JavaScript : Module ⑥

__/__/____

◊ Prototype :

- In JS, every object has a hidden property called [[Prototype]] (sometimes referred as __proto__).

•) If you try to access a property or method of an object and it is not found, JS looks for it in object's prototype.

•) This process continues upto prototype chain until it either finds the property or reaches the end (null).

:. To share methods & properties among many objects without duplicating code.

ii. Saves memory and keeps code efficient.

iii. Forms Backbone of inheritence in JS.

① Using __proto__ (old way):

```
let computer = {cpu: 12};
let lenevo = { screen: "HD", __proto__ : computer };
console.log (lenevo.cpu);  // 12
```

→ lenevo doesn't have cpu → JS check lenevo.__proto__ → finds computer → sees CPU : 12 → returns 12.

Similar approach

② Using object.setPrototypeOf (Modern way):

```
let genericCar = {tyres: 4};
let testa = {driver: "A1"};
object.setPrototypeOf (testa, genericCar);
```

→ tesla inherits from genericCar

__/__/____

Constructor:

- It is a special function used to create and initialize objects.
- It is a blueprint for objects.
- By convention, constructor function names start with capital letter.
- We call them with new keyword to create objects.

ex:
```
Function Person (name, age) {
    this.name = name; // 'this' refers to new object being created
    this.age = age;
    this.sayHello = function () {
        console.log('Hi, I'm ${this.name} and I'm
            ${this.age} years old.');
    };

let p1 = new Person ("Alex", 20);
let p2 = new Person ("Neha", 21);
p1.sayHello(); // Hi, I'm Alex and I'm 20 years old
p2.sayHello(); // Hi, Im Neha and I'm 21 years old
```

* constructor with prototype

- If we put methods inside the constructor, each object gets its own copy → not memory — efficient.

Better: put methods on the prototype.

ex:
```
Function Person (name, age) {
    this.name = name;
    this.age = age; }
```

// Shared by all objects

```
Person.prototype.sayHello = function () {
console.log (`Hi I'm ${this.name} and I'm ${this.age}
        years old.`);
};

let p1 = new Person ("Alex", 20);
let p2 = new Person ("Neha", 21);
p1.sayHello();
```

* Constructor safety check with new.target :

```
Function Drink (name) {
  if (!new.target) {
    throw new Error ("Drink must be called with
                    new keyword");
  }
  this.name;  }

let tea = new Drink ("chai");   // ✓
let coffee = Drink ("Coffee");   // ✗ Error
```

- new.target is a special built in reference available inside constructor.
- It's undefined if the function is not called with new.