

CS2204 Fundamentals of Internet Applications Development

Lecture 6 JavaScript Part 1

Computer Science, City University of Hong Kong
Semester B 2024-25

Announcement: Midterm

Mid-term is scheduled in two weeks, on **March 13th, Thursday** from **4:00 PM to 5:00 PM** (60 minutes)

Venue: LT-10

Coverage: Lec01 – Lec07, Lab01– Lab05

Form: 45 multiple choice questions; close book; no calculator allowed

Seat arrangement will be announced soon.

Reminder: CW1 due in on Mar 7th

2024/25 Semester B

Announcements

Discussions

Grades

Pages

Files

Syllabus

Quizzes

Modules

Collaborations

Attendance

Library Resources

Class List (AIMS)

LockDown Browser

uReply

CW1: A simple website for Hong Kong Disneyland

Published

Assign To

Edit

⋮

You are required to build a simple website for Hong Kong Disneyland by going through the 3Ss: structure, style, and script. This CW1 is the first step focusing on structure and basic style only. The Website, therefore, would not be fully functional until the second part (CW2) using advanced CSS and JavaScript are completed.

The detailed instructions for this assignment can be found here: [CS2204-2425B-CW1-Description.pdf](#) ↓ with corresponding materials under file > Assignment > CW1 > materials.

To create HTML files, you are recommended to use professional code editors for better syntax checking/code management (e.g., VS Code).

Important notes:

- Image sources needed in this assignment can be found under files > Assignments > CW1
- For videos, please link them directly to <https://personal.cs.cityu.edu.hk/~cs2204/video/>. Please **DO NOT** include them in your submission.

Points 10

Submitting a file upload

File Types zip

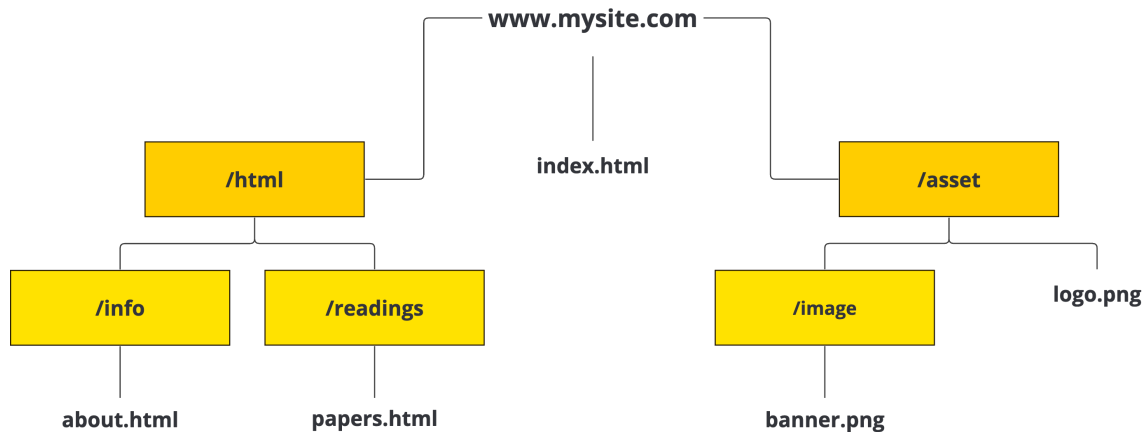
Some Rubric



Criteria	Ratings		Pts
Structure Have three main pages, including index.html, design.html, and order.html, which are organized in proper folders (e.g., html, image). A shared theme.css is linked to all three pages.	2 pts Full Marks	0 pts No Marks	2 pts
Use of HTML tags Use appropriate HTML structural tags (e.g., <header>, <footer>, <div>,), and give as much structure as feasible	4 pts Full Marks	0 pts No Marks	4 pts
CSS basic styles Follow the styling instructions to apply at least one tag, class, and id selector(s); also apply at least one contextual or advanced selector. This information should be specified in the design page and the style should be as consistent as possible	3 pts Full Marks	0 pts No Marks	3 pts
Good web dev practices Accessibility (e.g., alt-text for images), code readability (e.g., indentation), and page validation	1 pts Full Marks	0 pts No Marks	1 pts

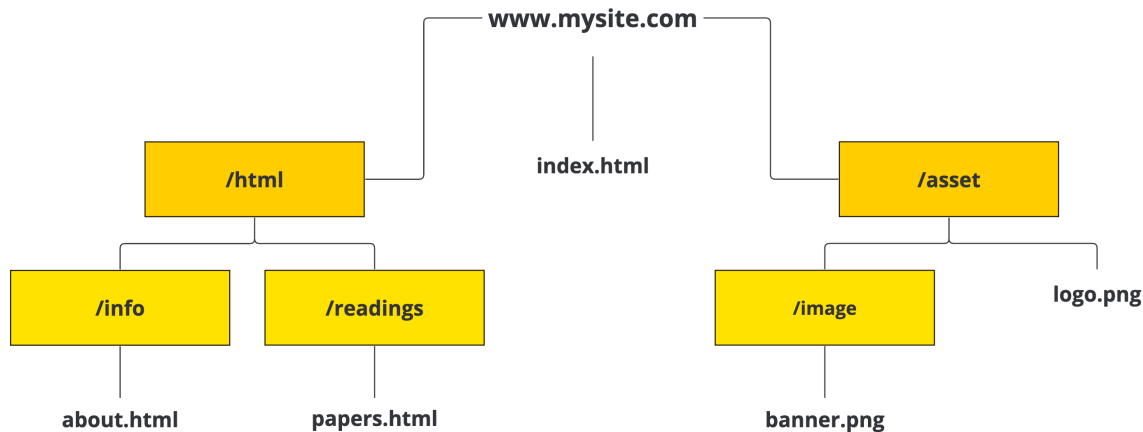
Total Points: 10

Post-lab Quiz 2 Review



Assuming that **the current working directory** is `/readings`, use relative path to access `papers.html`: _____

Post-lab Quiz 2 Review



Assuming that **the current working directory** is **/readings**, use relative path to access *logo.png*: _____



Question Time

Which statement of box model is true?

