JavaScript 2

Objectives

- To understand and apply the execution control structures
- To understand and apply the function
- To understand and apply the object orientation of JavaScript

Contents

- Execution Control Structures
- Functions
- Object Orientation
- Document Object Model (DOM)

Contents

- Execution Control Structures
- Functions
- Object Orientation
- Document Object Model (DOM)

Execution Control Structure

- So far, it can be seen that JavaScript code has executed in a series of sequential statements
- JavaScript has syntax that allows us to use condition and a particular block of code is selected to execute regarding the value of condition. This structure is called a decision.
- In addition, JavaScript has syntax that allows a code block to be repeatedly executed while some condition is true. It's called a repetition.

Execution Control Structures

- Decision control structures (if-then-else)
- Repetition control structures (loops)
- Using Repetition control structure with Arrays and User-defined Object's Properties

Decision control structures (if-then-else)

- https://www.youtube.com/watch?v=5i-xUNp5zMQ
- If statement
  - The if statement syntax offers the choice of doing something or not doing it
  - For example, suppose we define num1 = 1 and str1 = '1'

```
if(num1 == str1){
        console.log("The condition num1
== str1 is " + (num1 == str1))
    }
```

What is the output?

  - Normally, the condition of if statement need to be logical or comparison operators. In addition, it can be something else that return Boolean value (e.g. functions)

Decision control structures (cont.)

- If-else statement
  - Rather than if and condition, If-else statement allows us to use else with the choice to do something else when the condition of if is false.
  - For example, suppose num1 = 10 and str1 = ’1’

```
        if(num1 == str1){
            console.log("The condition num1
== str1 is " + (num1 == str1))
        }else{
            console.log("The condition num1
== str1 is " + (num1 == str1))
        }
```

What is the output?

Decision control structures (cont.)

- If-else-if statement
  - It is possible to have nesting an inner if-else statement in the else part of the outer if-else statement
  - For example,

```
var mark = 75;
var grade = "";

if ((mark >= 0) && (mark < 50)) {
    grade = "F";
} else if ((mark >= 50) && (mark < 60)) {
    grade = "D";
} else if (mark >= 60 && mark < 70) {
    grade = "C";
} else if (mark >= 70 && mark < 80) {
    grade = "B";
} else if (mark >= 80 && mark <= 100) {
    grade = "A";
} else {
    grade = "ERROR: range violation for mark: " +
mark;
}
console.log("The grade is " + grade);
```

What is the output?

Decision control structures (cont.)

- Switch statement
  - It match expression's value to a case clause and then execute the code block associated with that case
  - Syntax

```
switch (expression) {
case value1: //Statements executed when the result
of expression matches value1
[break;]
case value2: //Statements executed when the result
of expression matches value2
[break;]
...
case valueN: //Statements executed when the result
of expression matches valueN
[break;]
[default: //Statements executed when none of the
values match the value of the expression [break;]]
}
```

Decision control structures (cont.)

- Switch statement
  - For example,

What is the output?

```
var expr = "Soda";
switch (expr) {
  case 'Coke':
    console.log('Coke is 30 Baht.');
    break;
  case 'Diet Coke':
    console.log('Diet Coke is 30 Baht.');
    break;
  case 'Sprite':
    console.log('Sprite is 30 Baht.');
    break;
  case 'Ginger Ale':
    console.log('Ginger Ale is 30 Baht.');
    break;
  case 'Soda':
  case 'Sparking Water':
    console.log('Soda or Sparking Water is 30
Baht.');
    break;
  default:
    console.log('Sorry, we are out of ' + expr + '.');
}
```

Repetition control structures (loops)

- https://www.youtube.com/watch?v=I15K6r_29rs
- Types of loop
  - While loop
  - Do-While loop
  - For loop
  - Label

While loop

- The while statement allows us to have a loop that executes a code block repeatedly until the condition is evaluated to be false
- Syntax

```
while (condition){
 code block
}
```

While loop (cont.)

- For example,

```
var n = 0;
var x = 0;

//While loop
while (n < 10) {
    n++;
    x += n;
}
console.log("n = " + n + ", x = " + x);
```

What this loop does?

