FUNCTION EXPRESSION

Find the output of the following code:-

(A)

```
1
2   fun();
3
4   var fun = function() {
5       gun();
6   }
7
8   var gun = function() {
9       console.log("I am inside gun");
10  }
11
12  // print or line of error and what the error is
13
14
```

→ Function Expression

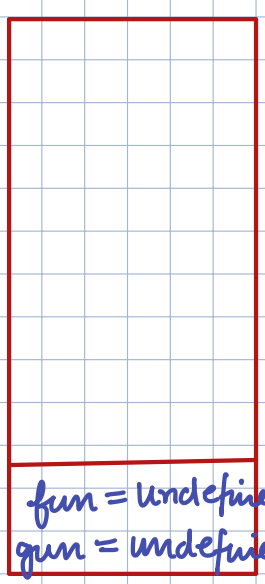
→ Function Expression

→ var is hoisted
→ fⁿ assignment is not hoisted

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE        zsh - Lecture_43

→ Lecture_43 git:(main) ✗ node 1_Ques1.js
/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/1_Ques1.js:2
fun();
^

TypeError: fun is not a function
    at Object.<anonymous> (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/1_Ques1.js:2:1)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47
→ Lecture_43 git:(main) ✗
```

① The memory is allocated for variable and function
② the code is executed.

① Stack



GlE·C → Global execution context

Here fun and gun variable will be alloted undefined set in G.E.C.

② So when the code is executed the first line of code that will run is fun().

But in GlE·C we have var fun = undefined, hence fun is not a function and as of now during memory allocation, fun variable does not contain address of function().

∴ fun() → this line gives an error that fun is not a fun.

var fun;
var gun;
→ fun();→ error
fun = { ——
——
——
}
gun = {
}

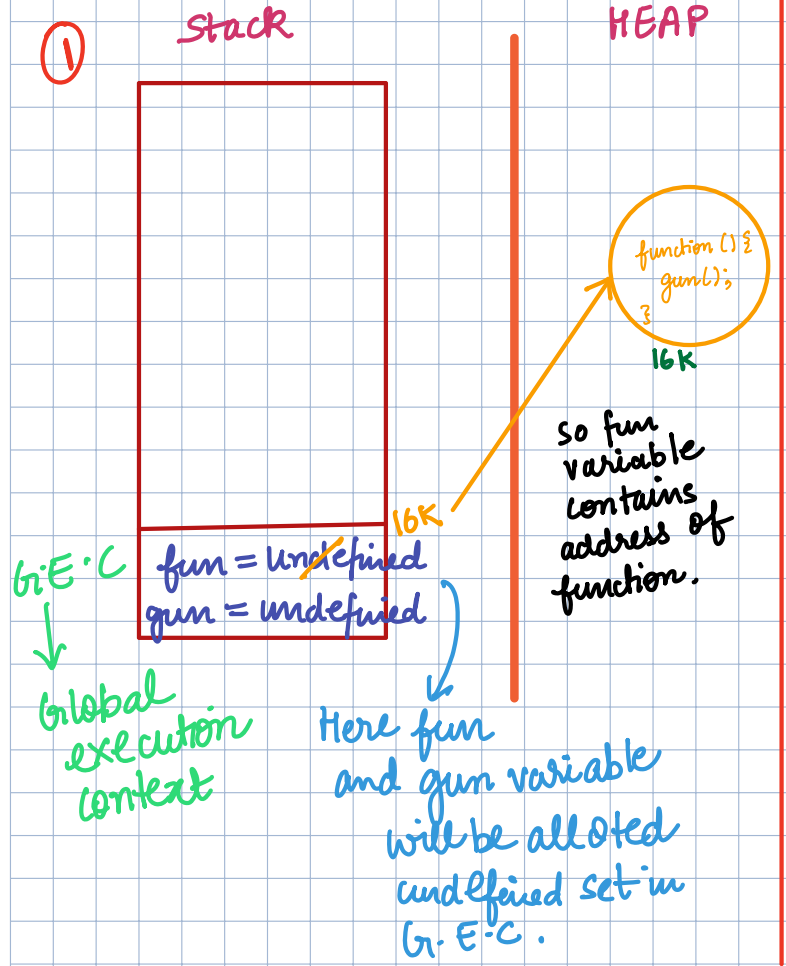
B

```
var fun = function() {
    gun();
}

fun();

var gun = function() {
    console.log("I am inside gun");
}

// print or line of error and what the error is
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                    zsh - Lecture_43

→  Lecture_43 git:(main) ✗ node 2_Ques1_part2.js
/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/2_Ques1_part2.js:3
    gun();
    ^

TypeError: gun is not a function
    at fun (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/2_Ques1_part2.js:3:5)
    at Object.<anonymous> (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/2_Ques1_part2.js:6:1
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47
→  Lecture_43 git:(main) ✗
```

(1) The memory is allocated for variable and function
(2) the code is executed.

(1)



(2) So, when fun() function will be called, inside fun, gun() function will be called.

But gun variable still contains undefined and is not alloted with the address of function yet.

Hence we will get error that gun is not a function.

