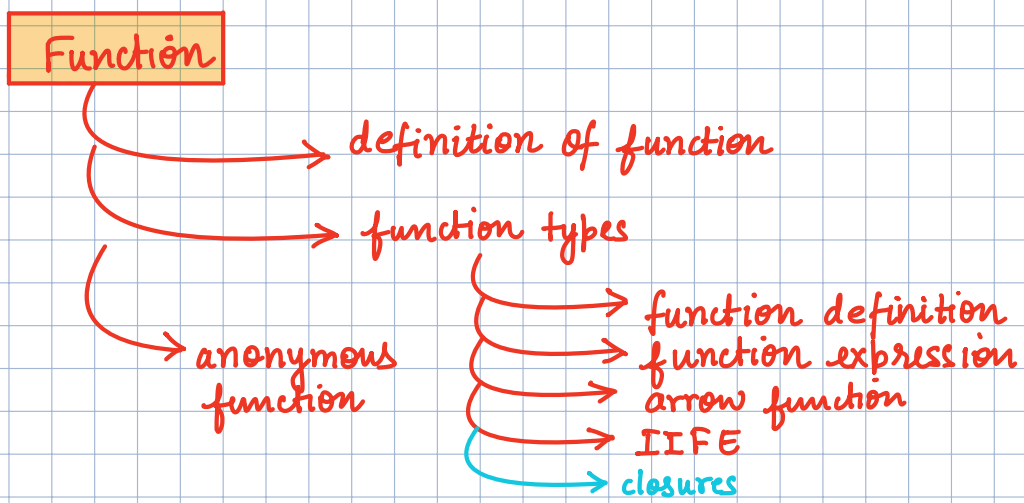
FUNCTION




```
// function definition -> code
function sayHello(name){
    console.log("Hello, ", name);
}
// function invocation -> this line actually runs that code
sayHello("Hemakshi");

// haven't called the function -> function name print
console.log(sayHello);
console.log("8", sayHello);
```

print the function

Before execution memory allocation takes place, Since the size of function cannot be decided before hand unlike string or integer, they are allocated in Heap.

SayHello = 8K

Stack

Heap

SayHello

8K

name = "Hemakshi"

SayHello

in every language we have tostring() function. In JS we can print the array in single line but not in Java.

```
→  Lecture_40 git:(main) ✗ node 1_intro_to_Functions.js
Hello,  Hemakshi
[Function: sayHello]
8 [Function: sayHello]
→  Lecture_40 git:(main) ✗
```

```
1
2   // function definition -> code
3   function sayHello(name){
4       console.log("Hello, ", name);
5   }
6   // function invocation -> this line actually runs that code
7   sayHello("Hemakshi");
8
9   // haven't called the function -> function name print
10  console.log(sayHello);
11  console.log("11", sayHello);
12
13  // print the function after calling it -> returned value
14
15  let rVal = sayHello("Janet");
16  console.log("16", rVal);
17
18  console.log("18", sayHello("Janet"));
19
20  // if you don't pass anything to the function -> parameter undefined
21  sayHello();
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_40 git:(main) ✗ node 1_intro_to_Functions.js
Hello,  Hemakshi
[Function: sayHello]
11 [Function: sayHello]
Hello,  Janet
16 undefined
Hello,  Janet
18 undefined
Hello,  undefined
→  Lecture_40 git:(main) ✗
```

```
23  // 2. With return value
24
25  function sayHello(name) {
26      console.log("Hello", name);
27      return "returned from a function"
28  }
29
30  let retVal = sayHello("Elena");
31  console.log(retVal);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_40 git:(main) ✗ node 1_intro_to_Functions.js
Hello Elena
returned from a function
→  Lecture_40 git:(main) ✗
```

