Module 01

Purwadhika
Digital Technology School

Full Stack Web Development

Array and Function

Job Connector Program

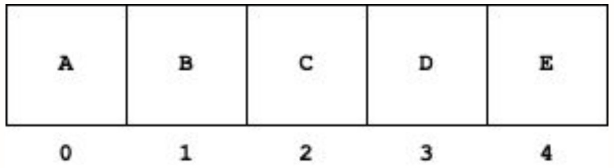
Outline

- Array
- Function

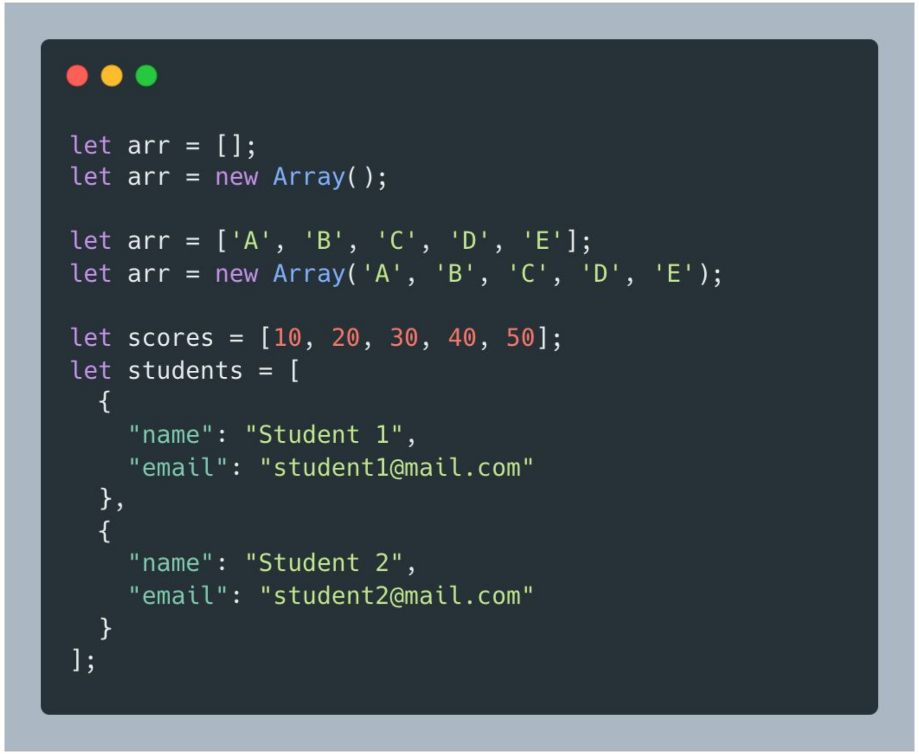
Array

An array is a **collection of similar data elements** stored at contiguous memory locations.

It is the simplest data structure where each data element can be accessed directly by only using its index number.



Array Declaration



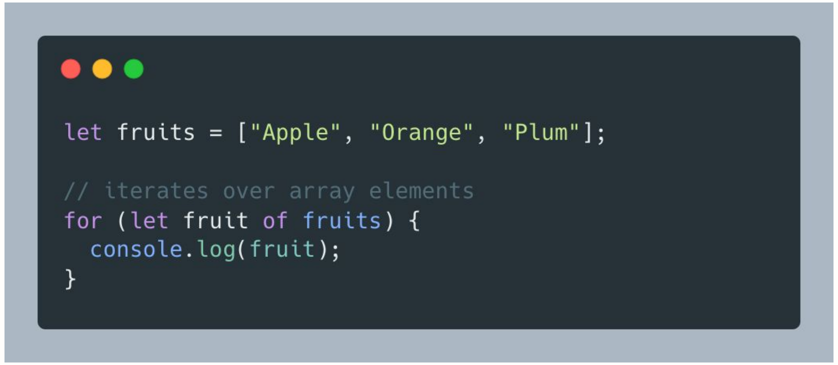
```
let arr = [];
let arr = new Array();

let arr = ['A', 'B', 'C', 'D', 'E'];
let arr = new Array('A', 'B', 'C', 'D', 'E');

let scores = [10, 20, 30, 40, 50];
let students = [
  {
    "name": "Student 1",
    "email": "student1@mail.com"
  },
  {
    "name": "Student 2",
    "email": "student2@mail.com"
  }
];
```

Array built-in methods

- toString
- join
- pop
- push
- shift
- unshift
- length
- concat
- splice
- slice
- indexOf
- lastIndexOf
- sort
- reverse
- forEach
- map
- filter
- find
- findIndex
- reduce
- reduceRight
- every
- some
- from
- keys
- entries
- includes

For … of loop

The for … of **doesn't give access to the number of the current element, just its value**, but in most cases that's enough. And it's shorter.



```
let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements
for (let fruit of fruits) {
  console.log(fruit);
}
```

For … in loop

Loops through the properties of an object. We will discuss about **for … in** loop in the next session.