What is the output?

Do-While loop

- The do-while statement allows us to have a loop that executes a code block repeatedly until the condition is evaluated to be false
- The condition is evaluated after executing the code block. Therefore, the code block is executed at least once.
- Syntax:

```
do{
    statement
}while (condition);
```

Do-While loop (cont.)

- For example,

What is the output?

```
//Do-While loop
n = 1;
x = 0;

while (n < 1) {
    n++;
    x += n;
}
console.log("n = " + n + ", x = " + x);

n = 1;
x = 0;
do {
    n++;
    x += n;
} while (n < 1);
console.log("n = " + n + ", x = " + x);
```

For loop

- The for statement allows us to have a loop that executes a code block repeatedly until the condition is evaluated to be false
- Syntax:

```
for ([initialization]; [condition]; [final-expression]){
   statement
}
```

For loop (cont.)

- For example,

What is the output?

```
//For loop
for (var i = 0; i < 10; i++) {
    console.log("i = " + i);
}
```

Label

- The labeled statement can be used as identifier of a code block that you can refer to
- Syntax:

Label :
    statement

- The label statement can be used with break and continue statement

Label (cont.)

- For example,

```
//Label
var i,j;
loop1:
    for (i = 0; i < 5; i++) { //The first for statement is labeled "loop1"
        loop2: for (j = 0; j < 3; j++) { //The second for statement is labeled "loop2"
            if (i === 1 && j === 1) {
                continue loop1;
            }
            console.log('i = ' + i + ', j = ' + j);
        }
    }
```

What is the output?

Break and Continue

- The break statement allows us to terminate the current loop, switch, or label.
- Syntax: break [*label*];

- The continue statement allows us to terminate execution of the current iteration of the current or labeled loop, and continues execution of the loop with the next iteration.
- Syntax: continue [*label*];

Counter-controlled loops

- Counter-controlled loops are loops that are controlled by a counter that is updated on every repetition.
- Loops are terminated when a counter reaches maximum counter value.
- Normally, we use counter to control for and while loops

Sentinel-controlled loops

- Sentinel-controlled loops are loops that are terminated, when a sentinel value occurs
- The sentinel value cannot be predicted before the loop is entered. Therefore, the number of required loop is not known.
- Normally, we use while loop to implement sentinel-controlled loops.

Nesting control structures

- Each control structure can be nested where one inside another to any depth
- For example, if statement in for loop or for loop in for loop

Note!!

- Ensure that the loop condition is satisfied before the first iteration
- Ensure that the code block in the loop change the loop condition. Therefore, the loop can terminate and infinite loops is avoided

Using Repetition control structure with Arrays and User-defined Object's Properties

- See example

Contents

- Execution Control Structures
- Functions
- Object Orientation
- Document Object Model (DOM)

Why use functions?

- There are two main reasons:
  - Reduce redundant code block
  - In the real application, there might be millions line of code. Using function, it is easy to understand, create, test, change, and navigate
- As software engineer, you can easily analyse the code and remove isolated code block

Functions

- There are two types of function:
  - Functions that do not return values
  - Functions that do return values

Scope and Lifetime of Variables

- Functions are like office departments.
- Scope (https://vimeo.com/202865903)
  - For variables that are declared outside of any function, their scope is all code associated with the web page. (Global variable)
  - For variables that are declared in a function, their scope is the entire function. (Local variable)
- Note! Variables should have the smallest scope.

Scope and Lifetime of Variables (cont.)

- Lifetime
  - Global variables: the time from they are declared until the web page is closed
  - Local variables: the time from a function, where they are declared in, is called until the function is terminated
- If the names of global variable and local variable are the same, the local variable is considered.

Constructor Functions

- Normally, we might want to make many objects that have exactly the same set of properties in OOP since they can be proceed in a similar way without error.
- Constructor functions offer a mechanism to create objects with identical sets of properties.
- Constructor functions can be created in the same way as normal function.

Constructor Functions (cont.)

