Day 4

Yogesh Yadav

Functions in JavaScript

- **A function is a block of reusable code.** A function allows us to write code once and use it as many times as we want just by calling the function.

- **JavaScript function syntax**

```
function functionName(parameter1, parameter2,..parameter_n)
{
  //function statements
}
```

  Points to remember
  1. Use the function keyword to define a function, followed by the name of the function. The name of the function should be followed by parentheses ().
  2. Function parameters are optional. The parameter names must be with in parentheses separated by commas.

- **Example :** JavaScript function to add 2 numbers. The following JavaScript function adds 2 numbers and return the sum

  ```
  function addNumbers(firstNumber, secondNumber)
  {
      var result = firstNumber + secondNumber;
      return result;
  }
  ```

  Calling the JavaScript function : Call the JavaScript function by specifying the name of the function and values for the parameters if any.

  ```
  var sum = addNumbers(10, 20);
  alert(sum);
  ```

  Output : 30

- The following function does not return any value. It simply writes the sum of two numbers to the page. However, we are assigning the return value of addNumbers() function to sum variable. Since addNumbers() function in this case does not have an explicit return statement, undefined will be returned.

```
function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    document.write(result);
}

var sum = addNumbers(10, 20);
alert(sum);
```

What happens when you do not specify values for the function parameters when calling the function
The parameters that are missing values are set to undefined

- **What happens when you specify too many parameter values when calling the function.**
  The extra parameter values are ignored.

In the example below, 30 & 40 are ignored.

```
function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}

var sum = addNumbers(10, 20, 30, 40);
alert(sum);
```

Should a javascript function always return a value
No, they don't have to. It totally depends on what you want the function to do. If an explicit return is omitted, undefined is returned automatically.

Defining a function using function declaration

- **Example 1 :** Declaring a function first and then calling it.

```
function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}

var sum = addNumbers(10, 20);
document.write(sum);
```

Output : 30

- **Example 2 :** A function call is present before the function declaration

  In Example 1, we are first defining the function and then calling it. The call to a JavaScript function can be present anywhere, even before the function is declared. The following code also works just fine. In the example below, we are calling the function before it is declared.

```
var sum = addNumbers(10, 20);
document.write(sum);

function addNumbers(firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

  Function Hoisting : By default, JavaScript moves all the function declarations to the top of the current scope. This is called function hoisting. This is the reason JavaScript functions can be called before they are declared.

- **Defining a JavaScript function using a function expression :** A Function Expression allows us to define a function using an expression (typically by assigning it to a variable). There are 3 different ways of defining a function using a function expression.

Anonymous function expression example : we are creating a function without a name and assigning it to variable add. We use the name of the variable to invoke the function.

```
var add = function (firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}

var sum = add(10, 20);
document.write(sum);
```

Functions defined using a function expression are not hoisted. So, this means a function defined using a function expression can only be called after it has been defined while a function defined using standard function declaration can be called both before and after it is defined.

```
// add() is undefined at this stage
var sum = add(10, 20);
document.write(sum);

var add = function (firstNumber, secondNumber)
{
    var result = firstNumber + secondNumber;
    return result;
}
```

Named function expression example : The difference is instead of assigning the variable to an anonymous function, we’re assigning it to a named function (computeFactorial).

```
var factorial = function computeFactorial(number)
{
    if (number <= 1)
    {
        return 1;
    }

    return number * computeFactorial(number - 1);
}

var result = factorial(5);
document.write(result);
```

The name of the function (i.e computeFactorial) is available only with in the same function. This syntax is useful for creating recursive functions. If you use computeFactorial() method outside of the function it raises 'computeFactorial' is undefined error.

```
var factorial = function computeFactorial(number)
{
    if (number <= 1)
    {
        return 1;
    }

