CS381
Web Application Development

JavaScript

These class notes are based on the material from our textbook,
Learning PHP, MySQL & JavaScript, 5th ed., by Robin Nixon

JavaScript

- Client-side scripting language that runs entirely inside the web browser.
- Brings dynamic functionalities to a website.
- Placed between <script> and </script> HTML tags.
- Gained new power when HTML elements got a more formal structured definition, called **the DOM (Document Object Model)**; an interface for HTML and XML.
- DOM represents the page as a document of objects so that programs can change the document structure, style, and content.

```
<html><head><title>Hello World</title></head><body>
<script>
        document.write("Hello World")
</script>
<noscript>Your browser doesn't support or has disabled
JavaScript</noscript>
</body></html>
```

JavaScript

- **When to use a semicolon (;)**
  - To separate more than one statement on a single line.
  - To write bookmarklets: small pieces of JavaScript which can be easily executed via "bookmarks" or "favorites" features in a web browser.

- **Automatic Semicolon Insertion**: When there is a place where a semicolon is needed, JavaScript adds it behind the scenes.

- **Including js Files** <script src="script.js"></script>

- **Head Scripts Use**
  - to execute a script when a page loads.
  - to write things such as meta tags into the <head> section. *Metadata is information about data.*
- **Debugging**
  - Chrome: Ctrl-Shift-J
  - Edge: F12

JavaScript is a very loosely typed language; the type is determined when a value is assigned and can change as the variable appears in different contexts.

Variables

Variables Names

- May include only the letters a-z, A-Z, 0-9, the $ symbol, and underscore (_).
- No other characters, such as spaces or punctuation, are allowed in a variable name.
- The first character of a variable name can be only a-z, A-Z, $, or _
- Reserved words (keywords) cannot be used as identifiers or property names.
- Names are case-sensitive. Count, count, and COUNT are all different variables.
- There is no set limit on variable name lengths.

String Variables

You may include a **single** quote within a **double-quoted** string or a double quote within a single-quoted string. But you must escape a quote of the same type by using the backslash character.

greeting = “Hello there“

warning = ‘Be careful‘

greeting = "\"Hello there\" is a greeting"

warning = '\'Be careful\' is a warning'

Functions

A function definition (called a function declaration, or function statement) consists of the

function name **(** *list of parameters*) **{** body **}**

```
function test(a, b )
{       return a + b;
}
```

function can also be created by a **function expression**. It also **can be anonymous**; does not have a name.

```
const sum = function(a, b)
{       return a + b;
}
```

Arrow function expressions

```
const sum = (a, b) => a + b;

document.write( sum(5, 6) );
```

Variables

Global Variables

- *Used* or declared outside of all functions. or
- *Used* within functions without declaration. (not good)

```
a = 123                                  // Global scope
var b = 456                              // Global scope
if (a == 123) var c = 789                // Global scope
```

Local Variables

- Parameters; except Arrays are passed to functions by reference.
- Declared within a function.

```
function test(t, w )                     // Local scope
{    a = 123                             // Global scope
     var b = 456                         // Local scope
     if (a == 123) var c = 789           // Local scope
}
```

Strict Mode **"use strict";** undeclared variables can not be used

- At the top: enables strict mode for an entire script.
- In function: enables strict mode for just this function.

```
function test()
{    "use strict";
     console.log(x);
     var x = 10;
     let y = 20;
     const name = 'Khaled';
}
```

Hoisting a mechanism where variables and function declarations are moved to the top of their scope before code execution.

var vs. let var is function scoped but let is block scoped.

unlike var, a let variable cannot be re-declared within its scope.

var is initialized as undefined, and let is not initialized.

const behaves just like let but it must be initialized with a value and cannot be changed.

Arrays and Types

Arrays

```
var toys = new Array('bat', 'ball', 'whistle');        var toys = ['bat', 'ball', 'whistle']

var face = [ ['R', 'G', 'Y'],  ['W', 'R', 'O'],  ['Y', 'W', 'G']  ]

t = ['R', 'G', 'Y'];      m= ['W', 'R', 'O'];      b = ['Y', 'W', 'G'];      face = [t, m, b]

document.write( face[0][1] )                  ➔          G
```

undefined a variable has been declared but its value has not been assigned.

null null is a value.

typeof operator used to look up types

```
n = '838102050';                document.write(  typeof n  )  ==> string
```

String to number

```
n = "123";                      n *= 1
```

Number to String

```
n = 123;                        n += ""
```

Explicit Casting

JavaScript has no explicit type casting such as (int) or (float).

