

Internet Programming I

Chapter 4 JavaScript: Functions



Group: 5
3rd Year Sec B Students,
Dept. of Software Eng.

Jan 2022, AASTU

Objectives



After success completion of the session you will be able:

- To Declare of function and understand its scope
- To understand about JavaScript nested function
- To understand the basics about parameters and arguments
- To describe different kinds of parameters
- To understand getter and setter methods.
- To know about the argument object property.
- To understand function Hoisting.
- To describe different kinds of functions
- To know mostly used methods in objects, date, and

Main Topic

- Overview of Function
- Components of Function
- Types of Function
- Function nesting and closure
- Hoisting
- Special function: setter and getter
- mostly used Built in method by category

4.1 overview of Function

- A function is a block of code that performs a specific task.
- A function declaration tells the JavaScript engine about a function's name, return type, and parameters.

- **Syntax:**

```
function name( parameter1, parameter2, parameter1){  
    statements;  
}
```

← This are the
components of
functions

- A function is declared using the **function** keyword.
- The basic rules of naming a function are similar to naming a variable. It is better to write a descriptive name for your function. For example, if a function is used to add two numbers, you could name the function add or addNumbers.
- The body of function is written within {}.

4.2 Parameters and Arguments

Default parameter – allows formal parameter to be initialized to default value if no value or undefined is passed:

Example:

```
function add(a, b = 1) {  
    return a + b;  
}
```

```
console.log(add(5, 2));  
// expected output: 7
```

```
console.log(add(5));  
// expected output: 6
```

2.What is the argument object

Argument objects – array-like local objects used to store arguments of non-arrow functions:

Example:

```
function findMax() {  
    let max = -Infinity;  
    for(let i = 0; i < arguments.length; i++) {  
        if (arguments[i] > max) {  
            max = arguments[i];  
        }  
    }  
    return max;  
    console.log(findMax(2, 3, 4, 5, 6)) // 6 is output  
}
```

Rest parameters

Rest parameters allows a function to accept indefinite number of arguments as an array. **Syntax:** `function add(num1, num2, ...theArgs) {}`

Example:

```
function add(num1, num2, ...otherNums) {  
    let normal_paramAdd = num1 + num2;  
    let other_paramsAdd = 0  
    let all_paramsAdd = 0
```

```
    for(let i = 0; i < otherNums.length; i++) {  
        other_paramsAdd += otherNums[i];  
    }  
}
```

```
add(1, 2, 3, 4, 5)  
//output 3, 12, 15
```

```
for (let j = 0; j < arguments.length; j++) {  
    all_paramsAdd += arguments[j];  
}
```

```
console.log(normal_paramAdd)  
console.log(other_paramsAdd)  
console.log(all_paramsAdd)text  
}
```

4.2.2. JavaScript Function Calling

- The **Function call** is a predefined JavaScript method, which is used to write methods for different objects. It calls the method, taking the owner object as an argument.
- JavaScript functions can be called:
 - As a function
 - As a method
 - As a constructor
 - via call and apply
- *Syntax:*
 call()
- *Return value:* It calls and returns a method with the owner object being the argument.

4.2.2. continued...

Calling a function as a function

```
<button
onclick="sayHello()">say
hello</button>
<script>
  function sayHello() {
    console.log(this);
    console.log(arguments);
    console.log('hello');
  }
</script>
```

Calling a function as a method

```
<button onclick="greeter.sayHello()">say
hello</button>
<script>
  greeter = {
    sayHello: function () {
      console.log(this);
      console.log(arguments);
      console.log('hello');
    }
  }
</script>
```

4.2.2. continued...

Calling a function as a constructor

```
function Greeter(name) {  
    console.info('begin constructor');  
    console.log(this);  
    console.log(arguments);  
    this.name = name;  
    console.info('end constructor')  
}  
  
function sayHello() {  
    var name =  
document.getElementById('name').value;  
  
    var grtr = new Greeter(name);  
    console.log('hello ' + grtr.name);  
}
```

Calling a function via call and apply

```
const person = {  
    fullName: function() {  
        return this.firstName + " " +  
this.lastName;  
    }  
}  
  
const person1 = {  
    firstName: "John",  
    lastName: "Doe"  
}  
  
const person2 = {  
    firstName: "Mary",  
    lastName: "Doe"  
}  
  
// This will return "John Doe":  
person.fullName.call(person1);
```

4.3 Types of function

Here are the major classification of functions

- classical Functions
- Function Expression
- Anonymous function
- Arrow Function

Classical Functions

This are the functions that are used normally in javascript with the following syntax

```
function nameOfFunction () {  
    // function body  
}
```