Function

A **function** is a block of reusable code written to perform a specific task.

Generally speaking, a function is a **"subprogram"** that can be called in another code.

Like the program itself, a function is composed of a sequence of statements called the function body.

Values can be passed to a function, and the function will return a value.

Defining Function

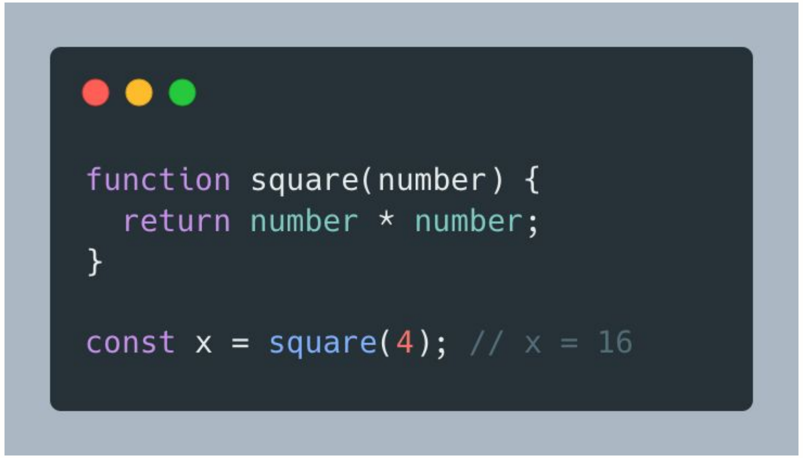
To define a **function** we have two ways:

- Function Declaration
- Function Expression

Function Declaration

Function declaration consists of the **function** keyword, followed by:

- The **name** of the function.
- A list of **parameters** to the function, enclosed in parentheses and separated by commas.
- The JavaScript statements that define the function, **enclosed in curly brackets** → {...}.

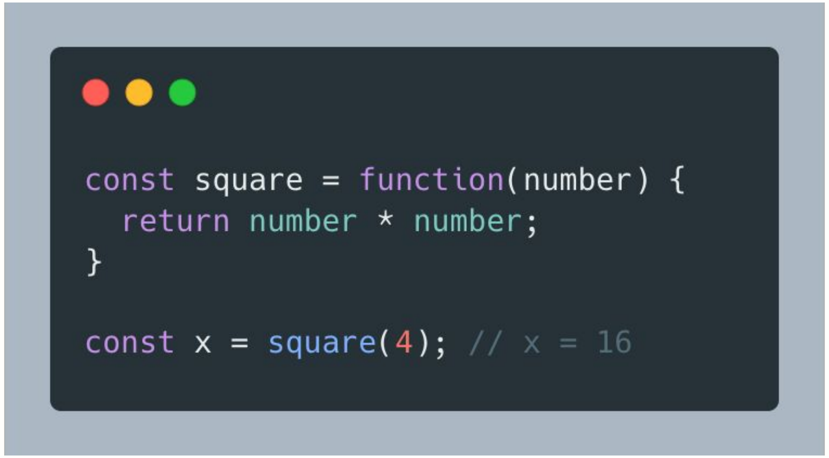


```javascript
function square(number) {
  return number * number;
}

const x = square(4); // x = 16
```

Function Expression

While the function declaration is syntactically a statement, functions can also be created by a **function expression**.

