CS2204 Fundamentals of Internet Applications Development

Lecture 8 JavaScript – Part 1

Computer Science, City University of Hong Kong

Semester A 2023-24

Topics

- Overview of programming
- JavaScript basics
- Control flow

Programming Overview

- Program is a set of instructions to tell a computer what to do. It usually involves:
  - 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
  - outputting results to screen, storage or in our case web page

- JavaScript is a bit different, which tells browsers what to do

Programming Overview

- Each **instruction** is a statement for :
  - defining or assigning values to variables
  - doing some operations (e.g., function calls, making decisions, etc.)
  - repeating some operations
- Writing a program is then 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

- **JavaScript** 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 between `<script>` `</script>` tags

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript First Example</title>
    <script>
        function myclick() {
            alert("Welcome to CS2204!");
        }
    </script>
  </head>
  <body>
  <!-- Page content begins here -->
    <h1>CS2204</h1>
    <h2 onclick="myclick();">Click Me</h2>
  <!-- Page content ends here -->
  </body>
</html>
```

In JavaScript, you define a function using the keyword **function**, and you enclose a block inside the function using the curly brackets {}

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

Code Example: lec08-01-JS-first-example.html

JavaScript

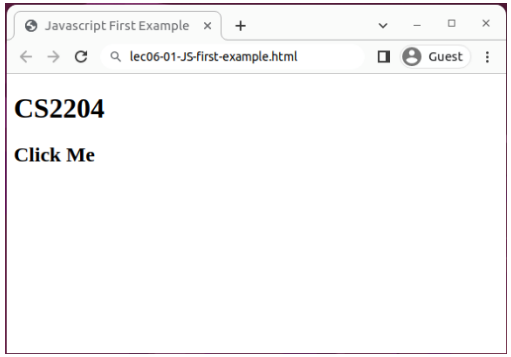
- Website interactivity can be enhanced by detecting a user event and defining the corresponding **event handler** to perform certain action
  - E.g., the following program has an event handler that detects if the user clicks on the h2 heading "Click Me" and then calls the function `myclick()` to pop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

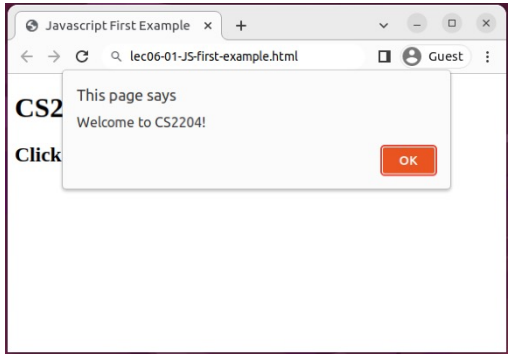
```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>Javascript First Example</title>
6      <script>
7          function myclick() {
8              alert("Welcome to CS2204!");
9          }
10     </script>
11   </head>
12   <body>
13   <!-- Page content begins here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

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: lec08-01-JS-first-example.html

How to include and run JavaScript

Two main ways to include JavaScript in a web page

- Embedded
- External

JS code can be executed when

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

Execution order is according to the order presented to the browser

Embedded Script

- Put in `<script> </script>` tags directly
  - scripts that contain functions usually go in the head section
    - it ensures that scripts are loaded before the function is called
    - scripts can be used by elements in the entire web page

