CS2204 Fundamentals of Internet Applications Development

Lecture 9 JavaScript – Part 2

Computer Science, City University of Hong Kong

Semester A 2023-24

Review: JavaScript (1)

1. JavaScript is a programming language that provides instructions for a browser to _____ and _____

Review: JavaScript (2)

2. What are the three basic (primitive) data types in JavaScript?

Topics

- Flow control
- Function
- Scope
- Objects

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

JavaScript: For-Loop

- A **repetition statement** *(also called a* ***loop****)* allows actions to be repeated while a certain condition is true.
  - Example: calculate summation from 1 to 10

```
var sum;
sum = 0;
sum = sum + 1;
sum = sum + 2;
 …
 sum = sum + 10;
```

sum = sum + i;

  - For-Loop statement

```
var sum = 0;

for (var i=1; i<=10; i++)
{
    sum = sum + i;
}
```

JavaScript: For-Loop

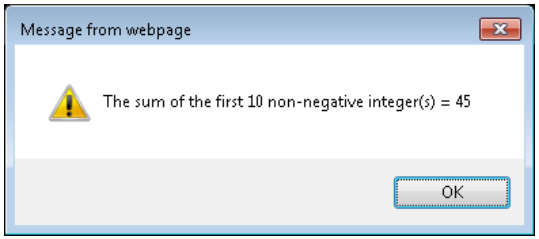
- The for-loop is often used to carry out a task for a finite number of times

The for-statement, e.g. **`for (i=0; i<N; i++)`**, contains 3 parts inside the parentheses:

1. Initialization: the code is executed at the beginning of the loop
   e.g., `i=0` assigns `0` to the variable `i` right after the statement `sum=0;`
2. Continuation condition: the tasks specified in the loop are carried out if the continuation condition in the form of Boolean expression is true, otherwise the loop ends if the continuation condition is false
   e.g., `i<10` is true when `i=0,1,2,3,4,5,6,7,8,9` but is false when `i=10;`
3. Increment statement: this part is executed at the end of each iteration of the loop
   e.g., `i++` means that the variable `i` is increased by `1` and it is executed at the end of each iteration of the loop after the statement `sum = sum + i;`

- Note that there should be no semi-colon after the parenthesis,
  i.e. `for (i=0; i<N; i++);` is wrong

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Javascript For-Loop</title>
    <script>
        function init() {
            var i, N, sum;
            N = 10;
            sum = 0;
            for (i=0; i<N; i++) {
                sum = sum + i;
            }
            alert("The sum of the first "+N+" non-negative integer(s) = "+sum);
        }
    </script>
  </head>
  <body onload="init();">
  <!-- Page content begins here -->
    <p>Adding the first N integers</p>
    <!-- Page content ends here -->
  </body>
</html>
```



The curly brackets after the for-statement enclose the statements that are executed at each iteration of the for-loop, in this example, `sum = sum + i;`
You can put multiple statements inside the curly brackets such that all of them will be executed at each iteration of the loop

Code Example: lec09-01-JS-for-loop.html

JavaScript: For-Loop (cont.)

```
N = 10;
sum = 0;
for (i=0; i<N; i++) {
    sum = sum + i;
}
```

The code on the left is executed according to the following sequence of operations:

```
1. N=10
2. sum=0
3. i=0
4. i<N ⇔ 0<10=true so the loop will continue to run
5. sum=sum+i ⇔ sum=0+0=0
6. i++ ⇔ i=0+1=1
```

1st iteration

2nd iteration

```
7. i<N ⇔ 1<10=true so the loop will continue to run
8. sum=sum+i ⇔ sum=0+1=1
9. i++ ⇔ i=1+1=2
```

3rd iteration

```
10.i<N ⇔ 2<10=true so the loop will continue to run
11.sum=sum+i ⇔ sum=1+2=3
12.i++ ⇔ i=2+1=3
```

4th iteration

```
13.i<N ⇔ 3<10=true so the loop will continue to run
14.sum=sum+i ⇔ sum=3+3=6
15.i++ ⇔ i=3+1=4
```

5th iteration

```
16.i<N ⇔ 4<10=true so the loop will continue to run
17.sum=sum+i ⇔ sum=6+4=10
18.i++ ⇔ i=4+1=5
```

6th iteration