- A. The margin attribute refers to the space surrounding an element, outside of its border.
- B. By default, the `width` of `<p>` takes up 100% width of the browser window
- C. If the value of the `overflow` property is `auto`, part of the contents will be visible, and the other part that takes up extra space of the container will be accessible via scrollbar
- D. The `width` attribute can be used to change horizontal space of all the HTML elements



Question Time

What does a hexadecimal color code like #FF5733 represent in terms of RGB values?

- A. RGB(255, 50, 75)
- B. RGB(0, 0, 0)
- C. RGB(255, 87, 51)
- D. RGB(15, 5, 7, 3)



Question Time

Consider the following CSS rule applied to an HTML element:

```
.element {  
    transform: translateX(100px) scale(1.5);  
}
```

What is the correct sequence of transformations applied to the .element class?

- A. The element was first expanded with additional 100 pixels horizontally, and scale up by 50%
- B. Current size - original size = 50% \times original size
- C. Current size - original size = 150% \times original size
- D. The element was moved 100 pixels to the right, and scale up by 150%



Question Time

What is the purpose of the @keyframes rule in CSS?

- A. To define the style and color of the text in an HTML document.
- B. To specify the timing function for transitions between styles.
- C. To configure the animations of an element by defining styles at various stages of the animation sequence.
- D. To create breakpoints for responsive web design.

Review: CSS3

Introduce `rgba()`: specifying a color using **red**, **green**, **blue**, and **alpha** (transparency) values.
e.g., `rgba(255, 0, 0, 0.5)` represents red color with 50% transparency

Specify element effects such as `box-shadow`, `border-radius`

Used in media queries to apply styles based on different media types

Specifies the transition effect for a CSS property over a specified duration.

Allows applying transformations to elements, such as scaling, rotating, skewing, or translating them.

`@keyframes`: Used to define animations by specifying keyframes at different stages (by percentages) of the animation's duration.

Review: CSS3

Welcome to CSS animations!



Rotating Circle



Bouncing Ball



Pulsing Color

```
.fade-in {  
  opacity: 0;  
  animation: fadeInAnimation ease-in-out 3s;  
  animation-fill-mode: forwards;  
  font-size: 20px;  
  color: #333;  
}
```

```
@keyframes fadeInAnimation {  
  from { opacity: 0; }  
  to { opacity: 1; }  
}
```

```
.rotate {  
  width: 100px;  
  height: 100px;  
  background: linear-gradient(to right, skyblue 50%, deepskyblue 50%);  
  border-radius: 50%;  
  animation: rotation 4s infinite linear;  
}
```

```
@keyframes rotation {  
  from { transform: rotate(0deg); }  
  to { transform: rotate(360deg); }  
}
```

```
.bounce {  
  width: 50px;  
  height: 50px;  
  background-color: coral;  
  border-radius: 50%;  
  animation: bounceAnimation 2s infinite;  
}
```

```
@keyframes bounceAnimation {  
  0%, 20%, 50%, 80%, 100% { transform: translateY(0); }  
  40% { transform: translateY(-30px); }  
  60% { transform: translateY(-15px); }  
}
```

```
.pulse {  
  width: 100px;  
  height: 100px;  
  background-color: violet;  
  border-radius: 50%;  
  animation: pulseAnimation 3s infinite;  
}
```

```
@keyframes pulseAnimation {  
  0%, 100% { background-color: violet; }  
  50% { background-color: pink; }  
}
```

Agenda

JavaScript Intro

- Execution environment
- Variable and scope
- Array

JavaScript Logic

- Expression and statement
- Operation and operator
- Functions

Agenda

JavaScript Intro

- Execution environment
- Variable and scope
- Array

JavaScript Logic

- Expression and statement
- Operation and operator
- Functions

About Programming language (1)

A set of instructions telling a computer what to do

- Getting data from user's **input** or **storage** (hard disk or memory)
- **Processing** data that may use temporary locations in memory (variable) to store results
- **Dynamically Outputting** results based on input or logical rules

In the context of web development, JavaScript tells the **browser** what to do

About Programming language (2)

What does each **instruction** (statement) do?

- Defining / assigning values to variables
- Logical / mathematical operations (e.g., function calls, calculation)

Writing a program is to :

- Think of a way to solve the problem first (i.e., the logic or algorithm)
- Build up instructions with flow control according to the designed algorithm

JavaScript Overview

JavaScript (also known as “JS”) is a **programming language** that can provide instructions for a browser to dynamically generate content for a website or enhance the website interactivity

JavaScript can be embedded in the head or body section of the webpage, with the code defined inside the `<script></script>` tags

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Javascript First Example</title>
6     <script>
7       function myclick() {
8         alert("Welcome to CS2204!");
9       }
10    </script>
11  </head>
12  <body>
13    <!-- Page content begins here -->
14    <h1>CS2204</h1>
15    <h2 onclick="myclick();">Click Me</h2>
16    <!-- Page content ends here -->
17  </body>
18 </html>
```

In JavaScript, a **function** is defined using the keyword **function**, and enclosed in **curly brackets { }**

The **semi-colon ;** is placed at the end of each JavaScript statement

How to include and run JavaScript (JS)

Different ways to include JavaScript in a web page (similar to include CSS)

- Embedded
- External
- Inline

JS code can be executed when

- The webpage is loaded
- Event is triggered (e.g., click action, timing, external call)

Execution order is according to the order presented to the browser

Embedded Script

Written in `<script>` `</script>` tags directly

- Usually go in the **head section**
 - scripts are loaded **before** the function is called
 - scripts can be used by elements in the entire web page

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>javascript First Example</title>
6     <script type="text/javascript">
7       function myclick() {
8         alert("Welcome to CS2204!");
9       }
10    </script>
11  </head>
12  <body>
13    <!-- Page content begins here -->
14    <h1>CS2204</h1>
15    <h2 onclick="myclick();">Click Me</h2>
16    <!-- Page content ends here -->
17  </body>
18 </html>
```