JavaScript provides the following built-in Functions:

- **parseInt()** parses a string and returns an integer. parseInt("50") => 50
- **parseFloat()** parses a string and returns a floating point number. parseFloat("50") => 50
- **Boolean()** finds whether expression is true or false
- **String()** returns a string representing the value of a number.
- **split()** splits a string into an array of substrings, and returns the new array.

```
var str = "How are you?";
var res = str.split(" ");
document.write( res );    // output: How,are,you?
```

JavaScript passes primitives types by value but arrays and objects by reference.

Checking Variable Scope

```
<html><head><title>Checking Scope Test</title></head><body>
<script>
        test()

        if( typeof a != 'undefined' ) document.write( 'a = '" + a + "'<br>' )
        if( typeof b != 'undefined' ) document.write( 'b = '" + b + "'<br>' )
        if( typeof c != 'undefined' ) document.write( 'c = '" + c + "'<br>' )

        function test()
        {       a = 123
                var b = 456
                if (a == 123) var c = 789
        }
</script>
</body></html>
```

The output:
a = "123"

Reading URL

```
<html><head><title>Reading  URL</title></head>
<body>
        <a id="mylink"  href="http://www.rcyci.edu.sa/en/">  Click me  </a><br>
        <script>                                                // or       var url = document.links[0].href
                var url = document.links.mylink.href            // or       var url = mylink.href
                document.write('The  URL is ' + url)
                document.write(  "<hr>Number  of items in the current session = "+ history.length  )

                // history.go(-3)       back 3 pages
                // history.back()       => Add button
                // history.forward()    => Add button
        </script>
</body>
</html>
```

Click me
The URL is http://www.rcyci.edu.sa/en/

Number of items in the current session = 2

document.write

Display all the links

```
for (j=0 ; j < document.links.length ; ++j)
        document.write(document.links[j].href + '<br>')
```

Change current page

```
document.location.href = 'http://rcyci.edu.sa'
```

With

- not recommended.
- It could create bugs and Ambiguity.

```
<script>
var str = "The quick brown fox jumps over the lazy dog“

with (str)
{
        document.write("The string is " + length + " characters<br>")
        document.write("In upper case it's: " + toUpperCase())
}
</script>
```

The string is 43 characters
In upper case it's: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

document.write

- not safe to be used.
- when called after a web page is fully loaded, it will overwrite the current document.

Exception Handling: onerror

element.onerror

an event that is triggered if an error occurs. It is a combination of the try and catch.

Event

action that can be detected by JavaScript. Every element has events that can trigger functions.

```
<html><head><title>onerror Test</title></head>
<body>
   <script>

      window.onerror = disErr;                    // assign a function to onerror event
      document.writ("Welcome");                   // deliberate error

      function disErr(message, url, lineNo)
      {
            alert(message + "\n\nURL:\n" + url + "\n\nLine:\n" + lineNo + "\n");
            return true;                          // true: prevents the firing of the default event handler.
      }

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

Exception Handling: Try and Catch

```
<html><head><title>Try  Catch </title></head><body>
<p>Enter a number between 50 and 100</p>
<input type="text" id="number" >
<button onclick="check()">Enter</button>
<p id="message"></p>
<script>      function check()
              {          var n = document.getElementById("number").value;
                         var msg = document.getElementById("message");
                         msg.innerHTML = "";

                         try {      if (n == "")              throw "Empty";
                                    if (isNaN(n))             throw "not a number";
                                    if (n > 100)              throw "too high";
                                    if (n < 50)               throw "too low";
                         }
                         catch (err) {  msg.innerHTML = "Input is " + err;
                         }

                         finally { // execute code, after try and catch, regardless of the result.
                         }
              }
</script>
</body></html>
```

Enter a number between 50 and 100

pp Enter

Input is not a number

Variable Arguments Functions

```
<html><head><title>Variable Arguments Functions</title></head><body>
<script>
        var words = fixNames("tHe", "DALLAS", "CowBoys");

        for (j = 0; j < words.length; ++j)
                document.write(words[j] + "<br>");

        function fixNames()
        {       var args = fixNames.arguments;
                var str = new Array();

                for (j = 0; j < args.length; ++j)
                        str[j] = args[j].charAt(0).toUpperCase() + args[j].substr(1).toLowerCase();
                return str;
        }
</script></body></html>
```

The
Dallas
Cowboys

Class Method Separately

```
<html><head><title> Class Method Separately </title></head><body><script>
        function User(name, username, password)          // functions are objects in JavaScript
        {       this.name = name;
                this.username = username;
                this.password = password;
                this.showUser = showUser;
        }
        function showUser()
        {       document.write("Name: " + this.name + "<br>");
                document.write("Username: " + this.username + "<br>");
                document.write("Password: " + this.password + "<br>");
        }
        var name = prompt("Name:");
        var u = new User(name, name+"user", name+"pass");
         u.showUser();
</script></body></html>
```

Class with Prototype

