FrontEnd Development basic

Module 2: Creating a logic operation with the page

2.1: Javascript Base

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Program Structure
4. Data Structures
5. Functions
6. String
7. Date
8. Errors

Module contents

- JavaScript Introduction
  - A brief history of JavaScript
  - JavaScript implementation
  - What is ECMAScript?

Module contents

- JavaScript introduction
  - A brief history of JavaScript
  - What is ECMAScript?
  - What is JavaScript?

A brief history of JavaScript

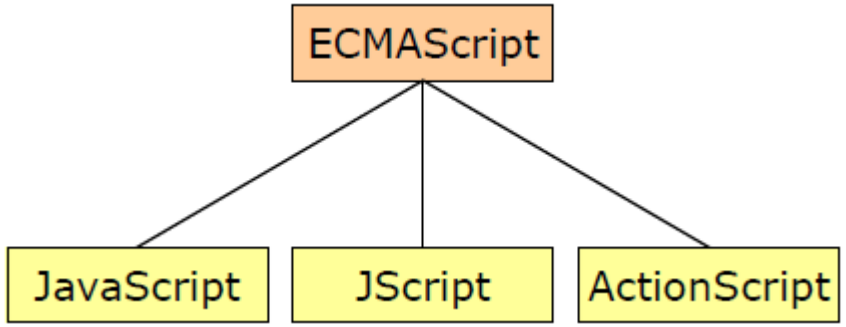
- Developed b Brendan Eich of Netscape, under the name of *Mocha*, then *LiveScript*, and finally *JavaScript*
- 1995 – JavaScript 1.0 in Netscape Navigator 2.0 (Dec)
- 1996 - JavaScript 1.1 in Netscape Navigator 3.0 (Aug), JScript 1.0 in Internet Explorer 3.0 (Aug). JavaScript had no standards governing its syntax or features
- 1997 – ECMAScript 1.0 (ECMA-262, based on JavaScript 1.1) (Jun), JavaScript 1.2 in Netscape Navigator 4.0 (Jun), JScript 3.0 in Internet Explorer 4.0 (Sep).
- 1998 – JavaScript 1.3 in Netscape 4.5 (ECMAScript 1.0) (Oct)
- 1999 - JScript 5.0 in Internet Explorer 5.0 (ECMAScript 1.0) (Mar) ECMAScript 3.0 (Regular expressions, error handling, etc) (Dec)
- 2000 – JScript 5.5 in Internet Explorer 5.5 (ECMAScript 3.0) (Jul), JavaScript 1.5 in Netscape 6.0 (ECMAScript 3.0) (Nov)
- 2001 – JScript 5.6 in Internet Explorer 6.0 (Aug)
- 2005 - JavaScript 1.6 I Firefox 1.5 (Nov)
- 2012 - all modern browsers fully support ECMAScript 5.1
- June 17, 2015 - ECMAScript 6 or ES6

Module contents

- JavaScript introduction
  - A brief history of JavaScript
  - What is ECMAScript?
  - What is JavaScript?

What is ECMAScript?

- ECMA stands for - European Computer Manufacturer's Association
- ECMAScript is a standard for a scripting language



- Each browser has its own implementation of the ECMAScript interface, which is the extended to contain the DOM and BOM

Module contents

- JavaScript introduction
  - A brief history of JavaScript
  - What is ECMAScript?
  - What is JavaScript?

JavaScript implementations 1/3

- JavaScript is:
  - Standardized (*based* on the ECMAScript standard)
  - A cross-platform
  - Dynamic typing
- A complete JavaScript implementations is made up of three distinct parts:
  - The Core (ECMAScript)
  - The Document Object Model (DOM)
  - The Browser Object Model (BOM)

JavaScript and Java 2/3

JavaScript	Java
Object-oriented (prototype-based). No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and methods can be added to any object dynamically.	Class-based. Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Variable data types are not declared (dynamic typing).	Variable data types must be declared (static typing).
Cannot automatically write to hard disk.	Can automatically write to hard disk.

