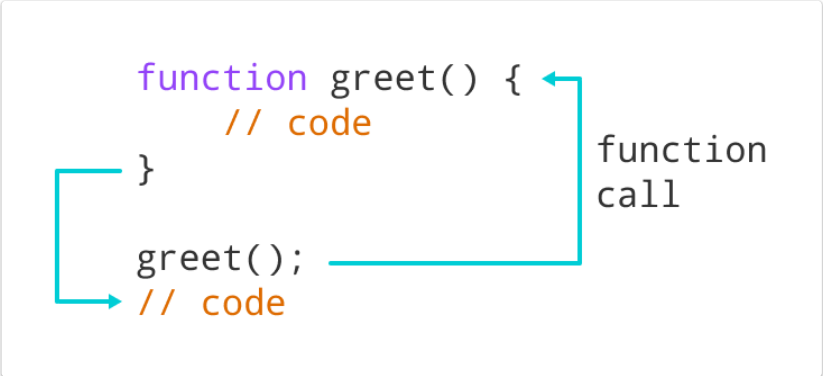
1-oy 8-dars. Funksiyalar va ularning turlari (Function Expression va Function Declaration, anonim function, arrow function, callback function). Jsda rekursiya

8-dars. Funksiyalar va ularning turlari (Function Expression va Function Declaration, anonim function, arrow function, callback function). Jsda rekursiya

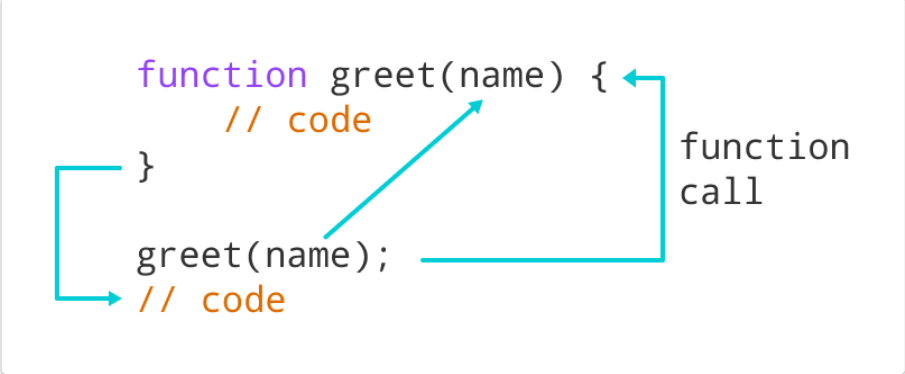
Declaring a Function

```
/// function yaratish
function nameOfFunction () {
    // function body
}

nameOfFunction()
```



Function Parameters



Function Expressions

```
let x = function (num) { return num * num };
console.log(x(4)) //=> 16

let y = x(3);
console.log(y) // => 9
```

Arrow Function

```
// function expression
let x = function(x, y) {
   return x * y;
}
x(1, 2) //=> 2
```

```
// using arrow functions
let x = (x, y) => x * y;

x(2, 3) //=> 6
```

```
                                                    rest parametr
let myFunction = (arg1, arg2,  ...argN) => {
    console.log(arg1, arg2, arg3) //=> 1,2,3,4,5,6
//→ arg1: 1, arg2: 2, arg3: [1,2,3]
}

myFunction(1,2,3,4,5,6)
```

Arrow Function with No Argument

```
let greet = () => console.log('Hello');
greet(); // Hello
```

Arrow Function with One Argument

```
let greet = x => console.log(x);
greet('Hello'); // Hello
```

Arrow Function as an Expression

```
let age = 5;

let welcome = (age < 18) ?
  () => console.log('Baby') :
  () => console.log('Adult');

welcome(); // Baby
```

Arguments Binding

```
let x = function () {
    console.log(arguments);
}
x(4,6,7); // Arguments [4, 6, 7]
```

Spread Operator

```
const arrValue = ['My', 'name', 'is', 'Jack'];

console.log(arrValue);   // ["My", "name", "is", "Jack"]
console.log( ... arrValue); // My name is Jack
```