```
<html><head><title> Class using prototype </title></head><body><script>
        function User(name, username, password)
        {       this.name  = name;
                this.username = username;
                this.password  = password;

                User.prototype.showUser = function()
                {       document.write("Name: " + this.name + "<br>");
                        document.write("Username: " + this.username + "<br>");
                        document.write("Password: " + this.password + "<br>");
                }
        }
        var name = prompt("Name:");
        var u = new User(name, name+"user", name+"pass");
        u.showUser();
</script></body></html>
```

prototype

refers to a single instance of the method instead of creating a copy of it in every objects. It can save a lot of memory.

So, instead	this.showUser = function()
We could write	User.prototype.showUser = function()

Class

```
<html><head><title> Class </title></head><body>
<div id='v'></div>
<script>
        class Person
        {       #firstName;
                #lastName;
                constructor(firstName, lastName)
                {       this.#firstName = firstName;
                        this.#lastName = lastName;
                }
                getfullName() { return this.#firstName + " " + this.#lastName; }
        }

        var p = new Person('Tom', 'Hanks');
        document.getElementById('v').innerHTML = p.getfullName();
</script></body></html>
```

Public:
All the members of an object are public members.

Private:
#vars are private members.

Class Extending JavaScript Objects

```
<!DOCTYPE html><html><head><title> Class Extending JavaScript objects </title></head>
<body>

<p><button onclick="runShow()">My Show</button></p>

<H1><p id='p1'>do not  do that</p></H1>

<script>
      document.addEventListener("dblclick", runShow);

      function runShow()
      {
            var myP1 = document.getElementById('p1');
            myP1.innerHTML = myP1.innerHTML.myReplace();
      }

      String.prototype.myReplace = function()
      {
            return this.replace(/ /g, '')      // or   /\s/g
      }