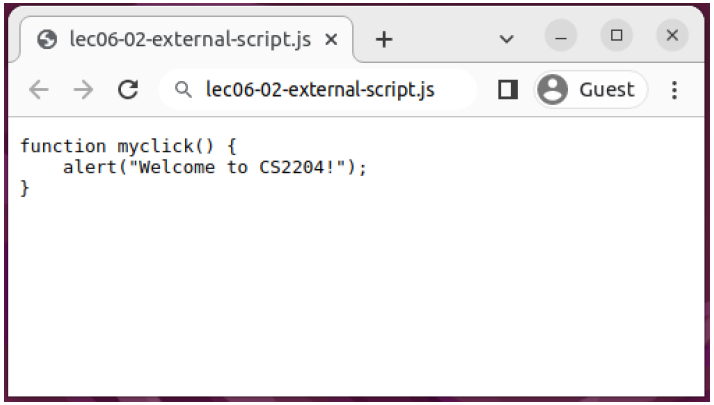
```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>Javascript First Example</title>
6      <script type="text/javascript">
7          function myclick() {
8              alert("Welcome to CS2204!");
9          }
10     </script>
11   </head>
12   <body>
13   <!-- 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: lec06-01-JS-first-example.html

External Script

- Scripts are 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)

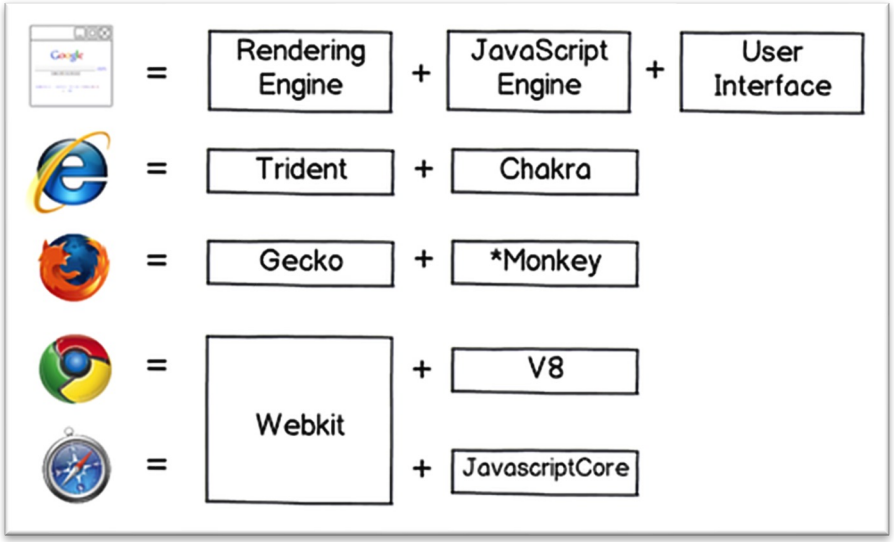
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript First Example</title>
    <script type="text/javascript" src="./lec06-02-external-script.js"></script>
  </head>
  <body>
  <!-- Page content begins here -->
    <h1>CS2204</h1>
    <h2 onclick="myclick();">Click Me</h2>
  <!-- Page content ends here -->
  </body>
</html>
```



Code Example: lec08-02-JS-external-script.html

Execution Environment

- Two **main** engines
  - Rendering engine processes HTML/CSS to render webpages
  - Script engine (e.g., Node.js) processes scripts, providing
    - operating system API
    - full networking features
    - can use JS to implement a Web Server



Execution Environment

- JavaScript contains three main components:
  - ECMAScript
    - For JavaScript syntax
  - DOM (Document Object Model)
    - For webpage operations
  - BOM (Browser Object Model)
    - For browser operations

How is JavaScript used in this course?

- Program Web pages
  - do something HTML & CSS (especially) **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

How is JavaScript taught in this course?

- **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

Topics

- Overview of programming
- JavaScript basics
- Control flow

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
  - e.g., *var, function, while, for, if, break, continue,* etc.

Variable

- Variable is one type of identifiers to store data values
  - Defined by the keyword var, e.g., var myMsg;
  - The name of this variable (or identifier) is 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
  - Valid names: myMsg1, my_msg, _myMsg, $myMsg, MyMsg
  - Invalid names: 3D_point, my-msg, my msg
  - hyphen (-) and space ( ) are not allowed

Variable Declaration

- Variable needs to be declared before we use it
- Declaration only:
  - var alertMsg;
  - var age1, age2, age3, AlertMsg;

Value of each variable 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

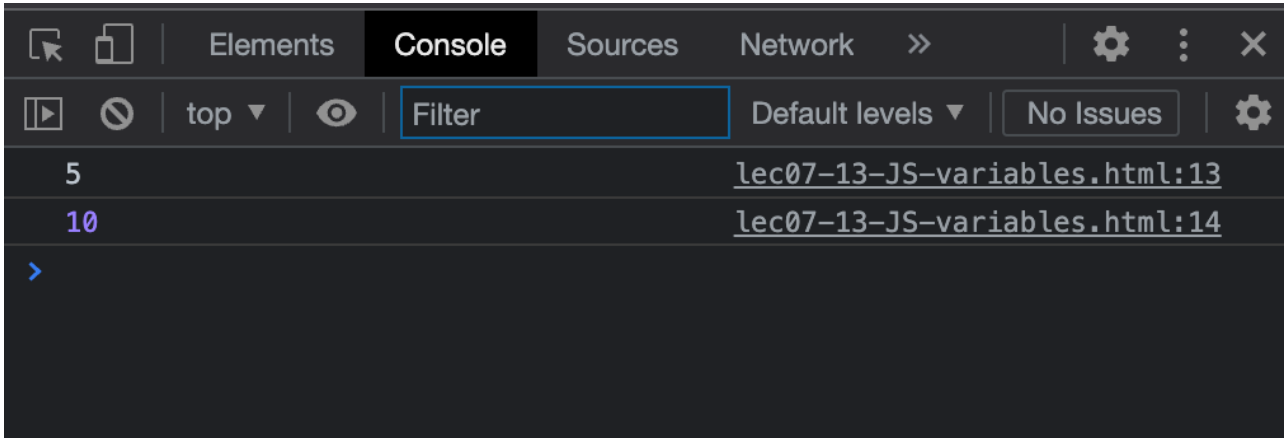
Variable names are case sensitive, e.g., age and Age are different variable names

