**7. Functions**

**1) Function Declaration vs. Expressions**:

**Function in JavaScript**:

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure a set of statements that performs a task or calculates a value. To use a function, you must define it somewhere in the scope from which we wish to call it.

**Define Function in JavaScript**:

In JavaScript there are two way to define a function.

1. Function declaration

**Example**:

function walk() {

console.log("walk");

}

walk(); //walk

1. Function Expression

**Example**: Anonymous function expression

let run = function() {

console.log("run");

};

run(); //run

**2) Hoisting**:

Hoisting is the process of moving function declarations to the top of the file. This is done automatically by the JavaScript engine, that is executing the code.

**Example**:

//valid

walk(); //run

function walk() {

console.log("walk");

}

**Example**:

//invalid

run(); //error

let run = function() {

console.log("run");

};

This statement is like

console.log(x);

let x = 10;

**3) Arguments**:

Arguments is an Array-like object accessible inside functions that contains the values of the arguments passed to that function.

**Note**:

“Array-like” means that arguments has a length property and properties indexed from zero, but it doesn't have Array's built-in methods like forEach and map.

**Example**:

function sum(a, b) {

return a + b;

}

console.log(sum(1, 2)); //3

console.log(sum(1, 2, 3, 4)); //3

console.log(sum(1)); //NaN -> 1 + undefined -> NaN

Every function in JavaScript have a special object called argument.

function sum(a, b) {

console.log(arguments);

return a + b;

}

console.log(sum(1, 2, 3, 4, 5));

**Output**:

/\*

Arguments(5) [1, 2, 3, 4, 5, callee: ƒ, Symbol(Symbol.iterator): ƒ]0: 11: 22: 33: 44: 5callee: ƒ sum(a, b)length: 5Symbol(Symbol.iterator): ƒ values()\_\_proto\_\_: Object

\*/

function sum() {

let total = 0;

for (let value of arguments) total += value;

return total;

}

console.log(sum(1, 2, 3, 4, 5)); //15

**4) The Rest Operator**:

Rest Operator is an improved way to handle function parameter, allowing us to more easily handle various input as parameters in a function. And it is aa actual array.

JavaScript has allowed a variable number of function parameters of a function but the problem is that it is not an array. It is an array like object. Therefore performing some operations on “arguments” will give an error.

Rest operator is added in ES2015 or ES6 which improved the ability to handle parameter.

**Example**:

function sum(...args) {

console.log(args);

}

console.log(sum(1, 2, 3, 4, 5)); //[1, 2, 3, 4, 5]

**Example**:

function sum(...args) {

return args.reduce((a, b) => a + b);

}

console.log(sum(1, 2, 3, 4, 5)); // 15

**Example**:

function sum(discount, ...prices) {

const total = prices.reduce((a, b) => a + b);

return total \* (1-discount);

}

console.log(sum(0.1, 20, 30)); //45

**Note**:

We must have to place the rest operator as the end argument as a function. If we place any argument after “rest operator” we will get error. For this reason this operator is called rest operator.

**Example**:

function sum(discount, ...prices, value) {

const total = prices.reduce((a, b) => a + b);

return total \* (1-discount);

}

console.log(sum(0.1, 20, 30));

/\*

Error: Uncaught SyntaxError: Rest parameter must be last formal parameter

\*/

**5) Default parameters**:

In JavaScript, function parameters default to [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined). However, it's often useful to set a different default value. This is where default parameters can help.

**Example**:

function interest(principal, rate, years){

return principal\*rate / 100\*years;

}

console.log(interest(10000, 3.5, 5)); //1750

We can use default value as

**Example**:

function interest(principal, rate, years) {

rate = rate || 3.5;

years = years || 5;

return ((principal \* rate) / 100) \* years;

}

console.log(interest(10000)); //1750

From ES6 we can write

**Example**:

function interest(principal, rate = 3.5, years = 5) {

return ((principal \* rate) / 100) \* years;

}

console.log(interest(10000)); //1750

**Note**:

If we give a default value of a parameter after that all value we have to give a default value otherwise we will get error.

**Example**:

function interest(principal, rate = 3.5, years) {

return ((principal \* rate) / 100) \* years;

}

console.log(interest(10000)); //NaN

**6) Getters and Setters**:

We can use getter method to access properties and setter method to change or mute them.

getters => access properties

getters => change (mute) them

**Example**:

const person = {

firstName: 'Mosh',

lastName: 'Hamedani',

get fullName(){

return `${person.firstName} ${person.lastName}`;

},

//value receive here is a valid string

set fullName(value){

const parts = value.split(" ");

this.firstName = parts[0];

this.lastName = parts[1];

}

};

console.log(person.fullName); //Mosh Hamedani

person.fullName = 'John Smith'; // set "John Smith"

console.log(person.fullName); // John Smith

**7) Try and Catch**:

The try/catch/finally statement handles some or all of the errors that may occur in a block of code, while still running code. Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The finally statement lets you execute code, after try and catch, regardless of the result.

**Example**:

const person = {

firstName: 'Mosh',

lastName: 'Hamedani',

get fullName(){

return `${person.firstName} ${person.lastName}`;

},

//value receive here is a valid string

set fullName(value){

const parts = value.split(" ");

this.firstName = parts[0];

this.lastName = parts[1];

}

};

console.log(person.fullName); //Mosh Hamedani

person.fullName = true;

*//Uncaught TypeError: value.split is not a function*

person.fullName = null;

*//* *Uncaught TypeError: Cannot read property 'split' of null*

For resolved this error we should add try-catch in the above program.

**Example**:

const person = {

firstName: "Mosh",

lastName: "Hamedani",

get fullName() {

return `${person.firstName} ${person.lastName}`;

},

//value receive here is a valid string

set fullName(value) {

//if(typeof value !== 'string') return

if (typeof value !== "string") {

throw new Error("Value is not a string");

}

const parts = value.split(" ");

this.firstName = parts[0];

this.lastName = parts[1];

}

};

try {

person.fullName = null;

} catch (e) {

console.log(e);

}

console.log(person);

/\*

Error: Value is not a string

at Object.set fullName [as fullName]

at index.js:20

{firstName: "Mosh", lastName: "Hamedani"}

\*/

**Example**:

Pass empty String as full name

const person = {

firstName: "Mosh",

lastName: "Hamedani",

get fullName() {

return `${person.firstName} ${person.lastName}`;

},

//value receive here is a valid string

set fullName(value) {

//if(typeof value !== 'string') return

if (typeof value !== "string") {

throw new Error("Value is not a string");

}

const parts = value.split(" ");

if(parts.length !== 2){

throw new Error('Enter a valid name');

}

this.firstName = parts[0];

this.lastName = parts[1];

}

};

try {

person.fullName = '';

} catch (e) {

console.log(e);

}

console.log(person);

/\*

Error: Enter a valid name

at Object.set fullName [as fullName]

{firstName: "Mosh", lastName: "Hamedani"}

\*/

7. Functions