CS2204 Fundamentals of Internet Applications Development

Lecture 8 JavaScript – Part3

Computer Science, City University of Hong Kong

Semester B 2024-25

About Mid-term

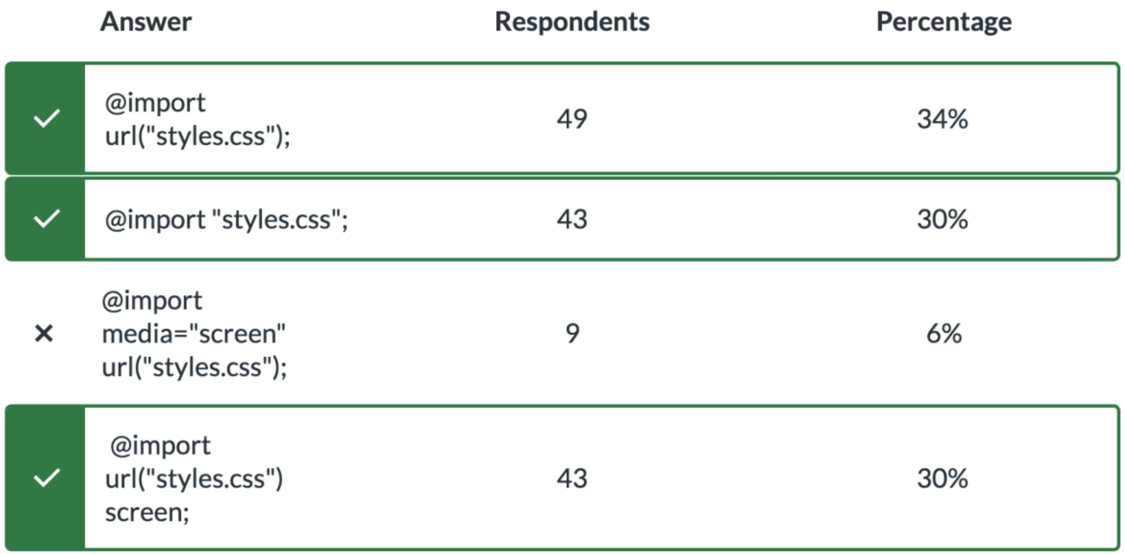
Three students did not take the exam or apply for a make-up: please apply mitigation: https://www.cityu.edu.hk/arro/asmt/mitg_main.htm and upload **solid justification materials (e.g., medical)** if you still want to make up the mid-term by **today (within 5 working days of the exam).**

We will release mid-term grade after the make-up is done and graded

Questions with high error rate will be covered in the last lecture as part of the course review

Post-lab Quiz 4 Review

Which of the following CSS `@import` statement are valid for importing media files? (See Lec 04 page 12)



	Answer	Respondents	Percentage
✓	@import url("styles.css");	49	34%
✓	@import "styles.css";	43	30%
✗	@import media="screen" url("styles.css");	9	6%
✓	@import url("styles.css") screen;	43	30%

Post-lab Quiz 5 Review

Which of the following selects all the links that are visited by the user?

A. `a:visited {}`
B. `a::visited {}`

Post-lab Quiz 5 Review

Given a paragraph within the `<body></body>` section of a HTML file: `<p>Hello World.</p>`.

Give this paragraph a solid border in green with a width of 3px on the top/bottom sides and 5px on the left/right sides

```
A. p { border: 3px 5px solid green; }
B. p { border: 5px 3px solid green; }
```

Post-lab Quiz 5 Review

Given a paragraph within the `<body></body>` section of a HTML file: `<p>Hello World.</p>`.

Make the paragraph take up a space commensurable to its length rather than the entire row

```
A. p {display: inline;}
B. p {display: block;}
```

Question time

1. Given the following code, what is the output?

```
var x=0, y, z;
y=x++;
console.log (y);
z=++y;
console.log (z);
```

2. How about the following output?

```
var x=0, y, z;
y=x++;
z=++y;
console.log (z);
console.log (y);
```

Question time

3. Given the following code, what is the output?

```
var s1 = 25;
var s2 = "1000";
var s3 = "abc";
console.log (s1 + s2);
console.log (s1 + s3);
console.log (s2 + s3);
```

4. How about **adding one more variable** `s4` to the above code:

```
var s4 = true;
console.log (s1 + s4);
console.log (s2 + s4);
console.log (s3 + (s1+s4));
```

Question time

5. Given the following code, what is the output?

```
var s1 = 25;
var s2 = “1000”;
var s3 = “hello”;
console.log (isNaN(s1));
console.log (isNaN(s2));
console.log (isNaN(s3));
```

6. How about Number.isNaN()?

```
console.log (Number.isNaN(s1));
console.log (Number.isNaN(s2));
console.log (Number.isNaN(s3));
```

`Number.isNaN()` method returns true only if the value is NaN **and** the type is Number

Agenda

Review of JS Conditionals and Loops

JavaScript functions

Object-oriented programming

DOM and Events

Types of Conditionals

One-way conditional: if

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

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

Two-way conditional: if-else

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

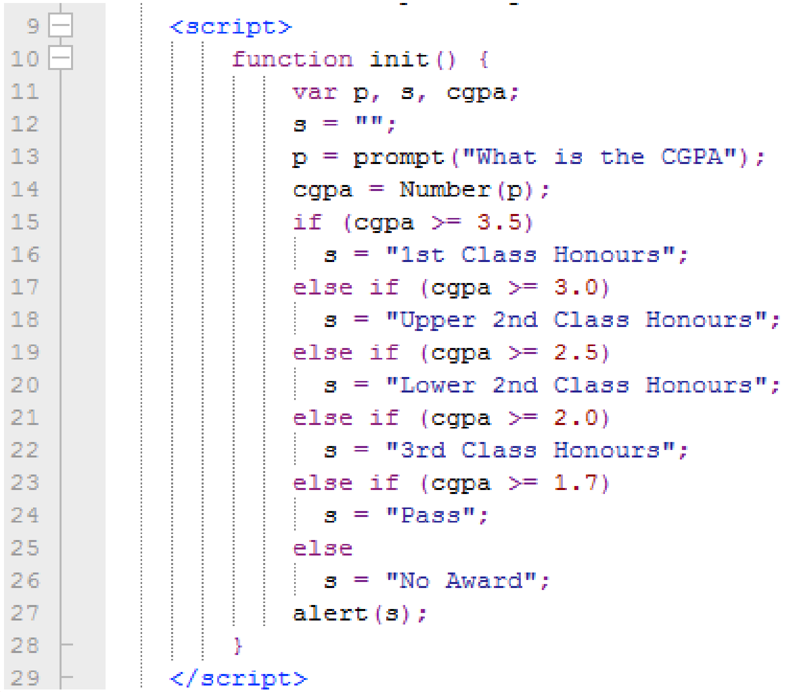
```
if (condition){
     statement1;
     statement2;
     …
} else {
     statement3;
     statement4;
     …
}
```

N-way conditional: multiple else-if

```
if (condition)
     statement1;
else if
     statement2;
else if
     statement3;
else if
     statement4;
        ...
```

Multiple else-if (N-Way Conditional)

