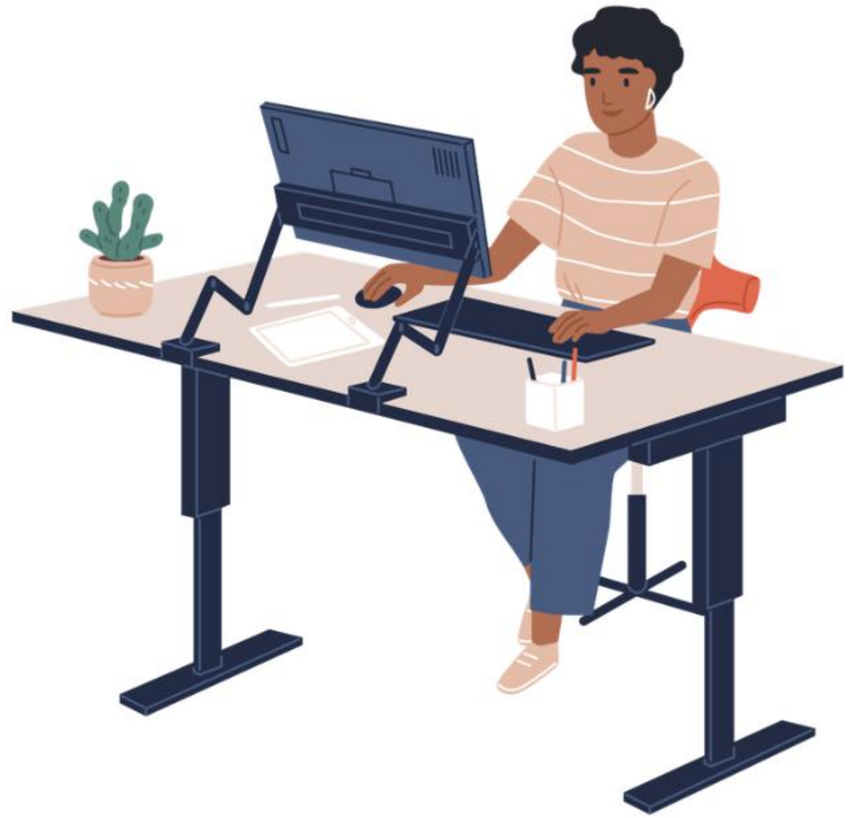
Slide Note

Learning Consolidation

Implement Modular Programming Using Functions



Slide Note

In this Sprint, you learned to:

- Define function as an independent unit to perform tasks
- Explain scope of a variable
- Differentiate between block scope, local scope and global scope
- Use default parameters in functions
- Explain predefined/built-in functions

Slide Note

Functions

- Modular programming is dividing the program into small and independent modules which have some specific functionality, instead of writing a program as one large block of code.
- A function is a block of organized, reusable code that is used to perform a specific action.
- Actions can be simple tasks like:
  - finding the sum of two numbers
  - calculating the maximum score
  - displaying a set of values to the user
  - validating data

