Lesson:

Function with a parameter (Single, and Multiple)

PW SKILLS

Topics Covered

1. **Function with parameter.**
2. **Function with default parameter.**
3. **Function with two parameters.**
4. **Function with an unlimited number of parameters.**
5. **Interview point**

Function with parameter

Before looking into the syntax and code let's understand what is a parameter and along with the parameter comes an argument.

A parameter is a variable in a function definition.

An argument is the actual value passed to the function when the function is called.

Inside a function, we can take one, two, multiple, or an unlimited number of parameters. In this lecture, we will be looking at how to take one parameter inside the function, and the rest will be seen in further lectures.

Let's look at an example where we will be writing a function that displays the message passed as a parameter onto the console.

JavaScript

```
// Function with Parameter

// Function Declaration

function displayMessage(messageToBeDisplayed) {
  console.log(messageToBeDisplayed);
}
```

JavaScript

```
// Function with Parameter

// Function Declaration

function displayMessage(messageToBeDisplayed) {
  console.log(messageToBeDisplayed);
}

// Calling in a function.

displayMessage("I am happy to learn Full Stack Web Development from PW Skills");
displayMessage("I am enjoying the journey of javascript");
```

In the above examples the strings “I am happy to learn Full Stack Web Development from PW Skills" and "I am enjoying the journey of javascript" are passed into the function during the function call. These are the arguments.

When a function is called, the values passed in as arguments are assigned to the corresponding parameters in the function definition.

Function with Default Parameters

Sometimes functions are written with default parameter values to avoid errors when the function is called without passing any arguments.

```
JavaScript
// Function with Parameter

// Function Declaration

function displayMessage(
  messageToBeDisplayed = "I am a proud student of PW Skills"
) {
  console.log(messageToBeDisplayed);
}

// Calling in a function.

displayMessage("I am happy to learn Full Stack Web Development from PW Skills");
// OUTPUT: I am happy to learn Full Stack Web Development from PW Skills

displayMessage();
// OUTPUT: I am a proud student of PW Skills
```

Here, the function "messageToBeDisplayed" has a default value of "I am a proud student of PW Skills".

When the argument "I am happy to learn Full Stack Web Development from PW Skills" is passed to the function and assigned to the parameter "messageToBeDisplayed". This results in the output "I am happy to learn Full Stack Web Development from PW Skills" being displayed in the console.

When the function is called without an argument or with undefined, the parameter "messageToBeDisplayed" is set to its default value “I am a proud student of PW Skills” and the output "I am a proud student of PW Skills" is displayed in the console.

Function with two parameters

To look at the function with two parameters, the best example would be of writing a function that gives us the sum of two numbers by taking the numbers as parameters.

```javascript
// Function with two parameters.

// Function Declaration

function sumOfTwoNumbers(num1, num2) {
  return num1 + num2;
}
```

The function "sumOfTwoNumbers" takes in two parameters, "num1" and "num2," and returns the sum of the two numbers.

```javascript
// Function with two parameters.

// Function Declaration

function sumOfTwoNumbers(num1, num2) {
  return num1 + num2;
}

// Calling a Function

let result = sumOfTwoNumbers(10, 40);
console.log(result);

// OUTPUT: 50
```

It is not necessary to always pass arguments as primitive values or variables containing primitive values like

```javascript
// Function with two parameters.

// Function Declaration

function sumOfTwoNumbers(num1, num2) {
  return num1 + num2;
}

// Calling a Function
```

```
let number1 = 10;
let number2 = 40;

let result = sumOfTwoNumbers(number1, number2);
console.log(result);

// OUTPUT: 50
```

We can also pass the values as an array.

```
JavaScript
// function with two parameters.

// Function Declaration

function sumOfTwoNumbers([num1, num2]) {
  return num1 + num2;
}
```

The function takes an array as a parameter, this array should contain two values, and it returns the sum of the two numbers.

```
JavaScript
// Function with two parameters.

// Function Declaration

function sumOfTwoNumbers([num1, num2]) {
  return num1 + num2;
}

// Calling a Function

let numbers = [10, 40];

let result = sumOfTwoNumbers(numbers);
console.log(result);

// OUTPUT: 50
```

The function parameter is defined as an array, so the two values inside the argument array are accessed using their index, in this case, it is assumed that the first element of the array (index 0) is num1, and the second element of the array (index 1) is num2.

Ideally, developers pass arguments as primitive values, but it is important to note that we can pass an array as an argument too.

Function with an unlimited number of parameters.

we will be looking at a situation where an unlimited number of parameters are passed as parameters to the function. As a developer, what are the options available to handle this situation?

Assume a condition where we want to find the sum of all the numbers passed as parameters to the function. In this case, we have no idea of the number of parameters. So, we cannot make use of what we have learned in previous lectures.

To do this we need to have an idea of the “arguments” keyword.

In JavaScript, the arguments keyword refers to an object that contains all the arguments passed to a function.

It is similar to an array, but not an actual array. We can make use of the “.length” method to know how many parameters are passed.

The arguments object is available within all function bodies and can be used to access the values of the arguments passed to the function.

Now to find out the sum of unlimited numbers passed parameter into the function we write.

```JavaScript
// Function declaration


function sumOfAllParameters() {
  let sum = 0;
  for (var i = 0; i < arguments.length; i++) sum += arguments[i];
  return sum;
}

// Function call

let result = sumOfAllParameters(1, 2, 3, 4, 5);
console.log(result);

// OUTPUT: 15
```

The function "**sumOfAllParameters**" accepts an unlimited number of parameters. The function uses the arguments object to access all the parameters passed to it, regardless of the number of parameters. Inside the function, a for loop iterates through the arguments object, adding each value to a variable sum. The final value of the sum variable is returned as the result of the function.

When the function is called, multiple arguments are passed to the function, in this case, it will be 1, 2, 3, 4, 5, The function will add all these numbers and return the total, which is 15, and this value gets stored in the variable result and then printed to the console using console.log(result).

It is not recommended to use the arguments object, as it is not as efficient as declaring the function parameters.

Interview Point

Q1. Write a function that takes user information [name, age, country] as an object and displays a welcome message using string literals. Example message: 'Hi Mithun, you are 21 years old from India and we are happy to have you onboard'.
Ans:

```
JavaScript

function welcomeUser(userInfo) {
  const { name, age, country } = userInfo;
   const welcomeMessage = `Hi ${name}, you are ${age} years old from ${country} and we are
happy to have you onboard.`;
  console.log(welcomeMessage);
}

 const userInformation = {
   name: 'Mithun',
   age: 21,
   country: 'India'
 };

 welcomeUser(userInformation);

 // Output: Hi Mithun, you are 21 years old from India and we are happy to have you onboard.
```

Q2. Create a JavaScript function that calculates the factorial of a given non-negative integer.
Ans:

Ans:

Ans:

```javascript
function factorial(n) {
 if (n === 0 || n === 1) {
  return 1;
 } else {
  return n * factorial(n - 1);
 }
}

// Testing the function
console.log(factorial(5)); // Output: 120
console.log(factorial(0)); // Output: 1
```