



**Persistent**

# JavaScript: Functions

Persistent Interactive | Persistent University



## Key learning points :

- Overview, importance and characteristics of functions
- Let and Const keyword
- Different types of functions :
  - Function Declaration
  - Function Expression
  - Anonymous Functions
  - Arrow functions
- Scopes
  - Local
  - Global
- Error Handling in JavaScript

## What if we don't write functions ?

- Without functions code is
  - redundant
  - any changes result lot of reworking
  - tedious to type in
  - problematic to read if code needs to change
  - not maintainable
  - error-prone
  - not reusable
  - not optimized
- Write functions to eliminate all above limitations !

# JavaScript Functions

- A function is
  - group of code which can be called anywhere on the page
  - used to reuse the code to compute different things based on different inputs
- Different approaches to write a function :-
  - function declaration
  - function expression
  - anonymous function

## How to write a function in JavaScript ?

- Follow below steps to write a function :-
  - 'function' keyword followed by
  - function name (optional)
  - parenthesis for function parameters '( )'
  - function parameters (optional) followed by
  - curly braces '{}'
  - function body inside curly braces (optional)

```
function display(message){  
    console.log(message);  
}
```

```
(function(){  
    console.log('learn functions');  
});
```

## How to invoke a function in JavaScript ?

- Function with names :-
  - function name followed by
  - parenthesis '()'
  - arguments within parenthesis (optional)
- Function without names :-
  - parenthesis '()'
  - arguments within parenthesis (optional)

```
display('learn function declaration');
```

```
(function(message)
{
    console.log(message);
})("learn anonymous functions");
```

# JavaScript Scope and Execution

- JavaScript with **ES5 syntax(i.e. using var)** does not support block level scope and only manages function level scope, there are only two scopes:
  - global scope
  - local scope(function level scope)
- In declaration scenario, JavaScript follows C-language declaration system, declaring at the top in the global context
- All function and variable declarations are maintained at the top, called **Declaration Hoisting**

## Block-scoped Variables...

- Block scoping is done via **let** and **const** in **ES6**.
- Block scoped variables exists only within the block that surrounds them.

```
function newFunc(){  
  if(true) {  
    const test =123;  
  }  
  console.log(test);    //ReferenceError: test is not  
                        defined  
}
```



## Example using let:

- let declared variable exists only inside the block where they are declared.

```
function newFun(){  
  if (a>b){  
    let test=a;  
    a=b;  
    b=test;  
  }  
  
  console.log(test===a);  
  //Reference Error : test is not defined  
  return a;  
}  
newFun();
```

## Example using const:

- Variables declared with const must be initialized immediately with a value and that value cannot be changed afterwards.

```
const test1;
```

```
//Syntax Error: missing = in const declaration
```

```
const test2= 123;
```

```
test2=456;
```

```
//TypeError: 'test2' is read-only
```

## Block-scoped Functions:

- Block scoped functions are similar to let function expressions in which function definition is removed once the function is done with the execution.
- These functions are hoisted to the top of the containing block. But the function expressions that use let are not hoisted.

```
{  
    function fun () { return 1 }  
    console.log(fun() === 1)  
    {  
        function fun () { return 2 }  
        console.log(fun() === 2)  
    }  
    console.log(fun() === 1)  
} // true true true
```

## Comparison of let and const to var:

- Unlike var, const and let are block level declarations.
- const and let aren't hoisted.
- Repeated declarations produce errors in case of let and const.
- They don't create properties of the global object when used at global scope that happens with var declaration.
- Unlike let, Attempting to assign a const to a previously defined constant will throw an error in both strict mode and non-strict mode.

## Function Declaration

- Function name is mandatory when writing function using function declaration syntax

Function  
Name

Function  
Parameters

Return value

```
<script type="text/javascript">
```

```
function sayHello(name, age) {
```

```
    alert( name + " is " + age + " years old.");
```

```
    return name.toUpperCase();
```

```
}
```

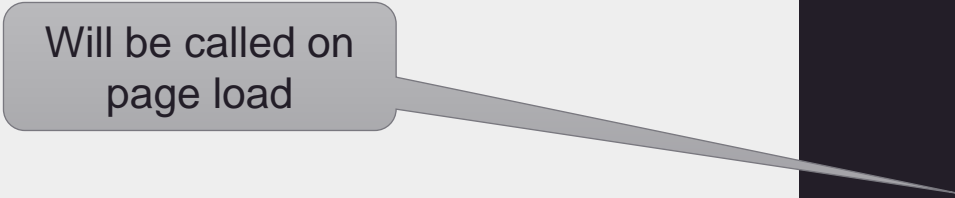
```
</script>
```

## Function Declaration continued..

- Function declarations get hoisted
- These can be executed before and after their definitions without any error
- In case of duplicates, last version will be considered by the browser
- Let us check some common usage of function :-