Comparison of prototype-based (JavaScript) object systems and class-based (Java) 3/3

Prototype-based (JavaScript)	Class-based (Java, C++, C#)
All objects can inherit from an other object	Class and instance are distinct entities.
Define and create a set of objects with constructor functions.	Define a class with a class definition; instantiate a class with constructor methods.
Same	Create a single object with the new operator.
Construct an object hierarchy by assigning an object as the prototype associated with a constructor function.	Construct an object hierarchy by using class definitions to define subclasses of existing classes.
Inherit properties by following the prototype chain.	Inherit properties by following the class chain.
Constructor function or prototype specifies an *initial set* of properties. Can add or remove properties dynamically to individual objects or to the entire set of objects.	Class definition specifies *all* properties of all instances of a class. Cannot add properties dynamically at run time.

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Program Structure
4. Data Structures
5. Functions
6. String
7. Date
8. Errors

Module contents

- Getting started with JavaScript
  - The Web Console
  - Syntax
  - Keywords & Reserved words
  - Primitive and Reference values
  - Primitive types
  - Operators

The Web Console

- Logs information associated with a web page: any network requests, JavaScript, CSS, security errors and warnings as well as error, warning and informational messages explicitly logged by JavaScript code running in the page context
- Enables you to interact with a web page by executing JavaScript expressions in the context of the page
- Consist of three main sections: toolbar, command line, message display pane
- **console.log** to print out Javascript objects
- **console.dir** to print out Javascript objects with properties

Lab work #1

- Get practice the Web Console First Using:
  - Console.log(‘hello’);
  - console.dir({one: 1, two: {three: 3}});

Module contents

- Getting started with JavaScript
  - The Web Console
  - Syntax
  - Keywords & Reserved words
  - Primitive and Reference values
  - Primitive types
  - Operators
  - JavaScript Math Object

The basic concepts of JavaScript Syntax

- Everything is case-sensitive
- Variables are loosely typed
  - Use the *var* keyword
  - Variables don't have to be declared before being used
- End-of-line semicolons are optional

```
var test1="red",
    test2="green"; //do this to avoid confusion
```

- Comments are

  //

  /* ...*/
- Braces indicate code blocks
  - Use { } for grouping; not a separate scope

Module contents

- Getting started with JavaScript
  - The Web Console
  - Syntax
  - Keywords & Reserved words
  - Primitive and Reference values
  - Primitive types
  - Operators
  - JavaScript Math Object

Keywords & Reserved words

- The keywords and reserved words cannot be used as variables or function names
- Keywords

  break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with
- Reserved words

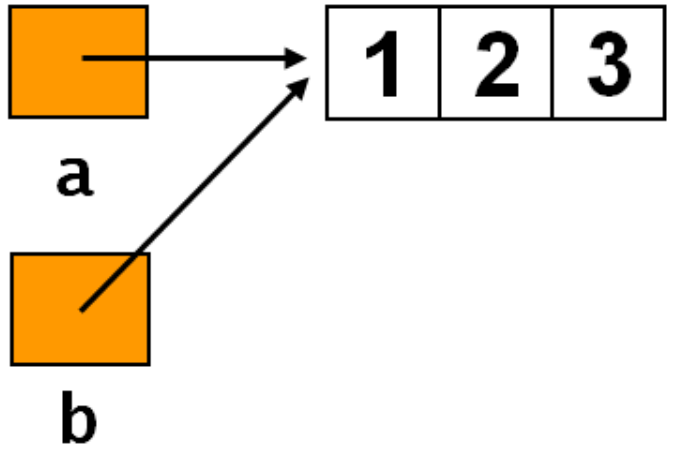
  abstract, boolean, byte, char, class, const, debugger, double, enum, export, extends, final, float, goto, implements, import, int, interface, long, native, package, private, protected, public, short, static, super, synchronized, throws, transient, volatile

Module contents

- Getting started with JavaScript
  - The Web Console
  - Syntax
  - Keywords & Reserved words
  - Primitive and Reference values
  - Primitive types
  - Operators
  - JavaScript Math Object

Primitive and Reference values

- Primitive values
  - are simple pieces of data that are stored on the *stack*
  - which is to say that their value is stored directory in the location that the variable accesses
  - the value is one of the JavaScript primitive values:
    - Undefined, Null, Boolean, Number, String
  - Many languages consider string as a reference type, but in JavaScript it is a primitive type
- Reference values
  - are objects that are stored in the *heap*
  - Meaning that the value stored in the variable location is a pointer to a location in memory where the object is store