HOISTING

Hoisting → Memory allocation → JS Function

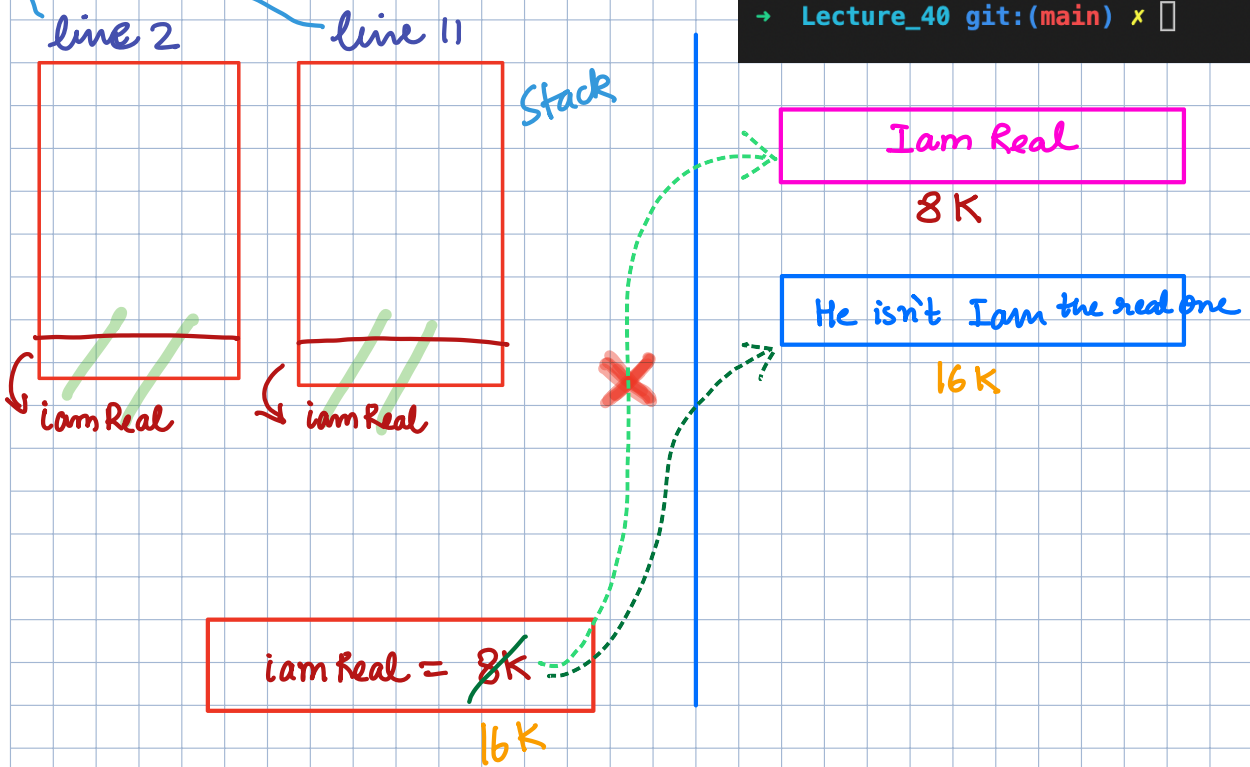
```
2   iamReal();
3   // 1
4   function iamReal() {
5       console.log("I am real");
6   }
7   // 2
8   function iamReal() {
9       console.log("He isn't -> I am the real one");
10  }
11  iamReal();
12
```

Javascript code runs in two steps

Step ① → Memory allocation for the function

Step ② → Code Execution

```
→  Lecture_40 git:(main) ✗ node 2_Hoisting.js
He isn't -> I am the real one
He isn't -> I am the real one
→  Lecture_40 git:(main) ✗
```



FUNCTIONS ARE VARIABLES

```
2   // fn definition
3   function fn(param) {
4       console.log("I am function definition", param);
5   }
6   // string -> value
7   fn("Hello");
8   // boolean -> value
9   fn(true);
10  //address
11  // object
12  let obj = { name : "Hemakshi" };
13  fn(obj);
14  // array
15  let arr = [10, 20, 30];
16  fn(arr);
17
18  // function are also variables -> functions are first class citizens in JS
19
20  function chotaFn() {
21      console.log("Hello i am a chota fn");
22  }
23
24  // function can also be passed as an argument in a function
25
26  fn(chotaFn);
27
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_40 git:(main) ✗ node 3_Types_of_Functions.js
I am function definition Hello
I am function definition true
I am function definition { name: 'Hemakshi' }
I am function definition [ 10, 20, 30 ]
I am function definition [Function: chotaFn]
→  Lecture_40 git:(main) ✗
```

```
2   // fn definition
3   function fn(param) {
4       console.log("I am function definition", param);
5       param();
6   }
7
8
9   // function are also variables -> functions are first class citizens in JS
10
11  function chotaFn() {
12      console.log("Hello i am a chota fn");
13  }
14
15  // function can also be passed as an argument in a function
16
17  fn(chotaFn);
18
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_40 git:(main) ✗ node 3_Types_of_Functions.js
I am function definition [Function: chotaFn]
Hello i am a chota fn
→  Lecture_40 git:(main) ✗
```

TYPES OF FUNCTIONS

```
// 1. function definition
function fn() {
    console.log("I am function definition");
}
fn();
// variable assignment
// let a = [10, 20, 30];
// let b = a;
// console.log(b);

// 2. function expression
let secondName = function originalName() {
    console.log("I am expression");
}
secondName();
originalName();

console.log("Before");

// 3. IIFE -> Immediately Invoked Function Expression
(function drawBoard(){
    console.log("board is immediately drawn");
})();

// 4. Anonymous function
(function () {
    console.log("board is immediately drawn");
})

console.log("After");

let secondName2 = function () {
    console.log("I am expression");
}
secondName2();
```

named function expression. One of the benefits of creating a named function expression is that in case we encountered an error. The stack trace will contain the name of the function making it easier to find the origin of the

runs as soon as the error browser encounters it. The benefit of this function is that it run immediately where its located in the code and produces a direct output.

don't have names. They need to be tied to something: variable or an event to run.

```
→  Lecture_40 git:(main) ✗ node 4_Types_of_functions.js
I am function definition
I am expression
Before
board is immediately drawn
After
I am expression
→  Lecture_40 git:(main) ✗
```