```
19.i<N ⇔ 5<10=true so the loop will continue to run
20.sum=sum+i ⇔ sum=10+5=15
21.i++ ⇔ i=5+1=6
```

7th iteration

```
22.i<N ⇔ 6<10=true so the loop will continue to run
23.sum=sum+i ⇔ sum=15+6=21
24.i++ ⇔ i=6+1=7
```

8th iteration

```
25.i<N ⇔ 7<10=true so the loop will continue to run
26.sum=sum+i ⇔ sum=21+7=28
27.i++ ⇔ i=7+1=8
```

9th iteration

```
28.i<N ⇔ 8<10=true so the loop will continue to run
29.sum=sum+i ⇔ sum=28+8=36
30.i++ ⇔ i=8+1=9
```

10th iteration

```
31.i<N ⇔ 9<10=true so the loop will continue to run
32.sum=sum+i ⇔ sum=36+9=45
33.i++ ⇔ i=9+1=10
34.i<N ⇔ 10<10=false so the loop will end
```

Example: For-Loop

- Given an array of numbers, store all the positive and even numbers in a new array and display them
  - If `arr` is an **array**, we can use `arr.length` to get the size (i.e., the number of elements) of the array to control the loop
  - `var res = [];` can create an empty array

```
<script>
    var nums = [-2, -1, 0, 6, 9, 12, 1];
    var res = [];
    var j = 0;

    for (var i=0; i<nums.length; i++) {
        if (nums[i] > 0 && nums[i] % 2 == 0) {
            res[j] = nums[i];
            j++;
        }
    }

    for (var i=0; i<res.length; i++) {
        console.log(res[i]);
    }


</script>
```

```
<script>
    var nums = [-2, -1, 0, 6, 9, 12, 1];
    var res2 = [];

    for (var i=0; i<nums.length; i++) {
        if (nums[i] > 0 && nums[i] % 2 == 0) {
            res2[res2.length] = nums[i];
        }
    }

    for (var i=0; i<res2.length; i++) {
        console.log(res2[i]);
    }
</script>
```

Code Example: lec09-02-JS-for-array.html

JavaScript: While-loop

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

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

}
```

The loop statements is executed as long as **expr2** is true. When **expr2** becomes false, the loop ends (e.g., ***i<11***).

expr1: Executed before entering the loop. Often used for variable initialization (e.g., ***i=1***).

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

JavaScript: While-Loop

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

Code Example: lec09-03-JS-while-loop.html

```
1   <!DOCTYPE html>
2   <html>
3     <head>
8       <title>Javascript While-Loop</title>
9       <script>
10          function init() {
11              var isInputValid, number;
12              isInputValid = false;
13              while (!isInputValid) {
14                  number = prompt("Input a positive integer");
15                  if (isNaN(number)) {
16                      alert("Please enter a NUMBER!");
17                  }
18                  else if (Number(number) <= 0) {
19                      alert("Please enter a POSITIVE number!");
20                  }
21                  else
22                      isInputValid = true;
23              }
24              alert("The positive number that you entered is "+number);
25          }
26      </script>
27    </head>
28    <body onload="init();">
29    <!-- Page content begins here -->
30      <p>Checking for Positive Number</p>
31      <!-- Page content ends here -->
32    </body>
33  </html>
```

`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()` is JavaScript built-in function and is used to check whether the given parameter is not a number. It will return `true` if it is not a number and `false` if it is a number, e.g.,