Slide Note

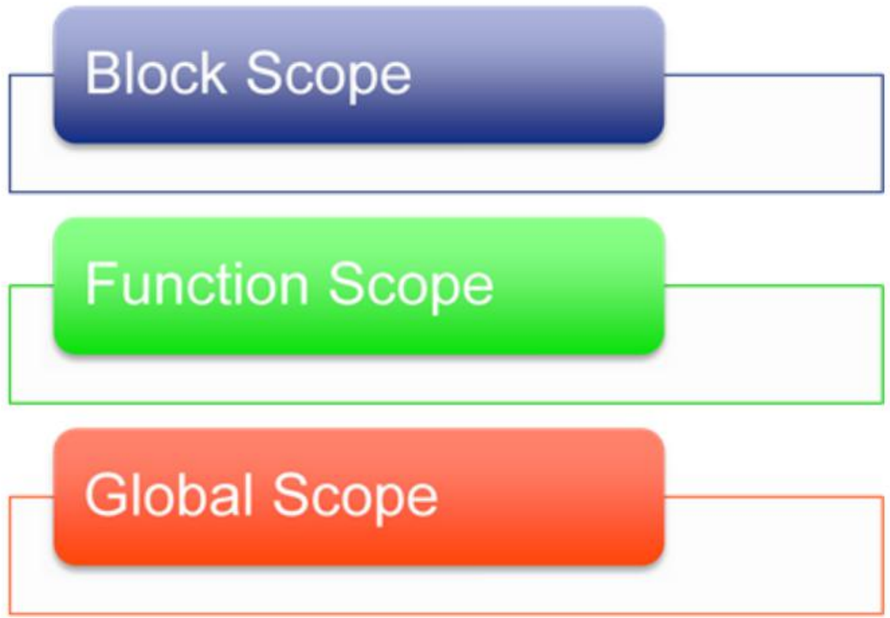
Function Definition

- A function definition consists of the **function** keyword followed by:
    - The name of the function
    - The Parameter list (Optional)
    - Function body which has the JavaScript statements enclosed in curly brackets, {...}
- Function parameters are just values you supply to the function so that the function can do something utilizing those values.
- The function body can optionally return some values to the calling script or function. To return some value, the function must have a return statement that specifies the value to return.
- A function must be called for to execute the tasks written inside the function body.
- Calling the function performs the specified actions with the passed parameters.
- The **parameters** are the aliases for the values that will be passed to the function. The **arguments** are the actual values.

Slide Note

JavaScript Scope

- Scope determines the accessibility or visibility of variables.
- JavaScript provides 3 types of scope:



Slide Note

Variable Scopes

- Block scope: Variables declared inside {} block cannot be accessed outside the block.
- Function scope: Local variables have function scope, and they can only be accessed within the function. A local variable lives until the function is under execution. The moment a function completes, the local variable gets deleted.
- Global scope: Global variables can be accessed from anywhere in a JavaScript program. A Global variable lives until the user closes the program or until the web browser is closed.

Slide Note

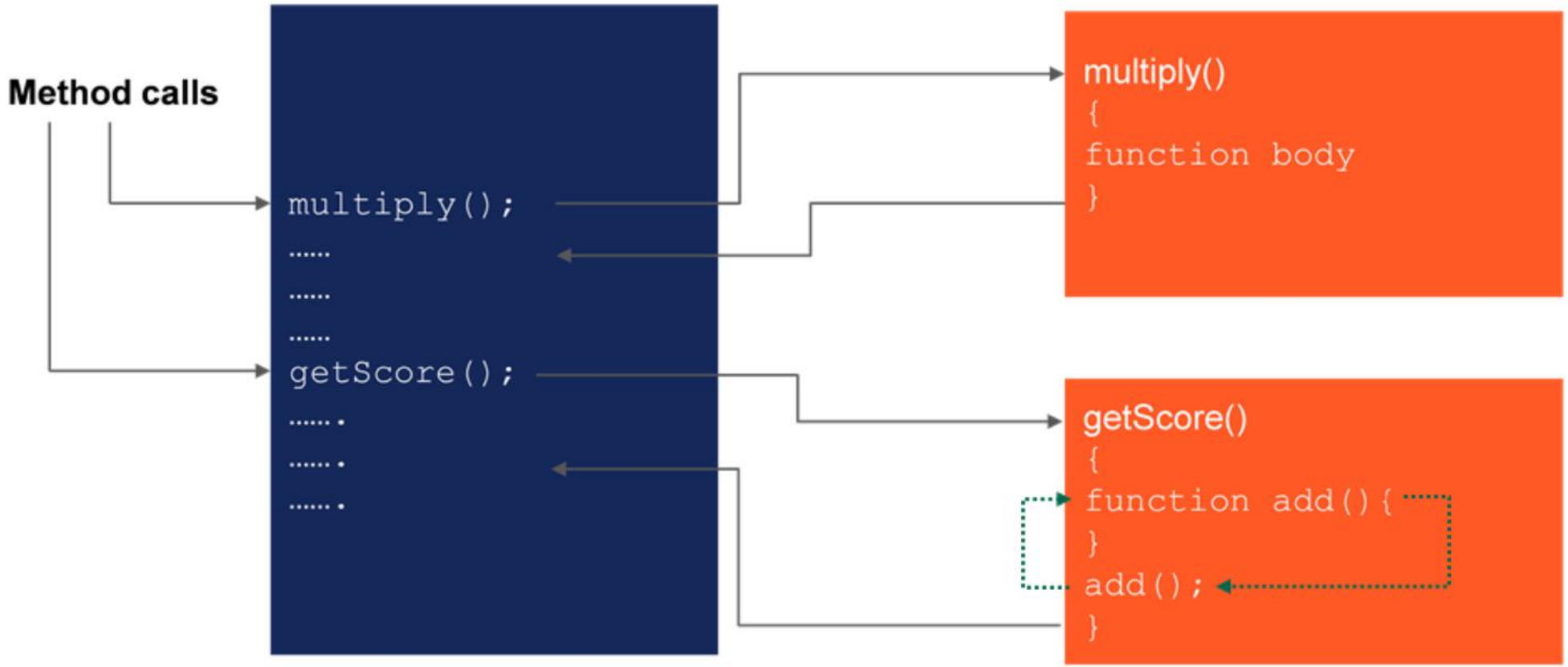
```
// variables are in the global scope
var num1 = 30, num2 = 5, name = 'joe';

//  function is in the global scope
function multiply() {
   return num1 * num2;
}

// Nested function example
function getScore() {
   var num1 = 30, num2 = 50;
   function add(){
   return name +'scored ' + (num1+num2);
   }
   return add();
 }

getScore(); // Returns "Joe scored 80"
```