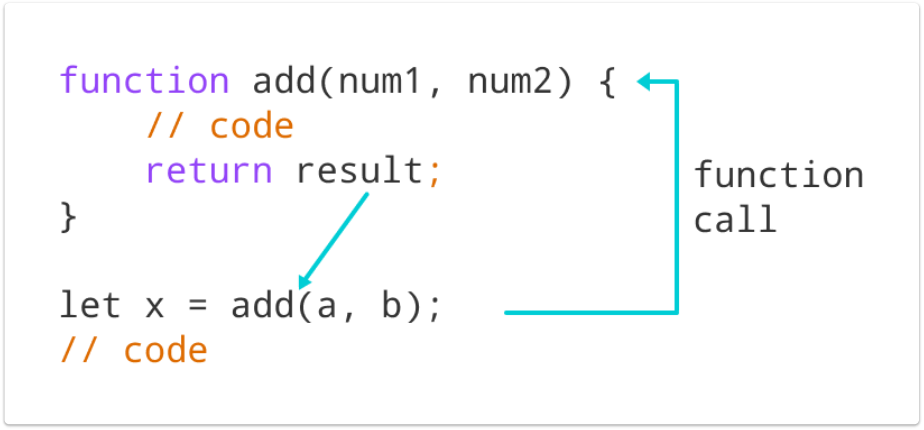
Rest Parameter

```
let func = function( ... args) {
    console.log(args);
}

func(3); // [3]
func(4, 5, 6); // [4, 5, 6]
```

JavaScript CallBack Function

Function Return




```
// function
function greet(name, callback) {
    console.log('Hi' + ' ' + name);
    callback();
}

// callback function
function callMe() {
    console.log('I am callback function');
}

// passing function as an argument
greet('Peter', callMe);
```

JavaScript Variable Scope

1. Global Scope
2. Local Scope

```
// program to print a text
let a = "hello";
```

```
function greet () {
    console.log(a);
}

greet(); // hello
```

Local Scope

```
// program showing local scope of a variable
let a = "hello";

function greet() {
    let b = "World"
    console.log(a + b);
}

greet();
console.log(a + b); // error
```

let is Block Scoped

```
// program showing block-scoped concept
// global variable
let a = 'Hello';

function greet() {

    // local variable
    let b = 'World';

    console.log(a + ' ' + b);

    if (b == 'World') {

        // block-scoped variable
        let c = 'hello';

        console.log(a + ' ' + b + ' ' + c);
    }

    // variable c cannot be accessed here
    console.log(a + ' ' + b + ' ' + c);
}

greet();
```

JavaScript Hoisting

```
// using test before declaring
console.log(test);   // undefined
var test;
```

```
// using test before declaring
var test;
console.log(test); // undefined
```

