- Now, you can do:

```
> Number("5") + Number("3");
<· 8
```

Bitwise Operators (Bonus)

Operator	Description	Example	Output
		a = 3	00000011 (3)
		b = 5	00000101 (5)
& (bitwise and)	Copies a bit if it exists in both.	a & b	00000001 (1)
| (bitwise or)	Copies a bit if it exists in either.	a | b	00000111 (7)
^ (bitwise XOR)	Copies a bit if it is in one number but not both.	a ^ b	00000110 (6)
~ (bitwise complement)	Flips all bits	~a	11111100 (-4)
<< (left shift)	Shift all bits to the left by a specified number.	a << 2	00001100 (12)

[Real Interview Question] Can you figure out how to know a number is even or odd using bitwise operators?

String Operators

String Operator(s)

- Javascript allows you to join strings together using the string concatenation operator +:

```
let helloWorld = 'Hello' + ' ' + 'World!';
```

- There are many string properties and functions built into javascript string objects which can be accessed using the . character. Here are a few examples:

```
aString = "Hello world!";
```

- `aString.length` returns length of the string
- `aString.charAt(posn)` returns the character found at a particular position
- `aString.indexOf(subString)` returns position of subString within aString (or -1 if not a substring)
- `aString.includes(subString)` similar to indexOf(), only a boolean true/false is returned
- `aString.trim()` returns a string with whitespace removed from both ends of the string
- `aString.toUpperCase()` returns an uppercase version of the string
- `aString.substr(start, num)` returns a substring from specified position, extracting num characters
- `aString.split(delimiter)` returns an array of strings split on a particular delimiter (ie character or substring)

String Operators Example

[Question] Can you explain each of these following outputs?

```
let aString = "Hello World!  ";
console.log(aString.length);
console.log(aString.charAt(2) );
console.log(aString.indexOf("lo"));
console.log(aString.includes("World"));
console.log(aString.trim());
console.log(aString.toUpperCase());
console.log(aString.substr(2, 4));
console.log(aString.split(" "));
console.log(aString.trim().split(" "));
```

```
14
l
3
true
Hello World!
HELLO WORLD!
llo
▶ (4) ['Hello', 'World!', '', '']
▶ (2) ['Hello', 'World!']
```

Control Structures

Controlling Program Flow

- Control structures help you control the flow of your program.
- In addition, we have structures which allow us to repeat actions (loops).
- Here are the control flow statements:
  - **if - then - else:**
    - This block allows us to make a decision and the else is optional.
  - **switch - case - break:**
    - Similar to if - then - else block, more efficient when checking a single variable for many different values.
  - **while and do - while:**
    - A simple loop that performs a condition check and allows the block to repeat.
  - **for and for - in:**
    - Similar to while loop, only incrementing is built in to the condition check.

Simple If Then Else

- This block can be simple, with an optional else clause:

```
if ( condition check ) {
    // code to execute if condition evaluates to true
}
```

```
if ( condition check ) {
  // code to execute if condition is true
} else {
  // code to execute if condition is false
}
```

Nested If Then Else

- Blocks can be nested inside each other when performing more than one check:

```
if ( first condition check ) {
  // code to execute if first condition is true
} else if ( second condition check ) {
  // code to execute if only second condition is true
} else {
  // code to execute if both conditions are false
}
```

```
if ( first condition check ) {
  if ( second condition check ) {
    // code to execute if both conditions are true
  } else  {
    // code to execute if only first condition is true
  }
} else {
  // code to execute if first condition is false
}
```

If Statement Example

- Date() function is a built-in constructor that creates a new instance of the Date object.
- When new Date() is called without any arguments, it creates a new Date object with the current date and time according to the system clock on the device where the code is running.

```
> new Date()
< Mon Apr 03 2023 09:21:41 GMT+0700 (Indochina Time)
```

```
let today = new Date().getDay();

if (today === 6) {
  console.log("Today is Saturday");
} else if (today === 0) {
  console.log("Today is Sunday");
} else {
  console.log("Looking forward to the Weekday");
}
```