Code Example: lec06-02-JS-first-example.html

External Script

Saved in an external “.js” JavaScript file

- easier for **maintenance**
- search engine friendly
- use codes from others (function libraries)
- provide codes for others (other pages)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript First Example</title>
    <script type="text/javascript"
src="./lec07-12-external-script.js"></script>
  </head>
  <body>
    <!-- Page content begins here -->
    <h1>CS2204</h1>
    <h2 onclick="myclick();">Click Me</h2>
    <!-- Page content ends here -->
  </body>
</html>
```

CS2204

Click Me

```
function myclick() {
  alert("Welcome to CS2204!");
}
```

Inline Script

Placed in the **body section** inside `<script></script>`

- Must use the `<script>` tag or event attribute - **CANNOT** put inside an element as **content**

```
<h1>alert ( )</h1>
```

this is wrong!

Advantages:

- useful for **diagnostics** (display alert messages)
- create HTML during page load depending on **different situations**, e.g., get current date time

Disadvantages:

- **difficult** to update

JS Execution Environment (1)

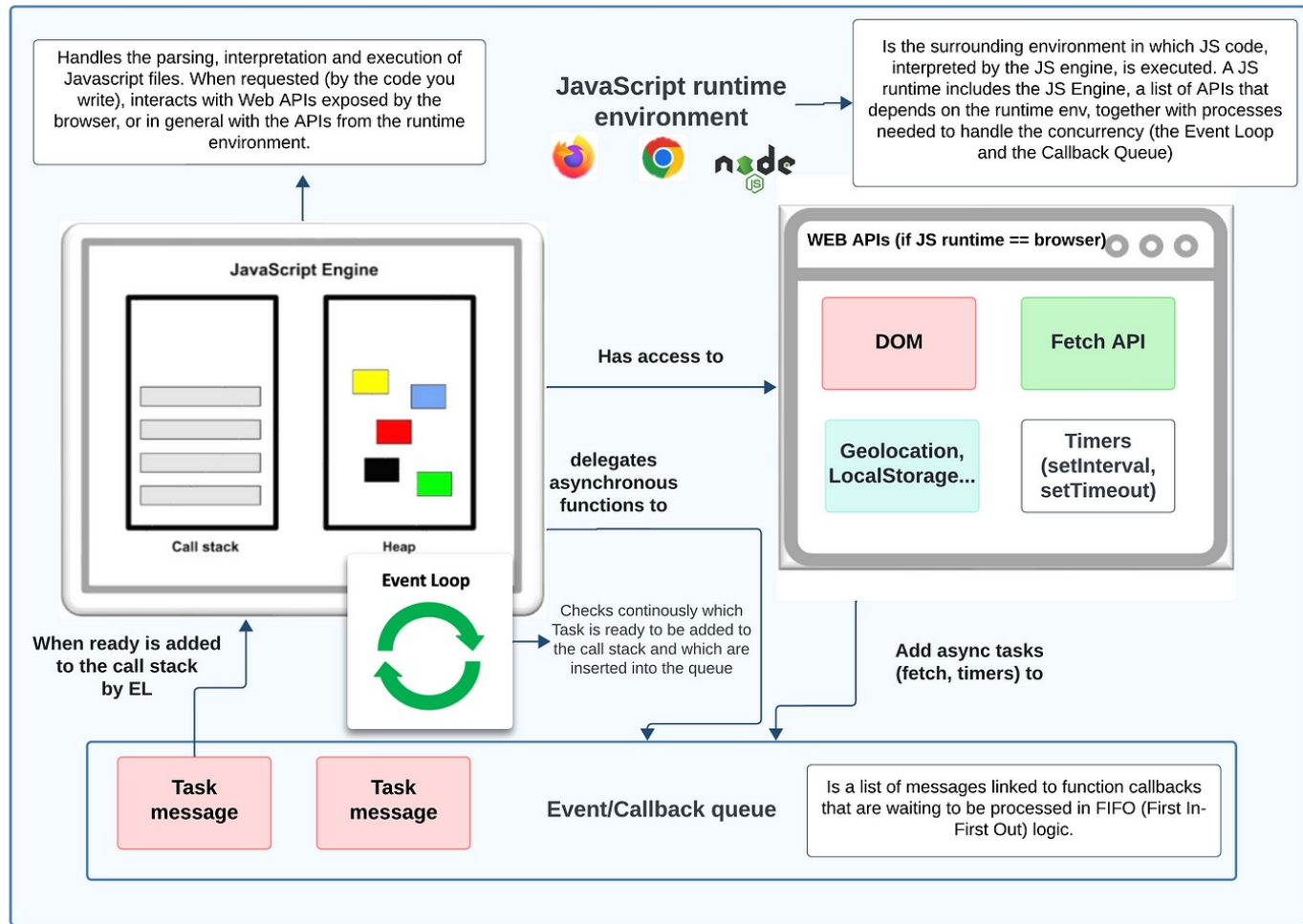
Two main engines

- **Rendering** engine processes HTML/CSS to render webpages
- **Script engine** (e.g., Node.js) processes scripts
 - Single-thread and synchronous execution (push task to stack)
 - Can lead to render-blocking

Can run asynchronously with events (e.g., http requests, SetTimeout, DOM)

- Push asynchronous task in queue
- Asynchronous methods: ES6 (promise), ES7 (async/await)

Execution Environment (2)



JS Execution Environment (3)

Three main components:

- **ECMAScript**
 - The core language syntax, defines how the code is written, interpreted, and executed, such as data types, operators, functions, and scopes
- **DOM** (Document Object Model)
 - Allows JS to access and manipulate the contents of a web page dynamically. It is an API (Application Programming Interface) that represents a web page as a tree of objects (recall what we learned in contextual selector in Lec 04)
- **BOM** (Browser Object Model)
 - For JS to interact with browser operations (e.g., get window size, open/close a window)

How will JS be used in this course? (1)

Program Web pages

- Dynamically generate content (HTML and CSS cannot do)
- Check user input, usually in forms
- Respond to events

DHTML (dynamic HTML) & HTML5 features

- **DHTML**: combine the power of CSS and JavaScript to enhance user experience; create HTML elements with JS
- **HTML5** features: scripted audio & video; canvas

How will JS be used in this course? (2)

Not as a full programming language

- just enough to work with Web pages
- similar to CSS & HTML - we focus on the elements
- 3S: Structure, Style & Script –use JS to manipulate or create elements, respond to events occurred for elements

Select an object in the DOM

- use JS to manipulate it
- attach JS to it (event handler/listener)

Identify events of an element

- execute a JS (event handlers) when they occur

JavaScript: Website Interactivity

Website interactivity can be enhanced by detecting a user event and defining the corresponding event handler to perform certain action, e.g., the following program has an event handler that detects if the user clicks on the h2 heading “Click Me” and then calls the function `myclick` to pop up a message.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Javascript First Example</title>
6     <script type="text/javascript">
7       function myclick() {
8         alert("Welcome to CS2204!");
9       }
10    </script>
11  </head>
12  <body>
13    <!-- Page content begins here -->
14    <h1>CS2204</h1>
15    <h2 onclick="myclick();">Click Me</h2>
16    <!-- Page content ends here -->
17  </body>
18 </html>
```