- 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: lec08-03-JS-variable.html

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



Data Types

- JavaScript has 3 basic (primitive) data types and other complex types:
  - **Boolean** - can contain **two values** only: *true* or *false*
  - **Number**
    - Integer - represented in JavaScript in *decimal* - 33, *hexadecimal* - 0x7b8 or 0X395, and *octal* - 071
    - 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 complex data types** (discussed later)

Critical thinking: Why there are different data types?

Data Type – Number

- Different **number systems**
  - Decimal: contains **single digits** 0 ~ 9
  - Octal: contains **single digits** 0 ~ 7, starting with 0o
    - e.g., `var num = 0o10;`
  - Hexadecimal: contains **single digits** 0~9 A~F, starting with 0x
    - e.g., `var num = 0x11;`
- Some special values
  - `Number.MAX_VALUE`: the maximum value
  - `Number.MIN_VALUE`: the minimum value
  - `Infinite`
  - `-Infinite`
  - `NaN`: Not a Number

Code Example: lec08-04-JS-numbers.html

Data Type – String

- Enclosed by double quote (“) or single quote (‘)
- Length of a string
  - ```
    var str = 'hello world';
    console.log(str.length);
    ```
- String concatenation using +
  - when a + b, if either a or b is a string, the meaning of + is not addition. It views them as two strings and concatenates them
  - Examples
    - console.log(‘hello’ + ‘world’);
    - console.log(‘CS’ + 2204);
    - console.log(‘220’ + 4);
    - console.log(220 + 4);
    - …

Code Example: lec08-05-JS-string.html
    ```

Data Type – Undefined

- If a variable is declared **but** not initialized yet, its data type is undefined
  - `var a; console.log(a);`
- Obtain the data type of a variable using `typeof`
  - `var num = 1; console.log(typeof num);`
  - `var str = 'hello'; console.log(typeof str);`
  - `var value = true; console.log(typeof value);`

Code Example: lec08-06-JS-undefined.html

Data Type Conversion

- Conversion to string, by using
  - toString()
    - e.g., `var num = 2204; var str = num.toString();`
  - String()
    - e.g., `var num = 2204; var str = String(num);`
  - +
    - e.g., `var num = 2204; var str = num + '';`

Code Example: lec08-07-JS-conversion.html

Data Type Conversion

- Conversion to number
  - parseInt()
    - e.g., `var num = parseInt('2204');`
    - e.g., `var num = parseInt('200px');`
  - parseFloat()
    - e.g., `var num = parseFloat('3.14');`
  - Number()
    - e.g., `var num = Number('2204');`
  - -, *, /
    - e.g., `var num = '2204' - 4;`
    - e.g., `var num = '2' * 5;`
    - e.g., `var num = '10' / '5';`
- Using `prompt()`, the obtained user input is a string
  - Add two numbers obtained from prompt()

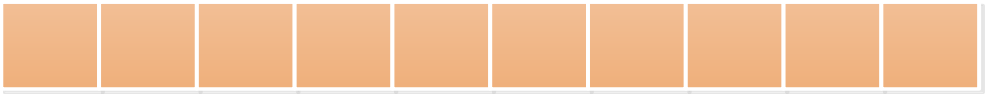
Code Example: lec08-07-JS-conversion.html

Array

- 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]`

```
var colors;
colors = ['red',
          'black',
          'white'];