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



```
<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>
```

```
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: lec07-10-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

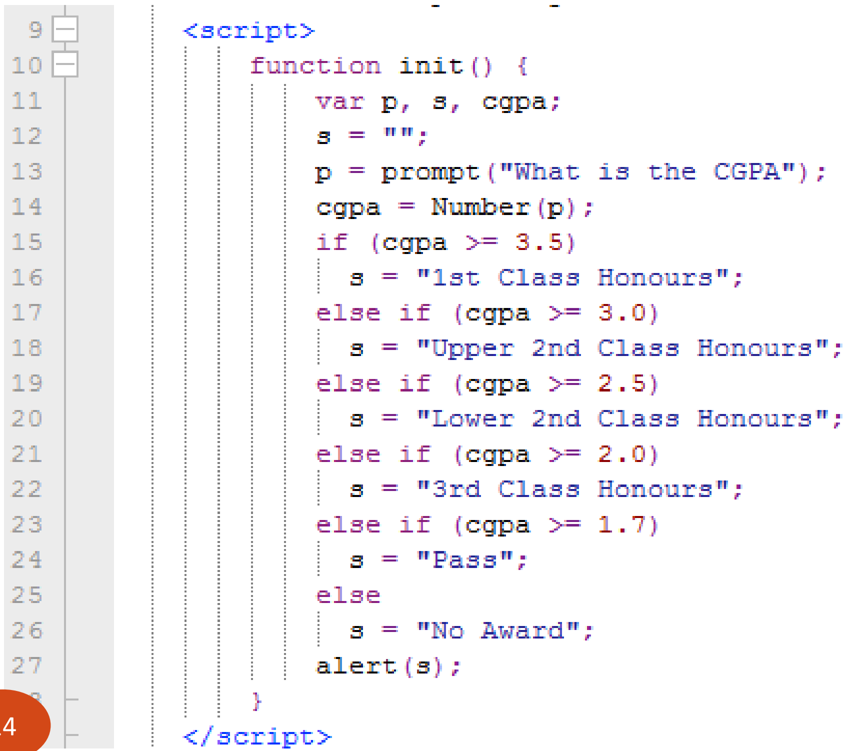
- If there is no break statement, execution "*falls through*" to the next statement in the succeeding case

```
switch (expression) {
  case label1:


  case label2:


  ...
  case labelN:


  default:


}
```

Code Example: lec07-11-JS-switch.html

switch - example

How to use **switch** to rewrite the following code?



```
<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>
```

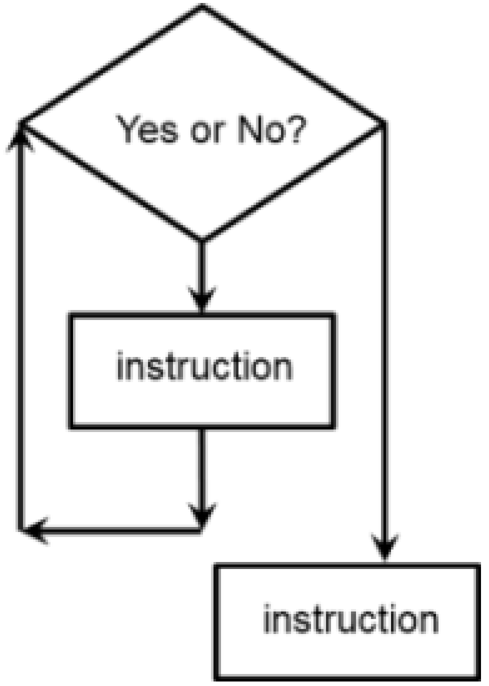
```
function init() {
    let category = Math.round(CGPA);
    if (CGPA < 1.7)
        category = 0;

    switch (category) {
        case 4:
          return '1st Class Honours';
        case 3:
          return 'Upper 2nd Class Honours';
        case 2:
          return 'Lower 2nd Class Honours';
        case 1:
          return '3rd Class Honours';
        case 0:
          return 'No Award';
        default:
          return 'Pass'; }
}
```

Loop

A **Loop** is a construct that allows you to **repeat a block of code multiple times.**

e.g., a countdown timer

A loop often constitutes of a set of instructions that the computer follows over and over until a certain condition is met.



Types of Conditionals

for-loop

```
for(expr1; expr2;
expr3)
{
loop statements;
}
```

while-loop

```
expr1;
while(expr2)
{
    loop statements;
    expr3;
}
```

do-while loop

```
expr1;
do{
loop statements;
    expr3;
}
while (expr2);
```

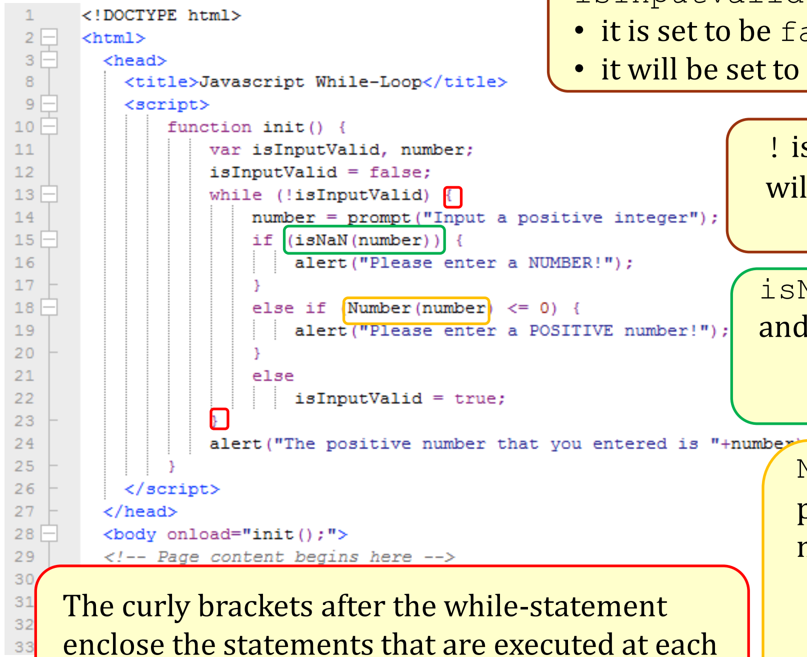
The loop statements is executed as long as **expr2** is true; when **expr2** becomes false, the loop ends.

expr1: Executed before entering the loop, often used for variable initialization

expr3: For each iteration, expr3 is executed after executing the loop body. Often used to **update** the counter variables (e.g., ***i++***).

Review: while-Loop

The while-loop is used to carry out a task repeatedly as long as a continuation condition is true



```
<!DOCTYPE html>
<html>
  <head>
    <title>Javascript While-Loop</title>
    <script>
      function init() {
        var isInputValid, number;
        isInputValid = false;
        while (!isInputValid) {
          number = prompt("Input a positive integer");
          if (isNaN(number)) {
            alert("Please enter a NUMBER!");
          }
          else if (Number(number) <= 0) {
            alert("Please enter a POSITIVE number!");
          }
          else
            isInputValid = true;
        }
        alert("The positive number that you entered is "+number
      }
    </script>
  </head>
  <body onload="init();">
  <!-- Page content begins here -->
```

`isInputValid` is a Boolean variable which has value `true` or `false`
- it is set to be `false` initially
- it will be set to `true` if the user inputs a positive number

`!` is the **NOT** operator and will negate its subsequent Boolean expression

`isInputValid`	`!isInputValid`
`true`	`false`
`false`	`true`

`isNaN()` returns `true` if the current input is NOT a number and `false` if it is a number, e.g.,

`isNaN(234)=false`

`isNaN("abc")=true`

`Number` is JS built-in function that converts the given parameter to a number according to its value such that numeric calculations can be applied, e.g.,

`Number("123")=123`

`Number("123")+1 = 124`

Critical thinking: what is the value of the expression `"123"+1 ?`

The curly brackets after the while-statement enclose the statements that are executed at each iteration of the while-loop

Code Example: lec07-13-JS-while-loop.html

Agenda

Review of JS Conditionals and Loops

JavaScript functions

Object-oriented programming

DOM and Events

JavaScript Functions

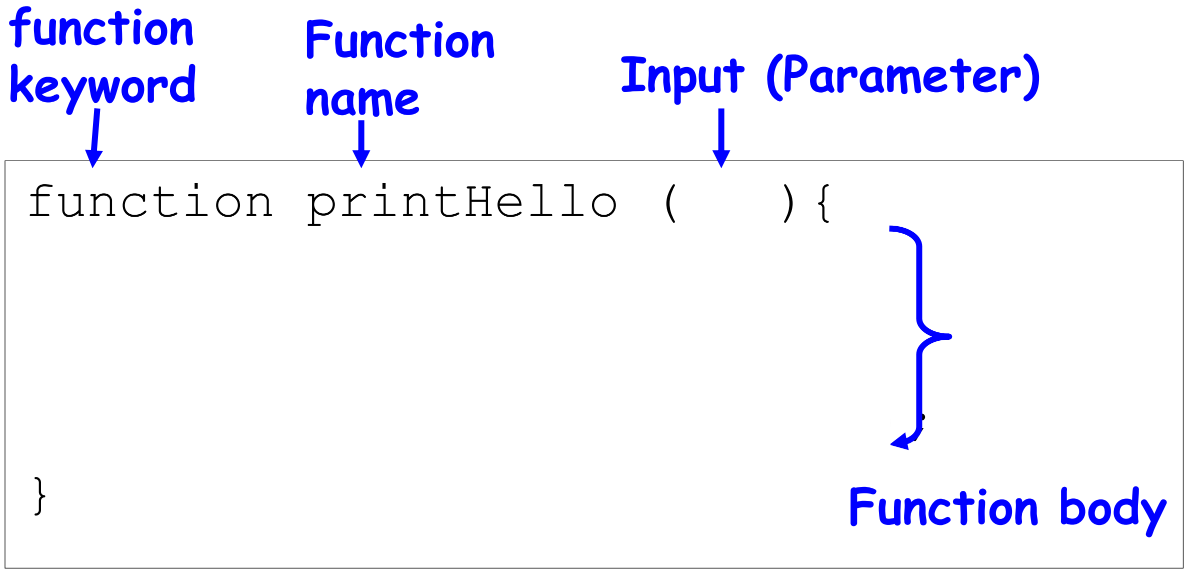
A function can be viewed as a "subprogram" that can be called by other codes. Commonly used for:

- Repeated use of a set of statements
- Event handler

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)