4.3.2 JavaScript Function Expression

- A function expression is very similar to and has almost the same syntax as a function declaration.
- The main difference between a function expression and a function declaration is that it can be stored in a variable.
- Such a function can be anonymous; it does not have to have a name.

```
const square = function(number) { return number * number }
```

```
var x = square(4)
```

- Also by Providing a name allows the function to refer to itself, and also makes it easier to identify the function in a debugger's stack traces:

```
const factorial = function fac(n) { return n < 2 ? 1 : n * fac(n - 1) }
```

```
console.log(factorial(3))
```

4.3.3 Anonymous Function

- ❑ In JavaScript, an anonymous function is that type of function that has no name or we can say which is without any name.
- ❑ We can also use an anonymous function as an argument for another function.

```
setTimeout(function () {  
    console.log('Execute later after 1 second')  
}, 1000);
```

- ❑ In order to invoke and execute a function immediately after its declaration, creating an anonymous function is the best way.

```
(function() { console.log('Hello'); })();
```

4.3.4 Arrow Function

- ❑ An arrow function expression is a compact alternative to a traditional function expression, but is limited and can't be used in all situation.

Differences & Limitations:

- Does not have its own bindings to **this** or **super**, and should not be used as methods.
- Does not have **new.target** keyword.
- Not suitable for **call**, **apply** and **bind** methods, which generally rely on establishing a **scope**.
- Can not be used as constructors.
- Can not use **yield**, within its body.

4.3.4 continued...

Arrow Function General Syntax:

let myFunction = (arg1, arg2, ...argN) => { statement(s) }

Additional syntax types include:

1. `param` `=>` `expression`

2. `(param1, paramN)` `=>` `expression`

3. `param => {`

`let a = 1;`

`return a + param;`

`}`

4.3.4 continued...

Comparing traditional functions to arrow functions

```
// Traditional Anonymous Function
function (a){
  return a + 100;
}
```

```
// Arrow Function Break Down
```

```
// 1. Remove the word "function" and place
arrow between the argument and opening body
bracket
```

```
(a) => {
  return a + 100;
}
```

```
// 2. Remove the body braces and word "return"
-- the return is implied.
```

```
(a) => a + 100;
```

```
// 3. Remove the argument parentheses
```

```
a => a + 100;
```

```
// Traditional Function
function bob (a){
  return a + 100;
}
```

```
// Arrow Function
```

```
let bob = a => a + 100;
```

```
// Traditional Anonymous Function
function (a, b){
  let chuck = 42;
  return a + b + chuck;
}
```

```
// Arrow Function
```

```
(a, b) => {
  let chuck = 42;
  return a + b + chuck;
}
```


4.4 Closure

- A closure is an expression (most commonly, a function) that can have free variables together with an environment that binds those variables (that "closes" the expression).
- A closure is a function that remembers its outer variables and can access them.

The closure has three scope chains listed as follows:

1. Access to its own scope.
 2. Access to the variables of the outer function
 3. Access to the global variables.
- In JavaScript, closures are created every time a function is created, at function creation time.

4.4.3 Nested Function

- A nested function is a function inside another function.
- one of the biggest feature in javascript is the ability to create function inside a function.
- Since a nested function is a closure, this means that a nested function can "inherit" the arguments and variables of its containing function.
- In other words, the inner function contains the scope of the outer function.

Example:

```
function addSquares(a, b) {  
    function square(x) {  
        return x * x;  
    }  
    return square(a) + square(b);  
}  
  
a = addSquares(2, 3); // returns 13  
b = addSquares(3, 4); // returns 25  
c = addSquares(4, 5); // returns 41
```

```
function outside(x) {  
    function inside(y) {  
        return x + y;  
    }  
    return inside;  
}  
  
fn_inside = outside(3); // Think of it like:  
give me a function that adds 3 to whatever  
you give  
// it  
result = fn_inside(5); // returns 8  
  
result1 = outside(3)(5); // returns 8
```

4.4.3 Function Scope

- Variables defined inside a function cannot be accessed from anywhere outside the function
- A function defined inside another function can also access all variables defined in its parent function, and any other variables to which the parent function has access

JavaScript has 3 types of scope:

- Block scope
- Function scope
- Global scope

```
function myFunction() {  
  let carName = "Volvo";  
  //Function Scope  
}
```

★ A function body creates a scope for let, const and even function declarations.