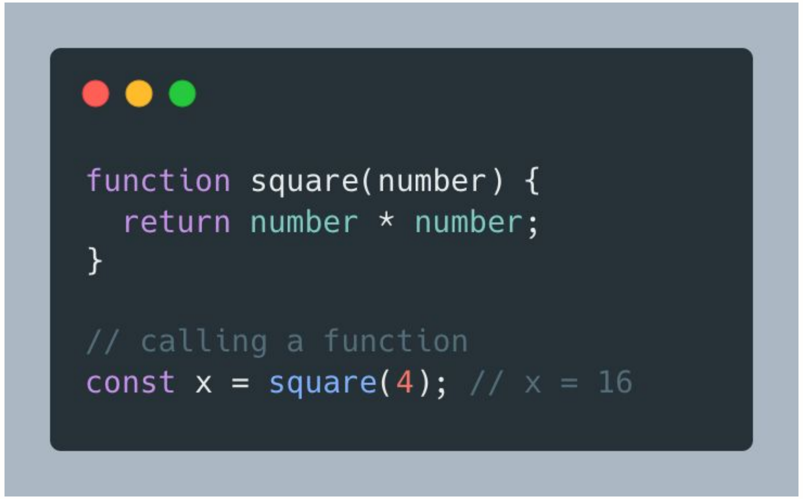
Such a function can be **anonymous**, it does not have to have a name. Then assign that anonymous function to a variable.



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

const x = square(4); // x = 16
```

Calling Function

Defining a function does not execute it. **If you want to call a function, just call the function's name and parentheses.**

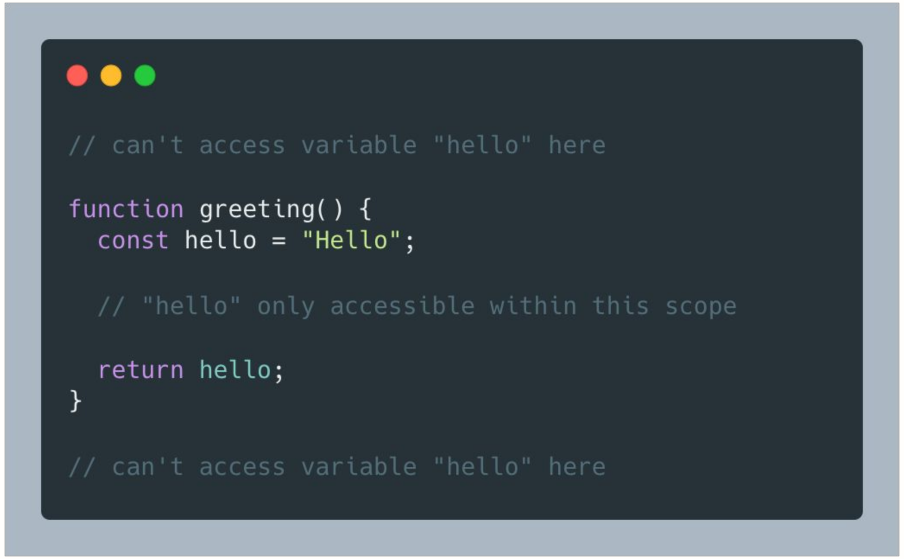


```
function square(number) {
  return number * number;
}

// calling a function
const x = square(4); // x = 16
```

Function Scope

Variables defined inside a function cannot be accessed from anywhere outside the function, because the variable is **defined only in the scope of the function**.

However, a function can access all variables and functions defined inside the scope.

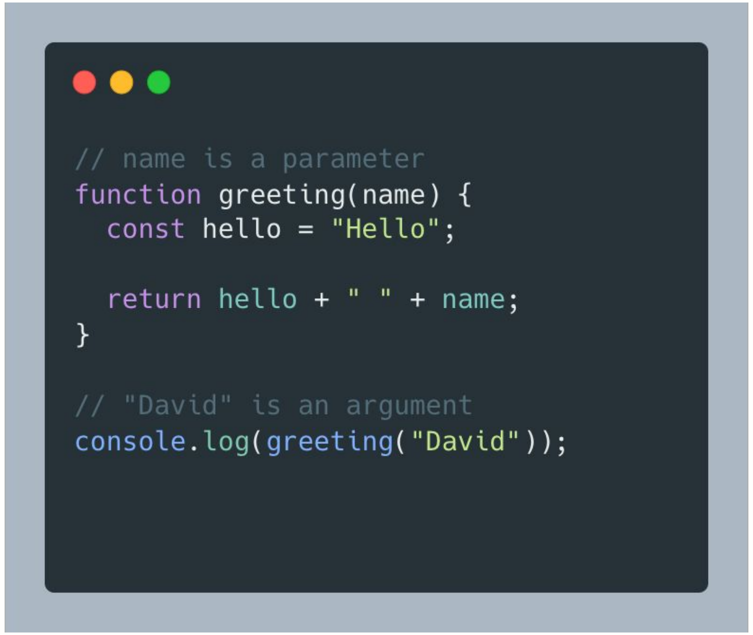