    return number * computeFactorial(number - 1);
}

var result = computeFactorial(5);
document.write(result);
```

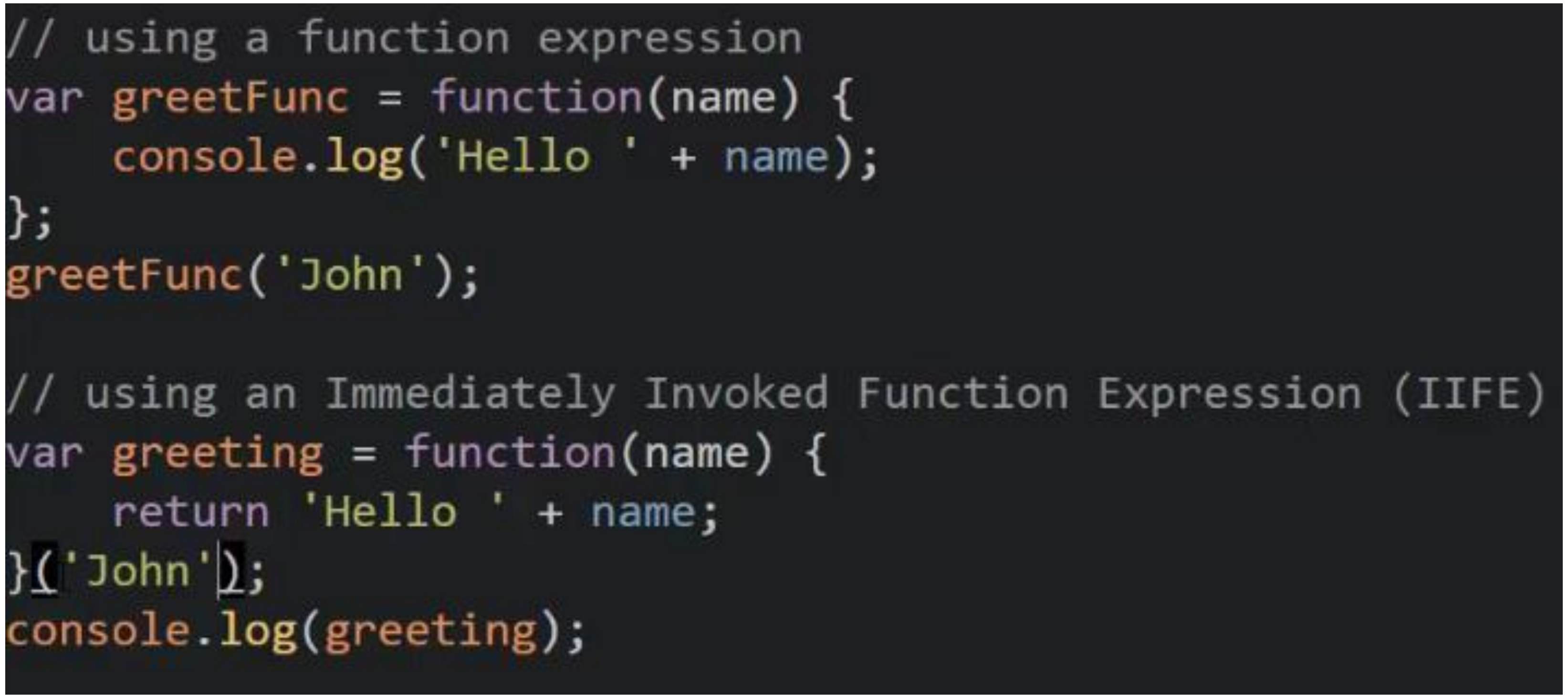
Output : Error - 'computeFactorial' is undefined.