Garbage collection

- Automatic reclamation of unused memory
  - JavaScript interpreter is able to detect when an object will never again be used by the program
  - If the object is no longer needed and its memory can be reclaimed
  - You can create all the garbage objects you want, and the system will clean up after you

Module contents

- Getting started with JavaScript
  - The Web Console
  - Syntax
  - Keywords & Reserved words
  - Primitive and Reference values
  - Primitive types
  - Operators
  - JavaScript Math Object

Primitive types

- Undefined
  - the Undefined type has only one value: *undefined* [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined]
- Null
  - the Null type has only one value: *null* [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN]
- Boolean
  - the Boolean type has two values: *true* and *false*
- Number
  - 32-bit integer and 64-bit floating-point values
  - Infinity → isFinite() [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Infinity]
  - Nan (Not a Number) → isNan() [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN]
- String
  - using either double quotes (") or single quotes (')
  - JavaScript has no character type

Primitive types Converting

- Automatic type conversion
  - console.log(8 * null)
    // → 0
  - console.log("5" - 1)
    // → 4
  - console.log("5" + 1)
    // → 51
  - console.log("five" * 2)
    // → NaN
  - console.log(false == 0)
    // → true
- Number()
- +String
- parseInt()
- parseFloat()

Module contents

- Getting started with JavaScript
  - The Web Console
  - Syntax
  - Keywords & Reserved words
  - Primitive and Reference values
  - Primitive types
  - Operators
  - JavaScript Math Object

Operators 1/10

- Binary

operand1 operator operand2

- Unary

operator operand

operand operator

- Conditional (ternary) operator

condition ? *val1* : *val2*

Operators 2/10

- Assignment operators
  =, +=, -=, *=, /=, **=, <<=, >>=, >>>=, &=, ^=, |=
- Arithmetic operators
  - +, -, *, /, % (remainder), ** (exponentiation)
  - Post/pre increment/decrement: ++, --
  - Unary operators: - (negation) ,+ (plus),
- Comparison operators
  - ==, !=, >, <, >=, <=
  - ===, !== (strictly equals and strictly not equals)

  Type and value of operand must match / must not match
- Logical operators
  - ! (logical NOT), && (logical AND), || (logical OR)
- Bitwise operators
  - &, |, ^, ~, <<, >>, >>>,

+ arithmetic operator behavior 3/10

```
"1"+1      // 11
1+"1"      //11
+"1"+1     // 2
+"123"+1          //124
0+"1"+1    // 011
1+true     //2
```

== (equal) vs === (strict equal) 4/10

```
//type conversation is performed before comparison
var v1 = (“5” == 5); //true

//no implicit type conversion
//true is only if both types and values are equal
var v2 = (“5” === 5); //false
var v3 = (5 === 5.0); //true
var v4 = (true == 1); //true
var v5 = (true == 2); //false
var v6 = (true == "1"); //true
```

Logical NOT 5/10

!op1

- Returns false if its single operand can be converted to true
- Otherwise, returns true
- Results may not be a boolean value

Logical AND 6/10

op1 && op2

- If op1 is true, expression evaluates to the value of op2.
- Otherwise the expression evaluates to the value of op1
- Results may not be a boolean value

Logical OR 7/10

op1 || op2

- If op1 is true, expression evaluates to the value of op1.
- Otherwise the expression evaluates to the value of op2
- Results may not be a boolean value

Typeof Operator 8/10

- It produces a string value naming the type of the value you give it

```
console.log(typeof 4.5)
// → number
console.log(typeof "x")
// → string
console.log(typeof true)
// → boolean
```

The Conditional Operator (?:) 9/10

- Syntax
  - ?:
- Operands:
  - The first operand must be (or be convertable to) a boolean value
  - The second and third operands may have any value
- The value returned by the conditional operator depends on the boolean value of the first operand

Operator precedence 10/10

Operator type	Individual operators
member	. []
call / create instance	() new
negation/increment	! ~ - + ++ -- typeof void delete
multiply/divide	* / %
addition/subtraction	+ -
bitwise shift	<< >> >>>
relational	< <= > >= in instanceof
equality	== != === !==
bitwise	In following order: &, ^, |
logical	In following order: &&, ||
conditional	?:
assignment	= += -= *= /= %= <<= >>= >>>= &= ^= |=
comma	,

