Prototypal Inheritance

Inheritance is an object getting access to the properties and methods of another object.

We already discussed that arrays and functions in javascript are basically objects

$$\begin{tikzcd}
 & \text{Object \{\}} \arrow[dl] \arrow[dr] & \\
\text{Array [ ]} & & \text{Function( )}
\end{tikzcd}$$

So the array object and the function object get access to the properties and methods of Object{}

const array~~Num~~ = [ ]

array.__proto__

▸ [concat: f, fill f, find: f, ...] $\rightarrow$ array

These are basically the methods we use on array right?

Now let's go up the prototype chain, what's on top of Array[],? it's Object {} (see above diagram).

▸ $\texttt{array.\_\_proto\_\_.}\underbrace{\texttt{\_\_proto\_\_}}_{\text{going one chain up(Object \{\})}}$

→ { hasOwnProperty: f, toString: f, valueOf: f, ... }

▸ array.toString()

▸ " "

@codeWithSimran

How are we able to use toString method on array? Well we just said inheritance is one object (array) can access methods (toString) of another object (Object {})

So whatever is on top of the prototype inheritance chain, you'll have access to it.

Try the same thing with functions. and objects

To understand why this concept is important let's take an example

Let's say we have 2 students, who say hi when they come to class and once they finish the assignment, they can leave by saying Bye

```
let student1 = {
        name : 'John',
        assignmentDone : true,
        sayHi : function(){
                    console.log("Hi");
                }
        sayBye: function(){
                    if(assignmentDone)
                    {   console.log("Bye")
                    }
}
```

@codeWithSimran

```
let student2 = {
        name : "Ria"
  }
```

And now we dont want to repeat code, so if student2 wants to use method of student 1, we learnt we can use bind.

$\text{const sayBye} = \underbrace{\text{student1.sayBye}}_{\text{we want to use sayBye method of student 1}}\text{.bind}(\underbrace{\text{student2}}_{\text{we want to use it for student2.}})$

But wait, we do have access to sayBye from student1, but sayBye needs assignmentDone variable and student2 does not have it.

So we need to find a solution that not just lets student2 have access to sayBye but also assignmentDone.

We basically want student2 to inherit all functions and variables [properties] of student 1.

@codeWithSimran

Soln

~~Lizar~~ student2.__proto__ = student1

- student2.sayBye()
  → Bye
- student2.sayHi()
  → Hi
- student2.assignmentDone
  → true

▸ student2.name
  → Ria

@codeWithSimran

That means whatever properties student2 already has (name) will be taken from student2 itself, but whatever is new (sayHi, sayBye, assignmentDone) will be inherited (taken) from student1.

Recap

$$\underbrace{\text{student2.\_\_proto\_\_}}_{\text{create a prototype chain and}} = \underbrace{\text{student1}}_{\text{inherit properties not present in Student2 from Student1}}$$

Exercise

①
```
for(let property in student2)
        console.log(property)
```

②
```
for(let property in student2)
    if(student2.hasOwnProperty(property)){
            console.log(property)
    }
```

After you execute these, you'll know that student2 does not actually have properties of student1 copied, instead we just have a reference to them through prototypal inheritance
(It looks up the protype chain if property is present)

Scanned with CamScanner