## Use case 1 : call functions on page load

Will be called on  
page load



```
<script type="text/javascript">  
    function sayHello(name, age) {  
        alert( name + " is " + age + " years old.");  
        return name.toUpperCase();  
    }  
  
    window.onload = sayHello("Vishal",35);  
</script>
```

## Use case 2 : call function from a function

Invoking a function

```
<script type="text/javascript">  
    function sayHello(name, age)  
    {  
  
        alert( name + " is " + age + " years old.");  
        return name.toUpperCase();  
  
    }  
  
    function display(){  
  
        var name = prompt("Enter your name");  
        sayHello("Vishal",35);  
  
    }  
</script>
```



## Use case 3 : calling a function on an event

- We can also call a function when user performs some actions
  - click of a button or types some key

**Register the handler  
display() to be called on  
button click**

```
<body>
```

```
<input type="button" size id="btn1" text="click me"  
onclick="display()"/>
```

```
</body>
```

# Function Arguments

- JavaScript maintains two objects inside functions :
  - arguments
  - this
- “arguments” is array like object that contains
  - all the arguments passed to a function
  - property called “callee” through which host function can be referenced
- ‘this’ represents current object, refer object constructors for more details

# Function Parameters and Arguments

- While calling a function, function arguments are not mandatory
- While writing a function, function parameters are not mandatory
- No error if a function is invoked
  - without passing any argument
  - with less number of arguments than the function parameters
  - with more number of arguments than the function parameters

## Function Expression

- Function expressions are not hoisted
- These can be executed only after their definitions

```
var display = function(){  
    console.log('welcome to persistent');  
};  
  
display();
```

# Anonymous Functions

- Anonymous functions :
  - functions without names
  - not available before its definition
  - if executed immediately, called as '**Immediately Invoked Function Expression**'

```
function(){  
    console.log('welcome to persistent');  
})();
```

## Some more features of JavaScript functions

- Functions act as a value like :
  - get assigned to variables which can invoke it
  - stored in objects or arrays
  - passed to or returned from other functions
- Only functions introduce scope :
  - function defined outside any function becomes global
  - function defined inside any function becomes local to that function boundary
  - if variables are declared using 'var' keyword , above rule gets applied
  - if variables are declared without using 'var' keyword then becomes global variables irrespective of location

## Identify scopes !!

Scope of 'a' is local to function 'findScope'

```
function findScope(){
```

```
  var a = 10; // ??
```

Scope of 'b' is global (irrespective of the location) as it is declared without using 'var' keyword

```
  b = 20; // ??
```

```
}
```

Scope of 'c' is global because it is declared outside the outermost function (here, only 1 function)

```
var c = 30; // ??
```

Scope of 'd' is global as declared without using 'var' keyword

```
d = 40; // ??
```

## Self-Invoking Functions

- Function expressions can be made "self-invoking".
- A self-invoking expression is invoked (started) automatically, without being called.
- Function expressions will execute automatically if the expression is followed by ().
- You cannot self-invoke a function declaration.
- You have to add parentheses around the function to indicate that it is a function expression

- `(function () {`
- `let x = "Hello!!"; // I will invoke myself`
- `})();`



## Closure

- [https://www.w3schools.com/js/js\\_function\\_closures.asp](https://www.w3schools.com/js/js_function_closures.asp)
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures>
- <https://www.freecodecamp.org/news/lets-learn-javascript-closures-66feb44f6a44/>
- <https://www.programiz.com/javascript/closure>

## What Are Arrow Functions ? (ES6 Feature)

- Arrow functions are a concise syntax for writing function expressions.
- They use a fat arrow syntax i.e. `=>` for writing functions.
- They are anonymous functions that simplify function scoping and the `this` keyword.
- These functions cannot be used as constructors so they will throw an error when used with `new`.
- The `prototype` property on arrow function does not exist.
- They are one-line mini functions , where we avoid having `function` keyword and `return` keyword because its implicit in arrow function.

