JavaScript Functions:-

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

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
}
```

Example save file (**simplefunction.html** file):-

```
<!DOCTYPE html>

<html>

<body>



<h2>JavaScript Functions</h2>



<p>This example calls a function which performs a calculation, and returns the result:</p>
```

```
<p id="demo"></p>


<script>

function myFunction(p1, p2)

{

 return p1 * p2;

}


document.getElementById("demo").innerHTML = myFunction(4, 3);


</script>


</body>

</html>
```

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:

```
let 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
}
```

to run above example save following code in file name returnfunction.html :-

```
<!DOCTYPE html>

<html>

<body>


<p id="demo"></p>
```

```
<script>

var x = myFunction(4, 3);

document.getElementById("demo").innerHTML = x;


function myFunction(a, b)

{

return  a * b;

}

</script>


</body>

</html>
```

Output:-

12

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

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;
  }


}
```

he 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);
```

Now write following code in file calssexample.html:-

```
<!DOCTYPE html>

<html>

<body>

<p id="demo"></p>


<script>

class Car {

  constructor(name, year) {

    this.name = name;

    this.year = year;

  }

}
```

```
const myCar = new Car("Ford", 2014);

document.getElementById("demo").innerHTML =

myCar.name + " " + myCar.year;

</script>


</body>

</html>
```

Output:-

Ford 2014

The example above uses the **Car class** to create two **Car objects**.

The constructor method is called automatically when a new object is created.

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 save file name Car.html :-

```
<html>

<body>

<p id="demo"></p>


<script>

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.";
```

</script>

</body>

</html>

Output:-

My car is 8 years old.

You can send parameters to Class methods:-

Example

<!DOCTYPE html>

<html>

<body>

<script>

```
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.";

</script>


</body>

</html>
```

Output:-

My car is 8 years old.

Class Inheritance:-

To create a class inheritance, use the `extends` keyword.

A class created with a class inheritance inherits all the methods from another class:

Example

Create a class named "Model" which will inherit the methods from the "Car" class:-

```
<!DOCTYPE html>

<html>

<body>
```

<script>

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

</script>

</body>

</html>

Output:-

I have a Ford, it is a Mustang

The `super()` method refers to the parent class.

By calling the `super()` method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

Inheritance is useful for code reusability: reuse properties and methods of an existing class when you create a new class.