`isNaN(234)=false`

`isNaN("abc")=true`

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

`Number("123")=123`

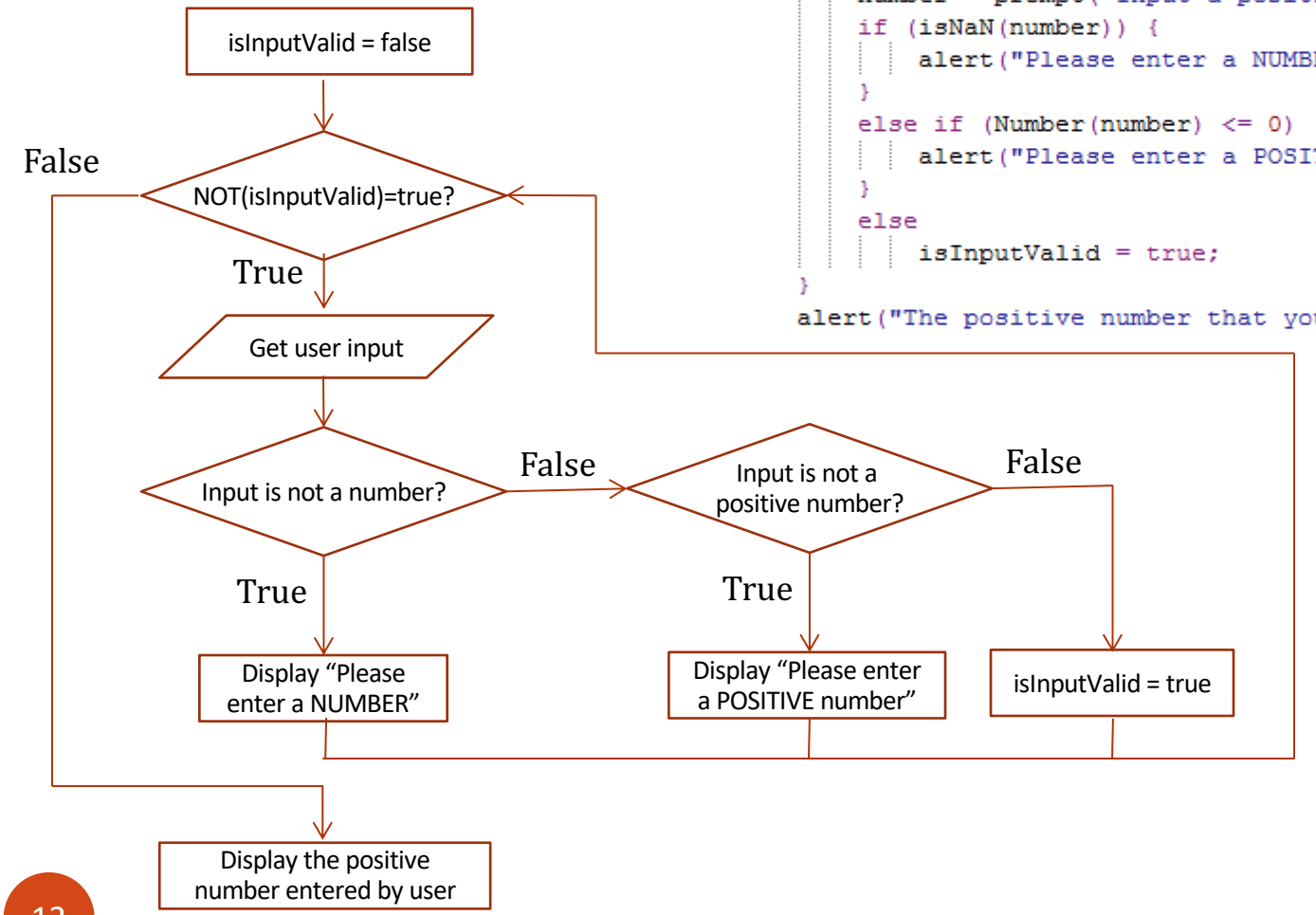
`Number("123")+1 = 124`

however, `"123"+1 = "1231"`

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

JavaScript: While-Loop (cont.)

```
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);
```