```
// can't access variable "hello" here

function greeting() {
  const hello = "Hello";

  // "hello" only accessible within this scope

  return hello;
}

// can't access variable "hello" here
```

Parameter & Argument

An **argument** is a value passed as input to a function. While a **parameter** is a named variable passed into a function.

Parameter variables are used to import arguments into functions.

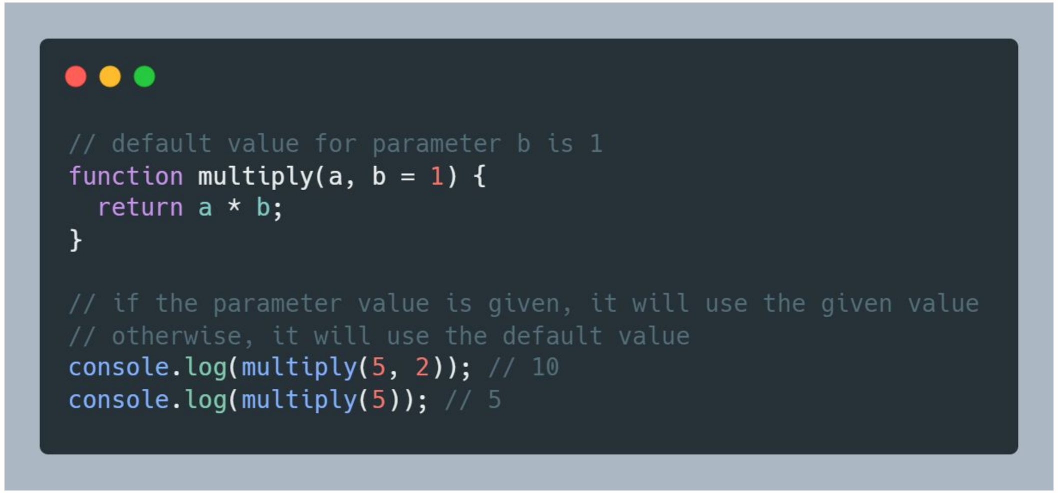


```javascript
// name is a parameter
function greeting(name) {
  const hello = "Hello";

  return hello + " " + name;
}

// "David" is an argument
console.log(greeting("David"));
```

Default Parameter

In JavaScript, function parameters default to **undefined**. However, it's often useful to set a different default value. This is where default parameters can help.

Default function parameters **allow named parameters to be initialized with default values if no value or undefined is passed**.



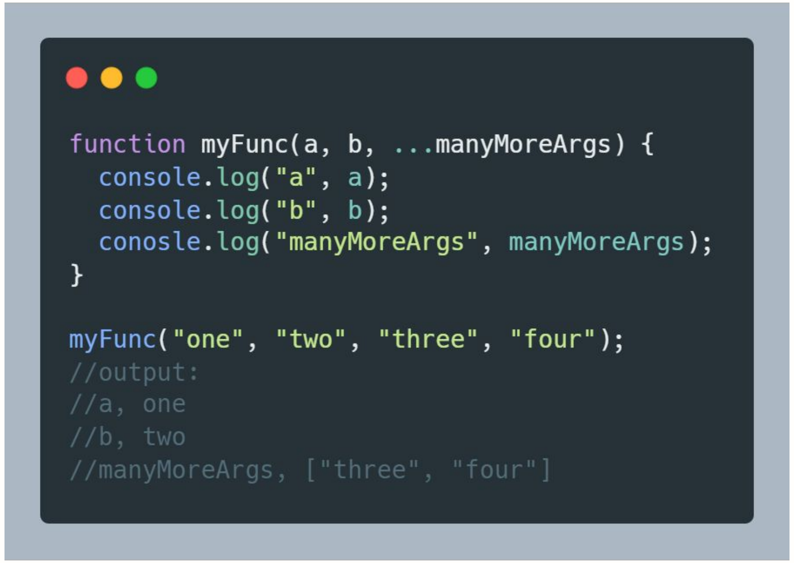
```
// default value for parameter b is 1
function multiply(a, b = 1) {
  return a * b;
}