Module contents

- Getting started with JavaScript
  - The Web Console
  - Syntax
  - Keywords & Reserved words
  - Primitive and Reference values
  - Primitive types
  - Operators
  - JavaScript Math Object

The Math Object

- A function that returns a new pseudorandom number between zero (inclusive) and one (exclusive) every time you call it
  - console.log(Math.random());
- A function that rounds down to the nearest whole number:
  - console.log(Math.floor(<digit>);
  - console.log(Math.round(<digit>);
- A function that rounds up to the nearest whole number:
  - console.log(Math.ceil(<digit>);
- A function that get a power of digit:
  - console.log(Math.pow(a,b);

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Program Structure
4. Data Structures
5. Functions
6. String
7. Date
8. Errors

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

Expressions 1/2

- An *expression* is any valid unit of code that resolves to a value
- JavaScript has the following expression categories:
  - Arithmetic
  - String
  - Logical
  - Primary expressions
  - Left-hand-side expressions
- Grouping operator ()
- Comprehensions (This is an experimental technology, part of the ECMAScript 2016 (ES7) proposal)
  - Array comprehensions
  - Generator comprehensions

Variables 2/2

- Multiple variables declaring syntax

  var *name_1* [= *value_1*] [,…, *name_n* [= *value_n*]]
- A variable name:
  - can be any word that isn't a reserved word
  - cannot include punctuation, except for the characters $ and _
  - Digits can also be part of variable names
- The = operator can be used at any time on existing variables to disconnect them from their current value and have them point to a new one.

Module contents

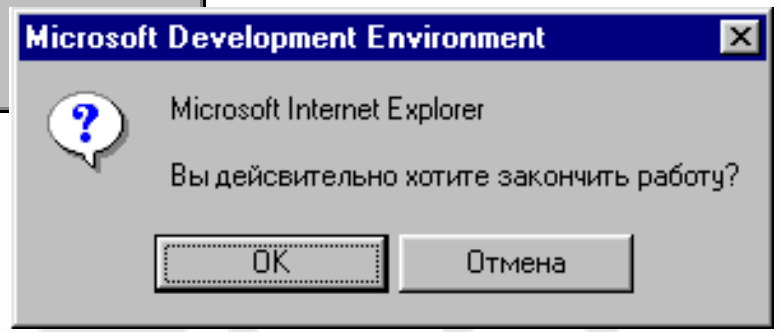
- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

Using JavaScript in HTML Page 1/3

- The <script> tag
- Attributes
  - async
  - charset
  - defer
  - src
  - type
    - type="text/javascript"

User input/output 2/3

- Input
  - confirm()
  - prompt()
- Output
  - alert()
  - document.write()
  - document.getElementById(id)



Using innerHTML 3/3

- To access an HTML element, JavaScript can use the method:

  document.getElementById(id);

- The **id** attribute defines the HTML element
- The **innerHTML** property defines the HTML content
- To "display data" in HTML, it will be set the value of an innerHTML property

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

Quality of a code 1/2

- Ability to debug a code and to correct mistakes
- Good style of a code
- To test a code, it is desirable — in the automatic mode

Maintainable code means code that:

- Is readable
- Is consistent
- Looks as if it was written by the same person
- Is documented

Online validate: Jshint.com 2/2

- **JSHint** is a community-driven tool to detect errors and potential problems in JavaScript code and to enforce your team's coding conventions.
- It is open source and will always stay this way.
- It was created and is maintained by Anton Kovalyov, a computer programmer from San Francisco

Lab work #2

- Please create the web page:
  - Ask to input two numbers with a prompt dialog window
  - Do the converting as it necessary
  - Ask to input an arithmetic operator
  - Output the calculated result in the web console
- Please validate code with online validator end errors

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

If/Else Statement

- Conditionally execute code
- Syntax

  if (*expression*) *statement1*

  [else *statement2*]
- Algorithm
  - *expression* is evaluated, and if it is true, *statement1* is executed;
  - otherwise, *statement2* is executed

Lab work #3

- Write a code for the following algorithm
  - Input balls from 0 to 100.
  - Display the received assessment according to the rules:

A	100-95
B	85-94
C	75-84
D	65-74
E	60-64
FX	0-59

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

Switch Statement

- Multiway branch to statements labeled with *case* or *default*
- Syntax

  switch (*expression*) {
  statements
  }
- Statements
  - case label_1: statements_1 [break;]
  - default: statements_def [break;]
- Algorithm
  - the value of *expression* is evaluated
  - Then it looks for a case label that matches that value
  - if it finds one, it starts executing the block of code following the case label
  - if it does not find a case label with a matching value, it starts execution the statement following a special-case default
  - if there is no default, it skips the block of code altogether.