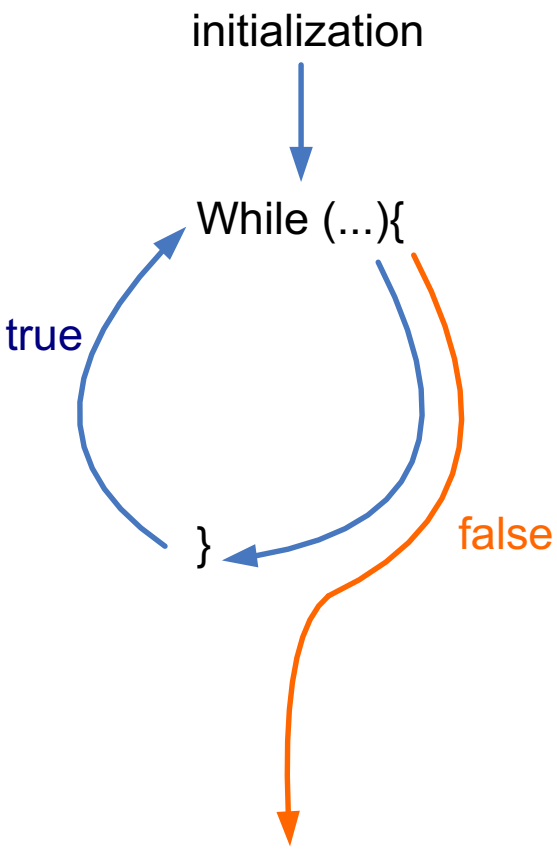
do statement

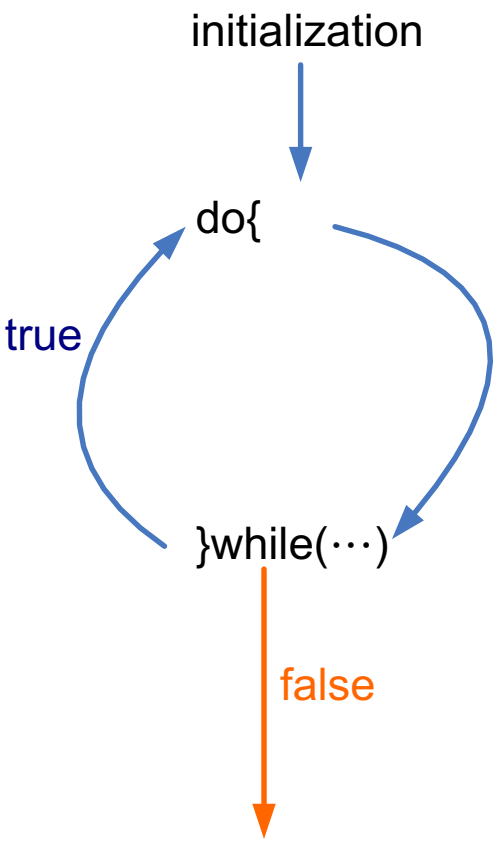
- General form of `do` statement (*repetition* statement)

```
do
{
    statement(s);
}
while (expression);
```

- Semantics:
  - `statement` is executed first; thus the **loop** body is run **at least once**
  - If the value of `expression` is non-zero (*true*), the loop repeats; otherwise, the loop terminates

While vs Do while





Break & Continue

- Break;
  - terminate the current loop, switch and transfer program control to the statement following the terminated statement
- Continue;
  - terminate execution of the statements in the current iteration of the current loop and go to the next iteration and continue with the loop

Example

- Input numbers using `prompt()` and store them in an array. Then select all the positive numbers in a new array and display them. Enter ‘e’ to exit the input

```
var inputs = [];
var temp;

do {
    temp = prompt('Input a number');
    if (temp != 'e') {
        inputs[inputs.length] = Number(temp);
    } else {
        break;
    }
} while(true);
```

```
var res = [];
for (var i=0; i<inputs.length; i++) {
    if (inputs[i] > 0) {
        res[res.length] = inputs[i];
    }
}

for (var i=0; i<res.length; i++) {
    console.log(res[i]);
}
```

Code Example: lec09-04-JS-do-while.html

Topics

- Flow control
- Function
- Scope
- Objects

JavaScript Functions

- 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

Keyword Function name Input (Parameter/Argument)

```
function printHello (    ){
   var i;
   for (i=0; i<n; i++)
      console.log("Hello\n”);
}
```

Function body

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

Code Example: lec09-05-JS-function-declare.html

Function Declaration

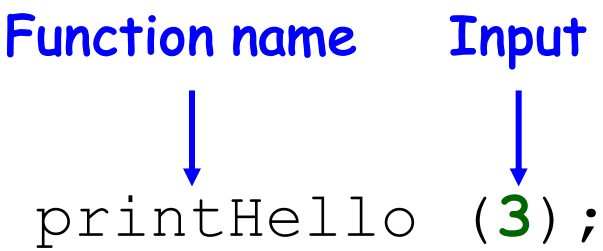
- A function must be declared before it can be used (called)