```
function printHello (    ){



}
```

<span>**n**</span> 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)

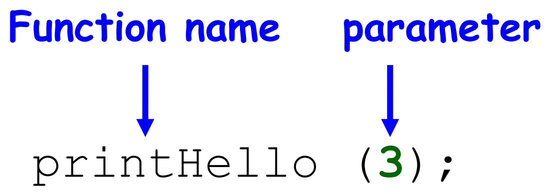
```
function name (par1, par2, ... parN) {
                statements
                [return statement]
        }
```

`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 ()



```
printHello (3);
```

Code Example: lec08-01-JS-function.html

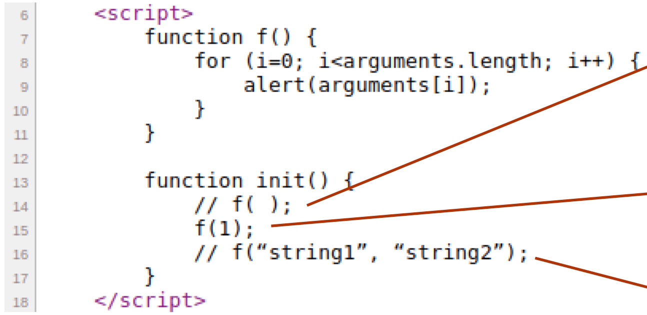
Parameter and arguments (1)

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

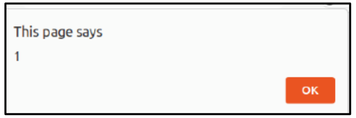
- arguments can be passed even they are NOT defined in declaration
- But the function should have an arguments object to get the actual arguments



```
<script>
    function f() {
        for (i=0; i<arguments.length; i++) {
            alert(arguments[i]);
        }
    }

    function init() {
        // f( );
        f(1);
        // f("string1", "string2");
    }
</script>
```

Code Example: lec08-02-JS-function-parameter.html

No output



```
This page says
1
OK
```

Critical thinking: what's the output?

Parameter and arguments (2)

Given the following code, what is the output?

```
  <script>
      function sum() {
      var total = 0;
      for (var i = 0; i < arguments.length; i++) {
        if (i%2 == 0)
          total += arguments[i];
      }
      return total;
    }

    function init(){
      console.log(sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));
    }
  </script>
</head>
<body onload="init();">
</body>
```

In JavaScript, the arguments variable is a **default variable** available within function bodies. It is an **"array-like" object** that contains all the arguments passed to a function when it is invoked.

A. 25

B. 30

- The variable ***i*** is the ***index*** of the argument, not its value
- Note that the index of the ***first*** argument is ***0***

Code Example: lec08-02-JS-function-parameter.html

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
```

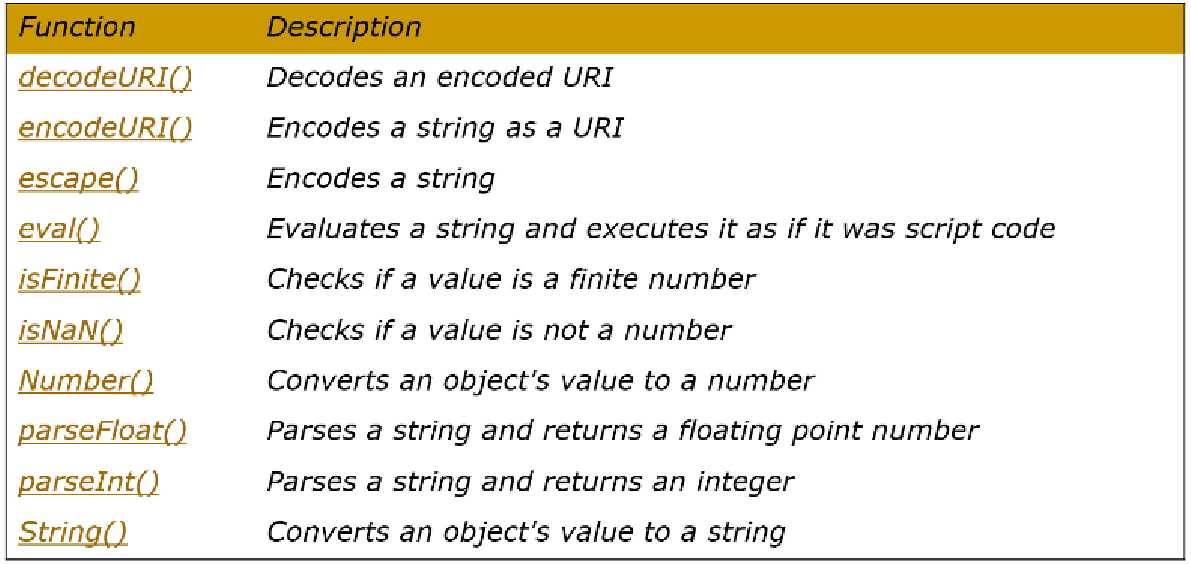
Function can have no name - **anonymous function**

```
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

Built-in JS function

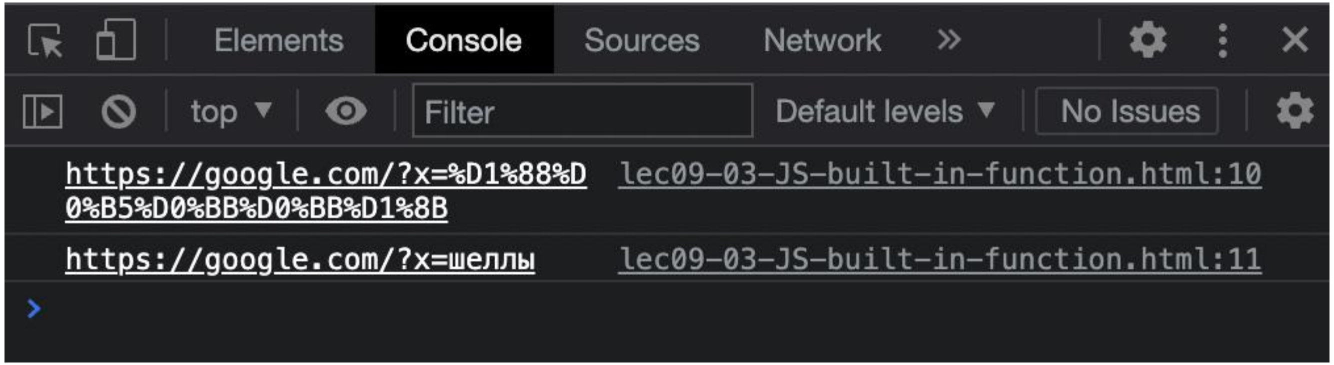
- A function needs declaration is a **self-defined function**.
- A **built-in function** is a function that is available as part of the JavaScript standard library, **without** explicitly declaring it.



Function	Description
decodeURI()	Decodes an encoded URI
encodeURI()	Encodes a string as a URI
escape()	Encodes a string
eval()	Evaluates a string and executes it as if it was script code
isFinite()	Checks if a value is a finite number
isNaN()	Checks if a value is not a number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string

Built-in function: `encodeURI()` and `decodeURI()`

```
const encoded = encodeURI('https://google.com/?x=шеллы') /*
Characters such as spaces, special characters, and reserved
characters (except :/?#[]@) are encoded. */
console.log(encoded);
Console.log(decodeURI(encoded)); /* Reverses the encoding
performed by encodeURI().*/
```



```
https://google.com/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B    lec09-03-JS-built-in-function.html:10
https://google.com/?x=шеллы                             lec09-03-JS-built-in-function.html:11
```

Code Example: lec08-03-JS-built-in-function.html

Built-in function: `eval()`

eval() *Evaluates a string and executes it as if it was script code*

Often used evaluate **mathematic expressions** or dynamically generate and execute code based on user input or other runtime conditions.