```
var fun;
var gun;
fun = function {
    gun()
}
fun();
gun = function {

}
```

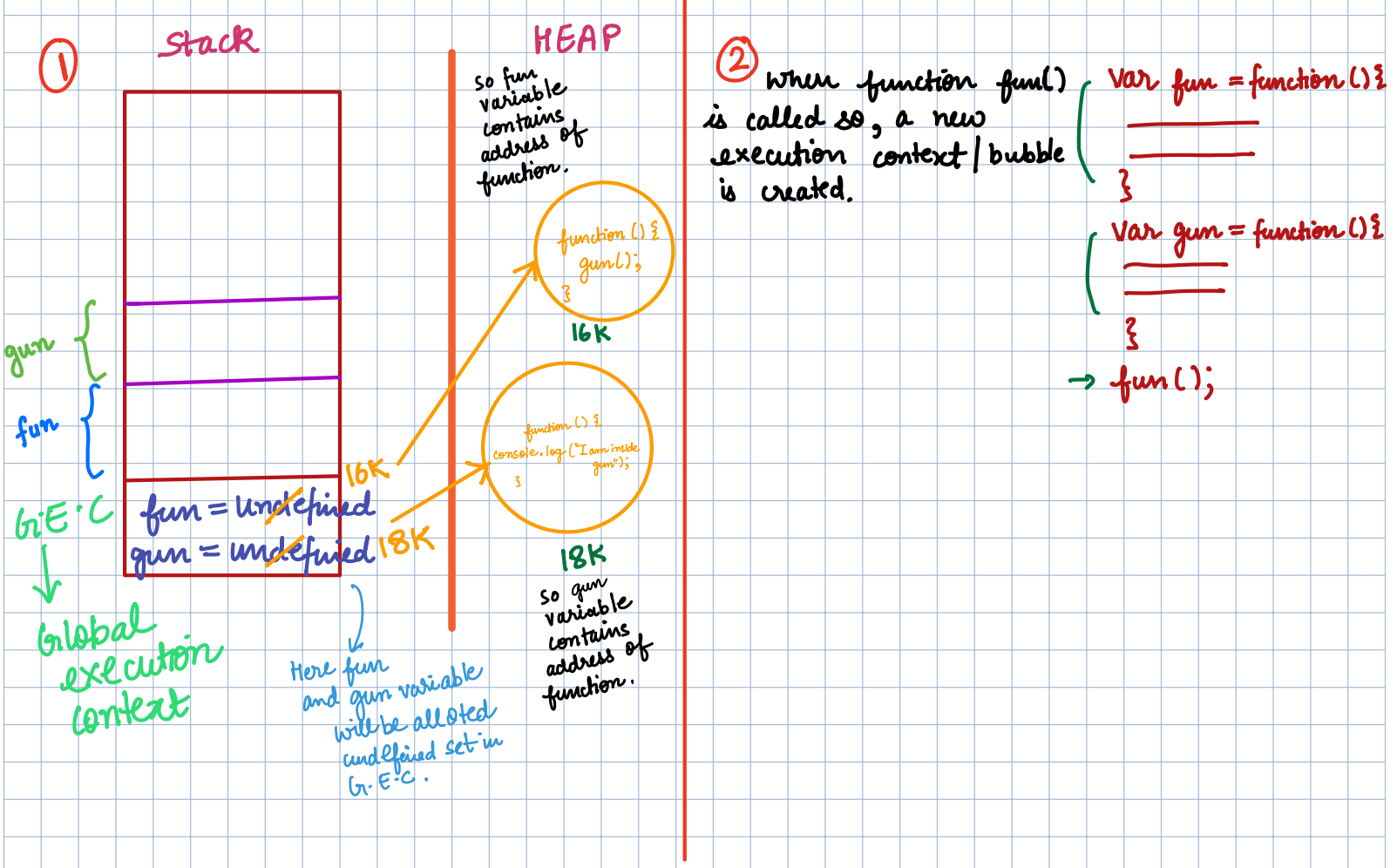
C

```
2   var fun = function() {
3       gun();
4   }
5
6   var gun = function() {
7       console.log("I am inside gun");
8   }
9
10  fun();
11
12  // print output or line of error and what the error is
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_43 git:(main) ✗ node 3_Ques1_part3.js
I am inside gun
→  Lecture_43 git:(main) ✗
```

① The memory is allocated for variable and function
② the code is executed.



D

```
2  fun();
3
4  function fun() {
5      gun();
6  }
7
8  function gun() {
9      console.log("I am inside gun");
10 }
11
12 // print output or line of error and what the error is
```

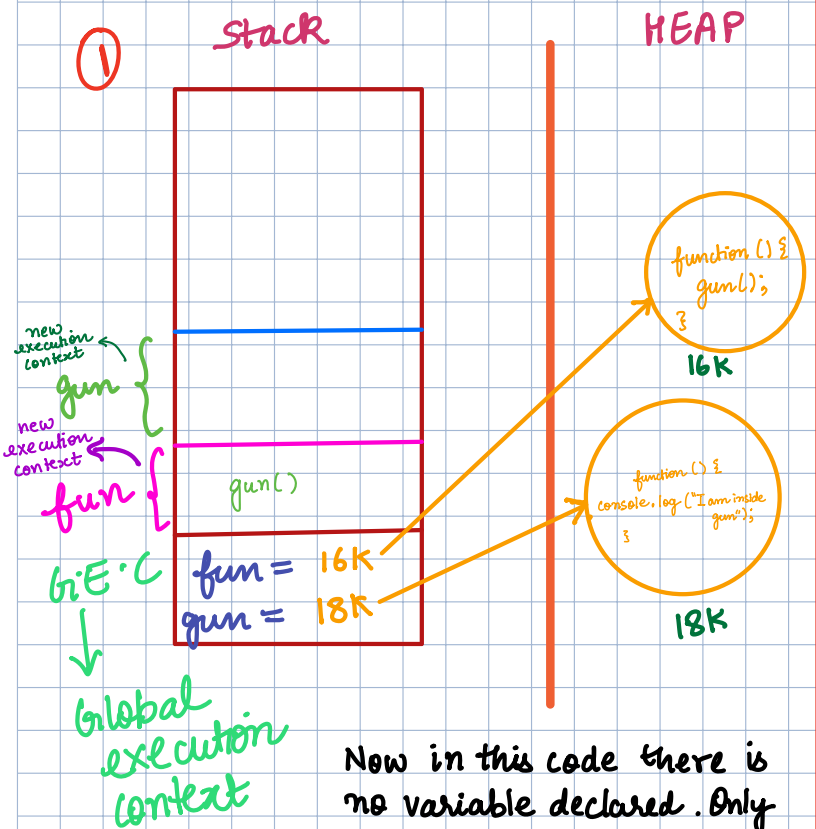
Function declaration

Both var and definition are hoisted

Function declaration

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

→  Lecture_43 git:(main) ✗ node 4_Ques1_part4.js
I am inside gun
→  Lecture_43 git:(main) ✗
```

① The memory is allocated for variable and function
② the code is executed.

① 

Now in this code there is no variable declared. Only functions are declared in this code
So, the memory for fun() and gun will be allocated in Heap.