</script>

</body></html>
```

prototype allows to add functionality to a built-in object as well.

Regular Expressions

are objects used as patterns for matching characters in strings.

Metacharacters	Description
/	Begins and ends the regular expression
.	Matches any single character except the newline
element*	Matches element zero or more times
element+	Matches element one or more times
element?	Matches element zero or one time
[characters]	Matches a character out of those contained within the brackets
[^characters]	Matches a single character that is not contained within the brackets
(regex)	Treats the regex as a group for counting or a following *, +, or ?
left|right	Matches either left or right
[l-r]	Matches a character from a range of characters between l and r
^	Requires match to be at the string's start
$	Requires match to be at the string's end

Regular Expressions

Metacharacters	Description
\b	Matches a word boundary
\B	Matches where there is not a word boundary
\d	Matches a single digit
\D	Matches a single nondigit
\n	Matches a newline character
\s	Matches a whitespace character
\S	Matches a nonwhitespace character (any char character but not a whitespace)
\t	Matches a tab character
\w	Matches a word character (a-z, A-Z, 0-9, and _)
\W	Matches a nonword character (anything but a-z, A-Z, 0-9, and _)
\x	Matches x (useful if x is a metacharacter, but you really want x)
{n}	Matches exactly n times
{n,}	Matches n times or more
{min,max}	Matches at least min and at most max times

Regular Expressions

\d	Whole Number 0 – 9 \d\d\d = 327 \d\d = 81 \d = 4	\d\d\d ≠ 24631
\w	Alphanumeric Char \w\w\w = dog \w\w\w\w = mule \w\w = to	\w\w\w = 467 \w\w\w\w = 4673 \w\w\w ≠ boat \w ≠ !
\W	Symbols \W = % \W = # \W\W\W = @#%	\W\W\W\W ≠ dog8
[a-z] [0-9]	One or no char from the set pand[ora] = panda pand[ora] = pando	pand[ora] ≠ pandora
(abc) (123)	matches abc or 123 in that exact order. pand(ora) = pandora pand(123) = pand123	pand(oar) ≠ pandora
|	OR pand(abc|123) = pandabc OR pand123	
?	zero or one times colou?r = colour colou?r = color	

Regular Expressions

*	zero or more times tre*= tree tre* = tr	tre* ≠ trees
+	one or more times tre+ = tree tre+ = tre	tre+ ≠ tr
.	any single character except the newline ton. = tone ton. = ton# ton. = ton4	ton. ≠ tones
.*	any character 0 or more times tr.* = tr tr.* = trees tr.* = treadmill	
{n}	exactly n times \d{3} = 836 \d{3} = 139 pand[ora]{2} = pandar pand[ora]{2} = pandoo pand(ora){2} = pandoraora	pand[ora]{2} ≠ pandora
{n,m}	at least min and at most max times \d{2,5} = 97430 \d{2,5} = 97	\d{2,5} ≠ 9
\+	Escaping RegEx \+[0-9]{11} any phone number stars with +	

Regular Expressions

\b	is used to find a match at the beginning or end of a word	str = "HELLO, LOOK AT YOU!"; /\bLO/ at the beginning of a word => location 7 /LO\b/ at the end of a word => location 3
\B	is used to find a match, but where it is NOT at the beginning or end of a word.	str = "HELLO, LOOK AT YOU!"; /\BLO/ not at a beginning of a word => location 3 /LO\B/ not at an end of a word => location 7
/^La +Voiture$/	Start with La at least one space and ends with Voiture	

Modifier	Meaning
/g	enables global matching. all matches, rather than only the first one.
/i	makes the regular expression match case-insensitive. Thus, instead of /[a-zA-Z]/, you could specify /[a-z]/i or /[A-Z]/i.

Using Regular Expressions

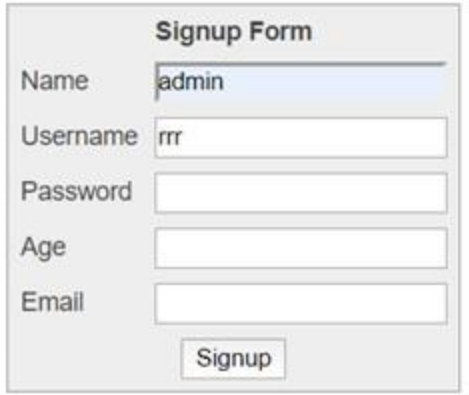
match()	searches a string for a match;	returns the matching array or null.
search()	searches a string for a specified value;	returns the position of the match or -1
replace()	searches a string for a specified value;	returns a new string after replacing the values.
test()	searches a string for a specified value;	returns true if found else false. re.test(str)

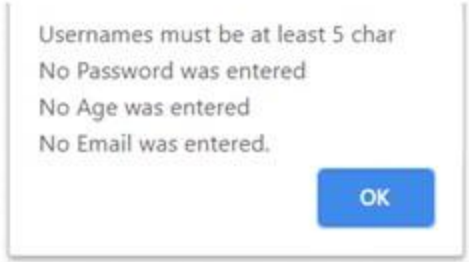
```
str = "The rain in SPAIN stays mainly in the plain";
```

`res = str.match(/ain/gi);`	→	ain,AIN,ain,ain
`res = str.search(/[a-c]/i);`	→	5
`res = str.replace(/[p]l?ain/gi, "");`	→	The rain in S stays mainly in the
`res = /y\b/.test(str);`	→	Yes

Form Validation

```
<!DOCTYPE html><html><head><title> Form Validation </title>
<style>.signup { border: 1px solid #999999; font: normal 14px helvetica;color: #444444;}</style>
<script src="FormValidation.js">
function validate(form)
{       fail = validateName(form.name.value);
        fail += validateUsername(form.username.value);
        fail += validatePassword(form.password.value);
        fail += validateAge(form.age.value);
        fail += validateEmail(form.email.value);
        if (fail == "")     return true;
        else { alert(fail); return false; }
}
</script></head><body>
<table class="signup" border="0" cellpadding="2" cellspacing="5" bgcolor="#eeeeee">
    <form method="post" action="#" onsubmit="return validate(this)">
        <th colspan="2" align="center"> Signup Form </th>
        <tr><td>Name                </td><td>  <input type="text" maxlength="32"  name="name"></td></tr>
        <tr><td>Username            </td><td>  <input type="text" maxlength="16"  name="username"></td></tr>
        <tr><td>Password            </td><td>  <input type="password" maxlength="12"  name="password"></td></tr>
        <tr><td>Age                 </td><td>  <input type="text" maxlength="3"   name="age"></td></tr>
        <tr><td>Email               </td><td>  <input type="text" maxlength="64"  name="email"></td></tr>
        <tr><td colspan="2" align="center">   <input type="submit" value="Signup"></td></tr>
    </form>
</table></body></html>
```





FormValidation.js