→ Looking forward to the Weekday

Switch Case Block

- This block is suitable when testing a variable for multiple values.
- Each mini "case-block" must be terminated with a break statement or it will run into the next case.
- An optional default option is to be run if no value is listed in the block:

```
switch ( aVariable ) {
  case value 1:
    // code to execute a variable has value #1
    break;
  case value 2:
    // code to execute a variable has value #2
    break;
  ...
  case value 105:
    // code to execute a variable has value #105
    break;
  default:
    // code to execute if value not listed
}
```

Run-on Switch Case Block

- If the same code is to be executed for a set of values, the break statement can be omitted to allow "run-on" execution.

```
switch ( dayName ) {
  case "Monday":
  case "Tuesday":
  case "Wednesday":
  case "Thursday":
  case "Friday":
    dayType = "Weekday";
    break;
  case "Saturday":
  case "Sunday":
    dayType = "Weekend";
    break;
  default:
    dayType = "Invalid";
}
```

[Question] What is the output if the value of variable “let dayName =” is:

- Monday
- Wednesday
- Friday
- Saturday
- Sunday
- Funday

While and Do - While Loops

- A while loop will perform a check and run code if the check is returned true.
- The loop should be designed so the condition being checked changes in the loop. For example, this will add up all of the numbers from 1 to 99 and stop when count is incremented past 99:

```
let sum1to99 = 0;
let count = 1;
let stopCount = 100;
while ( count < stopCount ) {
  sum1to99 += count;
  count++;
}
99
```

- A do - while loop is very similar except that **the check is performed at the end of each loop**, not the start, which **executes the code in the loop at least once**:

```
let sum1to99 = 0;
let count = 1;
let stopCount = 100;
do {
  sum1to99 += count;
  count++;
} while ( count < stopCount )
99
```

For Loops

- For loops combine an incrementer and a condition check into one area:

```
> let sum1to99 = 0;
  var stopCount = 100;
  for (let count = 1; count < stopCount; count++) {
    sum1to99 += count;
  }
<· 4950
```

[Task] Try to run this js script in the console of any website and see what happens to the existing content of the website?
[Note] We will learn more about document object next week!

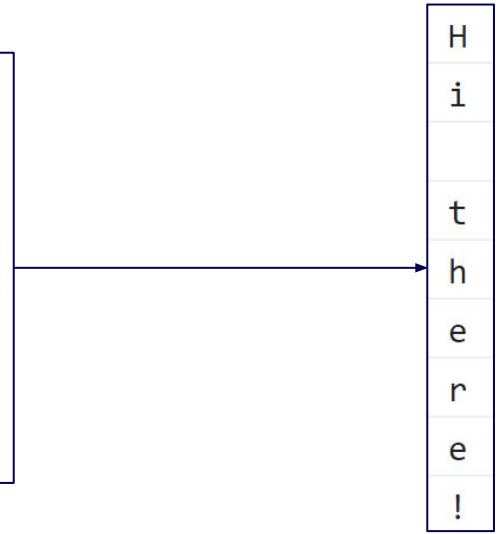
```
let weekDays = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Funday','Sunday'];
let dayMessage = '';

for (let dayNum = 0; dayNum < weekDays.length; dayNum++) {
  dayMessage = weekDays[dayNum] + ' is a ';
  switch ( weekDays[dayNum] ) {
    case "Monday":
      dayMessage += 'hateful ';
    case "Tuesday":
    case "Wednesday":
    case "Thursday":
    case "Friday":
      dayMessage += 'week day';
      break;
    case "Saturday":
    case "Sunday":
      dayMessage += 'weekend day';
      break;
    default:
      dayMessage += '... what day is that?';
  }
  document.write(dayMessage + '<br>');
}
```

Monday is a hateful week day
Tuesday is a week day
Wednesday is a week day
Thursday is a week day
Friday is a week day
Saturday is a weekend day
Funday is a ... what day is that?
Sunday is a weekend day

For ... of

- A for...of loop to iterate over the elements in an iterable object, such as arrays, strings, maps, and sets.
- It allows you to loop through each element of the iterable object without having to worry about its index.

```
let myString = 'Hi there!';

for (let char of myString) {
  console.log(char);
}
```



Functions

Functions

- **Javascript has many inbuilt functions**, and developers can write their own functions too.
- Functions can be called by other functions and by event handlers, such as when a page loads or when a user generated an event by interacting with the page.
- ✅ **Good programming tip:** A function should return something useful (eg a value or sting) or it should return a boolean (ie true or false) to let the caller know that the function executed successfully or failed.