② So now fun() function is called. Whenever a function is called, a new execution context is created.

```
var fun = function () {
    ____
    ____
}
```

Inside fun function, the gun() function is called. Now there will be a new execution context for gun.

```
var gun = function () {
    ____
    ____
}
→ fun();
```

Difference between Function Declaration and Function Expression:

	Function Declaration	Function Expression
1.	A function declaration must have a function name.	A function Expression is similar to a function declaration without the function name.
2.	Function declaration does not require a variable assignment.	Function expressions can be stored in a variable assignment.
3.	These are executed before any other code.	Function expressions load and execute only when the program interpreter reaches the line of code.
4.	The function in function declaration can be accessed before and after the function definition.	The function in function declaration can be accessed only after the function definition.
5.	Function declarations are hoisted	Function expressions are not hoisted

6.	**Syntax:**
	function geeksforGeeks(paramA, paramB) {
	// Set of statements
	}
		Syntax:
		var geeksforGeeks= function(paramA, paramB) {
		// Set of statements
		}

```
fun();

function fun() {
    gun();
}

var gun = function() {
    console.log("I am inside gun");
}

// print output or line of error and what the error is
```

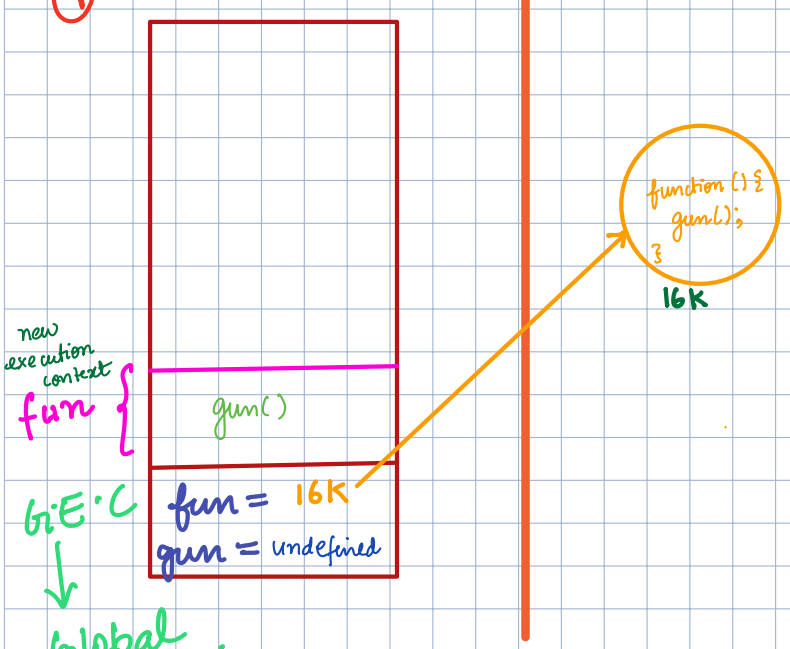
→ function declaration
∴ memory can be allocated

→ function expression
memory can't be allocated
variable gun will be declared but value will be set as undefined

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE        zsh - Lecture_43

→ Lecture_43 git:(main) ✗ node 5_Ques2_part1.js
/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/5_Ques2_part1.js:6
    gun();
    ^

TypeError: gun is not a function
    at fun (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/5_Ques2_part1.js:6:5)
    at Object.<anonymous> (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/5_Ques2_part1.js:3:1)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47
→ Lecture_43 git:(main) ✗
```

① The memory is allocated for variable and function.
② the code is executed.

① Stack



HEAP

② when fun() function is called a new execution context is created.

Now inside fun() function, the gun function is called. But gun is not a function yet. It is just a variable that contains undefined value.

So gun() will give error :- gun is not a function.

```
var fun = function(){
    gun(); error
}

var gun;
fun();
gun = function(){
}
```

F

```
1
2   function fun() {
3       gun();
4   }
5
6   fun();
7
8   var gun = function() {
9       console.log("I am inside gun");
10  }
11
12  // print output or line of error and what the error is
```

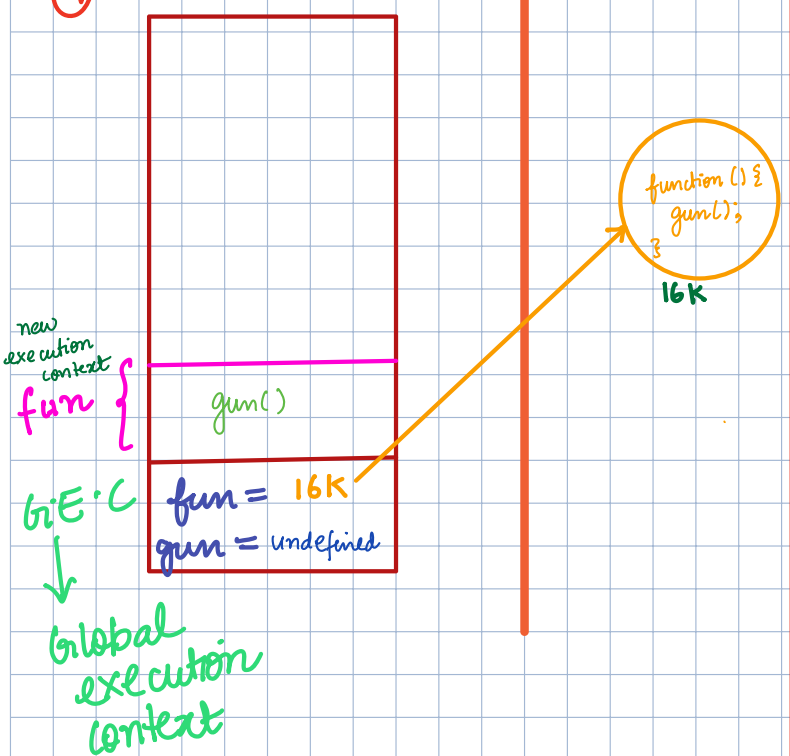
→ function declaration

→ function Expression

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE        zsh - Lecture_43

→  Lecture_43 git:(main) ✗ node 6_Ques2_part2.js
/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/6_Ques2_part2.js:3
    gun();
    ^

TypeError: gun is not a function
    at fun (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/6_Ques2_part2.js:3:5)
    at Object.<anonymous> (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/6_Ques2_part2.js:6:1)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47
→  Lecture_43 git:(main) ✗
```

① The memory is allocated for variable and function..
② the code is executed.

① Stack



② when fun() function is called a new execution context is created.

```
var fun = function(){
    gun(); error
}
var gun;
fun();
gun = function(){
}
```

Now inside fun() function, the gun function is called. But gun is not a function yet. It is just a variable that contains undefined value.