- They are called with the *new* operator
- In constructor function, the keyword *this* is used to refer to the object being created.
- Every newly created object is assigned with a consistent set of properties
- We can initialise the value to properties using the parameters of the function.
- We can add public (using *this*) and private (using *var*) functions in constructor functions

Constructor Functions (cont.)

- For example,

```
function Person(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.getInfo = function(){
        return this.firstName + " " + this.lastName + " Age is " + this.age;
    };
}
```

```
var person1 = new Person("Wudhichart", "Sawangphol", 30);
```

Constructor Functions (cont.)

- Interestingly, the properties of an object can be added on the fly.
- A constructor function can inherit another constructor function.

- See example.

First Class Functions

- A function can be assigned to a variable
- It can be passed as a parameter to a function. This can be used to great effect in a programming paradigm called functional programming.

Callback Functions

- APIs (Application Programming Interfaces) are implemented libraries of objects and their methods.
- Sometimes, API functions will require a function as a parameter. You are expected to code this function with a parameter list specified in the API documentation. This function is called Callback functions.

Contents

- Execution Control Structures
- Functions
- Object Orientation
- Document Object Model (DOM)

Object Orientation (OO)

- Generally, real-world objects and concepts cannot be implemented by a single function. Those objects and concepts need to involve multiple functions that usually need to share data.
- OO concepts in JavaScript
  - Classes
  - Inheritance and Polymorphism

Object Orientation (OO): Classes

- JavaScript classes was introduced in ECMAScript 2015
- This makes syntax to create objects in JavaScript clearer
- Classes are recipes for creating objects containing data items (known as *attributes* or *properties*) and functions that operate on this data (known as *methods*).
- The class syntax has 2 components:
  - Class declaration
  - Class expression
- An important difference between function declarations and class declarations is that function declarations are hoisted but class declarations are not

Classes (cont.)

- Class declaration:

```
class Person{
    constructor(firstName, lastName,
age){
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
}
```

- Class expression:

```
var Person = class {
    constructor(firstName, lastName,
age){
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
}
```

Classes (cont.)

- As mentioned, a class consists of:
  - Properties
  - Methods
  - In addition, we can have static methods (using keyword *static*).
- For example,

```
        getInfo(){
            return this.firstName + " " +
this.lastName + " Age: " + this.age;
        }

        static calculateBirthYear(age,
currentYear){
            return currentYear - age;
        }
```

Inheritance and Polymorphism

- Generally, we may want to create objects that have same general properties.
- Inheritance is an approach that a class inherits all the properties and methods of a particular base class.
- We use the keyword *extends.*
- For example,

```
class Student extends Person{
    constructor(firstName, lastName,
age, university, feePaid) {
        super(firstName, lastName, age);
        this.university = university;
        this.feePaid = feePaid;
    }
}
```

Inheritance and Polymorphism (cont.)

- Polymorphism is an approach that Subclasses of a class can define their own unique behaviors (methods) and yet share some of the same functionality of the parent class.
- For example,

```
//Person class
getInfo() {
    return this.firstName + " " +
this.lastName + " Age: " + this.age;
}
```

```
//Student Class
getInfo() {
    return this.firstName + " " +
this.lastName + " Age: " + this.age + "
University: " + this.university;
}
```

