UNIT IV

Functions : Defining Functions -Invoking Functions - Function Arguments and Parameters-Functions As Values - Functions As Namespaces - Closures - Function Properties, Methods, and Constructor - Functional Programming - Classes and Prototypes :Classes and Constructors - Java-Style Classes in JavaScript - Augmenting Classes - Classes and Types - Object-Oriented Techniques in JavaScript - Subclasses – Modules-JavaScript in Web Browsers : Client-Side JavaScript - Embedding JavaScript in HTML.

What is Function in JavaScript?

- Functions are very important and useful in any programming language as they make the code reusable
- A function is a block of code which will be executed only if it is called.
- If you have a few lines of code that needs to be used several times, you can create a function including the repeating lines of code and then call the function wherever you want.

How to Create a Function in JavaScript

1. Use the keyword **function** followed by the name of the function.
2. After the function name, open and close parentheses.
3. After parenthesis, open and close curly braces.
4. Within curly braces, write your lines of code.
5. Syntax:

```
function functionname()
{
lines of code to be executed
}
```

```
<html>
<head>
  <title>Functions!!!</title>
  <script type="text/javascript">
    function myFunction()
    {
      document.write("This is a simple function.<br />");
    }
      myFunction();
  </script>
</head>
<body>
</body>
</html>
```

Function with Arguments

- Functions can be created with arguments.
- Arguments should be specified within parenthesis
- Syntax:

```
function functionname(arg1, arg2)
{
 lines of code to be executed
}
```

```
<html>
<head>
   <script type="text/javascript">
      var count = 0;
      function countVowels(name)
      {
         for (var i=0;i<name.length;i++)
         {
           if(name[i] == "a" || name[i] == "e" || name[i] == "i" || name[i] == "o" || name[i]
== "u")
            count = count + 1;
         }

      document.write("Hello " + name + "!!! Your name has " + count + " vowels.");
      }
      var myName = prompt("Please enter your name");
      countVowels(myName);
   </script>
</head>
<body>
</body>
</html>
```

JavaScript Function Syntax

- A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas:
  (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets:
  - **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.
- A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return

- When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":
- Example
- Calculate the product of two numbers, and return the result:
- var x = myFunction(4, 3); // Function is called, return value will end up in x

```
function myFunction(a, b) {
  return a * b;           // Function returns the product of a and b
}
```

- The result in x will be:
- 12

Local Variables

- Variables declared within a JavaScript function, become **LOCAL** to the function.
- Local variables can only be accessed from within the function.
- Example
- // code here can NOT use carName

```
function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

- Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.
- Local variables are created when a function starts, and deleted when the function is completed.

JavaScript Classes

JavaScript Class Syntax

- Use the keyword class to create a class.
- Always add a method named constructor():

- Syntax

```
class ClassName {
 constructor() { ... }
}
```

- Example

```
class Car {
 constructor(name, year) {
  this.name = name;
  this.year = year;
 }
}
```

- The example above creates a class named "Car".
- The class has two initial properties: "name" and "year".
- A JavaScript class is **not** an object.
- It is a **template** for JavaScript objects.

Using a Class

- When you have a class, you can use the class to create objects:
- Example

```
let myCar1 = new Car("Ford", 2014);
let myCar2 = new Car("Audi", 2019);Try it Yourself »
```

- The example above uses the **Car class** to create two **Car objects**.
- The constructor method is called automatically when a new object is created.

The Constructor Method

- The constructor method is a special method:
- It has to have the exact name "constructor"
- It is executed automatically when a new object is created
- It is used to initialize object properties
- JavaScript will add an empty constructor method if constructor method is not defined.

Class Methods

- Class methods are created with the same syntax as object methods.
- Use the keyword class to create a class.
- Always add a constructor() method.
- Then add any number of methods.
- Syntax

```
class ClassName {
constructor() { ... }
```

```
    method_1() { ... }
    method_2() { ... }
    method_3() { ... }
    }
}
```

- Create a Class method named "age", that returns the Car age:
- Example

```
class Car {
constructor(name, year) {
this.name = name;
this.year = year;
}
age() {
let date = new Date();
return date.getFullYear() - this.year;
}
}
let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML =
"My car is " + myCar.age() + " years old.";
```

Sending parameters to Class methods:

- Example

```
class Car {
constructor(name, year) {
this.name = name;
this.year = year;
}
age(x) {
return x - this.year;
}
}

let date = new Date();
let year = date.getFullYear();

let myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML=
"My car is " + myCar.age(year) + " years old.";
```

Object-Oriented Techniques

- JavaScript is an object-oriented programming language and so a programming language can be called object-oriented when it provides programmers with at least four basic capabilities to develop:
    - Encapsulation: It is the capability for storing related information, whether data or methods, mutually in a single object.

    - Aggregation: It is the ability to store one object inside another.
    - Inheritance: A class can depend upon another class or number of classes and inherit their variables and methods for some specific use.
    - Polymorphism: It is the potential of the concept of OOP for writing one function or method which works in a variety of different ways.
- Objects are composed of attributes, and when an attribute contains a function, it is considered to be a method of the object else; the attribute is considered a property.

Object Properties in JavaScript

- Object properties can be any of the three basic data types or any of the abstract data types.
- Object properties are variables which are used within the object's methods, but as well can be globally visible variables which are used throughout the page.
- The syntax for including any property to an object is:

```
Obj_Name . obj_Property = property_Value;
```

Example:

```
var obj1 = project.title;
```

JavaScript in Web Browsers

Client-Side JavaScript

- Client-side JavaScript is the most common form of the language.
- The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.
- The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.
- JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

HTML <script> Tag

- Example

  Write “Hello JavaScript!” with JavaScript:

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
```

Definition and Usage

- The <script> tag is used to embed a client-side script (JavaScript).
- The <script> element either contains scripting statements, or it points to an external script file through the src attribute.

- Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

Attributes

Attribute	Value	Description
async	async	Specifies that the script is executed asynchronously (only for external scripts)
crossorigin		
defer	defer	Specifies that the script is executed when the page has finished parsing (external scripts)
integrity		
nomodule		
nonce		
referrerpolicy		
src	URL	Specifies the URL of an external script file
type	scripttype	Specifies the media type of the script

Global Attributes

- The <script> tag also supports the global attributes in html.

Default CSS Settings

- Most browsers will display the <script> element with the following default values:

```
script {
  display: none;
}
```

REFERENCES

1. www.w3schools.com
2. www.guru99.com/javascript
3. www.tutorialspoint.com/javascript

----------xxxxxxxxxxxx----------