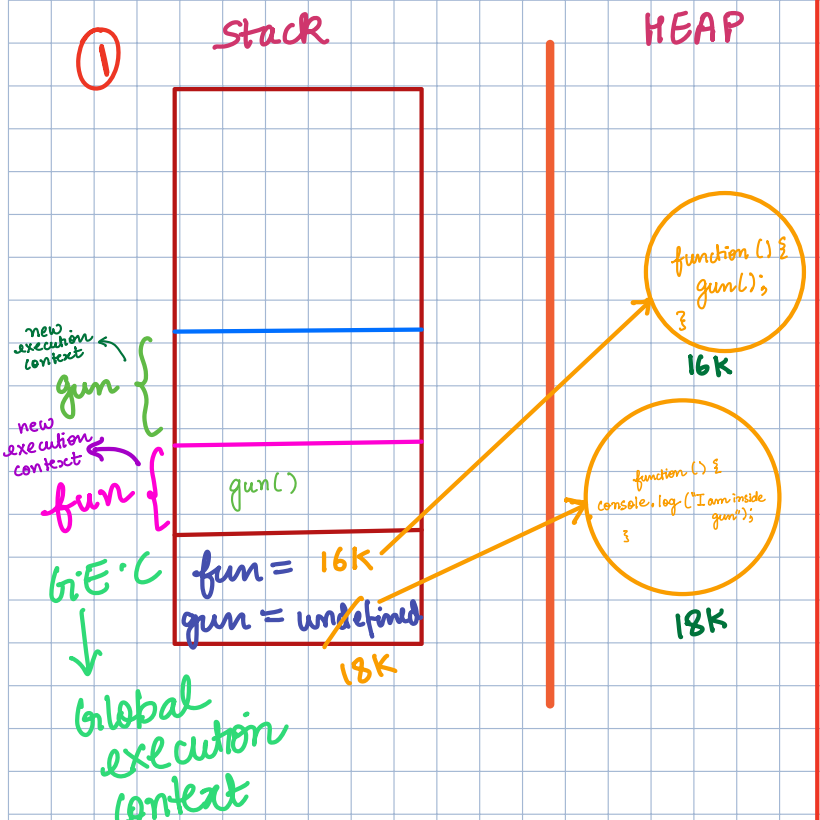
So gun() will give error :- gun is not a function.

```
3   function fun() {
4       gun();
5   }
6
7   var gun = function() {
8       console.log("I am inside gun");
9   }
10
11  fun();
12
13  // print output or line of error and what the error is
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_43 git:(main) ✗ node 7_Ques2_part3.js
I am inside gun
→  Lecture_43 git:(main) ✗
```

① The memory is allocated for variable and function
② the code is executed.



So, the memory for fun() will be allocated in Heap. The variable gun will be undefined set.

②

Now variable gun function will have the address function.

So now fun() function is called. Whenever a function is called, a new execution context is created.

Inside fun function, the gun() function is called. Now there will be a new execution context for gun.

```
var fun = function () {
}
var gun;
gun = function () {
}
→ fun();
```

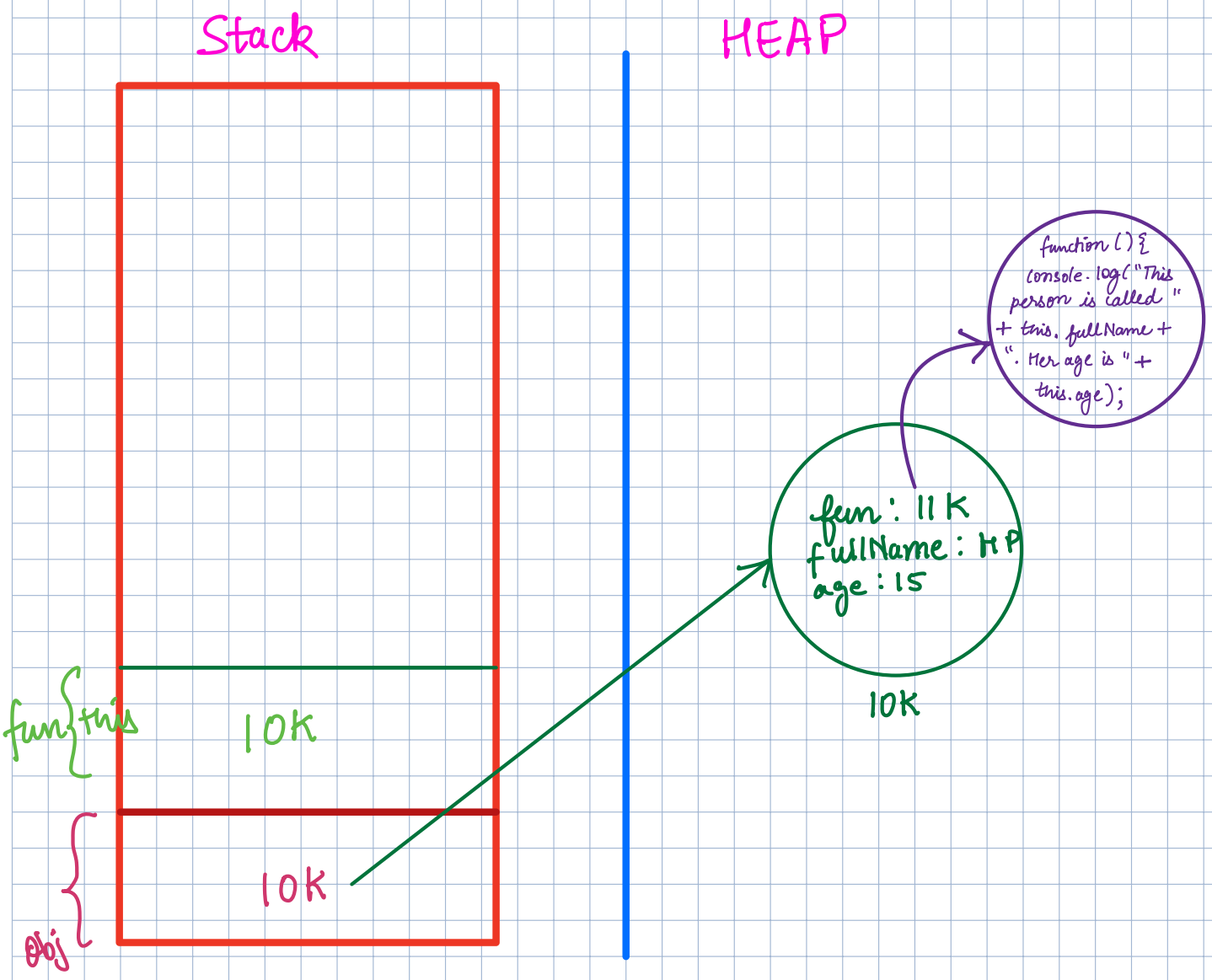
FUNCTION ON OBJECTS

```
2
3   let obj = {
4       fun: function(){
5           console.log("This person is called " + this.fullName + ". Her age is " + this.age);
6       },
7       fullName: "Hemakshi Pandey",
8       age: 15
9   };
10
11  obj.fun();
```