Self invoking function expression example :

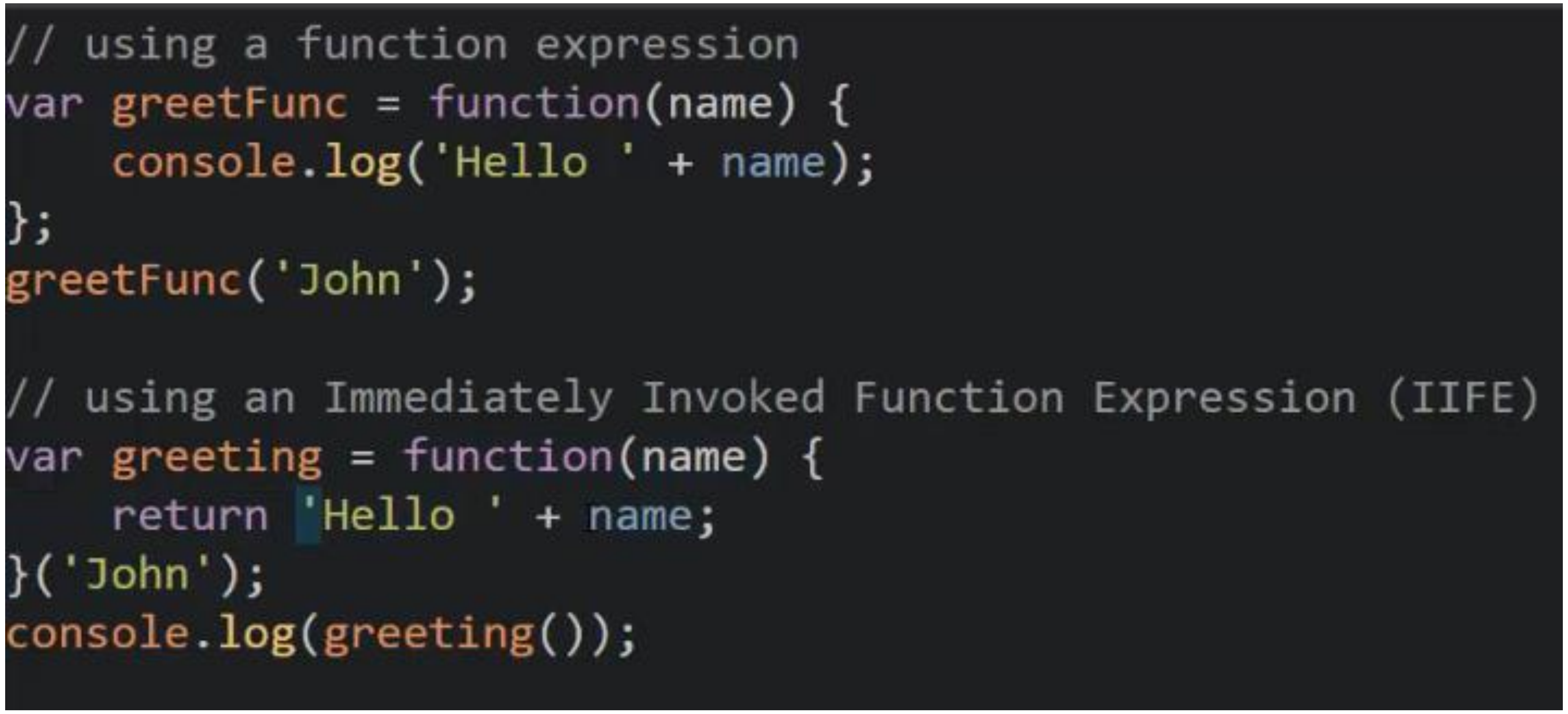
```
var result = (function computeFactorial(number)
{
    if (number <= 1)
    {
        return 1;
    }