## What Are Arrow Functions ?...

- Basic Syntax for a Function to calculate square of a number.

```
var calsq=(function(num){  
    return num*num;  
});
```

```
console.log(calsq(9));           //81
```

## What Are Arrow Functions ?...

- Using arrow function, the previous code can be rewritten as:.

```
let calsq= (num)=>{  
    return num*num;  
};  
  
console.log(calsq(6));           //36
```

## Expression Bodies

- Expression bodies-are a single line expression.
- They use => token and an implied return value.

```
let evenNos= [2,6,10];
```

```
let oddNos= evenNos.map( x => x+1);
```

```
console.log(oddNos);
```

```
//3,7,11
```

## Statement Bodies

- Statement Bodies are multiline statements
- Used for more complex logic.
- To have a return value is also possible from the statement in case of statement bodies of arrow functions.

```
let nines=[ ];  
let noArr=[1,2,9,18,36];  
  
noArr.forEach(v=>{  
    if(v%9===0)  
        nines.push(v);  
});  
console.log(nines);           //9,18,36
```

## Lexical this

- With arrow syntax, we can safely access the lexical this without worrying about its getting changed in the function assign for forEach. Previously, we would have had to create a separate “**that**” closure variable to enable accessing the correct **this**.

```
let obj={  
  name:"Sachin",  
  technologies:["JS","ANGULAR","NODE"],  
  printTech(){  
    this.tech.forEach( technology=>  
      console.log(this.name + " knows " +  
        technology));  
  }  
}  
obj.printTech();
```

## Default Parameter Values (Using ES6)

- It makes easier to handle function parameters.
- In JS, any number of parameters can be passed to a function.
- ES6 enables us to set a default parameter to a function.
- Parameters with default values are considered to be optional.

```
function add(x, y = 4, z){  
    return x + y + z;  
}  
  
console.log(add(1,6,2));           // 8-> y===4  
console.log(add(1, undefined,2)); // 5->y as default
```



## Default Parameter Values...

- Default parameters can also be set by executing function. Its not restricted to only primitive values

```
function getDefaultInc() {  
    return 1;  
}
```

```
function newInc(number, increment = getDefaultInc()) {  
    return number + increment;  
}
```

```
console.log(newInc(4, 4));           // 8
```

```
console.log(newInc(3));              // 4
```

## Rest Parameters (using ES6)

- JavaScript allows functions to call with more parameters than the function declares.
- If rest operator(...) is used in front of the last formal parameter, it specifies that it will receive all remaining actual parameters in an Array.
- It must always be the last parameter.
- If there are no remaining parameters, an empty Array is set to rest parameter.

```
function f(s, ...t){  
    ....                //function body  
}  
f('a', 'b', 'c');  
s='a'; t=['b', 'c']
```

## Spread Operator (using ES6)

- Spread operator looks exactly like the rest operator i.e.(...), but it works exactly opposite to rest operator.
- Spread operator convert the items of an iterable into arguments of a function call or into elements of an Array.
- Unlike rest operator, spread operator can be used anywhere in a sequence of parts.

```
Math.max(-1, 5, 11,3)           // 11
```

```
Math.max(-1, ...[-1, 5, 11], 3) // 11
```

## Spread Operator...

- Spread operator also works for constructor calls in addition to function and method calls
- It can also be used inside Array Literals. E.g

// Spread operator for Constructor

```
new Date(...[2007, 11, 24])    //Christmas Eve 2007
```

// Spread operator for Array Literals

```
[1, ...[2,3],4]                // [1,2,3,4]
```

## Summary : Session #

With this we have come to an end of our session, where we discussed :

- Importance of function, their characteristics and features
- Different ways to implement a function like function declaration, function expression & anonymous functions
- Local & global scope

-

At the end of this session, we expect you to :

- Understand functions along with all discussed concepts
- Implement all variants of functions as per requirement
- Understand scope of var with let keyword
- Implement Arrow functions
- Understand error handling inJavaScript

# Appendix

A decorative graphic consisting of a horizontal orange line that extends from the left edge of the slide. This line meets a vertical orange line that extends downwards to the bottom edge. At the top of the vertical line, there is a large, thin orange circle that overlaps with the horizontal line.

- References
- Key Contacts

## Reference Material : Books

### **Head First JavaScript Programming**

- By: Eric T. Freeman;  
Elisabeth Robson
- Publisher: O'Reilly Media, Inc.

### **Professional: JavaScript® for Web Developers**

- By: Nicholas C. Zakas
- Publisher: Wrox

# Persistent University

Tarun Kr. Joshi

[tarun\\_joshi@persistent.co.in](mailto:tarun_joshi@persistent.co.in)

Priya Singh

[priya\\_singh@persistent.co.in](mailto:priya_singh@persistent.co.in)

Shubhangi Kelkar

[shubhangi\\_kelkar@persistent.co.in](mailto:shubhangi_kelkar@persistent.co.in)





# Thank You !!!

Persistent Interactive | Persistent University