fun() function is called

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

→  Lecture_43 git:(main) ✗ node 8_Function_On_Object.js
This person is called Hemakshi Pandey. Her age is 15
→  Lecture_43 git:(main) ✗
```

output

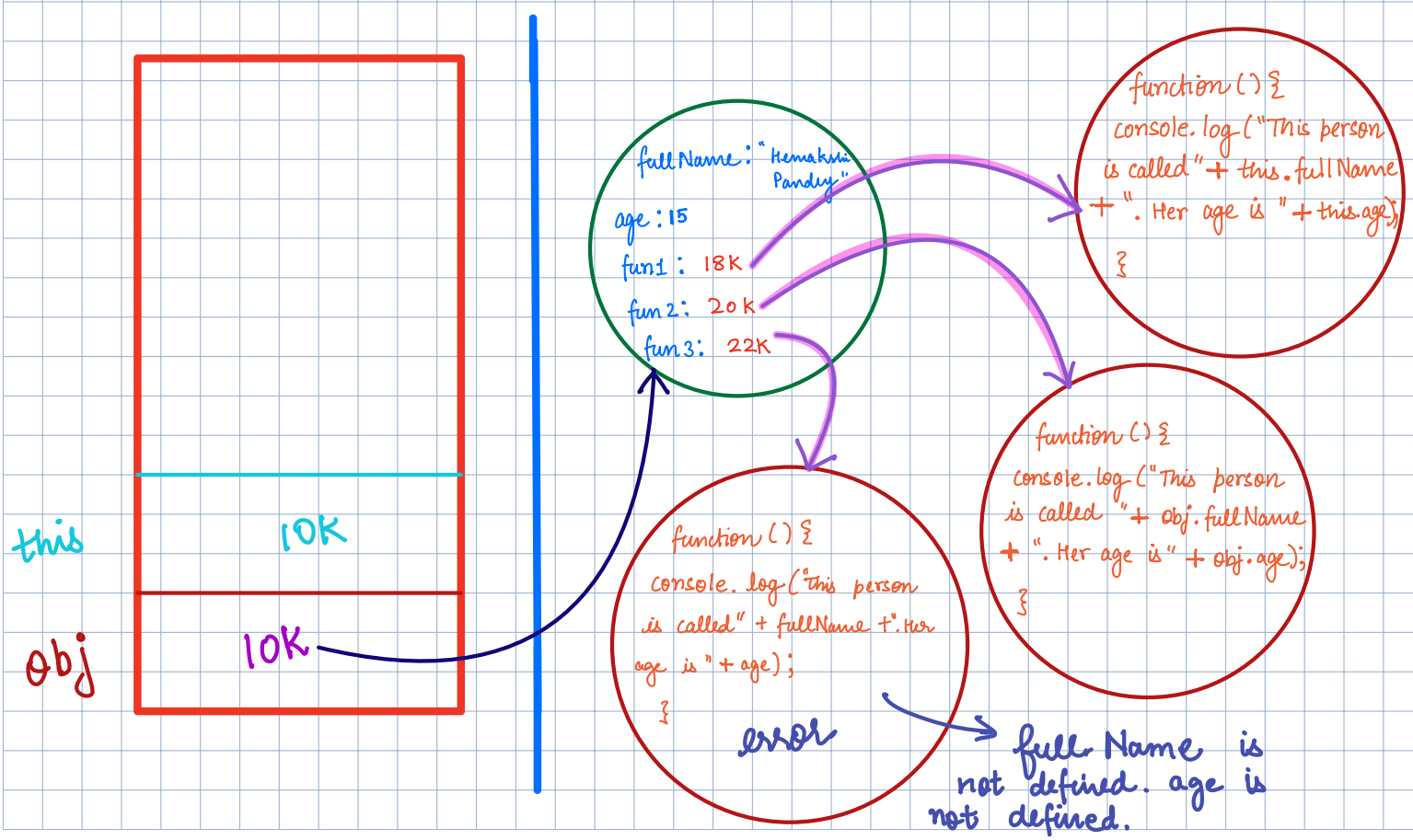


```js
let obj = {
    fun1: function(){
        console.log("This person is called " + this.fullName + ". Her age is " + this.age);
    },
    fun2: function(){
        console.log("This person is called " + obj.fullName + ". Her age is " + obj.age);
    },
    fun3: function(){
        console.log("This person is called " + fullName + ". Her age is " + age);
    },
    fullName: "Hemakshi Pandey",
    age: 15
};

obj.fun1(); // this is same in java and js
obj.fun2(); // this works in js but not in java
obj.fun3(); // this works in java but not in JS
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE — zsh - Lecture_43

```
→  Lecture_43 git:(main) ✗ node 8_Function_On_Object.js
This person is called Hemakshi Pandey. Her age is 15
This person is called Hemakshi Pandey. Her age is 15
/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/8_Function_On_Object.js:22
        console.log("This person is called " + fullName + ". Her age is " + age);
                                               ^

ReferenceError: fullName is not defined
    at Object.fun3 (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/8_Function_On_Object.js:22:48)
    at Object.<anonymous> (/Users/hemakshipandey/Desktop/Pepcoding-FJP1-Development/Javascript/Lecture_43/8_Function_On_Object.js:30
:5)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:12)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47
→  Lecture_43 git:(main) ✗
```

① The memory is allocated to Object in Heap.
② The memory is allocated to function during runtime.



FUNCTION CALLING

```
2   function fun(a, b){
3       console.log(a + " " + b);
4   }
5
6   fun();
7   fun(10);
8   fun(10, 20);
9   fun(10, 20, 30);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_43 git:(main) ✗ node 9_Function_Calling.js
undefined undefined
10 undefined
10 20
10 20
→  Lecture_43 git:(main) ✗
```

Parameter Rules

- JavaScript function do not specify data types for parameters.
- JavaScript functions do not perform type checking on the passed argument.
- JavaScript functions do not check the number of arguments received.