// if the parameter value is given, it will use the given value
// otherwise, it will use the default value
console.log(multiply(5, 2)); // 10
console.log(multiply(5)); // 5
```

Rest Parameters

The **rest parameters** syntax allows a function to accept an indefinite number of arguments as an array, providing a way to represent variadic functions in JavaScript.

A function definition's last parameter can be prefixed with "...", which will cause all remaining (user supplied) parameters to be placed within a standard Javascript array.

Only the last parameter in a function definition can be a rest parameter.

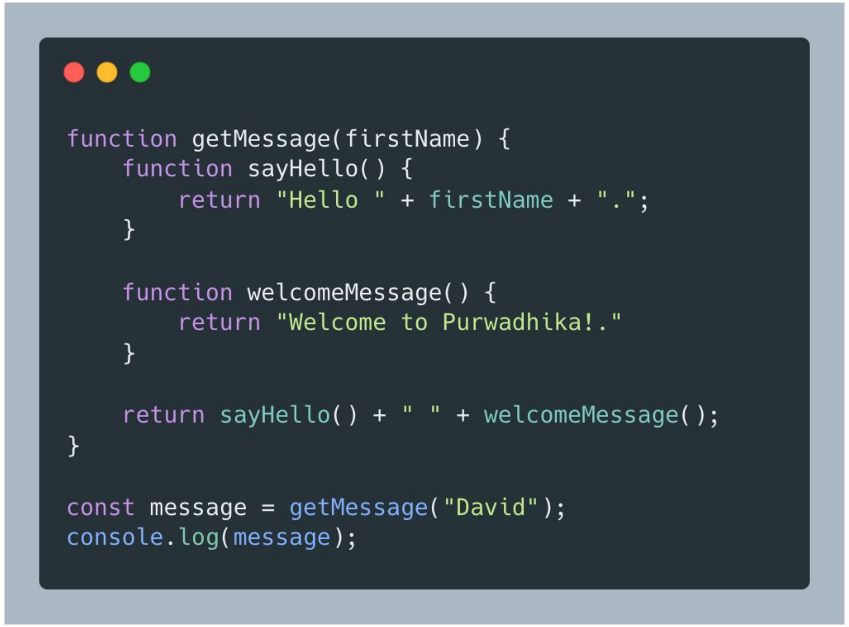


```
function myFunc(a, b, ...manyMoreArgs) {
  console.log("a", a);
  console.log("b", b);
  conosle.log("manyMoreArgs", manyMoreArgs);
}

myFunc("one", "two", "three", "four");
//output:
//a, one
//b, two
//manyMoreArgs, ["three", "four"]
```

Nested Function

In JavaScript, **a function can have one or more inner functions**. These nested functions are in the scope of outer function.

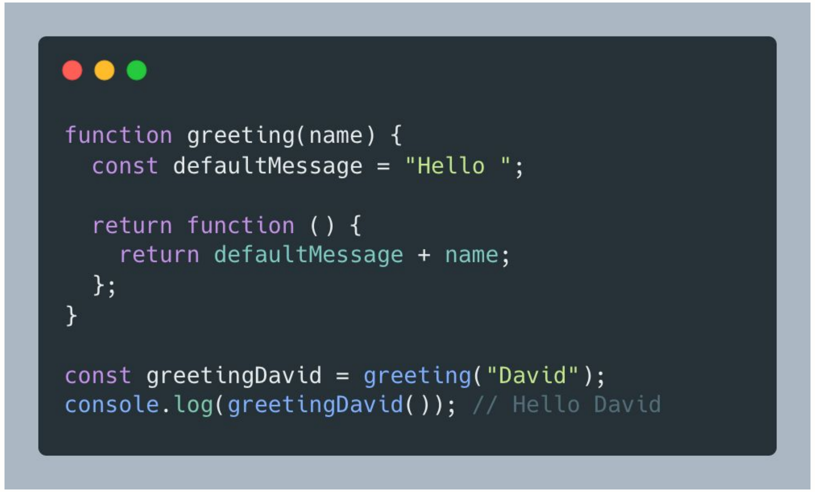
Inner function can access variables and parameters of outer function. However, outer function cannot access variables defined inside inner functions.



