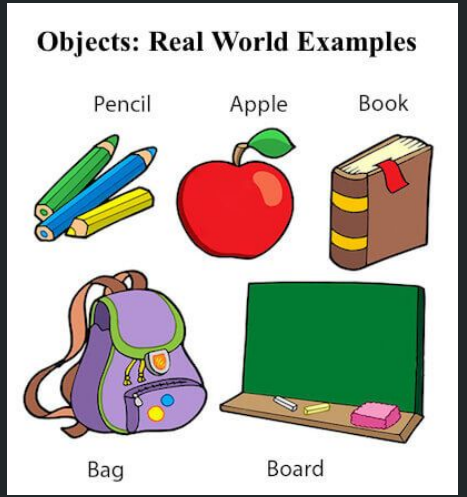
# Advanced Javascript

Day 3- JS Constructors, Call, Apply, Bind

# Creating Objects



There are different ways to create objects in Javascript.

1. var a = {} *//object literal*

2. var a= Object.create(b) *//Object.create() method*

But, there is one problem with creating objects through previous methods.

We need a better way to create multiple objects of same type!

# JS Constructors

# What are Constructors?

*Construct- To build something new or To create something from scratch*

The constructor method in JavaScript is a special function used to create and initialize objects.

# Js Constructor functions

The second method of creating a JavaScript object is using a constructor function. As opposed to object literals, here, you define an object type without any specific values.

We have to provide a blueprint of what properties an object should have.

To do that, we use functions.

```
function Player(name, team) {
this.name=name
this.team=team
}
var obj = new Player ( “Cristiano ronaldo” , “Portugal”)
```

But...

So, how did we move from simple functions that return values to creating objects using them...

It's all because of 'this'.

# Js Constructor functions

Important things to remember

- ‘new’ keyword is mandatory
- You can also pass function as argument values
- Best used for creating multiple objects of same type

We learned about creating custom Objects using Js functions. But..

We still don’t know much about behaviour of ‘this’ in functions

and we will be using functions a lot.

The value of *this* is determined by how the function was executed:

- In a method (function written inside an object), *this* refers to the owner object.
- Alone, *this* refers to the global object.
- In a function, *this* refers to the global object.

What if we could control where ‘this’ belongs?

# Call, Apply, Bind

All JavaScript functions has access to some very special methods which we can use to control where ‘*this*’ should refer.

Such methods are *call(), bind() & apply()*.

# Call()

The call() allows for a function/method belonging to one object to be assigned and called for a different object.

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.

# Apply()

The apply() method is literally the same as call() method.

They just both take arguments differently.

**Call() takes individual arguments seperated by comma.**

*printBio.call(obj, 20, ‘Pune’)*

**apply() takes a single array of arguments.**

*printBio.apply(obj, [ 20, ‘Mumbai’])*

Practical difference between Call and Apply.

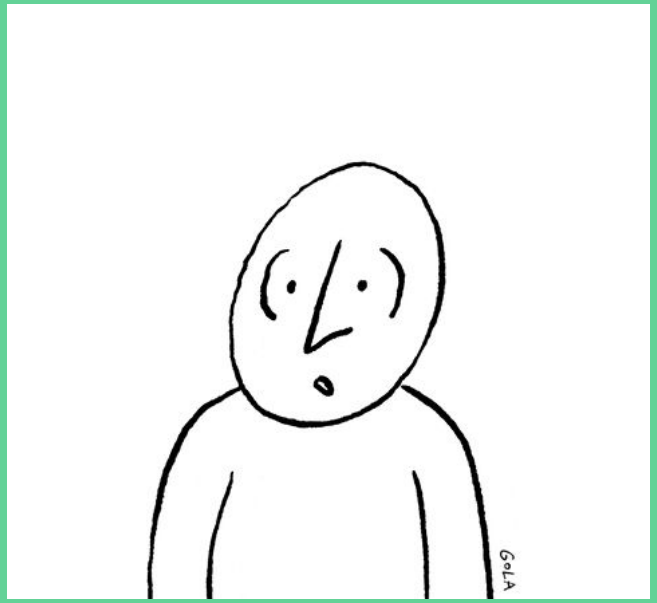
# Bind()

You can bind a particular object as ‘this’ to a function and use it later.

You cannot use call()  or apply() later, they run immediately.

# Let’s Recap

this -



# Function Constructors

# Call, Apply

# Bind