Default Parameters

- If a function is called with missing arguments (less than declared), the missing values are set to undefined.

```
1
2   function fun(a, b){
3       console.log(a + " " + b);
4       console.log(arguments);
5   }
6
7   fun();
8   fun(10);
9   fun(10, 20);
10  fun(10, 20, 30);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
→  Lecture_43 git:(main) ✗ node 9_Function_Calling.js
undefined undefined
[Arguments] {}
10 undefined
[Arguments] { '0': 10 }
10 20
[Arguments] { '0': 10, '1': 20 }
10 20
[Arguments] { '0': 10, '1': 20, '2': 30 }
→  Lecture_43 git:(main) ✗
```

THE ARGUMENTS OBJECT

If a function is called with too many arguments (more than declared), these arguments can be reached using the arguments object.

```
2
3   function fun(a, b){
4       console.log(a + " " + b);
5       console.log(arguments);
6       let res = Array.from(arguments);
7       let sq = res.map(x => x * x);
8       console.log(sq);
9   }
10
11  // fun();
12  // fun(10);
13  // fun(10, 20);
14  fun(10, 20, 30);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

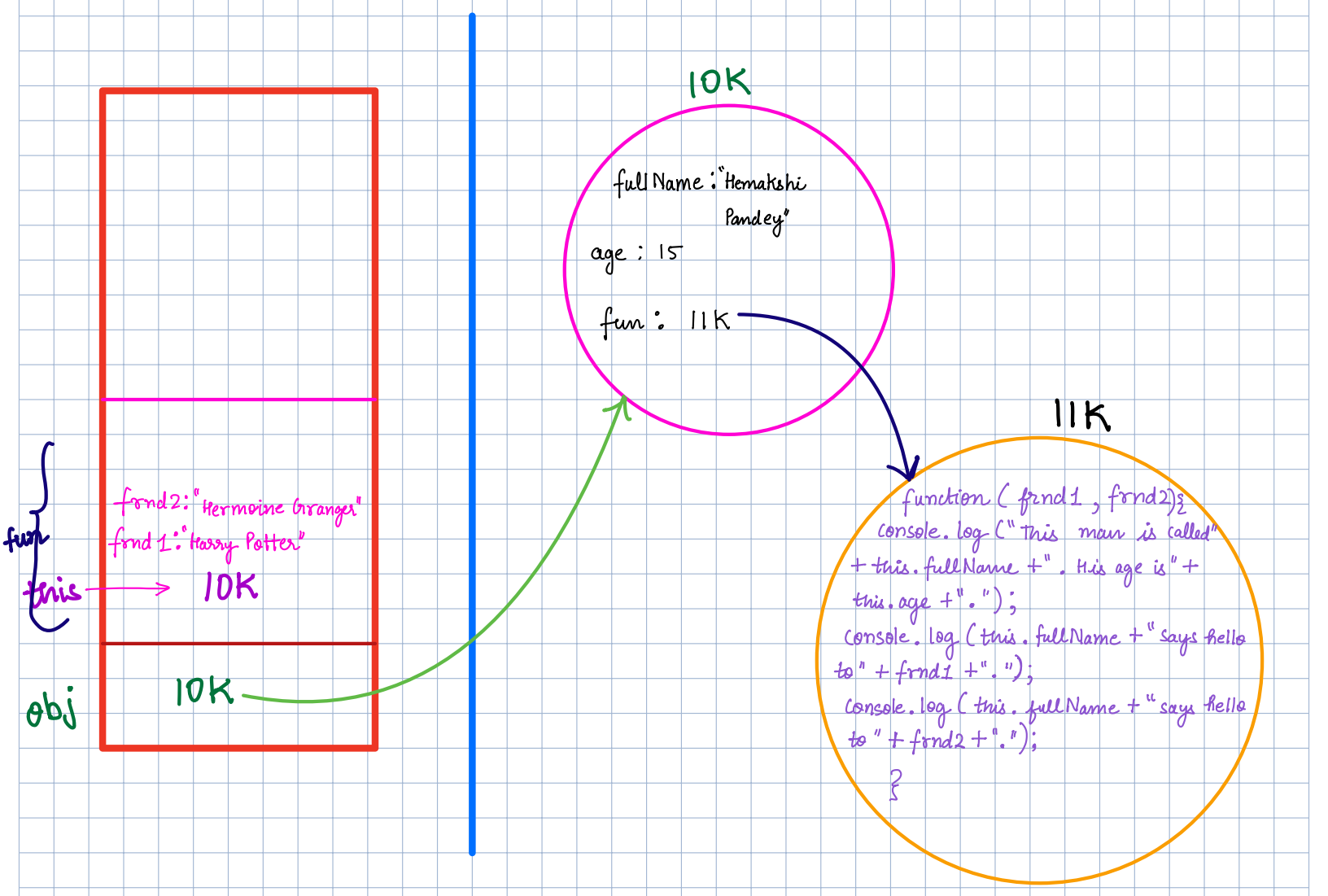
```
➜  Lecture_43 git:(main) ✗ node 10_Function_Calling2.js
10 20
[Arguments] { '0': 10, '1': 20, '2': 30 }
[ 100, 400, 900 ]
➜  Lecture_43 git:(main) ✗
```

WAYS OF CALLING

```
2   let obj = {
3       fun: function(frnd1, frnd2){
4           console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
5           console.log(this.fullName + " says hello to " + frnd1 + ".");
6           console.log(this.fullName + " says hello to " + frnd2 + ".");
7       },
8       fullName: "Hemakshi Pandey",
9       age: 15
10  };
11
12  obj.fun("Harry Potter", "Hermoine Granger");
13
```

Normal way of calling a function.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                    zsh

→  Lecture_43 git:(main) ✗ node 11_Ways_Of_Calling.js
This person is called Hemakshi Pandey. Her age is 15.
Hemakshi Pandey says hello to Harry Potter.
Hemakshi Pandey says hello to Hermoine Granger.
→  Lecture_43 git:(main) ✗
```



(A) THE JAVASCRIPT CALL() METHOD

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

```
let obj = {
    fun: function(frnd1, frnd2){
        console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
        console.log(this.fullName + " says hello to " + frnd1 + ".");
        console.log(this.fullName + " says hello to " + frnd2 + ".");
    },
    fullName: "Hemakshi Pandey",
    age: 15
};

// Normal Way
// obj.fun("Harry Potter", "Hermoine Granger");

// Using call() function
obj.fun.call({
    fullName: "Ginny Weasley",
    age: 14
}, "Fred", "George");

// call is a function. It is available on all functions. It can be used to call the functions.
// The use case is, if you want to override the default this.
```

→ The call() method is a predefined Java script method. It can be used to invoke (call) a method with an owner object as an argument (parameter).