```
var s1 = "10";
var s2 = " + 20";
console.log(s1 + s2);
console.log(eval(s1 + s2));
```

Output:

```
10 + 20
30
```

- s1 + s2 simply concatenates the two strings together
- eval (s1 + s2) evaluates the concatenated string as JS code, resulting the expression 10 + 20

Code Example: lec08-03-JS-built-in-function.html

Built-in function: `parseFloat()` and `parseInt()`

parseFloat()	*Parses a string and returns a floating point number*
parseInt()	*Parses a string and returns an integer*

- Only the **first** found number is returned
- Leading and trailing spaces are ignored
- Return NaN if the first character cannot be converted

```
var s3 = "10, 20, 30";
var s4 = "40 years ago";
var s5 = "He was 50";
var s6 = "60.9999";
```

What's the output of `parseInt(s3)`,`parseInt(s4)`, `parseInt(s5)`,`parseInt(s6)`? And how about `parseFloat()`?

Code Example: lec08-03-JS-built-in-function.html

Built-in JS function: notes

Built-in functions are convenient tools that allow developers to **easily get an output given the input**, so that they do not need to reinvent the wheels

They are **standardized**, **reliable**, and **extendable** than self-defined functions serving for the same purposes

Functions that we commonly call, such as `alert()`, `prompt()`, `console.log()`, `isNaN()` are also built-in functions

Agenda

Review of JS Conditionals and Loops

JavaScript functions

Object-oriented programming

DOM and Events

What is an **object**?

In programming languages, an object is a fundamental concept that represents a specific instance of a class or a data structure.

Objects are used to model real-world entities, concepts, or abstract constructs within a program. They encapsulate data and related operations, allowing for modular and organized programming. For example,

a **`car`** can be represented as an object with *properties* such as `color`, `brand`, `model`, and *behaviors* (i.e., functions) such as `drive()`, `stop()`

a **`person`** can be represented as an object with *properties* such as `name`, `gender`, `age`, and *behaviors* (i.e., functions) such as `walk()`, `sleep()`

JS Objects

JS is an **object-oriented programming language**: everything can be regarded as objects, including the primitive data types and functions

JS creates object with constructor or object literal

```
var currentDT = new Date(); /* variable currentDT is assigned to
Date(), a built-in JS object; new Date () is a constructor that
create an instance of Date() object */
var month = currentDT.getMonth(); /* getMonth() is a behavior (i.e.,
function) within the Date () object, which returns the month
information that is included in the current instance */.

var person = {name: "John", age: 25};
/* variable person is assigned to a self-defined object with two
properties */
```

JS Variables & Objects

Variables are name containers that hold values of any data type

- They are used to store temporary values when JS is running in the Web page
- These values will be **lost** once the page is reloaded!

Objects are complex data structures holding multiple values, properties, and functions.

- They are created using object literals ({}) or constructor functions
- They allow organizing related data and behaviors **into a single entity.**
- Objects consist of key-value pairs, where keys are property names and values can be of any data type, including other objects.

How Does An Object Look Like?

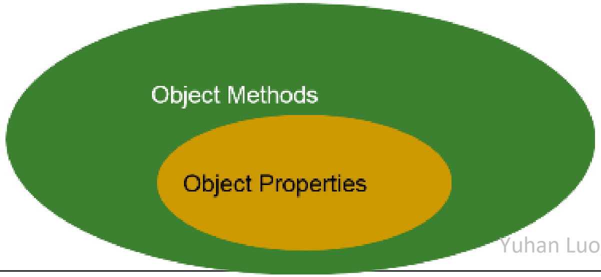
An object contains **two main** parts:

Properties

- values associated with an object, such as length, and width; styles and events are also properties
- can get/change their values by JS

Methods (i.e., functions)

- actions that can be preformed on objects, such as `write()` of the document object, i.e., `document.write()`



Four main kinds of objects in JavaScript

Primitive objects - number, string and Boolean

Built-in objects - Array, Date, and Math, etc.

- We can directly call the functions contained in these objects

```
new Array (). Length /* will return 0 because no elements is
pushed into the array */
```

Self-defined objects – defined by the programmer

- We can get the property values or call the functions declared for the object

```
var person = {name: “John”, age: 25};
var name = person.name; /* The output is “John”*/
```

DOM - provided by the browser as the host environment

```
document.getElementById (“xxx”).innerHTML  = “hello world”;
/*getElementById() is a function defined for document object; it also
returns an object that has properties such as innerHTML */
```

Define Your Objects: Literal

Syntax

```
var objName = {
    propertyName1: value1,
    propertyName2: value2,
    …
    methodName: function([pars]) {
          // function body
    }
};
```

Access property

- `objName.propertyName`
- `objName['propertyName']`

Access function

- `objName.methodName()`

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

```
const person = {
firstName: "John",
lastName: "Doe",
fullName: function() {
  return this.firstName + " " + this.lastName;
    }
};

console.log(person);
console.log(person.fullName());
```

What is **`this`**? Can it be removed?

The keyword `“this”`

In JavaScript, `“this”` keyword refers to an **object**

- Note that the value of `this` CANNOT be changed

Depending on how this is being invoked (or called), the object it refers to is different:

- In an object method, `this` refers to **this object**
- In an event, `this` refers to the **element** (such as a button, which is also an object in JavaScript) that receives the event
- When **used alone**, `this` usually refers to the **global object** (typically the `window` object in a web browser environment)

```
console.log (“Hello World”)/* This statement is equivalent to
window.console.log() or this.console.log () where window or
this can be ignored*/
```

Define Your Objects: New object ()

Syntax

```
var obj = new Object();
```

```
var obj = new Object();
obj.name = 'tony';
obj.studentID = '12345678';
obj.height = 170;

obj.writeCode = function() {
    console.log('hello world!');
}

console.log(obj.name);
console.log(obj['height']);
obj.writeCode();
```

We can declare properties and methods separately in a global context using

`objName.[variable/function name]= …`

Code Example: lec08-05-JS-new-obj.html

Define Your Objects: Constructor

Syntax

```
function funName ([values]) {
      this.property1 = value1;
      this.property2 = value2;
       …
      this.method = function([pars]) {
          // function body
      }
}
```

```
function Student(n, id, h) {
    this.name = n;
    this.stuID = id;
    this.height = h;
    this.writeCode = function(msg) {
        console.log(msg);
    }
}
```

```
var tony = new Student('tony', '12345678', 175);
console.log(tony.name);
console.log(tony['stuID']);
tony.writeCode('hello world');

var bob = new Student('bob', '23456789', 180);
console.log(bob.name);
console.log(bob['stuID']);
bob.writeCode('cs2204');
```

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

Key components

- Add each property and method within the **constructor** function
- this pointes to the object that the function declares

a **constructor** function is used to create new object instances, each with its own property values and method parameters

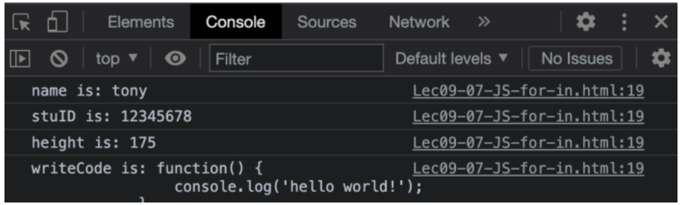
Iterate Elements in An Object (1)

Object elements includes all the properties and methods which are stored in an **array-like structure** as values of elements with an index

- Syntax