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

- name - the function name, should follow the rules for variable declaration
- par - the parameter names used in the function representing the actual value passed in when the function is called (argument); a function can have up to 255 parameters
- 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
- [ ] means optional - parameter 1 to N are optional and return is also optional

Calling a function

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



Special Characteristics of Function

- Function is an object and therefore can be assigned to a variable

```
function square(x) {return x*x;}
var a = square(4);
var b = square;
var c = b(5);
```

- Function can have no name - anonymous function

```
var d = function(x) {return x*x;}
var e = d(3);
```

- Therefore, there are two ways to declare a function
  - standard way, e.g., function fun1([pars]) { // statements; }
  - anonymous function, e.g., var fun2 = function([pars]) { // statements; }
- These characteristics are commonly used in event handler or object method set up

Code Example: lec09-05-JS-function-declare.html

Function Parameter

- There is no checking of the number of parameters

- Regardless of function declaration
  - `arguments` can be provided, even they are not defined in declaration, when the function is called
  - use the arguments object to get the actual arguments
    - arguments is an array-like object accessible inside function
    - e.g., `arguments.length`; `arguments[i]`;

```
function f1(a, b) {
    console.log(a + ' ' + b);
}

f1(1,2);
f1(1);
f1(1,2,3);
```

```
function f2() {
    for (var i=0; i<arguments.length; i++) {
        console.log(arguments[i]);
    }
}

f2();
f2(1,2,3);
f2('str1', 'str2');
```

Code Example: lec09-06-JS-function-parameters.html

return

- Used inside function, terminate the function execution and may return value to the function caller
- Find the minimum value of a set of numbers
  - e.g., `findMin(4, 2, 8, 6, 10);` or `findMin(8, 6, 10);`

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        function findMin() {
            var vMin = arguments[0];
            for (var i=1; i<arguments.length; i++) {
                if (arguments[i] < vMin) {
                    vMin = arguments[i];
                }
            }
            return vMin;
        }

        console.log(findMin(4, 2, 8, 6, 10));
        console.log(findMin(8, 6, 10));
    </script>
</head>
<body>

</body>
</html>
```

Code Example: lec09-07-JS-find-min.html

Built-in JavaScript Function

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

Topics

- Flow control
- Function
- Scope
- Objects

Variable Scope

- Variable scope tells where a variable can be referred to, used or valid
- Local variables
  - Declared in a function { }, which can only be accessed within {}
  - Try to access a local variable outside {} will produce unpredictable results
- Global variables
  - Declared not in the block. Can be accessed in the rest of the scripts
  - Will be override by a local variable with the same name in a function
  - If a variable is used in a function but without declared using keyword `var`, this variable is set to be a global variable

```
<script>
    var x = 1;

    function f(arg) {
        console.log(x);
        x = 0;
        console.log(x);

        var y = 2;
        console.log(y);

        z = 3;
    }

    f();
    console.log(x);
    console.log(z);
    console.log(y);
    console.log(arg);

</script>
```

Code Example: lec09-18-JS-scope.html

Variable and Function Hoisting

- When engine runs the JS code, it performs variable and function hoisting before running the code
  - Move the declaration of variable(s) and function(s) to the top of their current scope (only declaration, no initialization)

```
var num = 1;
function h() {
        console.log(num);   // Q1: what is the value of num?
        var num = 10;
        console.log(num);   // Q2: what is the value of num?
        date = "today";
}

h();                        // call function h()
console.log(date);
```

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

num = 1;
h();
console.log(date);
```

Code Example: lec09-xx-JS-hoisting.html

Variable and Function Hoisting

- What are the outputs of the following examples

```
console.log(course);
var course = ‘CS2204’;
```

```
f();
function f() {
  console.log(‘hello world’);
}
```


Code Example: lec09-10-JS-hoisting-2.html

var, let, and const

- var: global or local (function) scope
- let: block scope
- const: block scope but CANNOT be reassigned after initialization

(Both let and const are introduced in ES6)

```
const x = 1;
function f() {
    try{
        x ++;
    } catch (e){
         console.log(e.message);
    }
    console.log(x);
}
function g() {
    var x = 10;
    console.log(x);

    {
        let date = "today";
        console.log(date);
    }


}

f();
g();
```

Critical thinking

What are the outputs?