Contents

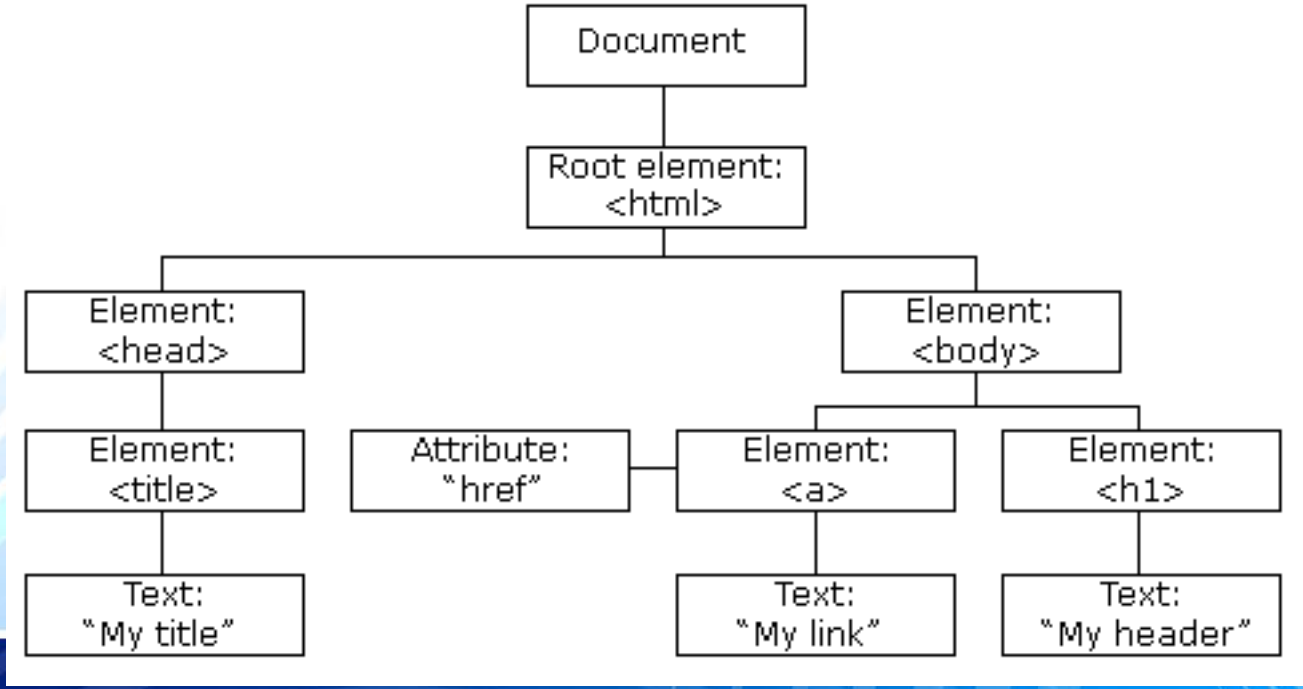
- Execution Control Structures
- Functions
- Object Orientation
- Document Object Model (DOM)

Document Object Model (DOM)

- The DOM is a W3C (World Wide Web Consortium) standard.
- *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- The DOM is separated into different parts:
  - **Core DOM** - defines a standard set of objects for any structured document.
  - **XML DOM** - defines a standard set of objects for XML documents.
  - **HTML DOM** - defines a standard set of objects for HTML documents

HTML DOM

- Basically, The HTML page is constructed as a tree of Objects.



Reference: https://www.w3schools.com/js/js_htmldom.asp

HTML DOM (cont.)

- The HTML DOM is a W3C standard and it is an abbreviation for the Document Object Model for HTML.
- The HTML DOM defines a standard set of objects for HTML, and a standard way to access and manipulate HTML documents.
- All HTML elements, along with their containing text and attributes, can be accessed through the DOM. The contents can be modified or deleted, and new elements can be created.
- The HTML DOM is platform and language independent. It can be used by any programming language like Java, JavaScript, and VBScript

HTML Document

- Everything in an HTML document is a node.
- Nodes
  - The entire document is a document node
  - Every HTML tag is an element node
  - The texts contained in the HTML elements are text nodes
  - Every HTML attribute is an attribute node
  - Comments are comment nodes
- Nodes have a hierarchical relationship to each other
- All nodes in an HTML document form a document tree (or node tree). The tree starts at the document node and continues to branch out until it has reached all text nodes at the lowest level of the tree.

HTML Document: Example

```
<html>

<head>
   <title>DOM Tutorial</title>
</head>

<body>
   <h1>DOM Lesson one</h1>
   <p>Hello world!</p>
</body>

</html>
```

- Every node except for the document node has a parent node. E.g. the parent node of the <head> and <body> nodes are the <html> node, and the parent node of the "Hello world!" text node is the <p> node.
- Most element nodes have child nodes. E.g. the <head> node has one child node: the <title> node. The <title> node also has one child node: the text node "DOM Tutorial".