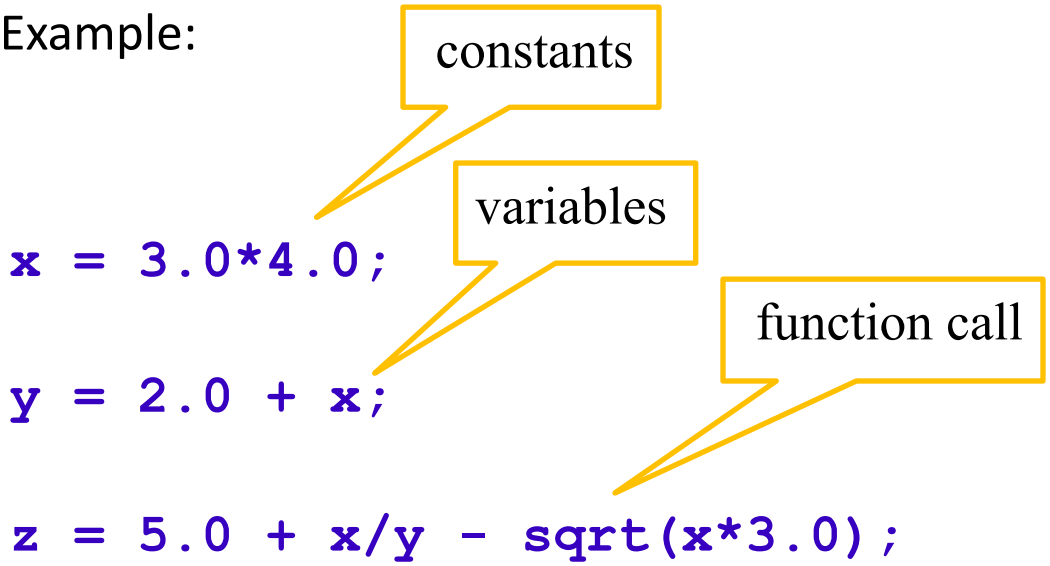
console.log(colors[1]);
colors[1]='green';
console.log(colors[1]);
```

Code Example: lec08-08-JS-array.html

Expressions

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

- Example:



Statement

- Each instruction is a JavaScript **statement**
  - each statement is ended with a semicolon ;
  - 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
- Usually, all instructions as a whole form a program but JavaScript in Web pages is a bit different
  - usually not as one single program
  - may spread out as different fragments
  - each fragment enclosed by <script> ... </script> or in a separate file (depending on whether embedded, inline and/or external scripts are used)

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

Operators

- An operator specifies an operation to be performed on some values
  - These values are called the operands of the operator
- Common JavaScript Operators:
  - Arithmetic Operators, e.g., +, -, * /, etc.
  - Assignment Operator
  - Comparison Operators
  - Logical Operators
  - …

Arithmetic Operators

- JavaScript arithmetic operators:
  - The 4 operators +, −, *, / are intuitive
  - % is the modulus operator. It returns the division remainder, e.g., 5 % 2 = 1
  - ++ is the increment operator, x++ will increase the value of the variable x by 1
  - −− is the decrement operator, x−− will decrease the value of the variable x by 1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        var a = prompt('Input a');
        var b = prompt('Input b');
        console.log(a + '%' + b + " is: " + a % b);

        a++;
        console.log(a);
    </script>
</head>
```


Code Example: lec08-09-JS-arithmetic-operators.html

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?

Assignment Operator

- Efficient assignments

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

Comparison and Logical Operators

- 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
Strict equal to	===	a === b
Not equal to	!=	b != 3

Code Example: lec08-10-JS-compare-logic.html

Comparison and Logical Operators

- Logical operators are used for combining **Boolean** (or **logical) values** and 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

- **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
- `Boolean()` can convert a value to Boolean. The above values will be converted to false; other values will be converted to true
  - `Boolean(NaN);`
  - `Boolean('2204');`

Comments

- HTML, CSS and JavaScript allow programmers to insert comments in the code but their syntax differs
- Comments will be ignored by the browser when the webpage is displayed but it is a good practice to insert comments to document what the code logic, such that it will be easy for others to understand your code or for you to revisit your code later

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

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

1) Similar to CSS, JavaScript comments can be placed between /* and */ and can span multiple lines
2) JavaScript comments can also be 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

```
<!DOCTYPE html>
<html>
  <head>
    <title>Comments</title>
    <style>
        /* The following CSS style sets the corresponding div element
           with color red */
        #course {
            color: red;
        }
    </style>
    <script>
        function init() {   // this function is called after the webpage has been loaded
            /* The following statment dynamically replaces the content
              of the corresponding div elemnt by the given string */
            document.getElementById("course").innerHTML="<h2>Fundamentals of Internet Applications Development</h2>";
        }
    </script>
  </head>
  <body onload="init();">
  <!-- Page content begins here -->
    <h1>CS2204</h1>
    <!-- the following div element's content is empty in the HTML
         and will be assigned by Javascript after the webpage has been loaded -->
    <div id="course"></div>
  <!-- Page content ends here -->
  </body>
</html>