Code Example

```
var text;
var fruits = "Banana";
switch(fruits) {
  case "Banana": text = "Banana is good!";break;
  case "Orange": text = "I am not a fan of orange.";break;    case
   "Apple":  text = "How you like them apples?";break; default: text
   = "I have never head of that fruit.";
}
console.log(text);
```

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Context (Local Scope, Global Scope)
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

Break/Continue Statement

- Break Syntax
  - break;
  - break *labelname*;
- Continue Syntax
  - continue;
  - continue *labelname*;

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Context (Local Scope, Global Scope)
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

Different Kinds of Loops

- *For* is an easy-to-use loop

  for (*initialize* ; *test* ; *increment*)*statement;*

  Initialize - is executed before the loop (the code block) starts

  Test - defines the condition for running the loop (the code block)

  Increment - is executed each time after the loop (the code block) has been executed

- *For/in* loops through the properties of an object

  for (*variable* **in** *object*) *statement;*

The **in** Operator

- How it works:
  - expects a lefthand operand that is or can be converted to a string
  - expects a righthand operand that is an object (or array)
  - evaluates to true if the lefthand value is the name of a property of the righthand object
- Code example

```
var person = {fname:"John", lname:"Doe", age:25}; //see later
 var text = "";
 var x;
 for (x in person) {
    text += person[x]+' ';
 }
console.log(text);
```

Module contents

- JavaScript Program Structure
  - Expressions and variables
  - User input/output
  - Quality of a code and online validate: Jshint.com
  - Context (Local Scope, Global Scope)
  - Conditional statements
    - If..else
    - Switch
    - Break/Continue
  - Loops and iteration
    - For, for..in
    - While, do..while

While and Do/While Statements

- A basic loop constructs
- While - loops through a block of code while a specified condition is true

  while(*expression*)

  statement
- do-while - also loops through a block of code while a specified condition is true

  do {

  statement } while(*expression*)

Lab work #4

- Prompt the user to input the marks of 5 subjects of a student. Display if he has passed in each subject if mark is above 60 else use break for
- Build a numerical calculator. Ask user for two inputs (numbers). Ask user for what function to perform:
    1. Addition,
    2. Subtraction,
    3. Multiplication
    4. Division.

  Display the result accordingly. And loop the code while user prompts – “That’s all”

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Program Structure
4. Data Structures
5. Functions
6. String
7. Date
8. Errors

Module contents

- Data Structures
  - Array and Associative Arrays
  - Object
  - The Difference Between Arrays and Objects
  - The delete Operator
  - Code Example

Module contents

- Data Structures
  - Array and Associative Arrays
  - Object
  - The Difference Between Arrays and Objects
  - The delete Operator
  - Code Example

Array 1/4

- A named collection of values, where all the values have the same type, and each value is identified by an index.
- The [ ] operator is used to access the numbered values of an array by index
- An integer variable or value used to indicate an element of an array: a[0] = 1; a[l] = 2;
- You create an array with the Array( ) constructor
- You can use array literal syntax to include arrays directly in a program
  - var *array-name* = [*item1*, *item2*, ...];

```
var points = new Array(); // Bad
var points = [];          // Good
```

Array property 2/4

- Every array has a *length* property that specifies the number of elements in the array
  - The first element is element 0
  - The last element is element length – 1
- A *prototype* property allows to add properties and methods to an Array object
- A *constructor* property returns the function that created the Array object's prototype

Main Array Methods 3/4

- To search the array for an element
  - indexOf()
  - lastIndexOf()
- To work with array as a stack
  - push()
  - pop()
- The **map()** method creates a new array with the results of calling a provided function on every element in this array
  - *arr*.map(*functionname* [, *thisArg*])

Associative Arrays (or Hashes) 4/4

- Array

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
console.log(person.length);
//person.length will return 3
console.log(person[0]);    //
person[0] will return "John"
```

- Hash