4.5 Special kinds of methods(Accessor Property)

- In JavaScript, accessor properties are methods that get or set the value of an object. For that, we use these two keywords:

get - to define a getter method to get the property value
- getter methods are used to access the properties of an object

set - to define a setter method to set the property value
- setter methods are used to change the values of an object.

```
const student = { firstName: 'Monica',  
  // accessor property(getter)  
  get getName() {return this.firstName; }  
};  
console.log(student.firstName); // Monica  
// accessing getter methods  
console.log(student.getName); // Monica  
// trying to access as a method  
console.log(student.getName()); // error
```

```
const student = {  firstName: 'Monica',  
  //accessor property(setter)  
  set changeName(newName) {  
    this.firstName = newName;  
  }  
};  
console.log(student.firstName); // Monica  
// change(set) object property using a setter  
student.changeName = 'Sarah';  
console.log(student.firstName); // Sarah
```

4.6 Hoisting

- Hoisting in JavaScript is a behavior in which a function or a variable can be used before declaration.

Variable Hoisting

```
a = 5;  
  
console.log(a);  
  
var a; // 5
```

Function Hoisting

```
greet();  
  
function greet() {  
    console.log('Hi, there.');
```

```
}
```

4.7 mostly used built in Functions by catagory

Object methods	
Object.create()	is used to create a new object and link it to the prototype of an existing object.
Object.keys()	creates an array containing the keys of an object.
Object.values()	creates an array containing the values of an object.
Object.entries()	creates a nested array of the key/value pairs of an object.
Object.assign()	is used to copy values from one object to another.
Object.freeze()	prevents modification to properties and values of an object, and prevents properties from being added or removed from an object.

continued...

Math methods	
log()	Returns the natural logarithm (base E) of a number.
max() and min()	Returns the largest and smallest of zero or more numbers repectively.
random()	Returns a pseudo-random number between 0 and 1.
round()	Returns the value of a number rounded to the nearest integer.
exp()	Returns E^N , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
ceil()	Returns the smallest integer greater than or equal to a number.

continued...

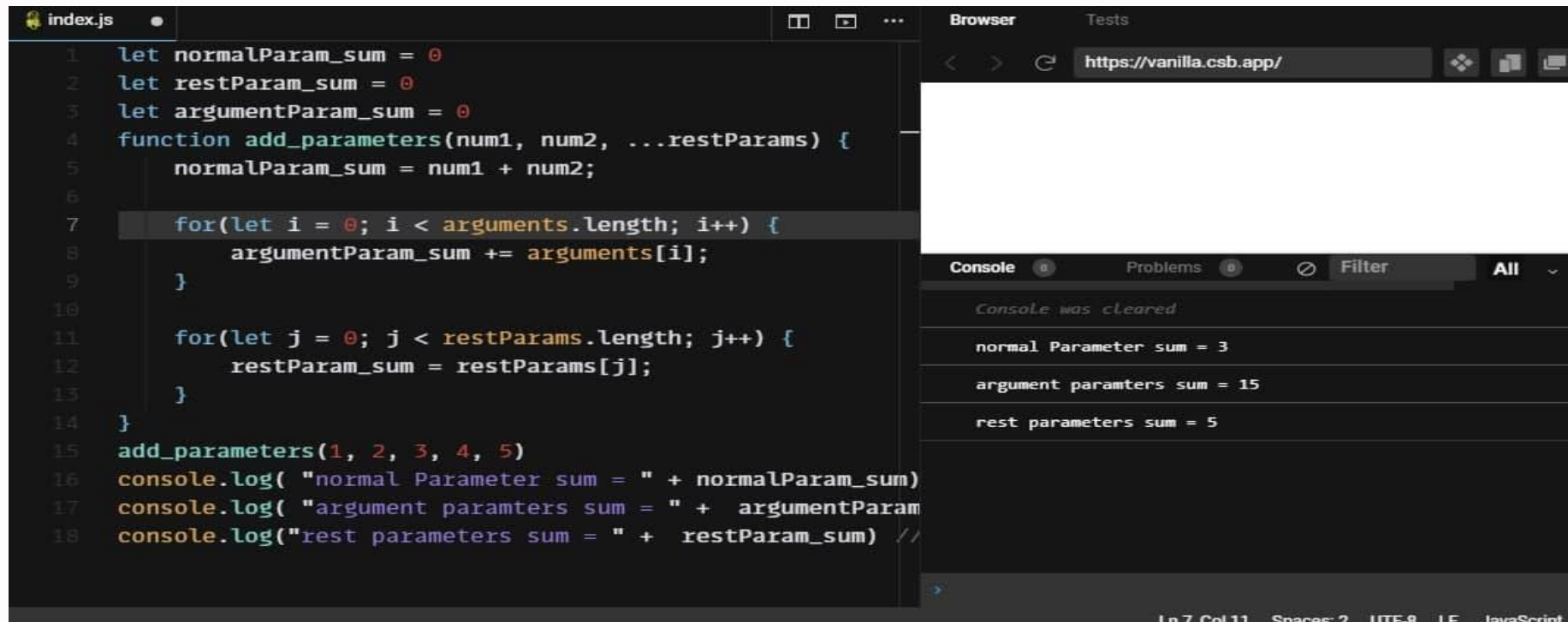
Date methods	
Date()	returns a string representation of the current date and time
getDate() and getDay()	Returns the day of the month/week for the specified date according to local time respectively.
setDate()	Sets the day of the month for a specified date according to local time.
getUTCDate()	Returns the day (date) of the month in the specified date according to universal time.
getTimezoneOffset()	Returns the time-zone offset in minutes for the current locale.
toSource()	Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.