`myclick` is a self-defined function which is defined with the keyword `function`

This page says

Welcome to CS2204!

The statement `alert (" [message] ")` will pop up a window and display the message specified inside the parentheses `()`

The attribute `onclick` is an event handler that is invoked when this h2 element is clicked. In this case, the JavaScript function `myclick()` will be called

Code Example:
lec06-02-JS-first-example.html

JavaScript: Dynamic Content

The innerHTML property allows the content of an HTML element to be changed dynamically

1. Originally the “welcome” div has no content: `<div id="welcome"></div>`
2. After the page has finished loading, the `onload` function is invoked which will call the `showDynamicContent()` function

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript InnerHTML Example</title>

    <script type="text/javascript">
      function showDynamicContent () {
        document.getElementById("welcome").innerHTML = "<h2>Welcome to CS2204!";
      }
    </script>
  </head>
  <body onload="showDynamicContent();">
    <h1>CS2204</h1>

    <div id = "welcome"></div>
  </body>
</html>
```

CS2204

Welcome to CS2204!

Usually the `onload` event handler is added as an attribute in the body tag and is used to call some JavaScript code to carry out some tasks right after the webpage has finished loading to initialize some settings

3. The statement `document.getElementById("welcome").innerHTML = "<h2>Welcome!</h2>"` replaces the current content of the “welcome” div with the string `<h2>Welcome!</h2>` such that the webpage will be displayed as if the html of the “welcome” div is `<div id="welcome"><h2>Welcome!</h2></div>`

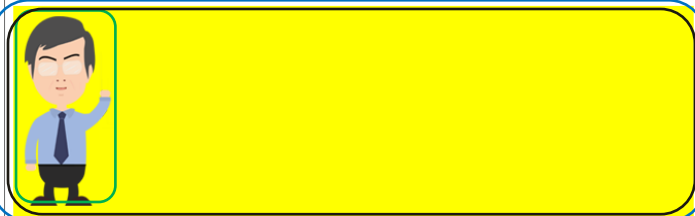
JavaScript: Change HTML elements

Javascript can modify an HTML element by

- changing its content
 - e.g., change the text
- changing its attribute
 - e.g., change img src attribute
- showing/hiding it
 - e.g., show/hide a div element
- changing its style
 - e.g., change the background

Code example:
lec06-05-JS-interactive-change.html

Javascript can change webpage interactively



Javascript can change the content of an HTML element:
Your lucky of the day is 15:03

Javascript can change the attribute of an HTML element:
Choose the character dress: ☐ Casual ☒ Formal

Javascript can show/hide an HTML element:
☒ Show box

Javascript can change the style of an HTML element:
Set the box color to

JavaScript: identifiers and keywords

Identifiers gives unique names to variables, functions, objects, etc.

Keywords are some pre-defined words, which

- have reserved meaning
- cannot be used as identifiers
- For example: *var, function, while, for, if, break, continue, ...*

Variable

One type of identifiers to store data values

- Defined by the keyword var, e.g., var myMsg;
- Keywords cannot be used as identifiers

Rules for variable names:

- Variable names can include the following characters only
 - all uppercase letters (A ~ Z), and lowercase letters (a ~ z)
 - digits (0 ~ 9)
 - underscore (_) and dollar sign (\$)
- Variable names **cannot** begin with a digit

Examples of variable names

- Valid names: myMsg1, my_msg, _myMsg, \$myMsg, MyMsg
- Invalid names: 3D_point, my-msg, my msg
- hyphen (-) and space () are not allowed

Data Types

JS has 3 basic (primitive) data types and other complex types:

- **Boolean** - can contain **two values** only: *true* or *false*
- **Number**
 - Integer - *decimal* or *hexadecimal*
 - Float - can be represented in either **standard** or **scientific** notation, e.g., 305.673, 8.32e+11, 1.2e2, 9.98E-12
- **String**
 - a **sequence** of zero or more characters
 - enclosed by double quote (") or single quote ('), e.g., "hello world!"
 - the quotes should be used with care: start double(single) quote matches with end double(single) quote
- **Other types**: array, date, object, function, etc.

Variable Declaration (1)

Variable needs to be declared before we use it

Declaration only:

```
var alertMsg;  
var age1, age2, age3, AlertMsg;
```

*Value of each variable **by default** is **undefined***

With value assignment

