WEB TECHNOLOGIES

JavaScript - Functions

Vinay Joshi
Department of
Computer Science and Engineering

JavaScript - Functions

- A function is a subprogram designed to perform a particular task. Functions are executed when they are called. This is known as invoking a function.
- Functions always return a value. In JavaScript, if no return value is specified, the function will return undefined.
- Whenever you have a relatively complex piece of code that is likely to be reused, you have a candidate for a function.
- A **Function Declaration** defines a named function. To create a function declaration you use the function keyword followed by the name of the function. When using function declarations, the function definition is hoisted, thus allowing the function to be used before it is defined.

```
function name(parameters){
    statements
    }
```

JavaScript - Functions

- A **Function Expression** defines a named or anonymous function. An anonymous function is a function that has no name. Function Expressions are not hoisted, and therefore cannot be used before they are defined.
- In the example below, setting the anonymous function object equal to a variable.

```
let name = function(parameters){
    statements
    }
```

- An **Arrow Function Expression** is a shorter syntax for writing function expressions.

```
let name = (parameters) => {
    statements
    }
```

JavaScript - Functions

What are Functions?

- Whenever you have a relatively complex piece of code that is likely to be reused, you have a candidate for a function.
- The general syntax for a function is:

```
function function_name([parameter [, ...]])
{
    statements
    //optional return statement
}
```

- The general syntax for calling a function is:

```
[retval =] function_name([argument [,…]])
```

JavaScript - Functions

Parameters and Arguments

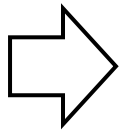
- Parameters are used when defining a function, they are the names created in the function definition. Parameters are separated by commas in the ().
- Arguments, on the other hand, are the values the function receives from each parameter when the function is executed (invoked).
- Argument list and parameter list mismatch does not give errors.
- Parameter that is not passed a value in arguments list is treated as **undefined**
- To access additional arguments, use the **arguments** array to access the values passed.

JavaScript - Functions

Hoisting – Variables and Functions

- Hoisting is JavaScript's default behavior of moving all variable and function declarations to the top of the current scope (to the top of the current <script> or the current function).
- Only declarations are hoisting not initializations

```
num = 4;
console.log(num);
var num = 9;
console.log(num);
```



```
var num;
num = 4;
console.log(num);
num = 9;
console.log(num);
```

JavaScript - Functions

Hoisting – Variables and Functions

- In JavaScript, a variable can be declared after it has been used.
- In other words; a variable can be used before it has been declared.

```
Example1
x = 5; // Assign 5 to x
document.write("Value of x "+x);  // Display x
var x; // Declare x*/

Example2
var x; // Declare x
 x = 5; // Assign 5 to
 document.write("Value of x "+x);
```

Both example 1 and example are same

JavaScript - Functions

Hoisting

- If we attempt to use a variable before it has been declared and initialized, it will return undefined.

Example:

```
console.log(x);
var x = 100;
```

```
// How JavaScript interpreted it
Var x;
Console.log(x);
x=100;
```

Output:

undefined

- However, if we omit the var keyword, we are no longer declaring the variable, only initializing it. It will return a ReferenceError and halt the execution of the script.

```
console.log(x);
x = 100;
```

Output:

ReferenceError: x is not defined

The reason for this is due to hoisting. Since only the actual declaration is hoisted, not the initialization, the value in the first example returns undefined.

JavaScript - Functions

Hoisting (continued)

- Function declarations are fully hoisted. This means you can call a function before its declaration in the code, and it will work as expected.

```
sayHello();
         function sayHello() {
         console.log("Hello!");
        }
```

- Output: Hello!

- Function declarations are fully hoisted, while variable declarations are hoisted with an initial value of **`undefined`**

JavaScript - Functions

Hoisting (continued)

- Hoisting can lead to unpredictable results.

Example:

```
var a = 10;
function hoist() {
    if (false) {
    var a = 20;
    }
console.log(a);
}
hoist();
```

Output:

undefined

- In this example, we declared a to be 10 globally. Depending on an if statement, a could change to 20, but since the condition was false it should not have affected the value of a. Instead, a was hoisted to the top of the hoist() function, and the value became undefined.

JavaScript - Functions

Hoisting (continued)

- Variables and constants declared with let or const are not hoisted as they are block-scoped.
- Duplicate declaration of variables, which is possible with var, will throw an error with let and const.
- // Attempt to overwrite a variable declared with var

```
var a = 3;
var a = 4;
console.log(a);       Output: 4
```

- // Attempt to overwrite a variable declared with let

```
let y = 1;
let y = 2;
console.log(y);      Output: Uncaught SyntaxError: Identifier 'y' has already been declared.
```

THANK YOU

Vinay Joshi
Department of Computer Science and Engineering

vinayj@pes.edu

+91 80 2672 6622