Nested Functions

- You may nest a function within another function.
- The nested (inner) function is private to its containing (outer) function.
- The inner function can be accessed only from statements in the outer function.

Slide Note

Function Call



Menu

00:08:00:00:18 08/ 18

Slide Note

Recursive Functions

- Functions that call itself are called recursive functions.
- Below is the function which computes factorials recursively.

```
function factorial(number) {
if ((number === 0) || (number === 1))
return 1;
else
return (number * factorial(number - 1));
}
//call the function
var result = factorial(5); //result is 120
```

Slide Note

JavaScript Hoisting

- Hoisting is JavaScript's default behavior of moving declarations of variables and functions to the top, before execution of the code.
- Hoisting allows functions to be safely used in code before they are declared.
- One of the advantages of hoisting is that it lets you use a function before you declare it in your code.

```
//call the function before it is defined
display('John'); //Name is John

//Function declaration
function display(name) {
console.log(`Name is ${name}`);
}
```

Slide Note

Built-in Functions

- Following are the global functions which are called globally.

Function Name	Description
eval()	Evaluates JavaScript code represented as a string.
isFinite()	Determines whether the passed value is a finite number.
isNaN()	Determines whether a value is NaN (Not a Number) or not.
parseFloat()	Parses an argument (converting it to a string first if needed) and returns a floating-point number
parseInt()	parses a string argument and returns an integer.

- JavaScript provides built-in methods on various predefined objects like Window, Number, String, Math, Date, etc.
- Note: JavaScript Objects will be covered in the following sprint.

Slide Note

Self-Check

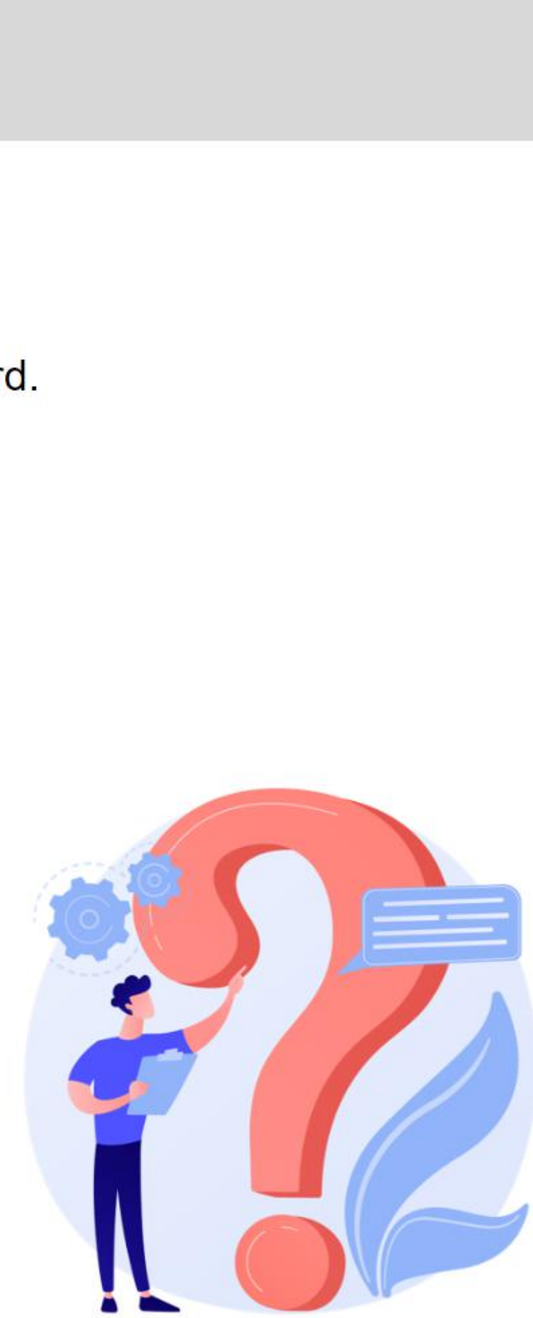
The most common way to define a function in JavaScript is by using _____ keyword.