```
var person = [];
person["firstName"]="John";
person["lastName"] = "Doe";
person["age"] = 46;
console.log(person.length);
//person.length will return 0
console.log(person[0]);    //
person[0] will return undefined
```

Module contents

- Data Structures
  - Array and Associative Arrays
  - Object
  - The Difference Between Arrays and Objects
  - The delete Operator
  - Code Example

Objects 1/2

- An object is a compound data type that contains any number of properties
- Each property has a name and a value
- The dot (.) operator is used to access a named property of an object

```
obj.x = 1; obj.y = 2; obj.total = obj.x + obj.y;
```

- Any object can be assigned any property
- JavaScript objects are associative arrays: they associate arbitrary data values with arbitrary names

```
obj["x"] = 1; obj["y"] = 2;
```

The new operator create an Object 2/2

- Objects are created with the new operator with no properties
  - var o1 = new Object( );
- An object literal is a comma-separated list of name/value pairs, contained within curly braces
  - var o1 = {x:l, y:2, total:31};

Module contents

- Data Structures
  - Array and Associative Arrays
  - Object
  - The Difference Between Arrays and Objects
  - The delete Operator
  - Code Example

The Difference Between Arrays and Objects

- Index
  - In JavaScript, **arrays** use **numbered indexes**
  - In JavaScript, **objects** use **named indexes**

Arrays are a special kind of objects, with numbered indexes

- When to Use
  - You should use **objects** when you want the element names to be **strings (text)**
  - You should use **arrays** when you want the element names to be **numbers**

Module contents

- Data Structures
  - Array and Associative Arrays
  - Object
  - The Difference Between Arrays and Objects
  - The delete Operator
  - Code Example

The delete Operator

- The **delete operator** removes a property from an object
- Syntax
  - delete expression
- Expression
  - variableName
  - objectExpression.property
  - objectExpression["property"]
  - array[index]

Module contents

- Data Structures
  - Array and Associative Arrays
  - Object
  - The Difference Between Arrays and Objects
  - The delete Operator
  - Code Example

Code Example. Use Array.

```
var cars = ["KIA", "Lexus", "Suzuki"];
cars.push("Ford");
console.log(Object.keys(cars));
var i, len, text;
for (i = 0, len = cars.length, text = ""; i < len; i++)     text += cars[i] + ' ';
console.log(text);
//use map()
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
// roots is now [1, 2, 3], numbers is still [1, 4, 9]
```

Code Example. Use Object.

```
var txt = "";
var person = {fname:"Ivanov", lname:"Ivan", age:25};
var x;
for (x in person) {
   txt += person[x] + " ";
}
console.log(txt);
console.log(person.fname+" "+person.lname +" is "+ person.age
   + " years old“);
```

Code Example.Use Hash.

```
var objHash = {
  object1: { name: "First", size: 10, color: 'green' },
  object2: { name: "Second", size: 15, color: 'red' }
}
var nameIndex = {
  green: objHash.object1,
  red: objHash.object2
};
console.dir({ objHash: objHash, nameIndex: nameIndex });
nameIndex['green'].name = 'My'; //edit array by index
console.dir({ objHash: objHash, nameIndex: nameIndex });
```

Lab work #5

- Write code to create an array of 5 Cars
  - Car has following properties:
    - FirmName (string)
    - ModelName (string)
    - EngineDisplacement (float)
  - All parameters to prompt by user
- Output all Cars that has Engine Displacement > 2.0

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Statements
4. Data Structures
5. Functions
6. String
7. Date
8. Object
9. Errors

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

Function defining and calling

- A function is a piece of JavaScript code that is defined once and can be executed multiple times by a program
- Defining syntax
  - function *funcname*([*arg1* [,*arg2* [..., *argn*]]]) { *statements*}
- Calling syntax
  - *funcname* ([*value1* [, *value2* [..., *valuen*]]])
- A function definition creates a new function object and stores that object in a newly created property named *funcname*
- Parameter passing
  - Basic types passed by value, objects by reference
- return *expression*;
- Allow function definitions to be nested within other functions

Code Example. Using Function.

- A function definition looks like this:
  - function sum(x,y) { return x + y; }
- A function calling result:
  - var total = sum(1,2); // Total is now 3
  - console.log(total);
- Or use an anonymous functions:
  - console.log((function(x,y){return x+y;})(1,2)); //see further

Code Example. Use of return with a multiple choice