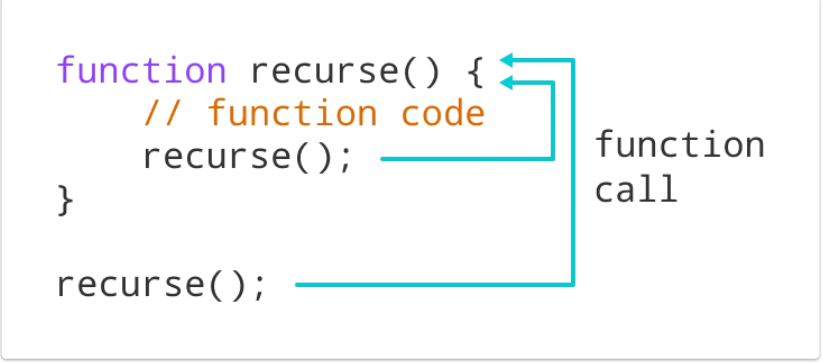
Variable Hoisting

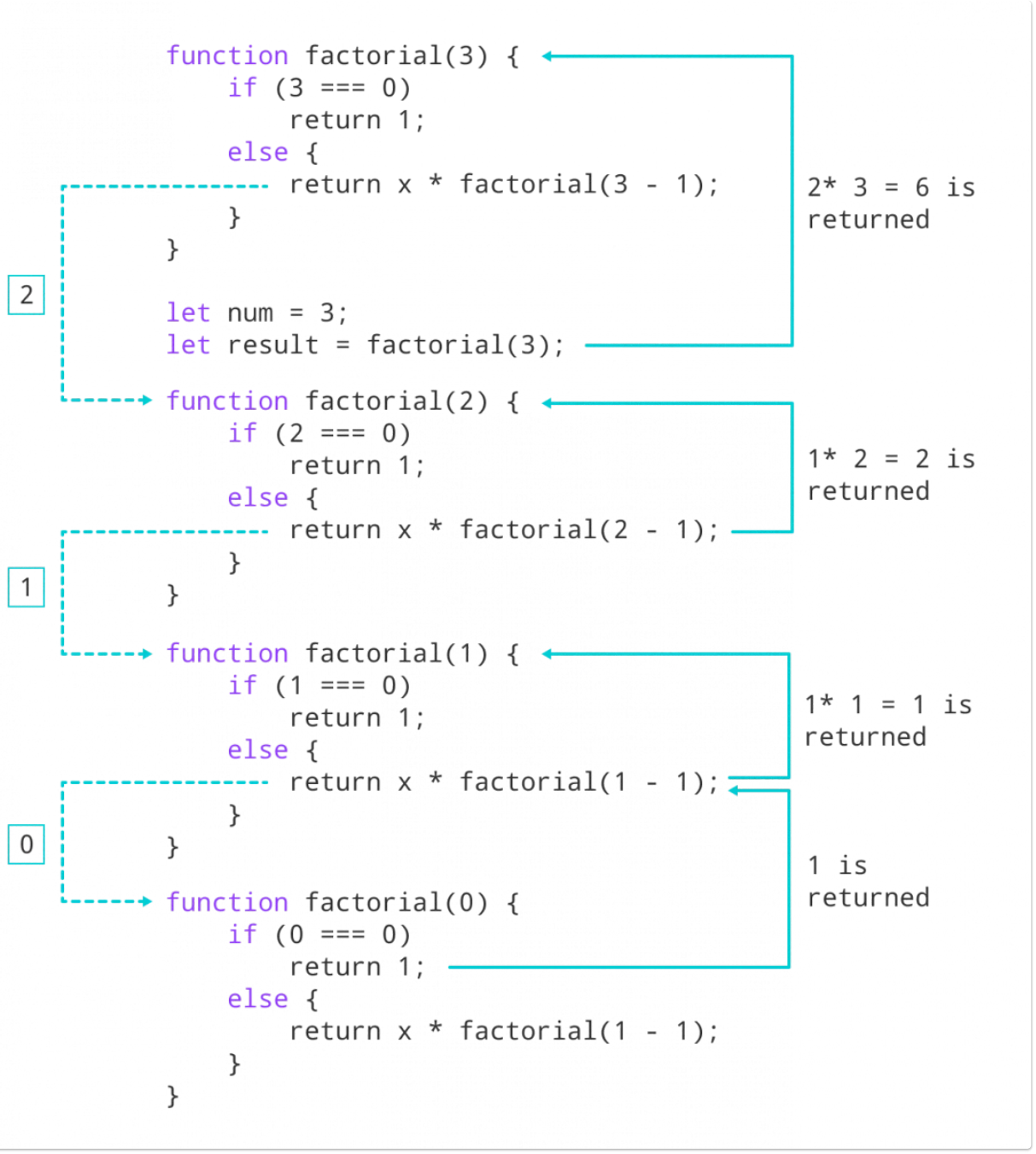
```
// program to display value
a = 5;
console.log(a);
var a; // 4
```

JavaScript Recursion

```
function recurse() {
    // function code
    recurse();
    // function code
}

recurse();
```




function factorial(3) {
if (3 === 0)
return 1;
else {
return x * factorial(3 - 1);
}
}
2
let num = 3;
let result = factorial(3);
2* 3 = 6 is returned
function factorial(2) {
if (2 === 0)
return 1;
else {
return x * factorial(2 - 1);
}
1
}
1* 2 = 2 is returned
function factorial(1) {
if (1 === 0)
return 1;
else {
return x * factorial(1 - 1);
}
0
}
1* 1 = 1 is returned
function factorial(0) {
if (0 === 0)
return 1;
else {
return x * factorial(1 - 1);
}
}
1 is returned