Lesson:

Important functions in JavaScript

PW SKILLS

Topics to be covered in

1. Anonymous Function
2. Named Function Expressions
3. First Class Function
4. IIFE
5. Pure and Impure Function
6. Callback Function
7. HOF
8. Lambda Function
9. Currying Function

Anonymous Function

An anonymous function in javascript is a function that is defined without a specified function name.
Let's take example to understand the declaration of anonymous function-

```javascript
// using the function expression
const greet = function () {
  console.log("Good morning");
};

// using the arrow function
const greet2 = () => {
  console.log("Good afternoon");
};
```

As mentioned In the above example we can use function expression or arrow function to declare an anonymous function as it does not have any function name defined over there.

Named Function Expression

A named function expression in javascript is a function expression that is assigned to a variable with a function name.
Let's take example to understand the declaration of named function expression-

```javascript
const greet = function namedFunctionExpression() {
  console.log("Named Function Expression");
};
greet();
```

In the above example, we can see that we had given the name to function after the function keyword and assigned the function to a variable named greet.

Difference between anonymous function and named function expression

Anonymous Function	Named Function Expression
• It is defined without a specified name. • It does not have any specified name that's why it cannot be self-referenced or called recursively. • It is not easy to debug as they do not have any specific identifier.	• It is defined with a specified name and will be assigned to a variable. • By using the name of the function we call the function recursively inside its block. • • It is not easy to debug as it has a name which can be used.

Example

```javascript
// anonymous function
const greet1 = function () {
  console.log("Anonymous Function");
};

// named function expression
const greet2 = function namedFunctionExpression() {
  console.log("Named Function Expression");
};
```

First Class Function

In JavaScript, functions are first class citizens which means that the functions are treated like other variables. In simple words, we can pass a function as an argument and can also return it from a function like a normal variable.

Example of passing a function as argument

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

function subtract(num1, num2) {
  return num1 - num2;
}

function calculator(num1, num2, operation) {
  return operation(num1, num2);
}
console.log(calculator(10, 5, add));
console.log(calculator(10, 5, subtract));

// output
// 15
// 5
```

In the above example, we have add and subtract function which is returning the addition or subtraction of two numbers respectively. The third function calculator is taking the parameters num1, num2 and operation in which the num1 and num2 are numbers while the operation is the function passed to it.
In the calculator function, it is calling the function passed to it with the given two numbers and returning their result.

Example for returning a function from another function

```javascript
function createGreeting(greet) {
  return function (name) {
    console.log(greet + " " + name);
  };
}

const morning = createGreeting("Good Morning");
morning("PW Skills");
const night = createGreeting("Good Night");
night("PW Live");

 // output
 // Good Morning PW Skills
 // Good Night PW Live
```

In the above example, we had a createGretting function which is taking a parameter for the greeting message and returning back a function which is taking the name of the user as a parameter and printing the greeting message and user name on console.So when createGreeting is called, it is returning the function which is being stored in the variable and then that variable is being invoked with the user name to print the greeting message and user name on the console.

IIFE

Introduction

IIFE stands for immediately invoked function expression. It is a javascript function that is executed as soon as it is defined. It is an anonymous function which is executed as soon as they are mounted by using the parenthesis ().

Syntax

```JavaScript
// using function keyword
(function () {
  // logic code
})()

// using the arrow function
(() => {
  // logic code
})()
```

Example

```JavaScript
(() => {
  console.log("I am an IIFE");
})();

// output
// I am an IIFE
```

Pure and Impure Function

Pure Function

A function which always returns the same output for the same input parameter is called pure function.
Pure function is only dependent on the parameter it will take, not on the other part of the program and it will also not affect the state of the other parts of the code.

Example

```
JavaScript
const add = (num1, num2) => {
  return num1 + num2;
};

const addition1 = add(10, 20);
const addition2 = add(10, 20);
const addition3 = add(15, 25);
const addition4 = add(15, 25);
console.log(addition1, addition2, addition3, addition4);

// output
// 30 30 40 40
```

In the above example, we can see that for the same parameters we are getting the same result and also our function is not affecting any variables which are before or after it while its execution. Hence the function add is a pure function.

Impure Function

A function which may return different values for the same input parameters is called impure function.
It may be dependent on the other code which is outside the function like any outside variable and can also modify the value of that particular variable while its execution.