```
var op = {
    plus:  function(a,b) {return a+b; },
    minus:  function(a,b) {return a-b; },
    inc: function(a) {return ++a; },
    dec:  function(a) {return --a; },
  };
//first calling
var f = op['dec'];console.log(f(5)); //display 4
//second calling
(function m(operation, first, second) {
  return op[operation](first, second);
})('plus',2,3); //display 5
```

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

Use anonymous functions

- Simulate blocks by function definition and call

(function([*arg1* [,*arg2* [..., *argn*]]]){*statements*})([*value1* [, *value2* [...,*valuen*]]])

- "Anonymous" functions (expressions for functions)
  - (function (x,y) {return x+y}) (2,3);

```
var u = { a:1, b:2 }
var v = { a:3, b:4 }
(function (x,y) {  // "begin local block"
 var tempA = x.a;
 var tempB =x.b;
  x.a=y.a;
  x.b=y.b;
  y.a=tempA;
  y.b=tempB
 }) (u,v)          // "end local block"
// Side effects on u,v because objects are passed by reference
```

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

Callback Pattern

- To transfer the function as the parameter of other function

```
function f(cb1, cb2) {
  setTimeout(cb1, 5000);
  cb2();
  return true;}
var r =  f(
function() {  document.write('callback 1 executed ');}, function()
{  document.write('callback 2 executed');}
                );
document.write('function returned: '+r+' ');
```

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

Closures and Curried functions

- Closure is the binding which defines the **scope** of execution
- F**unctions** create a closure context.
- As the function return the function

```
function Adder(x){ return function(y){return x+y} };
var a = Adder(5);
var b = Adder(10);
console.log(a(2)); // 7
console.log(b(2)); //12
```

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

The Function( ) Constructor

- new *constructor*(*arguments*);
- var f = new Function("x", "y", "return x+y;");
  same as:
  - function *f*(x,y) { return x+y; }
  - var sum = function(x,y) { return x+y; }
- The Function( ) constructor expects any number of arguments
- All arguments are strings that specify the names of the parameters
- The last argument is the body of the function

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

A recursive function

- A **recursive function** is a function that calls itself:

```
function loop(x) {
    if (x >= 10) return;
    loop(x + 1);
}
```

- Recursion is used to solve problems that contain smaller sub-problems.
- A recursive function can receive two inputs:
  - a base case (ends recursion)
  - a recursive case (continues recursion).

Module contents

- Functions
  - Function defining and calling
    - Use anonymous functions
    - Callback Pattern
    - Closures and Curried functions
  - The Function( ) Constructor
  - A recursive function
  - Function Properties and Methods

Arguments[] Property 1/3

- Call can supply any number of arguments
  - functionname.length : the number of arguments in definition
  - functionname.arguments.length : the number args in call
  - functionname.arguments.callee()
- The **arguments[ ]** contains the complete set of arguments passed to the function

The callee method 2/3

- This is useful, for example, to allow unnamed functions to invoke themselves recursively

```
function(x) {
if (x <= 1) return 1;
return x * arguments.callee(x-1);
}
```

- Another code example