```
var alertMsg = "The following error(s) is/are found.";  
var age1 = 10, age2 = 20.5, age3 = 30, AlertMsg = 'hello world';
```

Declaration and then initialization

```
var age1;    // declare variable age1  
age1 = 5;    // initialize age1  
age1 = 10;   // update age1
```


Variable Declaration (2)

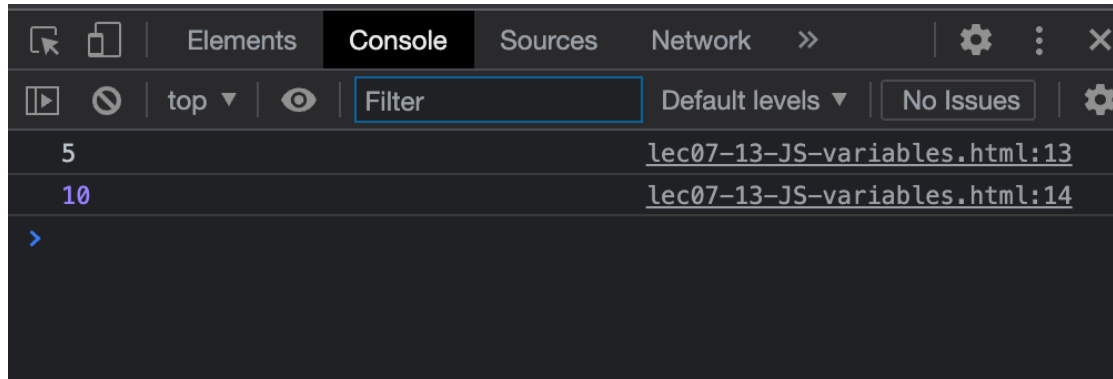
Variable names are **case sensitive**, e.g., age and Age are different variable names

- Web console can be opened by **right click > Inspect** and then select “**Console**” next to element

```
<script>
var name = 5;
var Name = 10;
console.log(name);
console.log(Name);
</script>
```

Code Example: lec07-06-JS-variable.html

Console.log () prints variable values in browser console for quick diagnosis



Variable Declaration (3)

In addition to **var**, you can also declare a variable using **const** or **let**.

```
const message = "hello world";
```

However, a variable defined with the **const** keyword

- **Must** be assigned a value when it is declared
- **CANNOT** be reassigned:

```
const message = "hello world";
```

```
message = "hello CityU"; // this will cause an error
```

Variable Declaration (4)

let (introduced in 2015) is another keyword used to declare a variable

```
let message = "hello world";
```

However, a variable defined with the **let** keyword

- **Must** be declared **before use**
- Have **block scope** (explained below)
- **CANNOT** be redeclared in the same scope

```
{let message = "hello world";}
```

```
// message variable cannot be used here
```

```
{var message2 = "hello world";}
```

```
// message2 variable can be used here
```



Question Time

```
<!DOCTYPE html>
<html>
<body>
<p id="x"></p>
<p id="y"></p>
<p id="z"></p>
```

```
<script>
var x = 10;
let y = 5;
const z = 8;
```

```
{
var x = 2;
let y = 2;
}
document.getElementById("x").innerHTML = x;
document.getElementById("y").innerHTML = y;
```

```
z = 1;
document.getElementById("z").innerHTML = z;
</script>
```

```
</body>
</html>
```

What will be the values of `x`, `y`, `z`, respectively?

What if we move the statement `let y = 2;` out of the block `{ }`?

Variable Scope (1)

Variable scope tells where a variable can be referred to, used or valid

Local variables

- Declared in a block { }, such as a function, which can only be accessed within the block
- Try to access a local variable outside the block will produce unpredictable results

Global variables

- Declared not in any block. Can be accessed in the rest of the scripts
- Will be override by a local variable with the same name in a function

```
var x = 1;
function f() {
    x = x + 1;           // using the global variable outside the function
    console.log(x);
}
function g() {
    var x = 10;          // using the newly declared local variable
    console.log(x);      // will be gone when function g finishes

    var myNum;
    console.log(myNum);
}
```

What will happen if **f()** and **g()** are called, respectively?

Variable Scope (2)

Variable within a function

- **variable hoisting:** if a variable is declared within a function, JS would implicitly make its declaration at the top of the function (only declaration, not initialization)
- if a variable is declared in a function but without using keyword var, this variable is set to be a global variable

```
var num = 1;
function h() {
  console.log(num); // Q1: what is the value of num?
  var num = 10;     // the local variable has the same name as the global one
  console.log(num); // Q2: what is the value of num?
  date = "today"
}
h();                // call function h()
console.log(date);  // try to print the value of variable date
```

What will be the output on the browser console?

```
function h() {
  var num;
  console.log(num);
  num = 10;
  console.log(num);
  date = "today"
}
```

Variable Scope (3)

Variables cannot be used in HTML since HTML **CANNOT** read them.

They are used in JS only. On the other hand, JS can use HTML definition through the DOM (to be discussed later)

```
<script>var myvar = 1; </script>  
<!-- this is wrong -->  
<h1 id = "myvar">Try to get variable declared in JS</h1>
```

Statements

While writing a JavaScript program, the instructions can be provided as a list of statements to call a function/method, perform assignment operation or arithmetic computations, etc.

Examples of statements are:

Note that JavaScript is case sensitive!

`Alert("welcome to CS2204!");` will be **incorrect**

```
alert("welcome to CS2204!"); // pop up an alert window
var a = 10; // assign 10 to variable a
document.getElementById("welcome").innerHTML = "<h2>welcome to
CS2204!</h2>" /*find the element with an id "welcome" and insert an
innerHTML content to it*/
```

Note that `// ...` or `/* ...*/` are ways to comment in JavaScript

Array (1)

An **array** is a special variable, which stores **a set of values**

```
var nums;  
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];  
  
console.log(nums[0]);
```



[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

There are ten elements in this array

`nums[0]`, `nums[1]`, ..., `nums[9]`

The i^{th} array element is `nums[i-1]`

The range of the subscript `i` ranges from 0 to `array_size-1`

The location `nums[10]` is invalid. Array out of bound!

Array (2): Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Array</title>
</head>
<body>
  <script>
    var nums;
    nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    console.log(nums[0]);

    var colors = ['red', 'white', 'black'];
    console.log(colors[2]);
    colors[2] = 'green';
    console.log(colors[2]);
  </script>
</body>
</html>
```

What will the browser console output?