```

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

Code Example: lec08-11-JS-comment.html

Flow Control Statements

Common Flow Control Statements

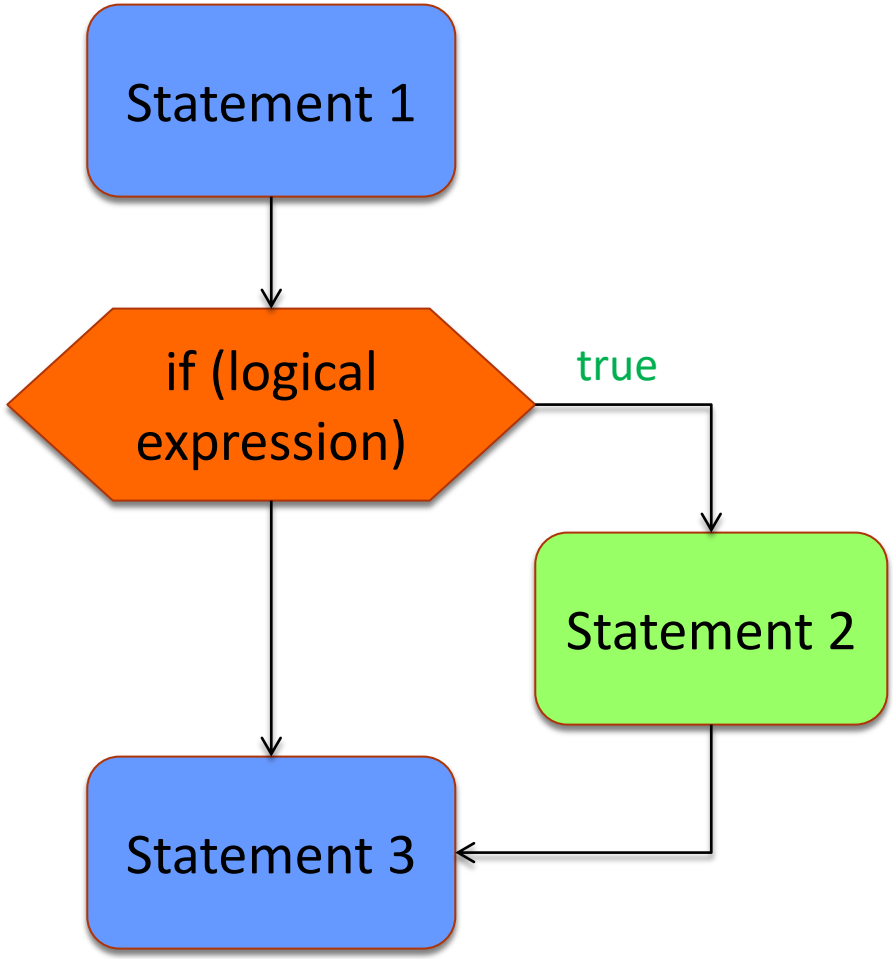
- if-else statement
- switch statement
- for statement
- while statement
- do-while statement
- break statement
- continue statement
- return statement
- block statement

if-statement: One-Way Conditional (1)

Execute a statement (or a block of statements) if a specified condition is true

```
statement1;
if (condition)
    statement2;
statement3;
```

```
statement1;
if (condition){
    statement2;
    statement22;
    …
}
statement3;
```



if-statement: One-Way Conditional (2)

Execute a statement (or a block of statements) if a specified condition is true

The variable `s` is initialized to be "`This is the end of summer`"

The variable `input` is assigned as a string (text) from the output of the **prompt function**. The expression `Number(input)` converts the string `input` to its numerical value so that it can be manipulated as a number

```
<script>
    function init() {
      var s = "This is the end of summer";
      var input = prompt("Enter a month");
      var month = Number(input);

      if(month>1 && month<=5) {
        s = "This month is in Semester B\n";
      }
      if(month ==12 || month ==1 ){
        s = "This is a winter month.\n";
      }
      if(month>=9) {
        s = "This month is in Semester A.\n";
      }
      if(month!=8) {
        s = "This month is the middle of summer.\n";
      }

      alert(s);
    }
</script>
```

The expression (month>1 && month <=5) is true if month > 1 AND month <=5, i.e., when month is equal to 2, 3, 4, 5