    return number * computeFactorial(number - 1);
})(5);

document.write(result);
```

Output : 120

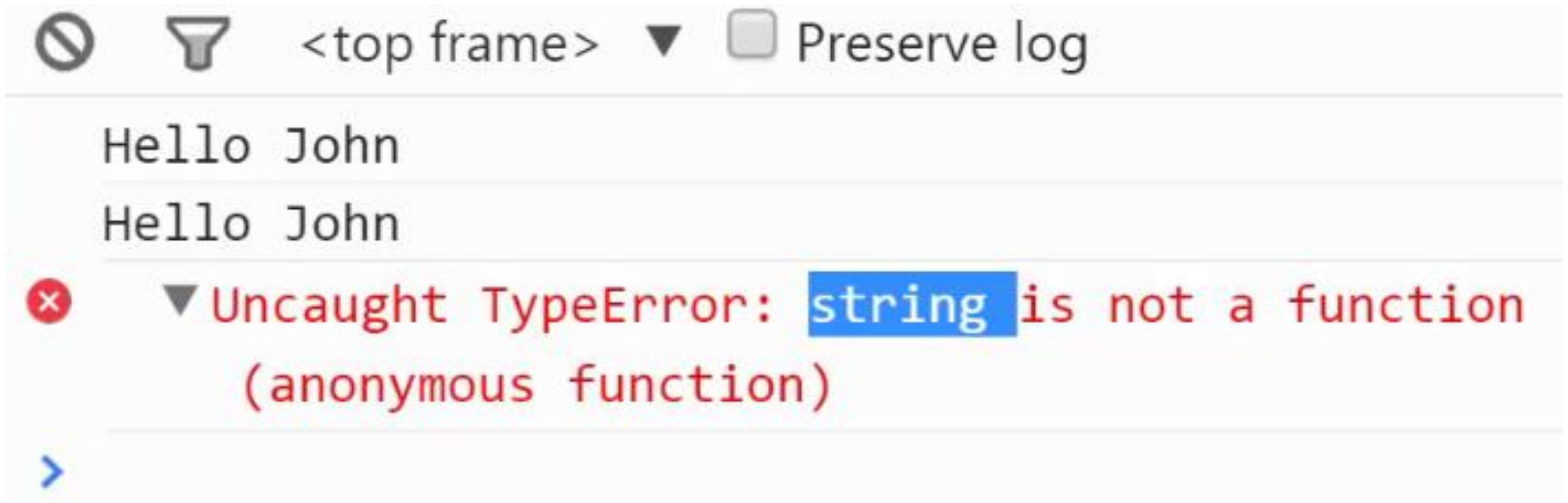
These are called with different names
Immediately-Invoked Function Expression (IIFE)
Self-executing anonymous functions
Self-invoked anonymous functions



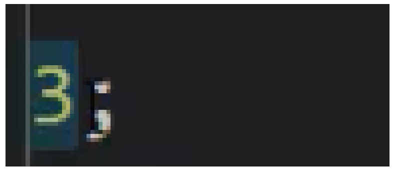
```
// using a function expression
var greetFunc = function(name) {
    console.log('Hello ' + name);
};
greetFunc('John');