A summary of JS basics

JavaScript is a programming language that can provide instructions for a browser to dynamically generate content for a website or enhance the website interactivity

There are multiple data types in JS – number, string, Boolean, and Array are commonly used

Variable declaration is important before using it

Be mindful to the variable scope, otherwise it can be tricky

You will not be able to learn everything about HTML, CSS and JavaScript after only a few lectures, but it serves as a starting point such that you can explore other features on your own

Agenda

JavaScript Intro

- Execution environment
- Variable and scope
- Array

JavaScript Logic

- Expression and statement
- Operation and operator
- Functions

JavaScript Expressions

An ***expression*** is a combination of constants, variables, and function calls that **evaluate to a result**

Examples:

```
x = 3.0*4.0;
```

4 is a constant

x is a variable

```
y = 2.0 + x;
```

```
z = 5.0 + x/y - sqrt(x*3.0);
```

sqrt() is function call, x*3.0 is the parameter we pass into the function

JavaScript Statement (1)

Each instruction is a JavaScript **statement**

- each statement ends with a **semicolon ;** For example,

```
var x;  
sqrt(3);
```

- a single statement may span multiple lines
- multiple statements may occur on a single line if each statement is separated by a semicolon (;) - NOT a good practice though

Critical thinking: What's the difference between statement and expression?

JavaScript Statement (2)

Differing from regular program consisting of all instructions as a whole, JavaScript:

- may spread out as different **fragments**
- each fragment enclosed by `<script> ... </script>` in a HTML file or in a separate file (depending on whether embedded, inline and/or external scripts are used)

JavaScript Operators

An operator specifies an operation to be performed on some values , which are called the **operands** of the operator

e.g., in `x = a + b;` `a` and `b` are operands; `+` and `=` are the operators

Common JavaScript **Operators**

- Assignment Operator (i.e., `=`)
- Arithmetic Operators (e.g., `+`, `-`, `*`, `/`)
- Comparison Operators (e.g., `>`, `<`, `==`, `!=`)
- Logical Operators (e.g., `&&`, `||`, `!`)
- String Operator (e.g., `+`)
- Conditional Operator (e.g., `? :`)

Assignment Operator =

An assignment operator assigns a value to its **left operand** based on the value of its right operand. Generic form is

variable = expression;

= is an assignment operator that is different from the **mathematical equality** (which is == in JavaScript)

```
x + 10 = y;
```

```
2=x;
```

```
var a, b, c;
```

```
a = (b = 2) + (c = 3);
```

```
var a, b, c;
```

```
b = 2;
```

```
c = 3;
```

```
a = b + c;
```

Critical thinking:
Are the following expressions with an assignment operator valid?

Arithmetic Operators

Four basic operators: $+$, $-$, $*$, $/$

Modulus operator: $\%$, returns the division remainder

`x = 5 % 2; /* x is 1 */`

Increment operator: $++$, e.g., `x++` will increase the value of the variable `x` by 1

Decrement operator: $--$, e.g., `x--` will decrease the value of the variable `x` by 1

Arithmetic Operators: example

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Javascript Arithmetic Operators</title>
5   <script>
6     function init() {
7       var s = "";
8       var x,y,z;
9
10      x = 7%3;
11      s = s + "x = 7%3 =" + x + "<br /><br />";
12
13      y = 1;
14      s = s + "Initially y = " + y + "<br />";
15      y++;
16      s = s + "After y++, y = " + y + "<br /><br />";
17
18      z = 2;
19      s = s + "Initially z = " + z + "<br />";
20      z--;
21      s = s + "After z--, z = " + z + "<br /><br />";
22
23      document.getElementById("display").innerHTML = s;
24    }
25  }
26 </script>
27 </head>
28 <body onload="init();">
29   <!-- Page content begins here -->
30   <h1>Javascript Arithmetic Operators</h1>
31   <div id="display"></div>
32   <!-- Page content ends here -->
33 </body>
34 </html>
```

Javascript Arithmetic Operators

$x = 7\%3 = 1$

Initially $y = 1$
After $y++$, $y = 2$

Initially $z = 2$
After $z--$, $z = 1$

$7\%3$ is the remainder when 7 is divided by 3 so it is equal to 1 because $7 = 2 \times 3 + 1$

Arithmetic Operators on strings

When “+” meets string

```
12 <script type="text/javascript">
13   s1="Hi, I am a CityU student. My student number is: ";
14   s2="20005";
15   s3=25;
16   document.write(s1+s2+"<br>");
17   document.write(s2+ " minus " + s3 + " is: ");
18   document.write(s2-s3);
19   document.write("<br>" + s2+ " plus " + s3 + " is: ");
20   document.write(s2+s3);
21 </script>
```

+ is not addition for strings!

Hi, I am a CityU student. My student number is:20005
20005 minus 25 is: 19980
20005 plus 25 is: 2000525

Note this special case - **meaning of operator depends on the operands**. Most other programming languages don't have this behavior

Critical thinking

What would be the output if we change the value of s2 to “a”?

JavaScript: NaN (1)

Short for "Not-a-Number"

`isNaN()` method returns **true** if **a value** is Not-a-Number.

`Number.isNaN()` method returns **true** if the value is NaN **AND** the type is Number (it means that the type of value NaN is a number)

Critical thinking

Then what's the difference between `isNaN()` and `Number.isNaN()`?

JavaScript: NaN (2)

