Functions

In JavaScript

CodeMaster Noyon

JS Functions

A JavaScript function is a block of code designed to perform a particular task. JavaScript functions are defined with the `function` keyword. Function types in JavaScript:

- Named functions
- Anonymous functions
- Arrow functions
- callback functions
- Higher order functions
- IIFE functions

CodeMaster Noyon

JS Named Function

Named functions are regular functions that have a name or identifier.

Name function syntax:

```
function addTwoValue(a, b){
    console.log("Sum: ", a+b);
};    //  It's called function declaration. Declared functions are not executed immediately.

addTwoValue(20, 10);  // function call
```

CodeMaster Noyon

JS Named Function Expression

Named functions are regular functions that have a name or identifier.

Function Expression:

```
let sum = function addTwoValue(a, b){
    console.log("Sum: ", a+b);
}; //  It's called function declaration. Declared functions are not executed immediately.


sum(10, 5);  // function expression call
```

CodeMaster Noyon

JS Anonymous Function

It is a function that does not have any name associated with it.

Anonymous function syntax:

```
const x = function(parameters){
    // code can be executed
};  // function define


x(); // function call
```

CodeMaster Noyon

JS Arrow Function

JavaScript Arrow functions were introduced in ES6.

Arrow function syntax:

```
const hello = (parameters) =>{
    return parameters;
};
```

Without parentheses:

```
const hello = a => a;
```

Return value by Default:

```
const hello = (para1, para2) => console.log(para1, para2);
```

CodeMaster Noyon

JS callback Function

Functions that are used as parameters to another function is called callback function.

```
const display = () =>{
    console.log('Hello world!');
};
// display() function defined, NOT call
```

```
const printSomething = (takeCallBack)=>{
    takeCallBack();
};
//printSomething() function defined, NOT call
```

```
printSomething(display);
```

// called printSomething() function & take function as a argument.

Here, display() is a callback function.

CodeMaster Noyon

JS Higher order Function

Functions that take another functions as parameters, are called Higher order function.

```
const display = () =>{
    console.log('Hello world!');
};
// display() function defined, NOT call
```

```
const printSomething = (takeCallBack)=>{
    takeCallBack();
};
//printSomething() function defined, NOT call
```

```
printSomething(display);
```

// called printSomething() function & take function as a argument.

Here, printSomething() is a Higher order function.

CodeMaster Noyon

JS IIFE Function

In JavaScript, IIFE refers Immediately Invoked Function Expression. IIFE function runs as soon as it defined.

IIFE function syntax:

```
(function(){
    console.log('Hello World!');
})();
```

Using Arrow function:

```
(()=>{
    console.log('Hello
World!');
})();
```

CodeMaster Noyon

JS Different Between functions & methods

Functions are used for performing tasks. Methods are used for manipulation objects.

```
const calSum = (a, b) =>{
    return a + b;
};

// calSum() is a function
```

```
let Name = 'Noyon Sarker';
Name.toUpperCase();

// toUpperCase() is a method
```

CodeMaster Noyon

JS function *rest* Parameters

The rest parameters (...) allows a function to treat an indefinite number of arguments as an array.

```
const sum = (...args) =>{
    // all arguments are received args parameter.
    let sum = 0;
    for(let i of args){
        sum += i;
    };
    return sum;
};

let result = sum(2,4,6, 5, 4, 6, 10, 5);

console.log(result);  // 42
```

CodeMaster Noyon

JS call() & apply() methods

With the `call()` method, you can write a method that can be used on different objects.

By the `call()` method, you can pass one object's value to another object. Objects keys must be same.

```
const person = {
    fullName: function(){
        return this.firstName+" "+this.lastName;
    }
};
```

```
const person1 = {
    firstName: 'Noyon',
    lastName: 'Sarker'
};
```

```
let result = person.fullName.call(person1);
console.log(result);  // Noyon Sarker
```

CodeMaster Noyon

JS call() & apply() methods

When objects keys are not same. The `apply()` method is similar to the `call()` method.

```
const person = {
    fullName: function(){
        return this.firstName+" "+this.lastName;
    }
};
```

```
const person1 = {
    fname: 'Noyon',
    lname: 'Sarker'
};
```

```
let result = person.fullName.call(person1);
console.log(result);  // undefined undefined
```

CodeMaster Noyon

JS bind() methods

The `bind()` method is similar to the `call()` method. But `bind()` method returns the function

```
const person = {
    fullName: function(){
        return this.firstName+" "+this.lastName;
    }
};
```

```
const person1 = {
    firstName: 'Noyon',
    lastName: 'Sarker'
};
```

```
let result = person.fullName.bind(person1);
result(); // Noyon Sarker
```

CodeMaster Noyon

JS function closures

A closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

```
function int(){
    let Name= 'Noyon';

    function display(){
        console.log(`My name is ${Name}`);
    };
    return display;
};  // int() returns display function

let result = int();  // Here, result holds a function.
result(); // My name is Noyon


int()();  // There will be same output
```

CodeMaster Noyon

JS Closure Scope Chain

Every closure has three scope:

- Local Scope (Own scope)
- Outer Function Scope
- Global Scope

```
// Global Scope
let Name = 'Noyon’;
function makeFunc(){
    //  Outer Function Scope
    let Age = 24;

    function display(country){
        //  Local Scope (Own scope)
        let Country = country;
        console.log(Name, Age,   Country);
    };
    return display;
};

makeFunc()('Bangladesh');
```

CodeMaster Noyon

JS Closure Scope Chain

```
// this task like, Every men calling their own son.

let text = 'Meeting calling: '; // global scope

function callMeeting(grandFather){
    return function(myFather){
        return function(me){
            // outer function scope
            return function(mySon){
                //local scope
                console.log(`${text} ${grandFather} --> ${myFather} --> ${me} --> ${mySon}`);
            }
        }
    }
};
callMeeting('Meghlal')('Vanu Ranjon')('Mithun')('Arjun’);
//Meeting calling: Meghlal --> Vanu Ranjon --> Mithun --> Arjun
```

CodeMaster Noyon