```
for (var k in objName) {
   console.log(k);            // property name
   console.log(objName[k]);  // property value
}
```

Code Example: lec08-07-JS-for-in.html



```
name is: tony                        Lec09-07-JS-for-in.html:19
stuID is: 12345678                   Lec09-07-JS-for-in.html:19
height is: 175                       Lec09-07-JS-for-in.html:19
writeCode is: function() {           Lec09-07-JS-for-in.html:19
            console.log('hello world!');
```

```
var myStudent = {
    name: 'tony',
    stuID: '12345678',
    height: 175,
    writeCode: function() {
        console.log('hello world!');
    }
};

for (var k in myStudent) {
    console.log(k + ' is: ' + myStudent[k]);
}
```

Iterate Elements in An Object (2)

Object elements includes all the properties and methods

- Syntax

```
for (var k in objName) {
   console.log(k);             // property name
   console.log(objName[k]);  // property value
}
```

- When it can be useful?
  - Data processing: filter/sort/transform data
  - Object manipulation: add/remove/modify properties
  - Debugging: inspect properties and values

Built-In Objects In JavaScript

Built-in objects are provided by JS instead of being defined by programmers or third-party Libs

- available globally in any JS environment

The following objects are built-in JavaScript:

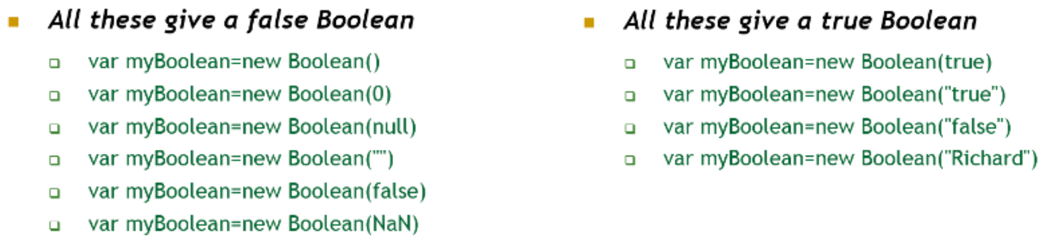
- Boolean
- Math
- Date
- Array
- String

Boolean

A primitive data type but can also be viewed as object

```
var myboolean = true;
var myboolean = new Boolean (value);
```

- It specifies the initial value `true` or `false` of the Boolean object
- The value is converted to a Boolean value, if necessary
  - if value is omitted or is `0`, `-0`, `null`, `false`, `NaN`, `undefined`, or the empty string (“”), the object has an **initial value of** `false`
  - all other values, including any object or the string “`false`”, create an object with an **initial value of** `true`



- *All these give a false Boolean*
  - var myBoolean=new Boolean()
  - var myBoolean=new Boolean(0)
  - var myBoolean=new Boolean(null)
  - var myBoolean=new Boolean("")
  - var myBoolean=new Boolean(false)
  - var myBoolean=new Boolean(NaN)
- *All these give a true Boolean*
  - var myBoolean=new Boolean(true)
  - var myBoolean=new Boolean("true")
  - var myBoolean=new Boolean("false")
  - var myBoolean=new Boolean("Richard")

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

Math Object (1)

Math object is **not** a function object (NO need to use “`new`”)

- can access it without using a constructor

Math object contains:

- **properties** - mathematical constants
  - e.g., Math.PI, Math.LN2, and Math.LN10
- **methods** - mathematical functions
  - e.g., Math.max(number), and Math.round(number)

```
console.log(Math.PI);
console.log(Math.SQRT2);

console.log(Math.min(3, 6, 9));
console.log(Math.min(3, 6, 9, 'cs2204'));
console.log(Math.min());

console.log(Math.floor(1.2));
console.log(Math.floor(1.9));
console.log(Math.ceil(1.2));
console.log(Math.ceil(1.8));
console.log(Math.round(1.4));
console.log(Math.round(1.5));
```

Code Example: lec08-09-JS-math.html

Math.SQRT2 is commonly used, but note that there is **NO** Math.SQRT3 or Math.SQRT4, …

Math Object (2)

Math object is **not** a function object (NO need to use "`new`")

- can access it without using a constructor

Math object contains:

- **properties** - mathematical constants
  - e.g., Math.PI, Math.LN2, and Math.LN10
- **methods** - mathematical functions
  - e.g., Math.max(number), and Math.round(number)

```
console.log(Math.PI);
console.log(Math.SQRT2);

console.log(Math.min(3, 6, 9));
console.log(Math.min(3, 6, 9, 'cs2204'));
console.log(Math.min());

console.log(Math.floor(1.2));
console.log(Math.floor(1.9));
console.log(Math.ceil(1.2));
console.log(Math.ceil(1.8));
console.log(Math.round(1.4));
console.log(Math.round(1.5));
```

Code Example: lec08-09-JS-math.html

Math.min() returns infinity, how about *Math.max()?*

Math Object (3)

`Math.random():` return a random value in [0, 1), where 1 is exclusive

What's the range of the random variable created by the following code create?v

```
Math.floor((Math.random()*10 + 1))
```

Given the following loop, what will be the range of the output?

```
res = '';
for (var i=0; i<10; i++) {
   res += Math.floor((Math.random()*10 + i)) + ' ';
  }
console.log(res);
```

Critical thinking: suppose that X and Y are two integers, how to create a random variable ranging from X – Y (including both X and Y)?

Code Example: lec08-09-JS-math.html

Date Object (1)

The Date object is used to work with dates and times. Create Date object by using new:

```
var mydate = new Date ();
new Date (milliseconds);
new Date (dateString);
new Date (yr_num, mo_num, day_num [, hr_num, min_num, sec_num, ms_num]);
```

`no argument`	the constructor creates a Date object for today's date and time based on **local time**
`milliseconds`	an integer value, representing the number of milliseconds since 1 January 1970 00:00:00 UTC
`dateString`	a string value, representing a date the string should be in a format recognized by the parse method
`yr_num, mo_num, day_num`	integer values, representing year, month, and day month is **representing by 0 to 11 with 0=January, and 11=December**
`hr_num, min_num, sec_num, ms_num`	integer values representing hours, minutes, seconds, and milliseconds

Code Example: lec08-10-JS-date.html

Date Object (2)

To create a date object, remember to use Date constructor

```
        today = new Date();
```

Some useful methods (functions) of the date object

- today.getDate() - returns `1-31`
- today.getDay() - returns `0-6`
- today.getMonth() - returns `0-11`
- today.getFullYear() - returns the current year
- today.getHours() – returns `0-23`

Except that `getDate()` returns values starting from 1, `getMonth()`, `getHours()`, `getDay()` all returns values starting from **0**

Array (1)

An array is a data structure that stores a collection of values that can be of any data type. We can create an array using literal