```javascript
function getMessage(firstName) {
    function sayHello() {
        return "Hello " + firstName + ".";
    }

    function welcomeMessage() {
        return "Welcome to Purwadhika!."
    }

    return sayHello() + " " + welcomeMessage();
}

const message = getMessage("David");
console.log(message);
```

Closure

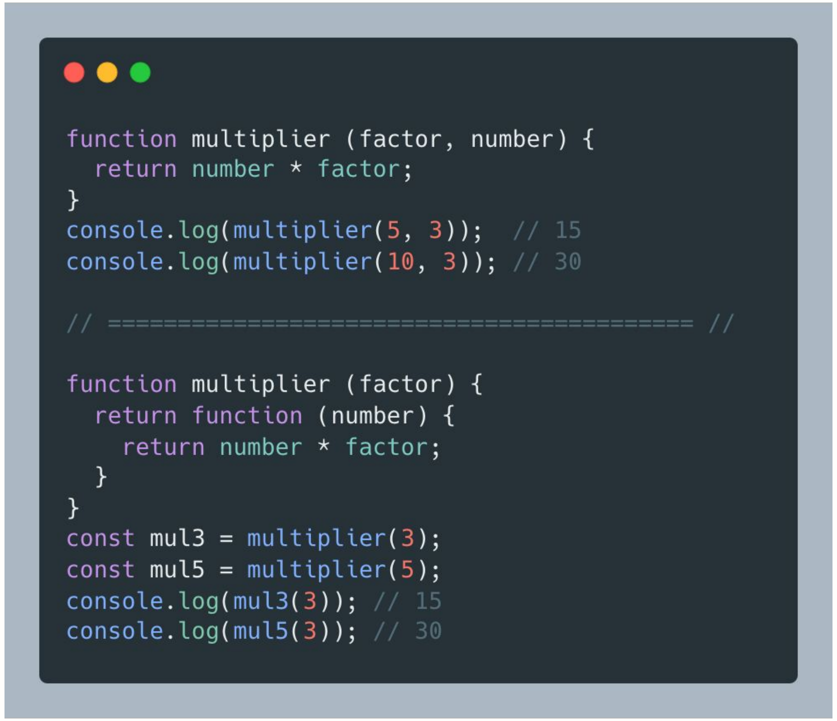
Closure means that an inner function always has access to the variables and parameters of it's outer function, even after the outer function has returned.



```
function greeting(name) {
  const defaultMessage = "Hello ";

  return function () {
    return defaultMessage + name;
  };
}

const greetingDavid = greeting("David");
console.log(greetingDavid()); // Hello David
```

Currying

- Currying is a transformation of functions that translates a function from callable as ***f(a, b, c)*** into callable as ***f(a)(b)(c)***.
- Currying **doesn't call a function. It just transforms it**.

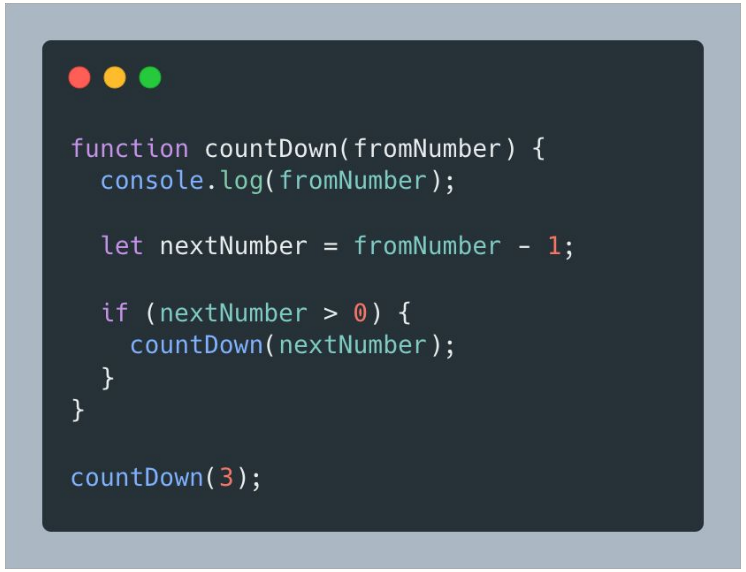


```
function multiplier (factor, number) {
  return number * factor;
}
console.log(multiplier(5, 3));  // 15
console.log(multiplier(10, 3)); // 30

