OOPS

class → blueprint

In IS, we have a class keyword
→ Inside the class →
(1) properties → data members
(2) Behaviours → member function

In case of a product a product can have properties as name, desc, price, rating
and it can have behaviour as update, buy, display

To define member func^n, we can just write normal func^n in the class.
But we need a very special Function as well which doesn't qualify as member function.

Constructor ← new objects can be created by calling the func^n.

Ex

```
Class product {
    Constructor() {
    }
}
```

/**
Constructor func^n to create new real life instances called as object
When we create an object this constructor is the first func^n that gets called **/

```
// member function
displayProduct() {
}
buyProduct() {
}
}
```

/**
In case of member functions function keyword is not necessary inside class
**/

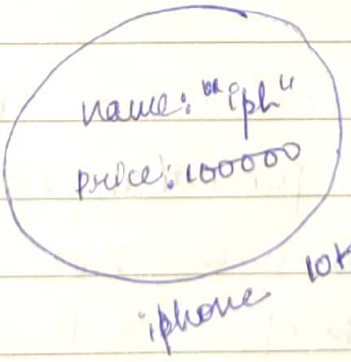
To define data members, we need to initialise inside the constructor function. We use the this keyword to do it.

Ex
```
Constructor(n,p){
    this.name = n;
    this.price = p;
}
```

How do we create real life objects?

```
let iphone = new Product("iph", 100000)
```

Note:- The above line cannot be written before the class is defined.



* We can though declare the datamembers outside the constructor without any let or const but by coding convention we do it inside constructor itself.

* If we need to access Data member inside member function this keyword comes handy.

* A class in JS can only have one constructor.

this — super important

Behaviour of this in JS is different than C++ or Java

⟹ In JS, this keyword references to the calling site i.e from where the function or class is called.

new

It looks like, new keyword only calls constructor, but it does a lot more than that

→ new keyword is not only associated with classes It can be used with function

W.r.t func^n what does new keyword do

```
function product(n, p){
    this.name = n;
    this.price = p;
    return this;
}
```

const p1 = new product("iphone", 100000);

The new keyword executes a 4 step process

(1) The new keyword creates a brand new empty JS object

(2) It does process of Linking (later) → it gets linked to its prototype object

(3) calls the func with this property assigned to the empty object it created before.

(4) the func execution starts (And we know that this keyword belongs to calling site)

{} {name: iphone}

this this

At last, if the func doesn't return any specific object, then it automatically returns this otherwise it returns the object returned by you.

But if we call product func without new keyword then it might return undefined when you have not returned anything.

Defining member fun's

function product(n, p) {

this.name = n;

this.price = p;

this.display = function() {

console.log(this.name, this.price);

}

}

When we call product without new keyword this points to global object.

```
const obj = {
    name: "abhi"
    display: function() {
        console.log(this, "is calling site");
    }
}
```

obj.display(); {name: 'abhi', display: ƒ}

is the calling site

* Once the object is made in case of arrow fun's now this doesn't point to calling site.

Prototypes

→ objects are created by constructor calls using new keyword classes → blueprint

objects → instances

In languages like C++, Java once the concrete object is made out of the blueprint we can't add properties to it we will be required to create new object.

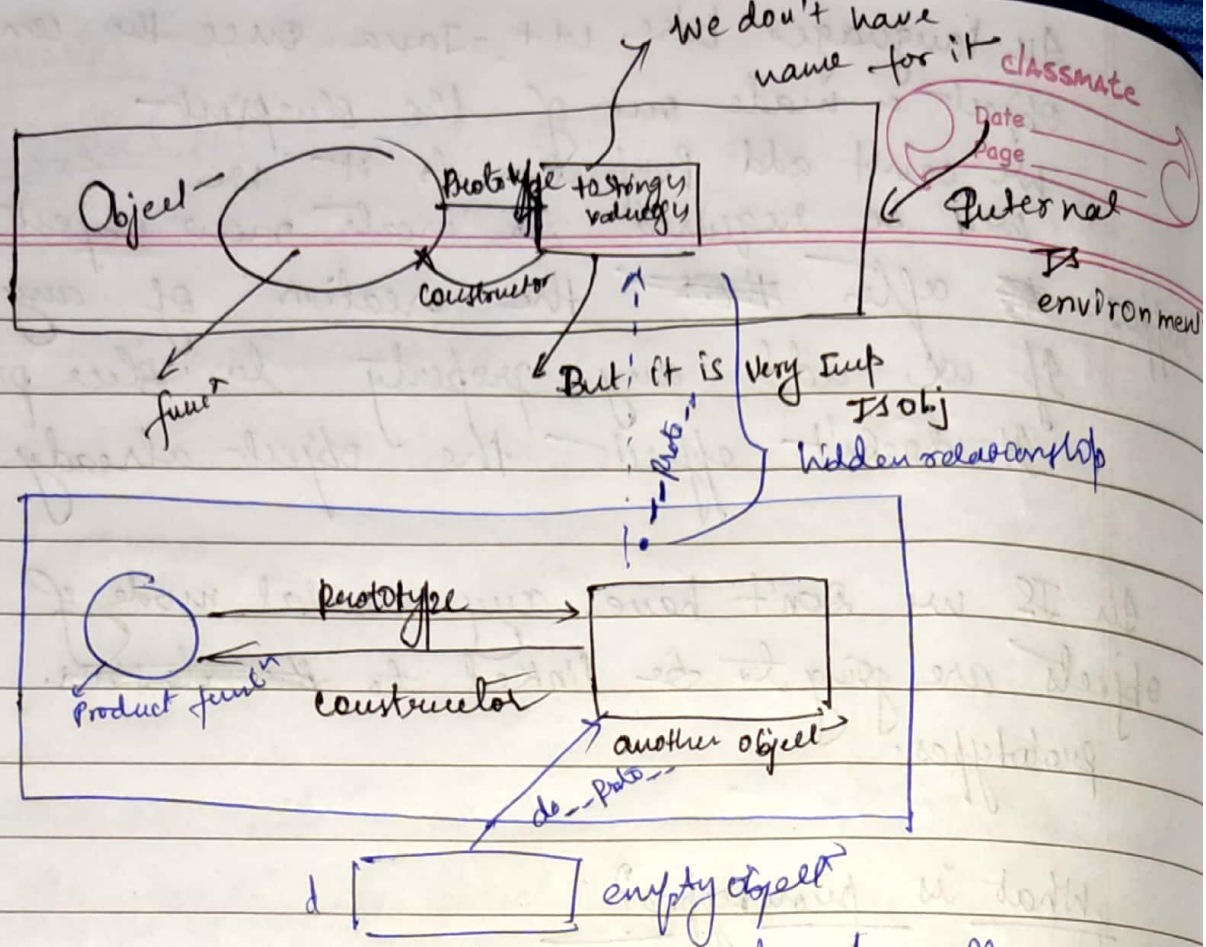
Suppose after the creation of any object if we add any property to blue print it doesn't effect the object already made.

In JS we don't have any mental mode of copy objects are going to be linked to ~~the classes~~. their prototypes.

What is prototypes?

→ It is a mechanism using which JS objects inherits features from one another.

* Every object will be having a property by default i.e prototype



In Arrow func^n this is resolved lexically.

~~Prototypal Inheritance~~

call(), apply(), bind() → mdn

freeze, seal

↓ neither we can add nor we can update old properties

↳ we can't add new property but we can update old property

Key, values, entries → Object property