// using an Immediately Invoked Function Expression (IIFE)
var greeting = function(name) {
    return 'Hello ' + name;
}('John');
console.log(greeting);
```



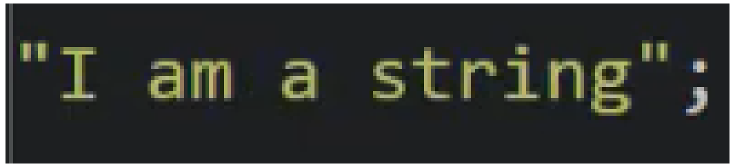
```
// using a function expression
var greetFunc = function(name) {
    console.log('Hello ' + name);
};
greetFunc('John');

// using an Immediately Invoked Function Expression (IIFE)
var greeting = function(name) {
    return 'Hello ' + name;
}('John');
console.log(greeting());
```



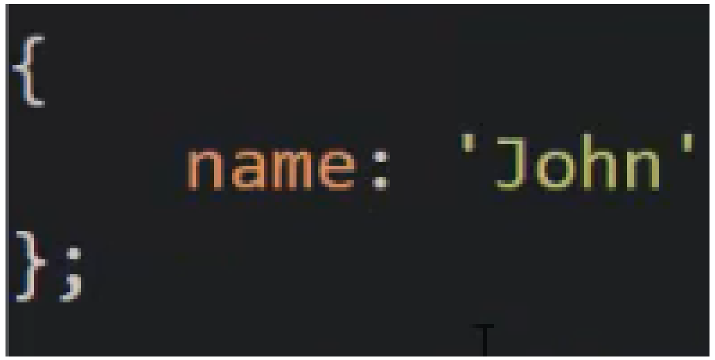
```
Hello John
Hello John
▼Uncaught TypeError: string is not a function
   (anonymous function)
>
```



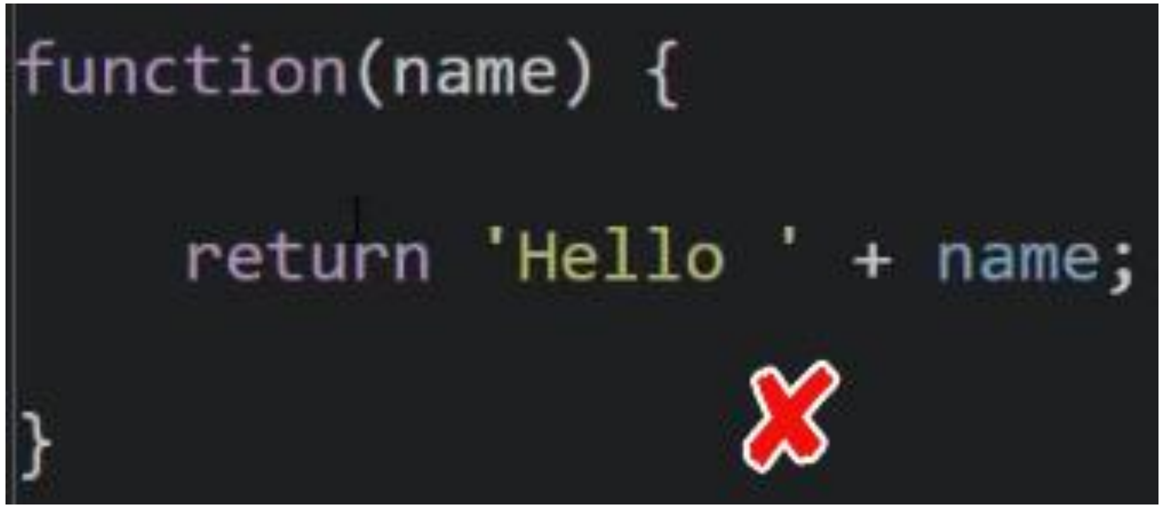
```
3;
```



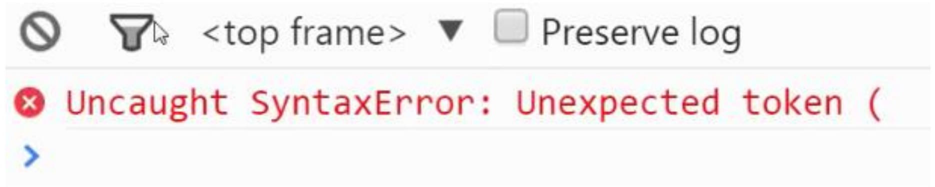
```
"I am a string";
```



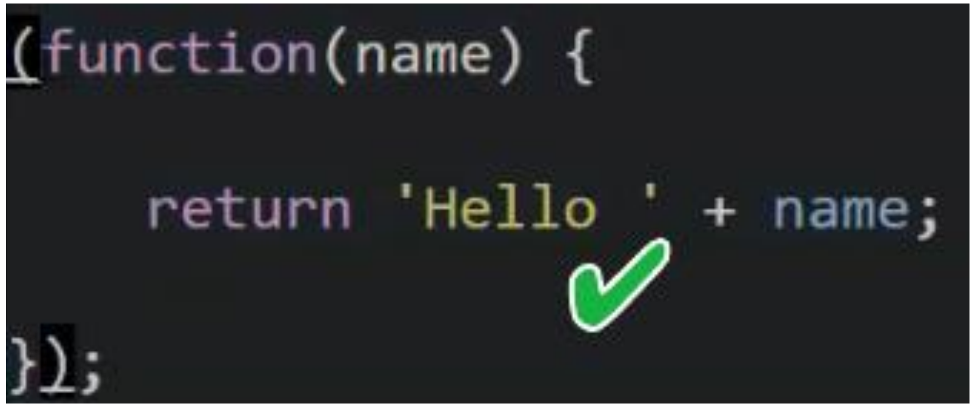
```
{
    name: 'John'
};
```



```
function(name) {

    return 'Hello ' + name;