Practical Exercises

1. What is the difference between JavaScript function declaration and function expression.
2. What is The difference between getter and setter methods in javascript?
3. What is The argument object?
4. What is Hoisting?
5. Write a JavaScript function that accepts two arguments, a string and a letter and the function will count the number of occurrences of the specified letter within the string?
6. Write a JavaScript program to pass a 'JavaScript function' as parameter?
7. Write a JavaScript function that checks whether a passed string is palindrome or not?

Demonstration question

1. Write a function that has two parameters and additional rest parameters. Your task is to write a function that finds summation of passed arguments for normal parameter, argument objects and rest parameters. [rest parameters, and argument object exercise]



The screenshot shows a code editor with a file named `index.js` and a browser window displaying `https://vanilla.csb.app/`. The code in the editor defines three variables for sums and a function `add_parameters` that takes two normal parameters and a rest parameter. It uses `arguments` to sum the first two parameters and the rest parameter array. The function is called with `add_parameters(1, 2, 3, 4, 5)`, and the results are logged to the console.

```
1 let normalParam_sum = 0
2 let restParam_sum = 0
3 let argumentParam_sum = 0
4 function add_parameters(num1, num2, ...restParams) {
5   normalParam_sum = num1 + num2;
6
7   for(let i = 0; i < arguments.length; i++) {
8     argumentParam_sum += arguments[i];
9   }
10
11   for(let j = 0; j < restParams.length; j++) {
12     restParam_sum = restParams[j];
13   }
14 }
15 add_parameters(1, 2, 3, 4, 5)
16 console.log( "normal Parameter sum = " + normalParam_sum)
17 console.log( "argument paramters sum = " + argumentParam
18 console.log("rest parameters sum = " + restParam_sum) //
```

The browser console shows the following output:

```
Console was cleared
normal Parameter sum = 3
argument paramters sum = 15
rest parameters sum = 5
```

Demonstration question

2. update activity 6 html code and write a built in function in your js code that will display the time of arrival for the plane by using the built in function DATE();. additional create additional object by using object.create() method and display the properties and values by using the built in methods object.keys() and object.values() in the text area.

```
const Airplane = {
  name: "none",
  model: "boing",
  date: Date(),
  print: function(){
    document.getElementById('text').innerHTML = "plane name: "
+ this.name;
  }
};
const plane = Object.create(Airplane);
plane.name = "emirates";
plane.model = "boing";
plane.date = Date();
```

```
let keys = () => {
  let keyValues = Object.keys(Airplane);
  document.getElementById('text').innerHTML =
"object keys are: " + keyValues;
}
```

```
let values = () => {
  let obValues = Object.values(plane);
  document.getElementById('text').innerHTML =
"object values are: " + obValues;
}
```

Demonstration question

OUTPUT:

object.create() =>

The image shows two side-by-side screenshots of a web application interface. Each screenshot has a light blue header bar with four buttons: 'LAND', 'TAKEOFF', 'Object keys', and 'Object values'. In the left screenshot, the 'Object keys' button is highlighted in black. Below the header is a text area containing the text 'plane name: emirates'. To the right of the text area is a black button labeled 'Plane information'. In the right screenshot, the 'Object values' button is highlighted in black. The text area now contains the text 'object keys are: name,model,date,print'. The 'Plane information' button is now light gray.

using Object.value() and Date() =>

The image shows a single screenshot of the web application interface. The 'Object values' button in the header is highlighted in black. The text area displays the following text: 'object values are: emirates,boing,Wed Feb 09 2022 08:06:14 GMT+0000 (Greenwich Mean Time)'. The 'Plane information' button is light gray.

object.key() ↑

Reading Resources/Materials

Links

- ✓ <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function>
- ✓ https://www.w3schools.com/js/js_scope.asp
- ✓ <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/get>

Thank You
For Your Attention!!

Any Questions

