8. OOP

* OOP = Object-Oriented Programming

→ OOP is a programming paradigm in computer science that relies on the concept of classes and objects.

→ It is used to structure a software program into simple, reusable pieces of code bluprints (usually called classes), which are used to create individiual instances of objects.

→ Benefits of using OOP

- Improved code organisation (structure of code)
- Reusability of code
- Better maintainability of code
- Closeness to real-world objects

* Object Prototypes

→ Prototypes are the mechanism by which Javascript objects inherit feature from one another.

→ It is like a single template object that all objects inherit methods and properties from, without having their own copy.

```
arr.__proto__      ⇐ reference
Array.prototype    } actual object
String.prototype
```

→ Every object in Javascript has built-in property, which is called its prototype.

→ The prototype is itself an object, so the prototype will have its own prototype, making what's called a prototype chain. The chain ends when we reach a prototype that has null for its own prototype.

e.g.
```
arr.__proto__
▸[ constructor: f, at: f, concat: f, ... ]
```

e.g.
```
arr.__proto__.push = (n) => { console.log("Pushing No.", n); };
arr.push(5);
pushing No. 5
```

changing default definition of push function

* Factor Functions

→ A function that creates objects

e.g.
```
const
function PersonMaker (name, age) {

    const person = {
            name: name,
            age: age,
            talk () {
                console.log(`My name is ${this.name}`);
                                                name
            }
    };
    return person;
}

let p1 = PersonMaker ("adam", 25);
let p2 = PersonMaker ( "eve", 24 );
p1.talk();
p2.talk();
```

Output: My name is adam
        My name is eve

→ Disadvantage : p1.talk === p2.talk ⇒ false
meaning, p1 stores talk() function and p2 stores talk() function at different locations. There two talk() function are made by p1, p2 individiually.

* New Operator

→ The new operator lets developers create an instance of a user-defined object type or of one of the built-in object types that has a constructor function.

```
e.g. function Person (name, age) {
                    name
             this.name = name;
             this.age = age;
     }              age

     Person.prototype.talk = function () {

           console.log(`Hi, My name is ${this.name}`);
     };                                  name

     let p1 = new Person ("adam", 26);
     let p2 = new Person ("eve", 25);
```

→ Now, p1.talk === p2.talk ⇒ true

→ When a function is called with the new keyword, the function will be used as a constructor.

→ new will do following things:

1. Creates a blank, plain JavaScript object. For convenience, let's call it newInstance.

2. points newInstance's [[Prototype]] to the constructor function's prototype property, if the prototype is an object; otherwise, newInstance stays as a plain object with object.prototype as its [[Prototype]].

~~2.~~Note: Properties / Objects added to the constructor function's prototype property are therefore accessible to all instances created from the constructon function.

3. executes the ~~construction~~ constructor function with the given arguments, binding newInstance as the this context (i.e. all references to this in the constructor function now refer to newInstance).

* Classes

→ classes are the template for creating objects

→ The constructor method is a special method of a class for creating and initialising an object instance of that class.

e.g.
```
class Person {

        constructor ( name, age ) {
                         name
              this.name = name;
              this.age = age;
     }            age

     talk () {

          console.log(`Hi, My name is ${this.name}`);
     }                                   name
}

let p1 = new Person ("adam", 27);
let p2 = new Person ("eve", 26);
```

* Inheritance

→ Inheritance is a mechanism that allows us to create new classes on the basis of already existing classes.

e.g.
```
class Student extends Person {

    constructor (name, age, marks) {
callback of
parent class's
constructor ——→   super (name, age);
                  this.marks = marks;
    }                    ↑marks.

    greet () { return "Hello!"; }

}

let s1 = new Student ("adam", 25, 90);
s1. talk ();
s1 . greet ();
```

output:
```
Hi, My name is adam.
Hello !
```

eg.
```
class Box {

    constructor (name, l, b){

        this.name = name;
        this.l = l;
        this.b = b;
    }

    area() {

        let area = this.l * this.b;
        console.log(`Box area is ${area}`);
    }
}

class Square Extends Box {

    constructor (a){

        super("square", a, a);
    }
    Same name then this method will override parent's method
    area () {

        let area = this.l * this.b;
        console.log(`Square area is ${ area}`);
    }
}
let sq1 = new Square(4);
sq1.area();
```
output : Square area is 16.