}
```



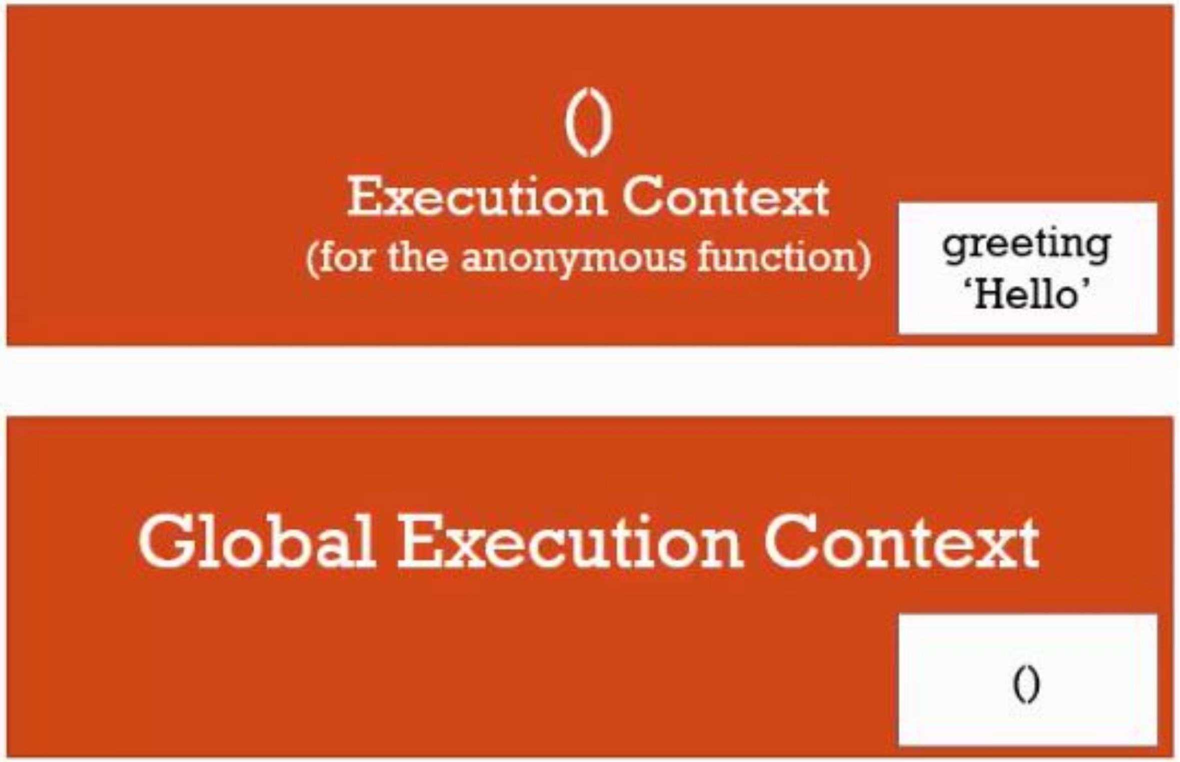
```
Uncaught SyntaxError: Unexpected token (
>
```



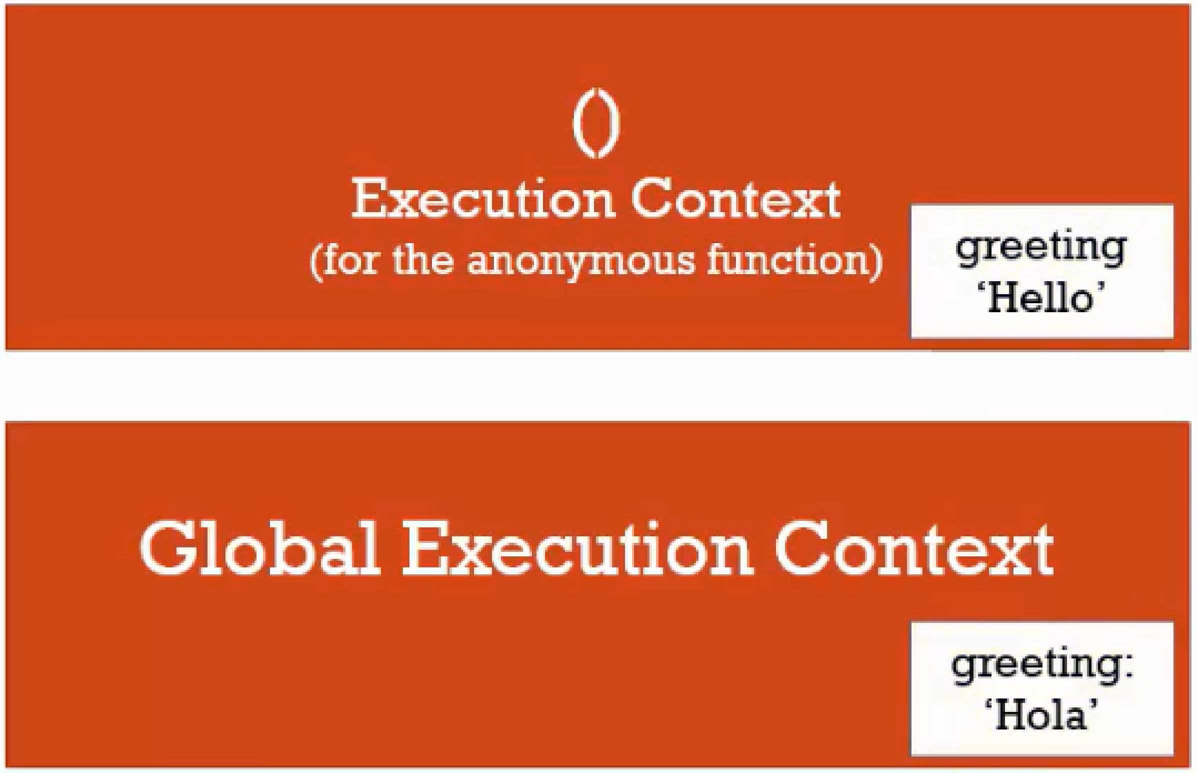
```
(function(name) {

    return 'Hello ' + name;