```
<!DOCTYPE html>
<html>
<body>
<h2>The isNaN() Method</h2>

<p>Is "Hello" NaN?</p>
<p id="textarea1"></p>
<p id="textarea2"></p>

<script>
let text = "Hello";
document.getElementById("textarea1").innerHTML =
'isNaN("Hello") = ' + isNaN(text);
document.getElementById("textarea2").innerHTML =
'Number.isNaN("Hello") = ' + Number.isNaN(text);
</script>

</body>
</html>
```

Critical thinking

What's the output in textarea1
and textarea2 respectively?

JavaScript: NaN (3)

```
console.log(Number.isNaN(NaN));  
  
console.log(Number.isNaN(Number.NaN));  
  
console.log(Number.isNaN(0 / 0));  
  
console.log(Number.isNaN(10 / "abc"));  
  
console.log(Number.isNaN("Hello"));  
  
console.log(Number.isNaN(42));  
  
console.log(Number.isNaN(true));  
  
console.log(Number.isNaN(null));  
  
console.log(Number.isNaN(undefined));  
  
console.log(Number.isNaN({}));
```

Critical thinking
What are the output in the browser console?

JavaScript `try` and `catch`

`try` statement: a block of code to be tested for errors while being executed

`catch` statement: a block of code to be executed, **if an error occurs** in the `try` block.

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}
```

The exception (`e`) is caught by the `catch` statement and an error message can be accessed by `e.message`

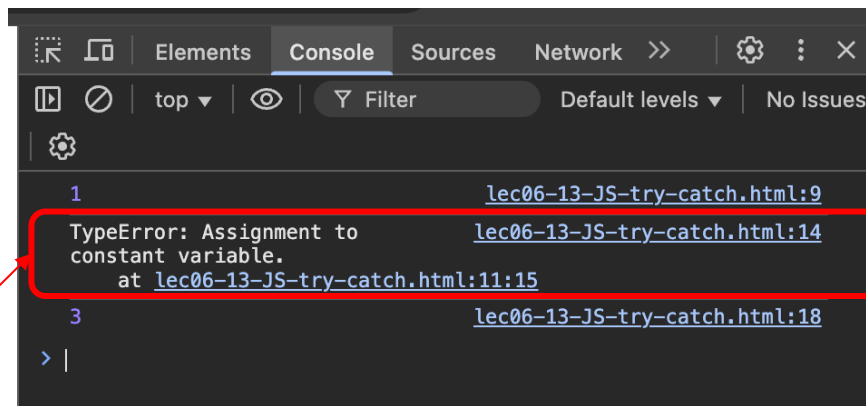
JavaScript `try` and `catch` example

`try` and `catch` statement is often used to prevent vulnerable statements from blocking the entire code

```
<script>
  const x = 1;
  console.log(x);
  try{
    x = 2;
    console.log(x);
  }catch(e){
    console.log(e);
  }

```

```
y = 3;
console.log(y);
</script>
```



What if we remove the `try {...}` and `catch (e) {...}` statement here?

Efficient Arithmetic Operators & Assignments

Operator	Example	Is the Same As
=	<code>x = y</code>	<code>x = y</code>
+=	<code>x += y</code>	<code>x = x + y</code>
-=	<code>x -= y</code>	<code>x = x - y</code>
*=	<code>x *= y</code>	<code>x = x * y</code>
/=	<code>x /= y</code>	<code>x = x / y</code>
%=	<code>x %= y</code>	<code>x = x % y</code>

Advantages

- **Conciseness**: reduce the code length
- **Efficiency/in-place modification**: optimize code execution without creating new variables

Increment/decrement Operators

- **Post-increment** and **post-decrement**: `k++` and `k--`
- **Pre-increment** and **pre-decrement**: `++k` and `--k`

```
var k=0, j, z;  
j=k++;  
z=++k;
```

Critical thinking

What are the results of `j` and `z` after the execution of the above code?

Comparison and Logical Operators (1)

Comparison operators accept **two** operands and compare them

The result is a **Boolean value**, i.e., true or false

Relational operators	Syntax	Example
Less than	<	x<y
Greater than	>	z>1
Less than or equal to	<=	b<=1
Greater than or equal to	>=	c>=2

Equality operators	Syntax	Example
Equal to	==	a==b
Not equal to	!=	b!=3

Comparison and Logical Operators (2)

Logical operators are used for combining **Boolean** (or **logical**) values to create **new logical values**

Logical AND (&&)

- return **true** if **both** operands are **true**, false otherwise (e.g., **a>1&&b<1**)

Logical OR (||)

- return **false** if **both** operands are **false**, true otherwise

Logical NOT (!)

- invert the Boolean value of the operand

x	y	x&& y
true	true	true
true	false	false
false	true	false
false	false	false

x	y	x y
true	true	true
true	false	true
false	true	true
false	false	false

x	!x
true	false
false	true

Comparison and Logical Operators (3)

Logical expressions can be `true` or `false` only

In JavaScript, if the value of an expression is one of the followings, this value can be treated as `false`; otherwise, the value is treated as `true`

- `false`
- `0`
- `""`
- `undefined`
- `NaN`
- `null`

JavaScript Comments

In **CSS**, comments can be placed between `/* ... */` and can **span multiple lines**