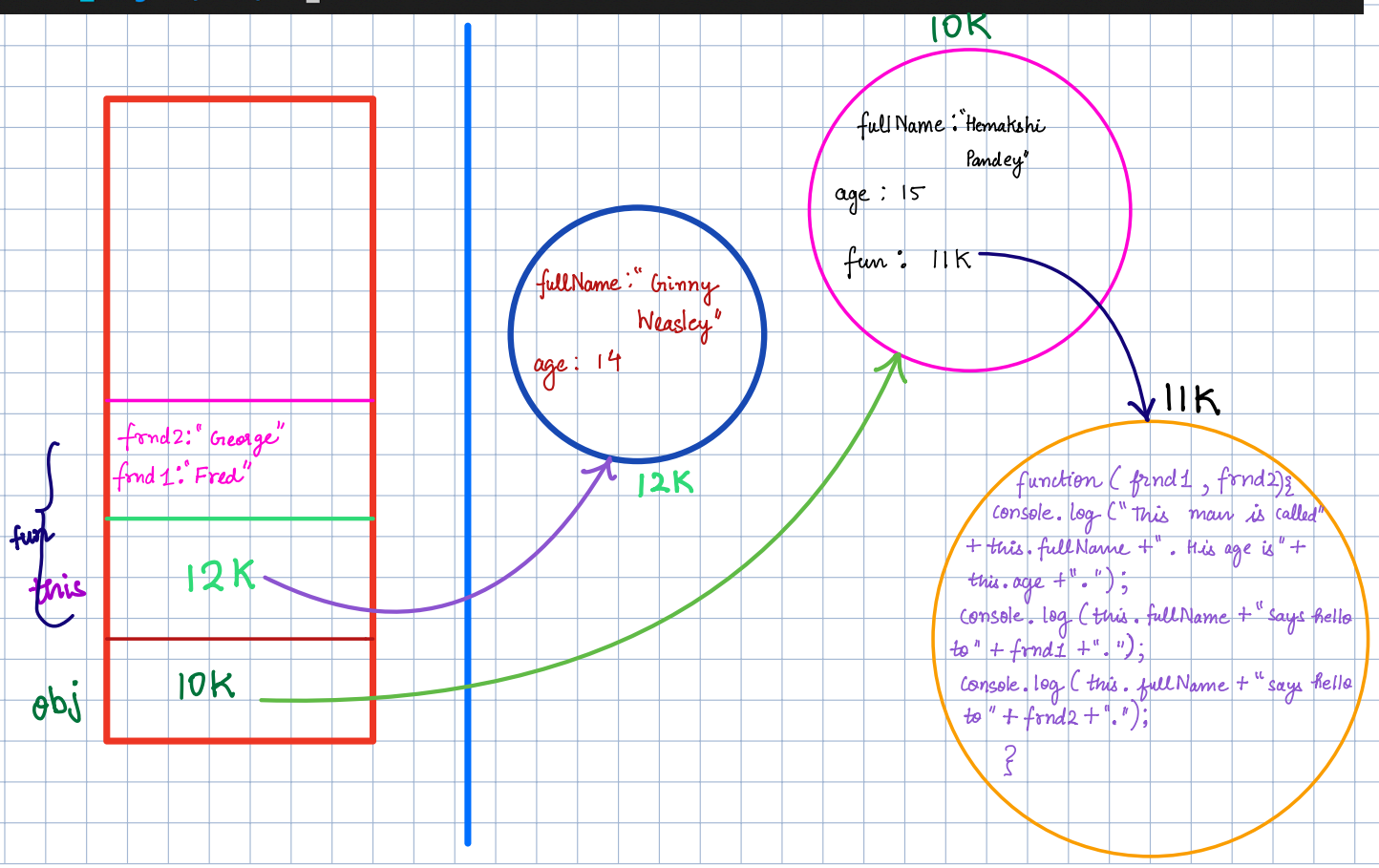
→ With call(), an object can use a method belong to another object

Object

→ The call() method calls a function with a given 'this' value and arguments provided individually.

→ call() provides a new value of this to the function / method. With call(), you can write a method once and then inherit it in another object, without having to rewrite the method for the new object.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                zsh -

→  Lecture_43 git:(main) ✗ node 11_Ways_Of_Calling.js
This person is called Ginny Weasley. Her age is 14.
Ginny Weasley says hello to Fred.
Ginny Weasley says hello to George.
→  Lecture_43 git:(main) ✗
```



(B) THE JAVASCRIPT APPLY() METHOD

```
let obj = {
    fun: function(frnd1, frnd2){
        console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
        console.log(this.fullName + " says hello to " + frnd1 + ".");
        console.log(this.fullName + " says hello to " + frnd2 + ".");

        console.log(arguments);
    },
    fullName: "Hemakshi Pandey",
    age: 15
};

// 3. Using apply method

let o3 = {
    fullName: "Luna Lovegood",
    age: 14
}

obj.fun.apply(o3, ["Dean Thomas", "Severus Snape", "Albus Dumbledore"]);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE zsh -

```
→  Lecture_43 git:(main) ✗ node 11_Ways_Of_Calling.js
This person is called Luna Lovegood. Her age is 14.
Luna Lovegood says hello to Dean Thomas.
Luna Lovegood says hello to Severus Snape.
[Arguments] {
  '0': 'Dean Thomas',
  '1': 'Severus Snape',
  '2': 'Albus Dumbledore'
}
→  Lecture_43 git:(main) ✗
```

The Difference Between call() and apply()

The difference is:—
The call() method takes arguments separately.
The apply() method takes arguments as an array.

The apply() method is very handy if you want to use an array instead of an argument list.