```
function func() {
    funcObj = arguments.callee
    funcObj.test++
    alert(funcObj.test)
}
func.test = 1;func();func();
```

The Difference Between Function and Method 3/3

- Function
  - Is a piece of JavaScript code that can be executed multiple times by a program
- Method
  - Belong to an object
- JavaScript includes many useful pre-defined methods
  - Combine with programmer-defined methods to make a program

Lab work #6

- Demonstration:
  - Please create function without arguments, that alerts the number of arguments and there names
  - Please call this function several times with a different values amount.

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Program Structure
4. Data Structures
5. Functions
6. String
7. Date
8. Errors

Module contents

- String
  - JavaScript Strings
  - Strings Methods
  - Regular Expression

Module contents

- String
  - JavaScript Strings
  - Strings Methods
  - Regular Expression

String

- Syntax
  - 'string text'
  - "string text“
  - String(thing)
    //Parameter **thing -** Anything to be converted to a string
  - new String(thing)
- Template string
  - `hello ${who}`
  - escape `<a>${who}</a>`
- Long literal strings
  - use the + operator to append multiple strings together
  - use the backslash character ("\")
- Comparing strings < or >

Module contents

- String
  - JavaScript Strings
  - Strings Methods
  - Regular Expression

The String Object Methods 1/2

- Syntax
  - <str-name>.<method-name>(parameters)

Method	Description
`charAt(index)`	Returns the character at specific position
`charCodeAt(index)`	Returns Unicode value of the character at specific position
`concat(string)`	Returns the string as the rezult of concatenating its argument
`fromCharCode(value1, valu2,...)`	Returns string created from series of Unicode values
`Indexof(substring, index)`	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found
`LastIndexf(Indexof(substring, index)`	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.

The String Object Methods 2/2

Method	Description
`slice( start, end )`	Extracts a section of a string and returns a new string.
`split( string )`	Splits a String object into an array of strings by separating the string into substrings.
`substr(start, length)`	Returns a string containing *length* characters starting from index *start* in the source string. If *length* is not specified, a string containing characters from *start* to the end of the source string is returned.
`substring(start, end )`	Returns a string containing the characters from index *start* up to but not including index *end* in the source string.
`toString()`	Returns the same string as the source string.
`valueOf()`	Returns the same string as the source string.
`toLowerCase()`	Returns lowercase version of string
`toUpperCase()`	Returns uppercase version of string

Module contents

- String
  - JavaScript Strings
  - Strings Methods
  - Regular Expression

Regular Expression

- Rules that govern which sequences of characters come up in a search
- It is also an object
- RegExp methods:exec(),test()
- String methods: match(),replace(),search(),split()
- How to create?
  - var re = /ab+c/;
  - var re = new RegExp("ab+c");
- Special characters: \ ^ $ * + ? . () {}, ect.

Code Example. Use Regular Expression

```
var text = "'Hello',23,\"Kyiv\",28.5,Kiev";
console.log(text);
var parsed = text.match(/[0-9a-zA-Z'"\\.]{1,}/g);
console.log(parsed);
```

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Program Structure
4. Data Structures
5. Functions
6. String
7. Date
8. Errors

Module contents

- Date
  - JavaScript Dates
  - Date Methods

Module contents

- Date
  - JavaScript Dates
  - Date Methods

The Date Object

- Constructor
  - new Date();
  - new Date(*value*);
  - new Date(*dateString*);
  - new Date(*year*, *month*[, *day*[, *hour*[, *minutes*[, *seconds*[, *milliseconds*]]]]]);
- Properties: prototype, length

Module contents

- Date
  - JavaScript Dates
  - Date Methods

The Date Object Methods

- All Methods descriptions are here:
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
- Demonstrate how to use main methods:
  - now()
  - getDate(),getDay(), getMonth()
  - setDate(), setMonth()
  - toJSON()

Training program

1. JavaScript Introduction
2. Getting started with JavaScript
3. JavaScript Program Structure
4. Data Structures
5. Functions
6. String
7. Date
8. Errors

Module contents

- Error
  - Try…catch
  - Throw
  - Finally

Module contents

- Error
  - Try…catch
  - Throw
  - Finally

Exception handling statements

- Throw statement
  - throw e //jump to catch, passing exception object as parameter
- try … catch statement
  - try { … //code to try
    } catch (e if e == …) { … //catch if first condition true
    } catch (e if e == …) { … //catch if second condition true
    } catch (e if e == …) { … //catch if third condition true
    } catch (e){ … // catch any exception
    } finally { … //code to execute after everything else
    }
- Exception types
  - Error [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error]
  - DOM Exception [https://developer.mozilla.org/en-US/docs/Web/API/DOMException]
  - DOM Error [https://developer.mozilla.org/en-US/docs/Web/API/DOMError]

try...catch statement

- The try block marks a block of statements to try, and specifies one or more responses should an exception be thrown

```
try { statements }
```

- The catch blocks handle all exceptions that may be generated in the try block

```
catch (catchID) { statements }
```

Module contents

- Error
  - Try…catch
  - Throw
  - Finally

Throw statement

- Syntax

```
throw expression;
```

- Note
  - *to specify an object when you throw an exception*
  - *then reference the object's properties in the* catch *block.*

Module contents

- Error
  - Try…catch
  - Throw
  - Finally

Finally statement

- The finally block executes whether or not an exception is thrown

```
finally{ statements }
```

- The finally block make a script fail gracefully when an exception occurs
- If the finally block returns a value, this value becomes the return value of the entiretry-catch-finally production