```
function validateName(field) { return (field == "") ? "No Forename was entered.\n" : ""}

function validateUsername(field) {
    if (field == "")                        return "No Username was entered\n"
    else if (field.length < 5)              return "Usernames must be at least 5 char\n"
    else if (/[^a-zA-Z0-9_-]/.test(field))  return "Only a-z, A-Z, 0-9, - and _ allowed in Usernames\n"
    return ""
}
function validatePassword(field) {
    if (field == "")                        return "No Password was entered\n"
    else if (field.length < 6)              return "Passwords must be at least 6 char\n"
    else if (!/[a-z]/.test(field) || ! /[A-Z]/.test(field) || !/[0-9]/.test(field))          return "one each of a-z, A-Z and 0-9\n"
    return ""
}
function validateAge(field) {
    if (field == "" || isNaN(field))        return "No Age was entered\n"
    else if (field < 18 || field > 110)     return "Age must be between 18 and 110\n"
    return ""
}
function validateEmail(field) {
    if (field == "")                        return "No Email was entered.\n"
    else if (!((field.indexOf(".") > 0) && (field.indexOf("@") > 0)) || /[^a-zA-Z0-9.@_-]/.test(field))
                                            return "The Email address is invalid.\n"
    return ""
}
```

Dynamic User Interface - DUI

An adaptable GUI that depends on some specifications like the utilization of XML/JSON to have an abstract interface, then each client displays its own representation of it.

```
<!DOCTYPE html><html>
<head><title> Dynamic User Interface</title> </head><body>
<script>
        const body = document.querySelector('body');          // returns the first element that matches a specified CSS selector
        body.setAttribute('align', 'center');

        const div = document.createElement('div');
        div.setAttribute('class', 'mydiv');
        body.append(div);

        const img = document.createElement('img');
        img.setAttribute('src', 'DUI.png');
        div.append(img);

        const a = document.createElement('a');
        a.innerHTML = "Students Information System";
        a.setAttribute('href', 'https://sis.rcyci.edu.sa/');
        body.append(a);

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

XML & JSON

Extensible Markup Language (XML)

is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable, used for data **storage** and **exchange**.

JSON (JavaScript Object Notation)

represents JavaScript objects as strings; an alternative method to XML for the data-exchange technique.

- It has **a simple format** making objects easy to read, create, and parse.
- Object — is represented as a **list** of property **names** and **values** contained in curly braces.
  - Object — { propertyName1 : value1, propertyName2 : value2 }
  - Array — [value0, value1, value2]
- Value — string, number, true, false, JSON object, null

Example — **{ "name":"John", "age":30, "cars":["Ford", "BMW", "Fiat"] }**

AJAX

Asynchronous JavaScript and XML (AJAX)

- A technique for creating fast and dynamic web pages.
- Allows web pages to be updated by exchanging small amounts of data with the server.
- Possible to update parts of a web page without reloading the whole page.
- Without AJAX the entire page must be reloaded if the content changed.

Steps

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The object sends a request to a web server.
4. The server processes the request.
5. The server sends a response back to the web page.
6. The response is read by JavaScript.
7. Proper action (like page update) is performed by JavaScript.

JSON

```
<!DOCTYPE html><html>
<head>
	<title>JSON</title>
</head>

<body>
	<div id='mydiv'></div>
	<script>
		var output = document.getElementById('mydiv');

		var jsonArrayString = '[ {"x": 7, "y": 42}, {"x": 0, "y": 0}, {"x": 2.7, "y": 3.14} ]';
		var pointsArray = JSON.parse(jsonArrayString);



		for(var i = 0; i < pointsArray.length; i++)
		{
			var point = pointsArray[i];
			output.innerHTML +=" <br>Point: " + point.x + ", " + point.y;
		}

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

JSON to JS Object:	var student	= JSON.parse(jsonStr);
JS Object to Jason:	var jsonStr	= JSON.stringify(student);

Point: 7, 42
Point: 0, 0
Point: 2.7, 3.14

JSON & AJAX

```
<!DOCTYPE html><html><head><title>JSON and AJAX</title></head>
<body>
<p><button type="button" onclick="getDisplay()"> Get Info </button></p>
<div id='mydiv'></div>
<script>

function getDisplay()
{    var r = new XMLHttpRequest();

     r.open('GET', 'https://support.oneskyapp.com/hc/en-us/article_attachments/202761627/example_1.json', true);
     r.send();

     r.onreadystatechange = function()
     {      if( r.readyState == 4 && r.status == 200)    // the response is ready
            {
                 var output = document.getElementById('mydiv');

                 var data = JSON.parse( r.responseText );
                 output.innerHTML += "<br><br>Fruit: " + data.fruit + "<br>Size: " + data.size + "<br>Color: " + data.color;

            }
     }
}
</script></body></html>
```

false if you want the code to stop and wait for the loading to finish.