HTML Document: Example

```
<html>

<head>
   <title>DOM Tutorial</title>
</head>

<body>
   <h1>DOM Lesson one</h1>
   <p>Hello world!</p>
</body>

</html>
```

- Nodes are siblings when they share a parent. E.g. the <h1> and <p> nodes are siblings, because their parent is the <body> node.
- Nodes can also have descendants. Descendants are all the nodes that are children of a node, or children of those children, and so on. E.g. all text nodes are descendants of the <html> node, while the first text node are descendant of the <head> node.
- Nodes can also have ancestors. Ancestors are nodes that are parents of a node, or parents of this parent, and so on. E.g. all text nodes have the <html> node as an ancestor.

Document Object

- JavaScript provides the document object for manipulating the document that is currently visible in the browser window.
- There are many methods provided in the document object. (can be found at https://developer.mozilla.org/en/docs/Web/API/Document)

Document Object (cont.)

- Summary of some methods

Methods	Descriptions
document.getElementById()	Returns the element that has the ID attribute with the specified value
document.getElementsByName()	Returns a NodeList containing all elements with a specified name
document.getElementsByTagName()	Returns a NodeList containing all elements with the specified tag name
document.write()	Writes HTML expressions or JavaScript code to a document
document.writeln()	Same as write(), but adds a newline character after each statement

Element Object

- Element Object is a part of a webpage. It is one of implementation of Node.
- Example:

```
<h1 style="color:blue;">DOM Lesson one</h1>
```

- Element object consists of:
  - Opening tag
  - Attributes
  - Text content
  - Closing tag

Element Object

- Summary of some methods/properties

Methods/Properties	Descriptions
Element.innerHTML	Sets or gets the HTML syntax describing the element's descendants.
Node.textContent	Returns / Sets the textual content of an element and all its descendants.
Node.nodeValue	Returns / Sets the value of the current node
Node.childNodes	Returns a live NodeList containing all the children of this node.
Node.firstChild	Returns a Node representing the first direct child node of the node, or null if the node has no child.
Node.getRootNode()	Returns the context object's root which optionally includes the shadow root if it is available.

Find and Access Nodes

- **getElementById()**
  - Returns the element that has the ID attribute with the specified value
  - Syntax:
    - elem = document.getElementById(’id');
- **getElementsByName()**
  - This method returns a collection of objects with the specified name in the format of array.

Find and Access Nodes (cont.)

- NodeList
- A nodeList usually stores the list in a variable, like this:

```
var nodeList =
document.getElementsByTagName('p');
```

- Now the variable nodeList contains a list of all <p> elements in the page, and we can access the <p> elements by their index numbers starting at 0.

```
for (var i = 0; i < nodeList.length; i++) {
    // do something with each paragraph
    output += nodeList[i].textContent + "<br/>";
}
```

Events

- Events are abilities to trigger actions in a browser.
- For example, What need to be done when we click on elements on a webpage.

Events (cont.)

- Summary of some events.

Event	Description
`onabort`	Fires when image transfer has been interrupted by user.
`onchange`	Fires when a new choice is made in a `select` element, or when a text input is changed and the element loses focus.
`onclick`	Fires when the user clicks using the mouse.
`ondblclick`	Fires when the mouse is double clicked.
`onfocus`	Fires when a form element gains focus.
`onkeydown`	Fires when the user pushes down a key.
`onkeypress`	Fires when the user presses then releases a key.
`onkeyup`	Fires when the user releases a key.
`onload`	Fires when an element and all its children have loaded.
`onsubmit`	Fires when a form is submitted.
`onunload`	Fires when a page is about to unload.

Events (cont.)

Event	Description
`onmousedown`	Fires when a mouse button is pressed down.
`onmousemove`	Fires when the mouse moves.
`onmouseout`	Fires when the mouse leaves an element.
`onmouseover`	Fires when the mouse enters an element.
`onmouseup`	Fires when a mouse button is released.
`onreset`	Fires when a form resets (i.e., the user clicks a reset button).
`onresize`	Fires when the size of an object changes (i.e., the user resizes a window or frame).
`onselect`	Fires when a text selection begins (applies to `input` or `textarea`).

References

- http://developer.mozilla.org/en-US/