A. fun

B. define

C. function

D. var



Slide Note

Self-Check: Solution

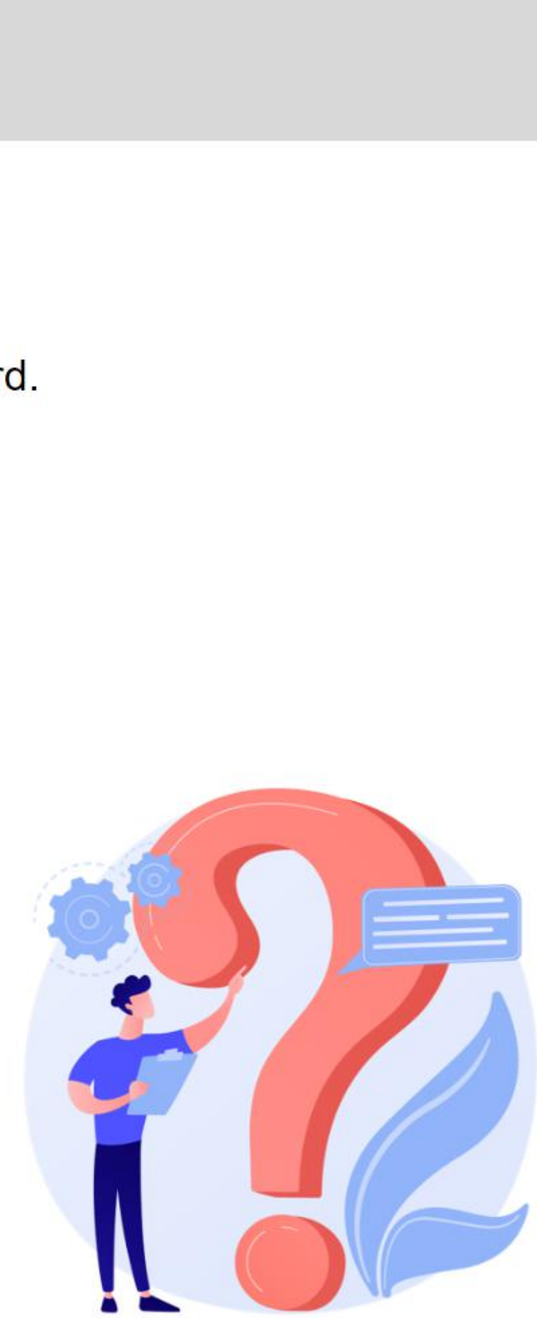
The most common way to define a function in JavaScript is by using _____ keyword.

A. fun

B. define

C. **function**

D. var



Slide Note

Self-Check

What will be the output of the code?

```
function compute(a = 30,b = 4) {
    return a * b;
}
console.log(compute(5));
```

1. 12
2. 120
3. 20
4. Error

Slide Note

Self-Check: Solution

What will be the output of the code?

```
function compute(a = 30,b = 4) {
    return a * b;
}
console.log(compute(5));
```

1. 12
2. 120
3. **20**
4. Error

Slide Note

Self-Check

What will be the output of the code?

```
let x = 10;
function compute(y) {
    let x = 20;
    return x + y ;
}
console.log(compute(5));
```

1. 15
2. 25
3. 5
4. Error

Slide Note

Self-Check: Solution

What will be the output of the code?

```
let x = 10;
function compute(y) {
    let x = 20;
    return x + y ;
}
console.log(compute(5));
```

1. 15
2. **25**
3. 5
4. Error