The expression (month==12 || month ==1) is true if month is equal to 12 OR 1, i.e., when month is equal to 12, 1

The expression (month>=9) **will be executed** if month is 9, 10, 11

The expression (month!=8) **will be executed** if month is 6, 7

Code Example: lec08-12-JS-if.html

if-else: Two-Way Conditional (1)

Execute a statement (or a block of statements) if a specified condition is true. Otherwise, another statement (or a block of statements) will be executed

```
if (condition)
    statement1;
else
    statement2;
```

```
if (condition){
    statement1;
    statement2;
    …
} else {
    statement3;
    statement4;
    …
}
```

if-else: Two-Way Conditional (2)

Execute a statement (or a block of statements) if a specified condition is true. Otherwise, another statement (or a block of statements) will be executed

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript Two-Way Conditional</title>
    <script>
      function init() {
            var p, s;
            s = "";
            p = prompt("Enter a positive integer: ");
            if(Number(p)>0) {
              s = p +" is a positive integer\n";
            } else {
              s = p + " is NOT a positive integer\n";
            }
            s = s + "Two-way conditional example.";
            alert(s);

        }
    </script>
  </head>
  <body onload="init();">
    <!-- Page content begins here -->
    <h1>Two-Way Conditional</h1>
    <!-- Page content ends here -->
  </body>
</html>
```

Code Example: lec08-13-JS-if-else.html

```
if (condition){
     statement1;
     statement2;
     …
} else {
     statement3;
     statement4;
     …
}
```

\n specifies that a line break should be added such that the subsequent text will be displayed in a new line when shown by the alert() function

Multiple else-if (N-Way Conditional)

You can have as many nested "else if" statements as you want.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Javascript N-Way Conditional</title>
    <script>
        function init() {
            var p, s, cgpa;
            s = "";
            p = prompt("What is the CGPA");
            cgpa = Number(p);
            if (cgpa >= 3.5)
              s = "1st Class Honours";
            else if (cgpa >= 3.0)
              s = "Upper 2nd Class Honours";
            else if (cgpa >= 2.5)
              s = "Lower 2nd Class Honours";
            else if (cgpa >= 2.0)
              s = "3rd Class Honours";
            else if (cgpa >= 1.7)
              s = "Pass";
            else
              s = "No Award";
            alert(s);
        }
    </script>
  </head>
  <body onload="init();">
  <!-- Page content begins here -->
    <h1>N-Way Conditional</h1>
    <!-- Page content ends here -->
  </body>
</html>
```

```
if (logical expression 1)
        //action for true
        statement a;
else if (logical expression 2)
        //action for true
        statement b;
else if (logical expression 3)
        //action for true
        statement c;
… …

else
        //action for false
        statement;
```

CGPA	Boolean Expression	Award Classification
3.5 or above	CGPA>=3.5	1st Class Honours
3.0-3.49	CGPA>=3.0 AND CPGA<3.5	Upper 2nd Class Honours
2.5-2.99	CGPA>=2.5 AND CPGA<3.0	Lower2nd Class Honours
2.0-2.49	CGPA>=2.0 AND CPGA<2.5	3rd Class Honours
1.7-1.99	CGPA>=1.7 AND CPGA<2.0	Pass
<1.7	CPGA<1.7	No Award

Code Example: lec08-14-JS-N-way-conditional.html

switch

A multi-branch flow control is easier to follow that multiple (nested) statements

- Execute statements associated with the case where its label matches the expression's value; if no matching label is found, the default case will be executed

- Break statement: ensures the program breaks out of switch once the matched statement is executed
  - If there is no break statement, execution "*falls through*" to the next statement in the succeeding case

```
switch (expression) {
    case label1:

    case label2:

    ...
    case labelN:

    default:

}
```

switch

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        var input = prompt('Input your 4-digit course code');
        var str = 'You are taking CS';

        switch(input) {
            case '1102':
                str += input;
                break;
            case '2204':
                str += input;
                break;
            default:
                str = 'You are not taking any CS courses';
        }

        console.log(str);
    </script>
</head>
<body>

</body>
</html>
```

Code Example: lec08-15-JS-switch.html

Personal Webpage (Verson-07)

- Add neon light animation to the top of the page
  - **Step 1**: Add a div element to the top of the page
  - **Step 2**: Use linear-gradient to create the gradient effect for the block
  - **Step 3**: Change the color and add the "*box-shadow*" when the top of the page is hovered
  - **Step 4**: Add a "*SwingBackAndForth*" animation function to simulate the neon light effect