© THE JAVASCRIPT BIND() METHOD

```js
let obj = {
    fun: function(frnd1, frnd2){
        console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
        console.log(this.fullName + " says hello to " + frnd1 + ".");
        console.log(this.fullName + " says hello to " + frnd2 + ".");

        console.log(arguments);
    },
    fullName: "Hemakshi Pandey",
    age: 15
};

// 4. Using Bind Method
let o4 = {
    fullName: "Lily Evans",
    age: 16
}
let boundFunction = obj.fun.bind(o4, "James Potter", "Sirius Black", "Remus Lupin");
boundFunction();
boundFunction("Harry Potter");
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                    zsh -

→  Lecture_43 git:(main) ✗ node 11_Ways_Of_Calling.js
This person is called Lily Evans. Her age is 16.
Lily Evans says hello to James Potter.
Lily Evans says hello to Sirius Black.
[Arguments] {
  '0': 'James Potter',
  '1': 'Sirius Black',
  '2': 'Remus Lupin'
}
This person is called Lily Evans. Her age is 16.
Lily Evans says hello to James Potter.
Lily Evans says hello to Sirius Black.
[Arguments] {
  '0': 'James Potter',
  '1': 'Sirius Black',
  '2': 'Remus Lupin',
  '3': 'Harry Potter'
}
```

The bind() method creates a new function that, when called has its "this" keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

The bind function creates a new bound function, which is an exotic function object that wrap the original function object. Calling the bound function generally results in the execution of its wrapped function.

CUSTOM BIND METHOD

```
Function.prototype.myBind = function(){
    let orgFun = this;
    let args = Array.from(arguments);

    let boundFun = function(){
        let thisForOrgFun = args[0];
        let argsForOrgFun = args.slice(1);
        let argsFromInvocation = Array.from(arguments);

        argsForOrgFun = argsForOrgFun.concat(argsFromInvocation);

        orgFun.apply(thisForOrgFun, argsForOrgFun);
    }
    return boundFun;
}

let obj = {
    fun: function(frnd1, frnd2){
        console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
        console.log(this.fullName + " says hello to " + frnd1 + ".");
        console.log(this.fullName + " says hello to " + frnd2 + ".");
        console.log(arguments);
    },
    fullName: "Hemakshi Pandey",
    age: 15
};

// 4. Using myBind or custom bind
let o4 = {
    fullName: "Lily Evans",
    age: 16
}
let boundFunction = obj.fun.myBind(o4, "James Potter", "Sirius Black", "Remus Lupin");
boundFunction();
boundFunction("Harry Potter");

// Bind is not similar to call function. It doesn't make any call. It gives you a new function to call.
```

CUSTOM CALL METHOD

```js
Function.prototype.myCall = function(){
    let orgFun = this;
    let args = Array.from(arguments);

    let thisForCall = args[0];
    let params = args.slice(1);

    orgFun.apply(thisForCall, params);
}

let obj = {
    fun: function(frnd1, frnd2){
        console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
        console.log(this.fullName + " says hello to " + frnd1 + ".");
        console.log(this.fullName + " says hello to " + frnd2 + ".");
        console.log(arguments);
    },
    fullName: "Hemakshi Pandey",
    age: 15
};

// 2. Using call() method
obj.fun.myCall({
    fullName: "Ginny Weasley",
    age: 14
}, "Fred", "George", "percy");

// call is a function. It is available on all functions. It can be used to call the functions.
// The use case is, if you want to override the default this.

```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                zsh - L

→  Lecture_43 git:(main) ✗ node 14_Custom_call.js
This person is called Ginny Weasley. Her age is 14.
Ginny Weasley says hello to Fred.
Ginny Weasley says hello to George.
[Arguments] { '0': 'Fred', '1': 'George', '2': 'percy' }
→  Lecture_43 git:(main) ✗
```

CUSTOM CALL METHOD (without Apply)

```
11  Function.prototype.myCall2 = function(){
12      let orgFun = this;
13      let args = Array.from(arguments);
14      let thisForCall = args[0];
15      let params = args.slice(1);
16
17      thisForCall.fun = orgFun;
18      thisForCall.fun(...params);
19      delete thisForCall.fun;
20  }
21
22  let obj = {
23      fun: function(frnd1, frnd2){
24          console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
25          console.log(this.fullName + " says hello to " + frnd1 + ".");
26          console.log(this.fullName + " says hello to " + frnd2 + ".");
27          console.log(arguments);
28      },
29      fullName: "Hemakshi Pandey",
30      age: 15
31  };
32
33  // 2. Using call() method
34  obj.fun.myCall2({
35      fullName: "Ginny Weasley",
36      age: 14
37  }, "Fred", "George", "percy");
38
39  // call is a function. It is available on all functions. It can be used to call the functions.
40  // The use case is, if you want to override the default this.
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                zsh -

→  Lecture_43 git:(main) ✗ node 14_Custom_call.js
This person is called Ginny Weasley. Her age is 14.
Ginny Weasley says hello to Fred.
Ginny Weasley says hello to George.
[Arguments] { '0': 'Fred', '1': 'George', '2': 'percy' }
→  Lecture_43 git:(main) ✗
```

CUSTOM APPLY METHOD

```js
Function.prototype.myApply = function(){
    let orgFun = this;
    let args = Array.from(arguments);
    let thisForApply = args[0];
    let params = args[1];

    thisForApply.fun = orgFun;
    thisForApply.fun(...params);
    delete thisForApply.fun;
}
let obj = {
    fun: function(frnd1, frnd2){
        console.log("This person is called " + this.fullName + ". Her age is " + this.age + ".");
        console.log(this.fullName + " says hello to " + frnd1 + ".");
        console.log(this.fullName + " says hello to " + frnd2 + ".");
        console.log(arguments);
    },
    fullName: "Hemakshi Pandey",
    age: 15
};
// 3. Using apply method
let o3 = {
    fullName: "Luna Lovegood",
    age: 14
}
obj.fun.myApply(o3, ["Dean Thomas", "Severus Snape", "Albus Dumbledore"]);
// apply is similar to call method. It just uses an array to pass arguments.
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE zsh -

```
→  Lecture_43 git:(main) ✗ node 15_Custom_Apply.js
This person is called Luna Lovegood. Her age is 14.
Luna Lovegood says hello to Dean Thomas.
Luna Lovegood says hello to Severus Snape.
[Arguments] {
  '0': 'Dean Thomas',
  '1': 'Severus Snape',
  '2': 'Albus Dumbledore'
}
→  Lecture_43 git:(main) ✗
```