```
JavaScript
let totalValue = 100;
const increaseValue = (increment) => {
  return (totalValue += increment);
};

const value1 = increaseValue(50);
const value2 = increaseValue(50);
console.log(value1, value2);

// output
// 150 200
```

In the above example, we can see that we have passed the same argument in value1 and value2 but not getting the same result, also we are manipulating the value of totalValue which is defined outside the function. That's why this one is an impure function.

Difference between pure and impure function

Pure Function	Impure Function
• It does not modify others which are outside the function. • It is only dependent upon the input parameters. • It is easier to test. • It is easier to maintain.	• It can modify others which are outside the function. • It may be dependent upon the other parts of the code. • It is hard to test. • It is harder to maintain.

Callback Function

A callback is a function that is passed as an argument to another function. It will run after the main function has finished its execution. While invoking the main function, the callback function will be passed to the function as an argument in it.
Callback functions are quite important in handling the asynchronous code in javascript, which we will be discussing later in this course.

Example

JavaScript

```
const displayResult = (result) => {
  console.log("Your result is", result);
};
const add = (num1, num2, callback) => {
  const result = num1 + num2;
  callback(result);
};

const subtract = (num1, num2, callback) => {
  const result = num1 - num2;
  callback(result);
};

add(20, 10, displayResult);
subtract(20, 10, displayResult);

// output
// Your result is 30
// Your result is 10
```

As in the above example, we can see that we had a function to display the output to which we are passing as an argument to the add and subtract function and that one is executed at the end to display the result after the calculation. So displayResult is a callback function here.

Note: We will see more examples and real life use cases of callback functions in advanced javascript modules.

HOF

HOF stands for Higher Order Function. A HOF is a function that takes one or more functions as arguments or returns a function as a result. This is the reason that the functions in javascript are called first class citizens as they can be treated as a variable.
In other words, we can say that HOF operates on function, either by accepting them as a parameter or by returning them.

Example of returning a function from another function

```javascript
function createGreeting(greet) {
  return function (name) {
    console.log(greet + " " + name);
  };
}

const morning = createGreeting("Good Morning");
morning("PW Skills");
const night = createGreeting("Good Night");
night("PW Live");

// output
// Good Morning PW Skills
// Good Night PW Live
```

In the above example, the createGreeting function is returning a function from it which is being stored in the morning and night variable and being invoked after it to get the output.

Example of passing a function to another function

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

function subtract(num1, num2) {
  return num1 - num2;
}
```

```
function calculator(num1, num2, operation) {
  return operation(num1, num2);
}

console.log(calculator(10, 5, add));
console.log(calculator(10, 5, subtract));

// output
// 15
// 5
```

Lambda Function

In javascript, lambda function is a short and anonymous function that takes one or more parameters and contains a single expression. Lambda functions are also referred to as arrow functions used for writing short and readable code.

Example:

JavaScript

```
const add = (num1, num2) => num1 + num2;
console.log(add(10, 20));

// output
// 30
```

In the above example, we are using the arrow function and getting two parameters in it and have a single expression which will be returned by default as it is an arrow function.

Function Currying

It is a technique used to transform a function of multiple parameters into several functions of a single argument in sequence. By using this technique we can easily make our code more readable, reusable and easy to test and debug.

Example

JavaScript

```
// using simple arrow function
function calculateVolume(length, breadth, height) {
  return length * breadth * height;
}
```

```
const volume = calculateVolume(2, 5, 10);
console.log(volume);
// output
// 100

// using the function currying method
function calculateNewVolume(length) {
  return function (breadth) {
    return function (height) {
      return length * breadth * height;
    };

  };
}

const newVolume = calculateNewVolume(2)(5)(10);
console.log(newVolume);
// output
// 100
```

In the above example, inside the function currying we had just separated all the parameters in an individual function which makes it more readable. Also we can see that we are using function chaining inside the function currying to invoke them one after another.