

# Functions are first class citizens in JS

① We can assign functions to variables

```
var sayHello = function() { }
```

② We can pass functions as arguments

```
function sayHello(message) {
```

```
  message()
```

```
}
```

↳ accepting function as an argument

```
sayHello(function() { console.log("Hello") })
```

passing function as an argument

③ Return functions from another function

```
function sayHello() {
```

```
  return function message() { console.log("Hey") }
```

```
}
```

```
sayHello()();
```

① Has the return value of sayHello

② The returned value is a function and we want to call that function that is returned.

We're able to achieve all this because, well functions are just objects right? And we can pass objects as arguments, return them etc

@codeWithSimran

# Higher order functions @codeWithSimran

A function that takes a function as an argument, returns a function, or both

Let's say we want to multiply 2 numbers, but before that we want to check if they're numbers

function multiply(num1, num2)

```
{ if(num1 && num2 && typeof num1 === 'number'
    && typeof num2 === 'number')
{
    return actualMultiplication(num1, num2);
}
}
```

@codeWithSimran

So basically instead of doing everything in multiply we first check if arguments are valid and then call the multiplication method.

Now let's say we decide that we want to add both the numbers, will we write another function doing the same check? Can we pass add or multiply as arguments? [assume add & multiply exist]

function operation(num1, num2, performOperation)

```
{ if(checkIfValid(num1, num2)) {
    performOperation(num1, num2)
}
```

operation(5, 2, multiply) // we can now pass a func as argument to decide what we want to do at runtime  
operation(5, 2, add)



# CLOSURES

@codeWithSimran

Closures allow functions to access variables from the enclosing scope even after it leaves the scope in which it was declared.

Confusing right?



function first() {

let breakfast = 'breakfast'

return function second() {

let lunch = 'lunch'

        let random = 'xyz'

return function third() {

let dinner = 'dinner'

            return `\${breakfast} → \${lunch} →  
  \${dinner}`

        }

    }

}

first()()();

@codeWithSimran

→ breakfast → lunch → dinner

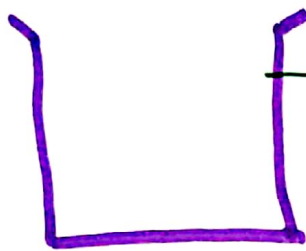
- ① When we called first() → it returned second  
→ first got pushed on the call stack and when it returned the result, it got popped off the stack
- ② We invoked the result of first() [which is a function] by first()() so second got invoked  
→ Second was pushed on the call stack and returned function three. So second is popped off the stack

③ We finally invoked function three by `first()()()`

Function third printed breakfast, lunch and dinner  
But wait!! How did function third get access to breakfast and lunch when the functions owning them were popped off the stack?

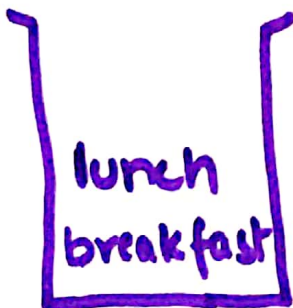
Well, the magic lies in closures @codeWithSimran

Closures is actually a feature by Javascript



\* Let's say we have a closure box somewhere, when JE sees that breakfast is accessed somewhere down the line, it puts breakfast into the closure box

- \* Even though first is done executing, it keeps variable breakfast in the closure box
- \* Similarly if sees lunch is being accessed later, and puts it in closure box
- \* Function third will now get access to breakfast and lunch from closure box



Well, we also declared a variable called random inside second.

Why is it not in the closure box? Because it's not being referenced

later and can be cleaned up the garbage collector :)

@codeWithSimran



So what enables this features?

@codeWithSimran

- ① Functions are first class citizens  
function can be returned from another function
- ② Lexical scope  
what variables we have access to depends on  
where the function was declared.

So using the concept of higher order functions and scope chaining, we can enable closures

Note

- ① first and second are both higher order functions as they both return functions.
- ② Before we actually execute the code, the JE knows what variables your code has access to

@codeWithSimran