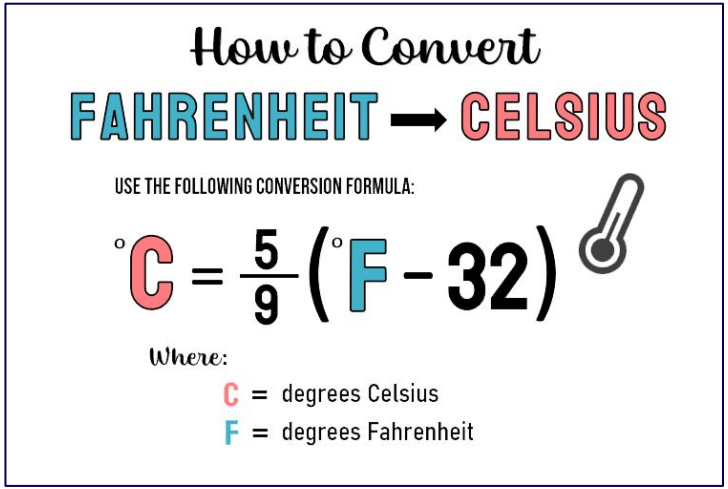
```
function suitableFunctionName (optional parameters){
    // code goes here
    // last line: return something
}
```

```
function addNumbers(num1, num2) {
    return num1 + num2;
}

let result = addNumbers(5, 7);
console.log(result); // Output: 12
```

A simple function example

- Here is a simple function converting Fahrenheit temperature to Celsius temperature.

```
> function toCelsius(fahrenheit) {
      return (5 / 9) * (fahrenheit - 32);
  }

  let x = toCelsius(77);
  let text = "The temperature is " + x + " Celsius";
  console.log(text);
  The temperature is 25 Celsius
```



Function as a variable

- **Functions can be assigned to variables** just like any other data type, such as numbers or strings.
- When a function is assigned to a variable, the variable can be used to call the function just like you would call the function directly.

```
// Define a function
function sayHello(name) {
    console.log("Hello " + name);
}

// Assign the function to a variable
let welcome = sayHello;

// Call the function using the variable
welcome('Alice'); // Output: Hello, Alice!
```

Higher-order Function

- **A higher-order function** is a function that takes one or more functions as arguments, or returns a function as its result.
- It is a **powerful feature** in JavaScript because they allow for more **modular** and **reusable code.**
  - By passing functions as arguments or returning them as results, we can write **generic functions** that can be customized by passing in **different functions** with **different behaviors.**

Pass a function as an argument inside another function

```
// define two functions
function add(a, b) {
    return a + b;
}

function subtract(a, b) {
    return a - b;
}

// define a higher-order function that takes in a function as an argument
function performOperation(operation, a, b) {
    return operation(a, b);
}

// use the higher-order function to perform different operations
let result1 = performOperation(add, 5, 3); // returns 8
let result2 = performOperation(subtract, 5, 3); // returns 2

console.log(result1); // 8
console.log(result2); // 2
```

A function returns another function

```
function createMultiplier(multiplier) {
    return function (number) {
        return number * multiplier;
    }
}

const double = createMultiplier(2);
const triple = createMultiplier(3);
console.log(double(5)); // Output: 10
console.log(triple(5)); // Output: 15
```

JavaScript Style Guide

Good JavaScript Style Guide

- To understand the best practices of javascript, please follow:
  - Principles of Writing Consistent, Idiomatic JavaScript
  - Google JavaScript Style Guide
- Criteria for good javascript code:
  - Filename
  - Syntax Formatting
  - Naming (variables, functions, etc)
  - JSDoc & Comments