```
var myarray = [ ];
```

Create an array using the new operator

```
var myarray = new Array(); /* can add elements later using
loops or depending on the development needs; recommended */
var myarray = new Array (element0, element1, … , elementN);
var myarray = new Array (arrayLength);
```

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

Array (2)

Array length

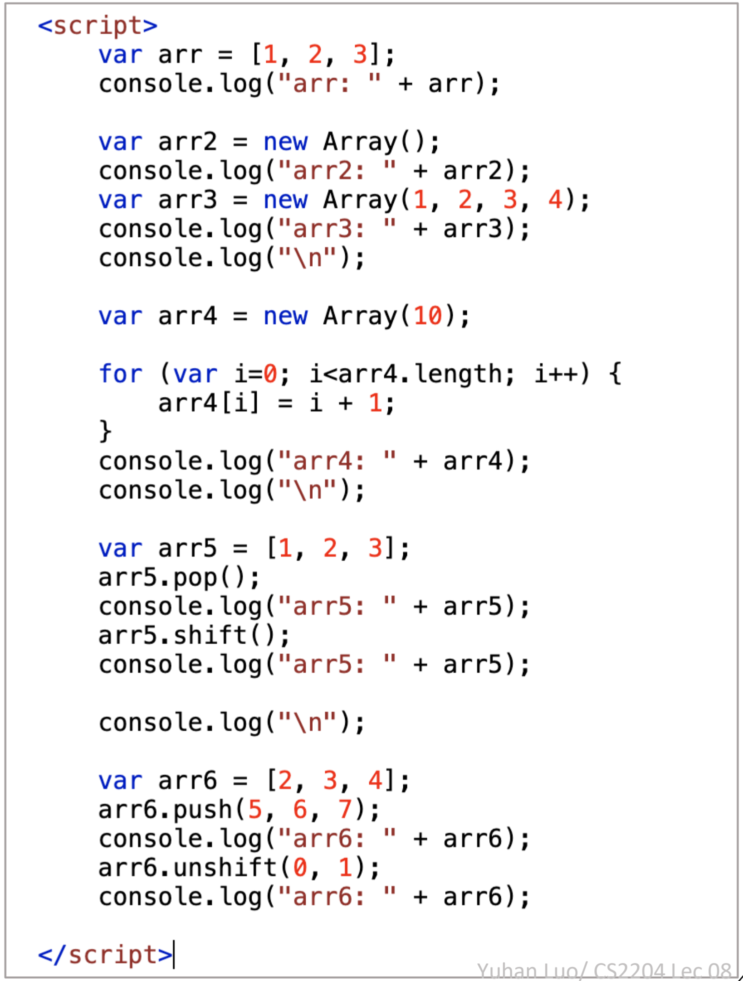
- Specifies the length of the array
- Can be accessed using the `length` property

Delete an element

- `pop();` delete the **last** element
- `shift();` delete the **first** element

Add element(s)

- `push(values);` add values to the **end** of the array
- `unshift(values);` add values to the **beginning** of the array



```
<script>
    var arr = [1, 2, 3];
    console.log("arr: " + arr);

    var arr2 = new Array();
    console.log("arr2: " + arr2);
    var arr3 = new Array(1, 2, 3, 4);
    console.log("arr3: " + arr3);
    console.log("\n");

    var arr4 = new Array(10);

    for (var i=0; i<arr4.length; i++) {
        arr4[i] = i + 1;
    }
    console.log("arr4: " + arr4);
    console.log("\n");

    var arr5 = [1, 2, 3];
    arr5.pop();
    console.log("arr5: " + arr5);
    arr5.shift();
    console.log("arr5: " + arr5);

    console.log("\n");

    var arr6 = [2, 3, 4];
    arr6.push(5, 6, 7);
    console.log("arr6: " + arr6);
    arr6.unshift(0, 1);
    console.log("arr6: " + arr6);

</script>
```

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

String

Similar to Boolean, string is a primitive type and can be viewed as an object.

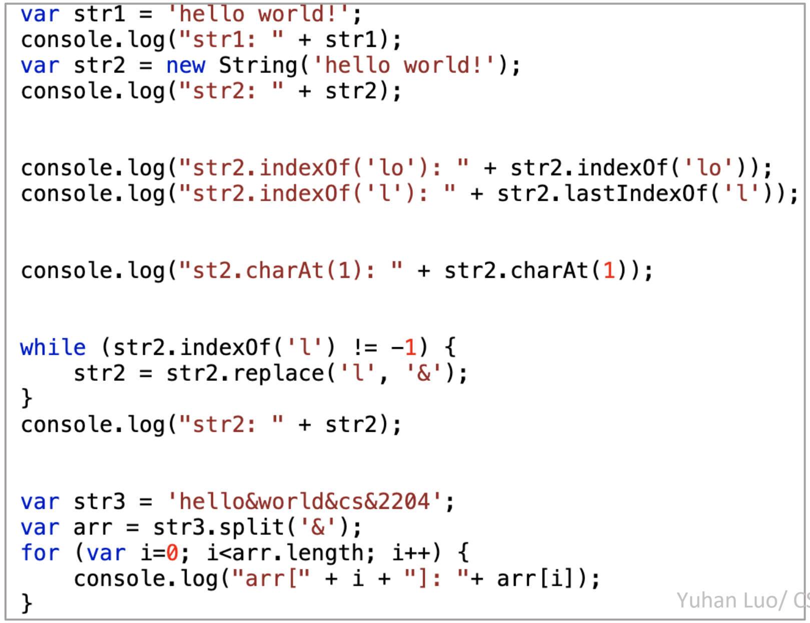
```
var str = 'string';
var str = new String('string');
```

Properties:

- `length`

Methods:

- `indexOf() and lastIndexOf()`
- `charAt()`
- `substr(start, length)`
- `replace(pattern, replacement)`
- `split(separator)`



```
var str1 = 'hello world!';
console.log("str1: " + str1);
var str2 = new String('hello world!');
console.log("str2: " + str2);


console.log("str2.indexOf('lo'): " + str2.indexOf('lo'));
console.log("str2.indexOf('l'): " + str2.lastIndexOf('l'));


console.log("st2.charAt(1): " + str2.charAt(1));


while (str2.indexOf('l') != -1) {
    str2 = str2.replace('l', '&');
}
console.log("str2: " + str2);


var str3 = 'hello&world&cs&2204';
var arr = str3.split('&');
for (var i=0; i<arr.length; i++) {
    console.log("arr[" + i + "]: "+ arr[i]);
}
```

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

Agenda

Review of JS Conditionals and Loops

JavaScript functions

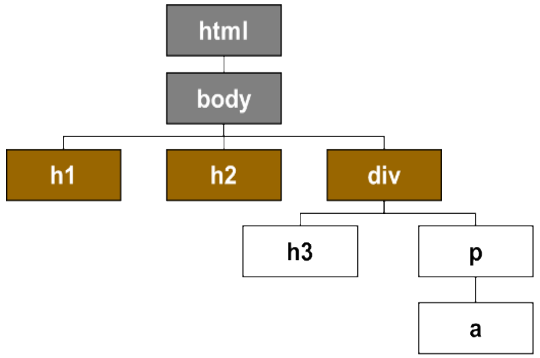
Object-oriented programming

DOM and Events

DOM

The DOM (Document Object Model) is the main bridge between the Web page and JavaScript

- Document
- Element: tag
- Node: element, attribute, etc.
- Each node can be viewed as an **object**



JavaScript would find/select an object and then changes its properties/content or call its methods

How to Select an Element? (1)

By **ID**

`document.getElementById('id') /* get the corresponding element` **`object */`**

To get the text content, we should call the property **innerHTML**

```
<div id="block1">
    this is a block
</div>

<div id="block2">
    <h1>heading1</h1>
    <h1>heading2</h1>
    <h1>heading3</h1>
    <p>this is a pargraph</p>
</div>

<ol>
    <li id="list1">The first item</li>
    <li id="list2">The first item</li>
</ol>
```

```
<script>
    var myDiv = document.getElementById('block1').innerHTML;
    console.log(myDiv); // output the text value of the object
    console.dir(myDiv); // output all the properties of the object

    var hs = document.getElementsByTagName('h1');
    console.log(hs);
    var ps = document.getElementsByTagName('p');
    console.log(ps);

    var divhs = document.getElementById('block2').getElementsByTagName('h1');
    console.log(divhs);

    var lis = document.getElementsByTagName('ol').getElementById('list1');
    console.log(lis);
</script>
```

Code Example: lec08-13-JS-byID.html

How to Select an Element? (2)

By **tag name**

```
document.getElementsByTagName('tagname') /* get an array
of elements objects of this tag */
```

```
<div id="block2">
    <h1>heading1</h1>
    <h1>heading2</h1>
    <h1>heading3</h1>
    <p>this is a pargraph</p>
</div>
```

Code Example: lec08-13-JS-byID.html

Critical thinking: Will the following code show the HTML content? of h1?

```
var hs = document.getElementsByTagName('h1').innerHTML;
```

If no, how to make it show the HTML content?

How to Select an Element? (3)

By **combination**

```
<div id="block2">
    <h1>heading1</h1>
    <h1>heading2</h1>
    <h1>heading3</h1>
    <p>this is a pargraph</p>
</div>
```

Code Example: lec08-13-JS-byID.html

The following code selects all the h1 elements in “block 2”

```
var divhs = document.getElementById('block2').getElementsByTagName('h1');
console.log(divhs);
```

Given the html, does the following code selects the element “list1”? Why or why not?

