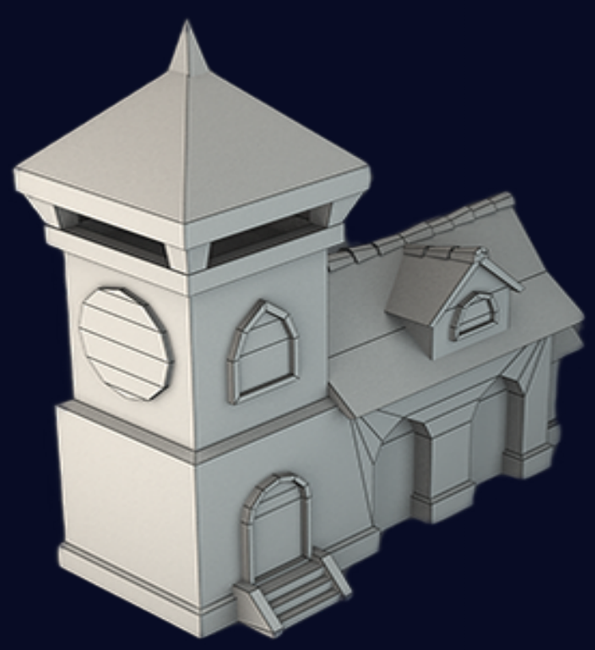
@startwithmani

@M.serisha Kothapalli

Constructor Functions

IN JAVASCRIPT

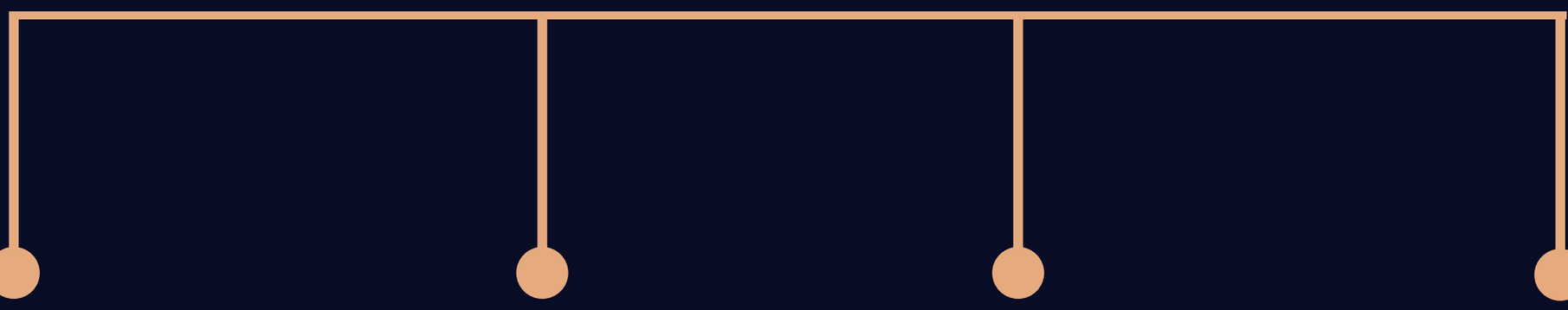


swipe

@startwithmani

@M.serisha Kothapalli

Creation of Objects



Why do we have multiple ways?

Before we get into that...Lets see what exactly are constructor functions and discuss few points.

swipe

@startwithmani

@M.serisha Kothapalli

Constructor functions:

JavaScript constructor function are special functions used to create multiple similar objects.

Two conventions ✌️

1) The name of a constructor function starts with a **capital letter** like Person, Employee.

2) A constructor function should be called only with the **new** operator.

swipe

Behind the scene of calling constructor functions:

When you call or invoke this constructor function, these sequence of operations take place:

- A new empty object gets created.
- The this keyword begins to refer to the new object and it becomes the current instance object.
- The new object is then returned as the return value.

swipe

Creation of constructor function:

- Follow the convention of capital letter for function name.
- Commented part happens behind the scene

```
function Employee(name)
{

  // this = { }
  this.name = name;
  this.greet = function (){
          console.log( `Hi! this is ${this.name}`);
    };
  // return this
}
```

swipe

@startwithmani

@M.serisha Kothapalli

Calling the constructor function:

- Follow the convention of capital letter for function name.
- Commented part happens behind the scene

```
const employee1 = new Employee('mani');
const employee2 = new Employee('serisha');
console.log(employee1)
console.log(employee2)
```

```
▼ Employee {name: 'mani', greet: ƒ} ⓘ
  ▶ greet: ƒ ()
    name: "mani"
  ▶ [[Prototype]]: Object
▼ Employee {name: 'serisha', greet: ƒ} ⓘ
  ▶ greet: ƒ ()
    name: "serisha"
  ▶ [[Prototype]]: Object
```

swipe

Downside

- Perfect we have created similar objects.But there is a downside in this approach.

- Every time an instance of Employee is created, a new function is defined and assigned to the greet property of that object.

- If we create 5 Employee objects, they will all have their own greet method that does the same thing.

- A more efficient way to do this is to define greet once, so that each Employee object use that same function reference.

- Incase if you wanted to update the method, you need to do it at one place instead of updating for each single instance.

swipe

Using prototype

- Every function in JavaScript has property called prototype. Whenever any Employee instance is created, the object will inherit any properties or methods defined on Employee.prototype.

```
function Employee(name) {
  this.name = name;
}

Employee.prototype.greet = function () {
   console.log( `Hi! this is ${this.name}`)
};

const employee1 = new Employee('mani');
const employee2 = new Employee('serisha');
console.log(employee1)
console.log(employee2)
```

swipe

@startwithmani

@M.serisha Kothapalli

```
▼ Employee {name: 'mani'} ℹ
    name: "mani"
  ▼ [[Prototype]]: Object
    ▶ greet: ƒ ()
    ▶ constructor: ƒ Employee(name)
    ▶ [[Prototype]]: Object
▼ Employee {name: 'serisha'} ℹ
    name: "serisha"
  ▼ [[Prototype]]: Object
    ▶ greet: ƒ ()
    ▶ constructor: ƒ Employee(name)
    ▶ [[Prototype]]: Object
```

Now the greet method is in prototype. So anything on the prototype is **shared** across all object instances of that constructor. I repeat **SHARED**

The greet method is defined only once, so that each Employee object instance use that same function reference.

ES6 classes - they are just syntactic sugar of constructor functions.

So stay connected for more

swipe

Did you find it helpful??

Like this post!

Share with your friends

Save it for later

Follow for more!

@startwithmani

@M.serisha Kothapalli