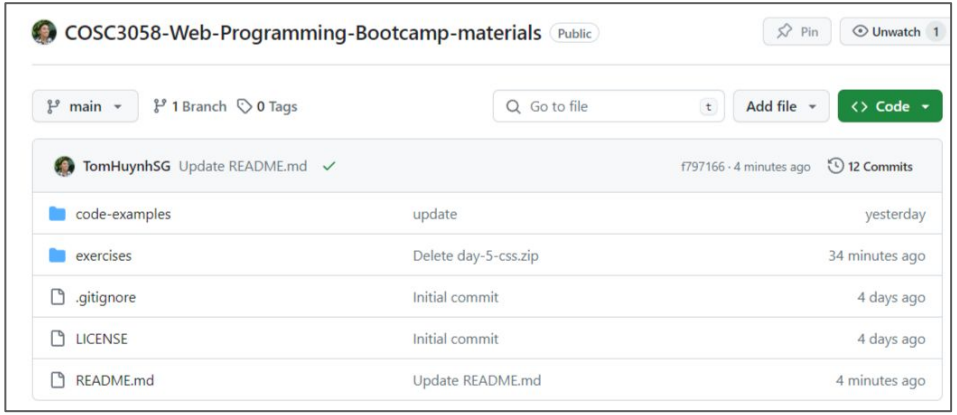
```
> typeof NaN
< "number"
> 9999999999999999
< 10000000000000000
> 0.5+0.1==0.6
< true
> 0.1+0.2==0.3
< false
> Math.max()
< -Infinity
> Math.min()
< Infinity
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true===3
< true
> true-true
< 0
> true==1
< true
> true===1
< false
> (!+[]+[]+![]).length
< 9
> 9+"1"
< "91"
> 91-"1"
< 90
> []==0
< true
```

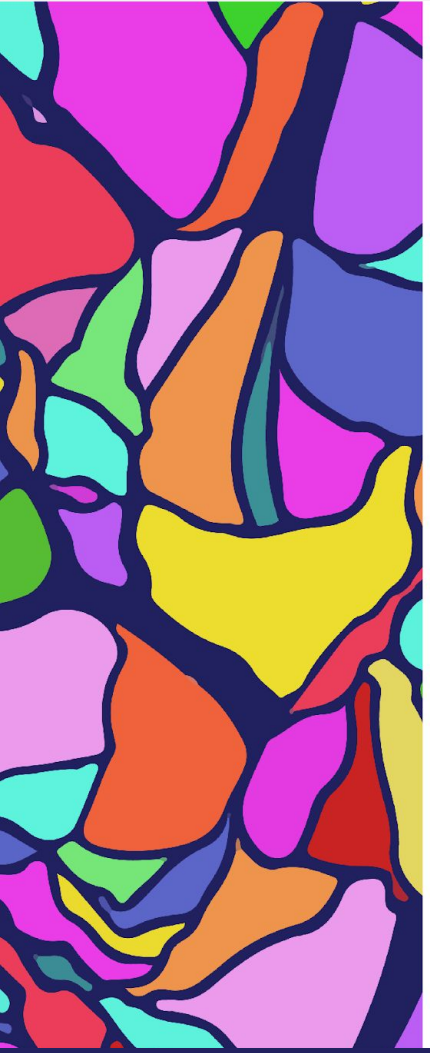
JS Reference

- MDN (Mozilla Developer Network) is always the best reference source.
  - Link: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Comprehensive Reference Devdoc.io:
  - Link: https://devdocs.io/javascript/
- W3School for Javascript:
  - Link: https://www.w3schools.com/js/default.asp
- JS Cheat Sheet:
  - Link: https://ilovecoding.org/blog/js-cheatsheet
- DOM Documents:
  - Link: https://www.w3schools.com/jsref/dom_obj_document.asp
- DOM Style objects:
  - Link: https://www.w3schools.com/jsref/dom_obj_style.asp

COSC3058 Materials Github Repo

- All of the file examples of the lectures and tutorials will be uploaded weekly and available for everyone to check them out.
- [Github repository link](#).





“Motivation is what gets you started.
Habit is what keeps you going.”

Jim Ryun

RMIT UNIVERSITY