```
<ol>
    <li id="list1">The first item</li>
    <li id="list2">The first item</li>
</ol>
```

```
var lis = document.getElementsByTagName('ol').getElementById('list1');
console.log(lis);
```

How to Select an Element? (4)

By **CSS selector**

Select one each time

```
document.querySelector('CSS selector')
document.querySelector('#id')
document.querySelector('.className')
document.querySelector('tagName')
document.querySelector('#id1, #id2') /* return either one found
the first element object */
```

Select all

```
document.querySelectorAll('CSS selector')
```

get an array of all selected elements objects

Code Example: lec09-14-JS-query.html

How to Select an Element? (5)

By **CSS selector: Combination** also works

```
<div id="block1">this is a block</div>

<ol>
    <li class="list">item1</li>
    <li class="list">item2</li>
    <li class="list">item3</li>
</ol>

<div id="block2">
    <p id="ps1">this is the first paragraph another block</p>
    <p id="ps2">this is the second paragraph another block</p>
</div>
```

```
var divp = document.querySelectorAll('div > p');
console.log(divp);

var pp = document.querySelector('p ~ p');
console.log(pp);

var divpli = document.querySelectorAll('div, #ps2, .list');
console.log(divpli);
```

Code Example: lec08-14-JS-query.html

How does JS Works With Web objects?

Properties

- the properties of an object, such as font properties, color properties, and box properties can be read or changed with JS

Methods - use them to do something, such as:

- `alert()` or `window.alert()` in JS contexts where the object is window
- *`document.write()`* - write to the document object, i.e., the Web page
- `document.querySelector(“video”).play()`

Often times, the interaction is trigger by **event handlers**

- **Functions** "attached" to objects and used to trap events happening to their owning object
- Example events: *`onclick`*, *`onmouseover`*, *`onchange`*, etc.

What is an event?

An event occurs when the **user** or **browser** manipulate the page

Event	Description
`onchange`	An HTML element has been changed
`onclick`	The user clicks an HTML element
`onmouseover`	The user moves the mouse over an HTML element
`onmouseout`	The user moves the mouse away from an HTML element
`onkeydown`	The user pushes a keyboard key
`onload`	The browser has finished loading the page

For full list, you can refer to: https://www.w3schools.com/jsref/dom_obj_event.asp

Three important aspects of an event

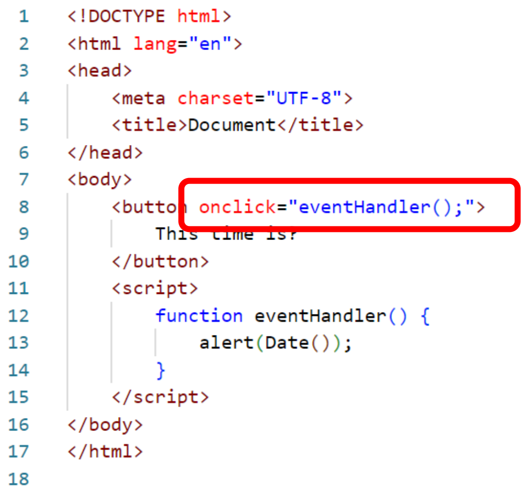
1) **Where** will the event happen?
   - Source of the event: a button, link, or an input field?

2) **Which type** of the event to be handled
   - An user action (e.g., *onclick*)?
   - Or a browser setting (e.g., *setTimeout()*)?

3) **How** to handle the event?
   - Event handler, which is usually a function

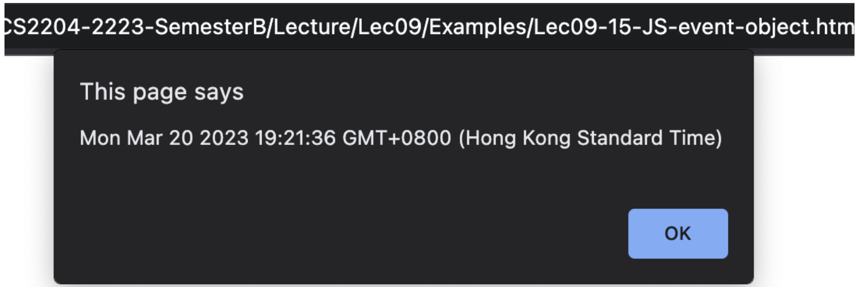
Refer to an event (1)

HTML element’s attribute



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <button onclick="eventHandler();">
        This time is?
    </button>
    <script>
        function eventHandler() {
            alert(Date());
        }
    </script>
</body>
</html>

```

Code Example: Lec08-15-JS-event-attribute.html



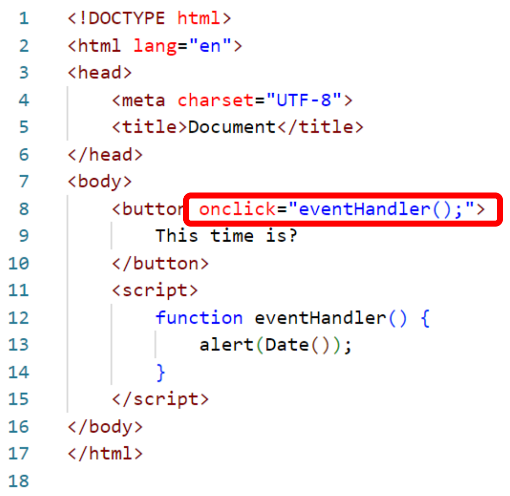
S2204-2223-SemesterB/Lecture/Lec09/Examples/Lec09-15-JS-event-object.htm

This page says

Mon Mar 20 2023 19:21:36 GMT+0800 (Hong Kong Standard Time)

Refer to an event (2)

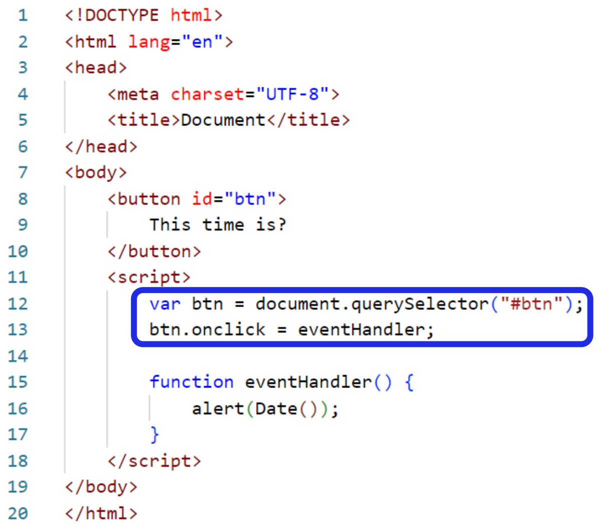
HTML element’s attribute



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <button onclick="eventHandler();">
        This time is?
    </button>
    <script>
        function eventHandler() {
            alert(Date());
        }
    </script>
</body>
</html>

```

Code Example: Lec08-15-JS-event-attribute.html

An object's property

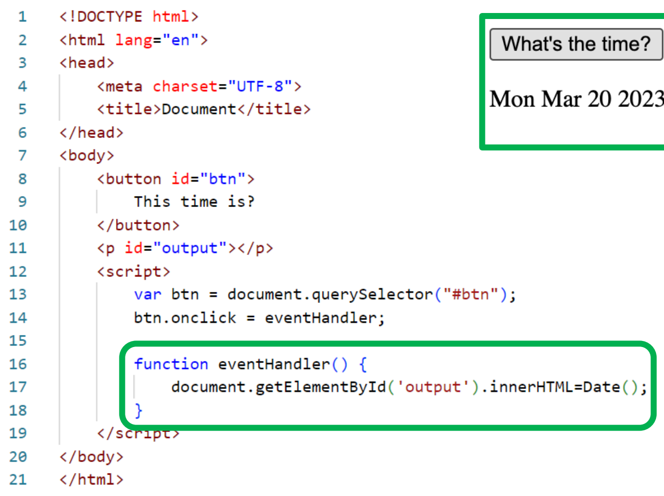


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <button id="btn">
        This time is?
    </button>
    <script>
        var btn = document.querySelector("#btn");
        btn.onclick = eventHandler;

        function eventHandler() {
            alert(Date());
        }
    </script>
</body>
</html>
```