});
```

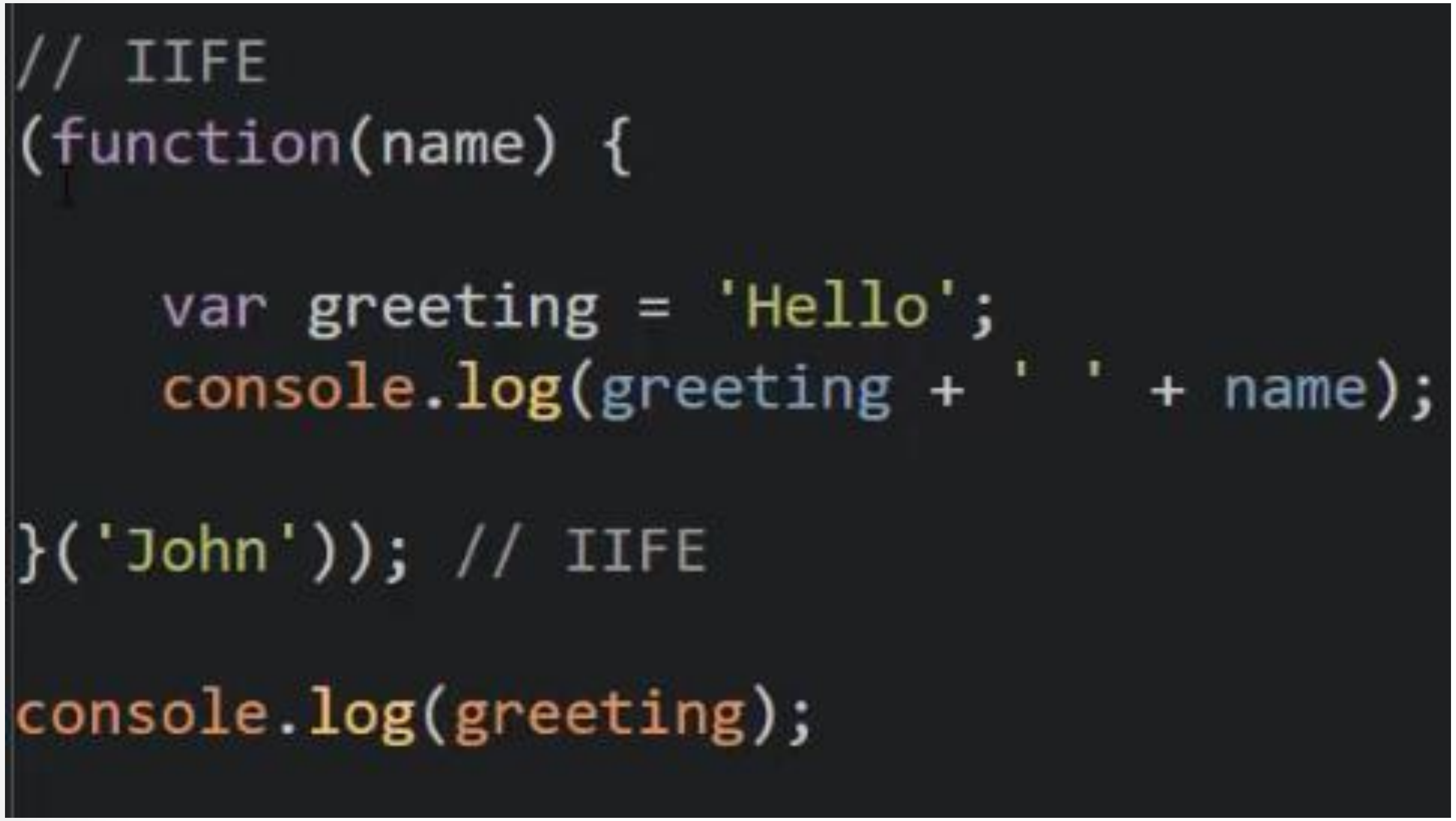
() use for expression

```
(function(name) {

    var greeting = 'Hello';
    console.log(greeting + '
' + name);