// ========================================== //

function multiplier (factor) {
  return function (number) {
    return number * factor;
  }
}
const mul3 = multiplier(3);
const mul5 = multiplier(5);
console.log(mul3(3)); // 15
console.log(mul5(3)); // 30
```

Recursive

A **recursive** function is a function that calls itself until it doesn't.

In this example, the count down will stop when the next number is zero.

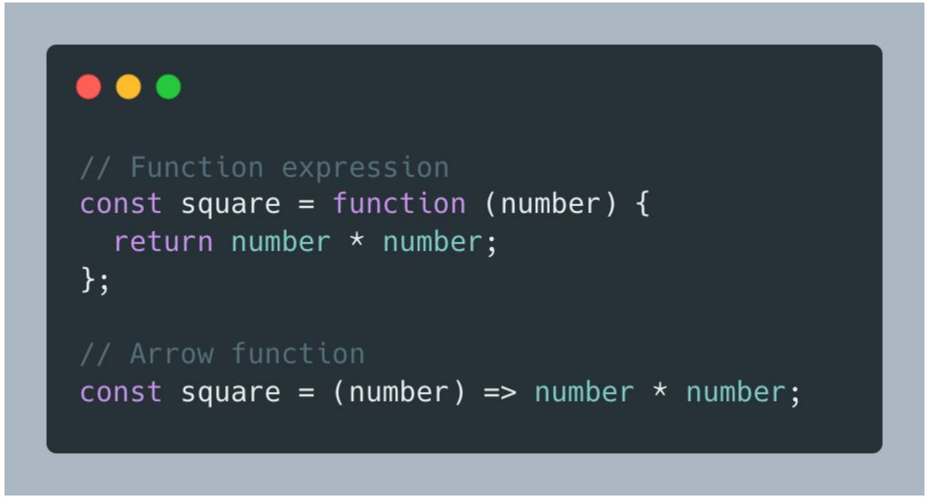


```
function countDown(fromNumber) {
  console.log(fromNumber);

  let nextNumber = fromNumber - 1;

  if (nextNumber > 0) {
    countDown(nextNumber);
  }
}

countDown(3);
```

Arrow Function

Arrow function provide you with an alternative way to write a shorter syntax compared to the function expression.

Arrow Function vs Function Expression



```
// Function expression
const square = function (number) {
  return number * number;
};

// Arrow function
const square = (number) => number * number;
```

Arrow Function Limitation

There are differences between *arrow functions* and *traditional functions*, as well as some limitations:

- Arrow functions don't have their own bindings to this, arguments or super, and should not be used as methods.
- Arrow functions don't have access to the new.target keyword.
- Arrow functions aren't suitable for call, apply and bind methods, which generally rely on establishing a scope.
- Arrow functions cannot be used as constructors.
- Arrow functions cannot use yield, within its body.

Predefined Function

JavaScript has several top-level, built-in functions:

- isFinite(), The global **isFinite()** function determines whether the passed value is a finite number. If needed, the parameter is first converted to a number.
- isNaN(), The **isNaN()** function determines whether a value is Nan or not.
- parseFloat(), The **parseFloat()**, function parses a string argument and returns a floating point number.
- parseInt(), The **parseInt()** function parses a string argument and returns an integer of the specified radix (the base in mathematical numeral systems).
- etc.

Exercise 1

- Create a function that can create a triangle pattern according to the height we provide like the following :

```
01
02  03
04  05  06
07  08  09  10
```

- Parameters : **height** → triangle height

Exercise 2

- Create a function that can loop the number of times according to the input we provide, and will replace **multiples of 3** with "**Fizz**", **multiples of 5** with "**Buzz**", **multiples of 3 and 5** with "**FizzBuzz**".
- Parameters : **n** → total looping

Exercise 3

- Create a function to calculate Body Mass Index (BMI)
- Formula : **BMI = weight (kg) / (height (meter))²**
- Parameters : **weight** & **height**
- Return values :
  - < 18.5 return “**less weight**”
  - 18.5 – 24.9 return “**ideal**”
  - 25.0 – 29.9 return “**overweight**”
  - 30.0 – 39.9 return “**very overweight**”
  - > 39.9 return “**obesity**”

Exercise 4

- Write a function to remove all odd numbers in an array and return a new array that contains even numbers only

Exercise 5

- Write a function to split a string and convert it into an array of words
  - Example : “Hello World” → [“Hello”, “World”]

Thank You!