Code Example: lec09-11-JS-let.html

var, let, and const (2)

Difference between var and let

- Variables defined with let are also hoisted to the top of the block
- Using a let variable before it is declared will result in a ReferenceError

```
var num = 1;
let num2 = 2;
function h() {
    console.log(num);
    try{
        console.log(num2);
    } catch (e){
        console.log(e.message);
        }
    var num = 10;
    let num2 = 12;
    console.log(num);
}
```

Critical thinking

In the this and prior examples with var, let, and const, what if we remove the try and catch statements?

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`

Topics

- Flow control
- Function
- Scope
- Objects

Objects & Variables

- In JavaScript, everything can be regarded as objects, including the *primitive data types* and *functions*
  - In other Object-Oriented languages, objects (object instances) are created from classes (template) through instantiation. However, the class concept is not obvious in JavaScript
- Object creation is done by the literal or new operator

```
var myVar = 123; //123 is a literal
var currentDT = new Date();
```

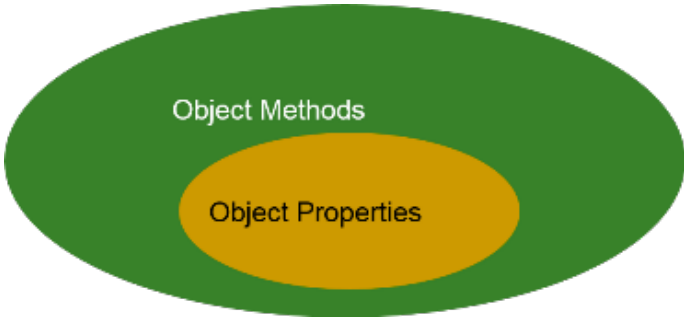
- Variables are then used to "store" or point to objects

Objects & Variables

- **Variables** (in the form of objects) are therefore used to store temporary values when JavaScript is being run in the Web page
  - These values will be lost once the page is reloaded!
- There are 4 main kinds of objects commonly used:
  - simple primitive objects - number, string and boolean
  - built-in objects - Array, Date and Math, etc.
  - self-defined objects - we define the structure of the object
  - DOM - provided by the browser as the host environment

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
  - actions that can be preformed on objects, such as write() of the document object, i.e., document.write()
  - use them in JS to do something



Define Your Objects - JSON

- The first way to define an object
  - JavaScript Object Notation (JSON)
- Syntax

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

- Access property
  - objName.propertyName
  - objName[‘propertyName’]
- Access function
  - objName.methodName()

Code Example: lec09-12-JS-JSON-object.html

Define Your Objects - New

- The second way to define an object
  - Use `new Object()` to create an object
- Syntax

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

Code Example: lec09-13-JS-new-object.html

- Add properties and methods
  - `obj.pName = 'tony';`
  - `obj.fName = function() {};`

Critical Thinking

- Any limitations of the previous two methods?

Define Your Objects - Constructor

- The third way to define an object
  - Use constructor function
- Syntax
  - this
  - no return

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

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

- Create an object
  - `new funName([values]);`

Define Your Objects - Constructor

- **Four** steps when "new" is used
  - Create an empty object
  - this is pointed to the created empty object
  - Execute the code of the **constructor** to add each property and method
  - Return this object

Iterate Elements in An Object

- A special for-loop
- Syntax

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

Code Example: lec09-15-JS-for-in.html

Built-In Objects In JavaScript

- The following objects are built-in JavaScript:
  - Boolean
  - Math
  - Date
  - Array
  - String
- Online documents

Personal Webpage (V09)

Personal Webpage (V09)

- Support Multi-languages: Pre-prepare the contents in English and another language and use a switch button to flip the language
  - **Task 1:** Use the same tags to place contents in different languages and add the same class attribute to the same language contents
  - **Task 2:** Set display to none for one language class in "*basicStyle.css*"
  - **Task 3:** Write a function *changeLanguageTo()* to toggle language by exchanging the display style of two classes
  - **Task 4:** Add a button to call the function when the button is clicked