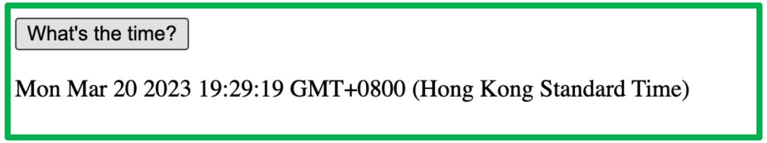
Code Example: Lec08-16-JS-event-object.html

Event Handler

A piece of JavaScript codes, usually a **function** tells the object how to react when that event occurs



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <button id="btn">
        This time is?
    </button>
    <p id="output"></p>
    <script>
        var btn = document.querySelector("#btn");
        btn.onclick = eventHandler;

        function eventHandler() {
            document.getElementById('output').innerHTML=Date();
        }
    </script>
</body>
</html>
```



Mon Mar 20 2023 19:29:19 GMT+0800 (Hong Kong Standard Time)

`eventHandler` replaces the content of `<p>` whose id is "output" by the current time.

Code Example: Lec08-17-JS-event-handler.html

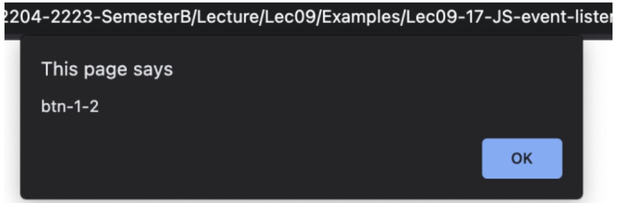
Event Listener

A function that is registered to listen for a specific event on an element

Syntax `event.addEventListener(type, listener[, useCapture])`

- **type**: a **string** to represent an event (no prefix *on*)
  - e.g., ‘click’, ‘mouseover’, etc.
- **listener**: **function** to handle this event

```
<button id="btn1">Button 1</button>
<button id="btn2">Button 2</button>
```



```
2204-2223-SemesterB/Lecture/Lec09/Examples/Lec09-17-JS-event-lister
This page says
btn-1-2
```

When Button 1 is clicked

How about when the **Button 2** is clicked?

```
var btn1 = document.getElementById('btn1');
btn1.onclick = f1;

function f1() {
    alert('btn-1-1');
}
btn1.onclick = f2;

function f2() {
    alert('btn-1-2');
}

var btn2 = document.getElementById('btn2');
btn2.addEventListener('click', f3);
function f3() {
    alert('btn-2-1');
}
btn2.addEventListener('click', f4);
function f4() {
    alert('btn-2-2');
}
```

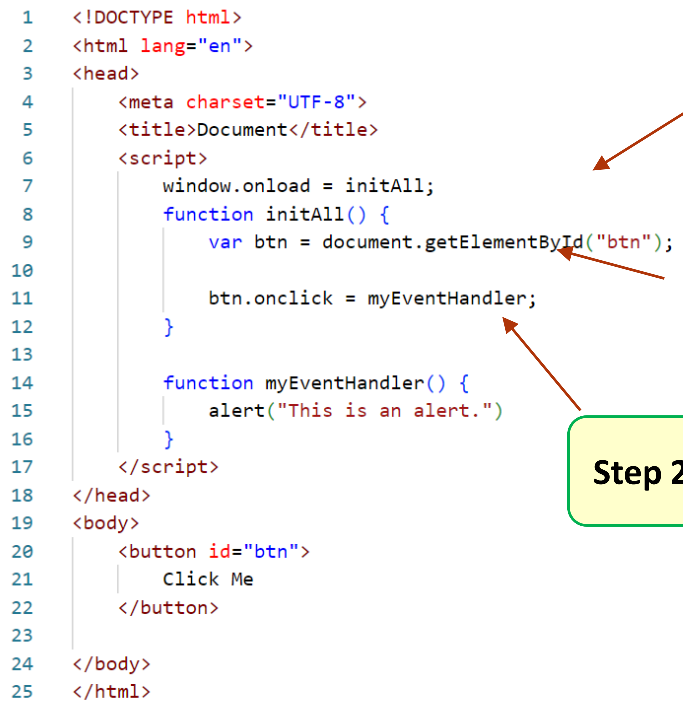
Code Example: Lec08-18-JS-event-listener.html

Event Handler vs. Listener

Question time:

- What is the key difference between event handler and event listener?
- Which one is better?

Example: Steps to Set Up Event



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        window.onload = initAll;
        function initAll() {
            var btn = document.getElementById("btn");

            btn.onclick = myEventHandler;
        }

        function myEventHandler() {
            alert("This is an alert.")
        }
    </script>
</head>
<body>
    <button id="btn">
        Click Me
    </button>

</body>
</html>
```

Step 1: find the object that you want to set up event for

Step 3: assign the event handler to the object. This can be replaced by `btn.addEventListener("click", myEventHandler);`

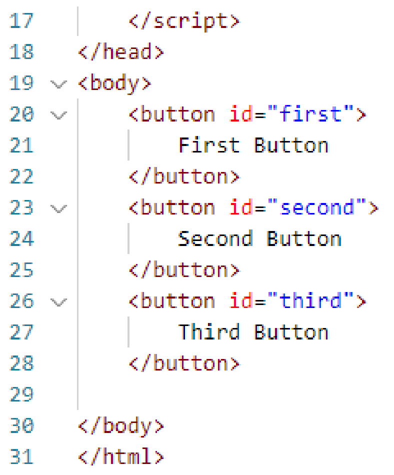
Step 2: define the event handler for it

Lec08-19-JS-event.html

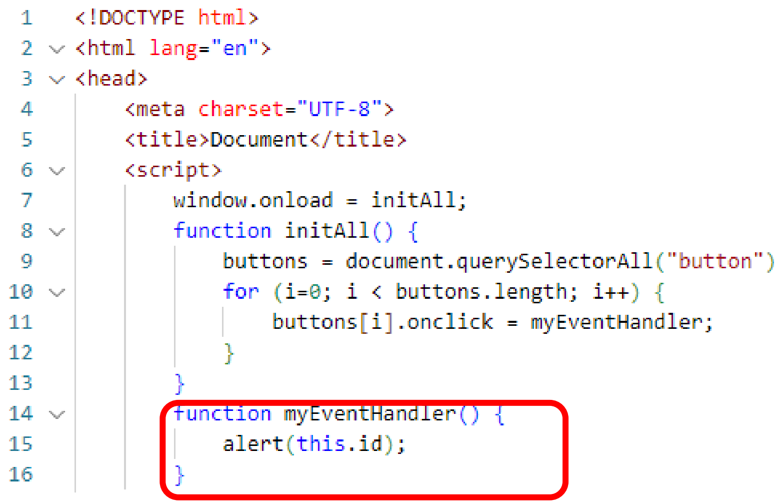
Object `this`

`this` can be used in the event handler to refer to the assigned object

- Advantages: same event handler may be used for many similar objects



```
    </script>
</head>
<body>
    <button id="first">
        First Button
    </button>
    <button id="second">
        Second Button
    </button>
    <button id="third">
        Third Button
    </button>

</body>
</html>
```



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        window.onload = initAll;
        function initAll() {
            buttons = document.querySelectorAll("button")
            for (i=0; i < buttons.length; i++) {
                buttons[i].onclick = myEventHandler;
            }
        }
        function myEventHandler() {
            alert(this.id);
        }
```

Code Example: Lec08-20-JS-object-this.html

Refer to the object assigned with this event handler

Lecture summary

JavaScript has two types of function: **self-defined** functions and **built-in** functions

JavaScript is an object-oriented programming languages. Objects can be the primitive data types or complex data structures.

Each object has its properties and methods (i.e., functions). Likewise, there are also built-in objects and self-created objects.

`this` refers to an object depending on how the object is being invoked

JavaScript interacts with the HTML elements through DOM (Document Object Model) by assigning/changing its properties and call its methods. We can select HTML elements in JavaScript by ID, tag names, CSS queries, and combinations of these methods

EventHandlers are a special type of functions that tells the web object how to react when an event occurs.