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Comments</title>
5     <style>
6       /* The following CSS style sets the corresponding div element
7        with color red */
8       #course {
9         color: red;
10      }
11    </style>
12    <script>
13      function init() { // this function is called after the webpage has b
14        /* The following statment dynamically replaces the content
15         of the corresponding div elemnt by the given string */
16        document.getElementById("course").innerHTML="<h2>Fundamentals of I
17      }
18    </script>
19  </head>
20  <body onload="init();">
21    <!-- Page content begins here -->
22    <h1>CS2204</h1>
23    <!-- the following div element's content is empty in the HTML
24         and will be assigned by Javascript after the webpage has been loaded
25    <div id="course"></div>
26    <!-- Page content ends here -->
27  </body>
28 </html>
```

In **JavaScript**, there are 2 ways to add comments:

- 1) comments can be placed between `/* ... */` and can **span multiple lines**
- 2) placed after `//` until the end of line so this **is a single line** comment. Note that **CSS does not support this single line comment style**

In **HTML**, comments can be placed between `<!-- ... -->` and can **span multiple lines**

JavaScript Functions

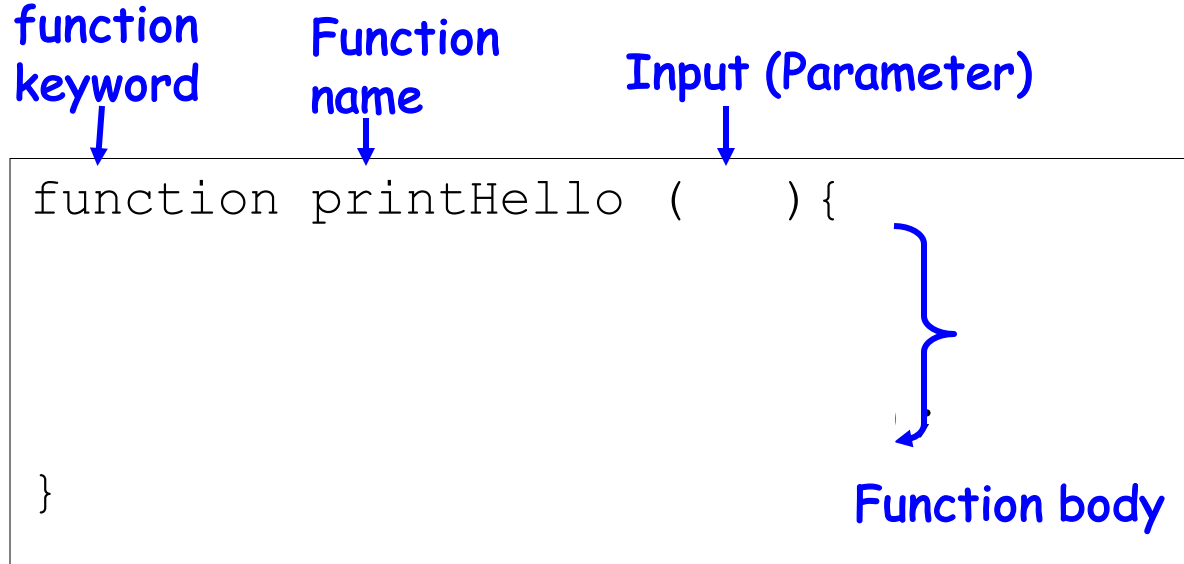
A function can be viewed as a “subprogram” that can be called by other codes. Commonly used for:

- Repeated use of a set of statements
- Event handler (will cover in detail later)

There are 2 types of function in JavaScript:

- Self-defined function - declared by the programmer
- Built-in function - defined in JavaScript, can be used directly without declaration

Function Declaration (1)



`n` is defined as a parameter, therefore there is no need to declare `n` in the function body again

Function Declaration (2)

```
function name (par1, par2, ... parN) {  
    statements  
    [return statement]  
}
```

A function must be declared **before** it can be used (called)
name - the function name, should follow the rules for variable declaration

par - the variable that the function expects to receive as input
parameter 1 to N are optional

Function Declaration (3)

```
function name (par1, par2, ... parN) {  
    statements  
    [return statement]  
}
```

statements - refer to the statements comprising the body of the function (the actual work to be done)

return statement - to specify the value to be returned (if any, the result) from the function, which is **optional**

Call a function

To make a function call, we only need to specify a function name and provide **parameter(s)** in a pair of ()

Function name	parameter
↓	↓
printHello	(3);

Parameter and arguments (1)

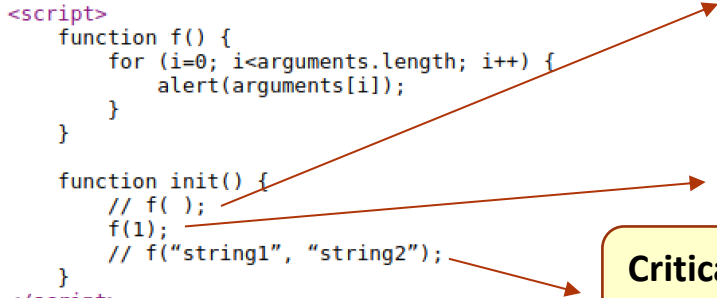
Parameter: a **variable** defined **in the function declaration** and represents the variable that **the function expects to receive as input**

JavaScript is loosely typed: no checking of parameter types (you must check on your own)

Argument: the **actual value** that is passed to the function when it's called

- arguments can be passed even they are NOT defined in declaration
- But the function should have an arguments object to get the actual arguments

```
6  <script>
7    function f() {
8      for (i=0; i<arguments.length; i++) {
9        alert(arguments[i]);
10     }
11   }
12
13   function init() {
14     // f( );
15     f(1);
16     // f("string1", "string2");
17   }
18 </script>
```



No output



Critical thinking: what's the output?

Special Characteristics of function

Function can be assigned to a variable with a **return** statement

```
function square(x) {return x*x;}  
var a = square(4); // pass 4 to the function named square  
var b = square; // assign variable b to the function named square  
var c = b(5); // pass 5 to the function named b
```

Function can have no name - **anonymous function**

```
var b = function(x) {return x*x;} /* assign variable b to the an  
anonymous function that expects to receive one parameter */  
var d = b(3);
```

Anonymous functions are commonly used to create a closure where allow the functions to access variables **outside** the function scope

Lecture Summary

- JS code consists of expressions, statements, and operators
- Expression is a combination of constants, variables, or functions calls that evaluate to a result
- Each instruction is a JS statement
- An operator specifies an operation to be performed on some values, which are called operands
- Arithmetic operators can be used for numbers and strings
- There are also comparisons and logical operators that returns true or false
- Functions are “subprograms” that can be called by other codes