}('John'));
```



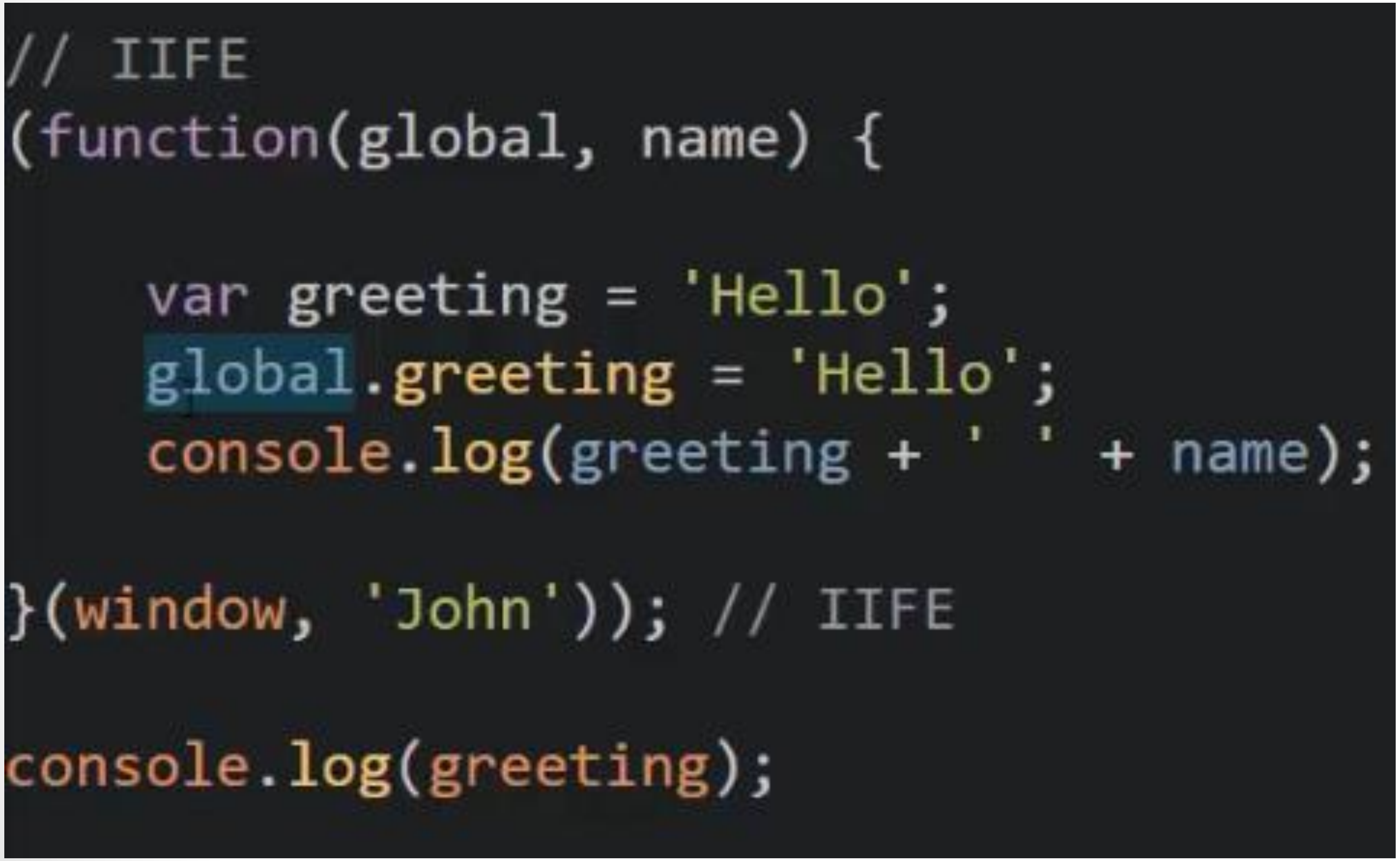
```
var greeting = 'Hola';

(function(name) {

    var greeting = 'Hello';
    console.log(greeting + '
' + name);

}('John'));
```





```
// IIFE
(function(name) {

    var greeting = 'Hello';
    console.log(greeting + ' ' + name);

}('John')); // IIFE

console.log(greeting);
```

Safe Code expose global object like jquery



```
// IIFE
(function(global, name) {

    var greeting = 'Hello';
    global.greeting = 'Hello';
    console.log(greeting + ' ' + name);

}(window, 'John')); // IIFE

console.log(greeting);
```