# Constructor Functions

This lesson introduces constructor functions, their syntax, and gives examples to explain the concept.

WE'LL COVER THE FOLLOWING


- Introduction
- Syntax
  - Explanation
- Example
  - Explanation


In the previous lesson, we discussed why the use of *constructor functions* is necessary, but we still don't know what they are. Let's delve into the details in this lesson.

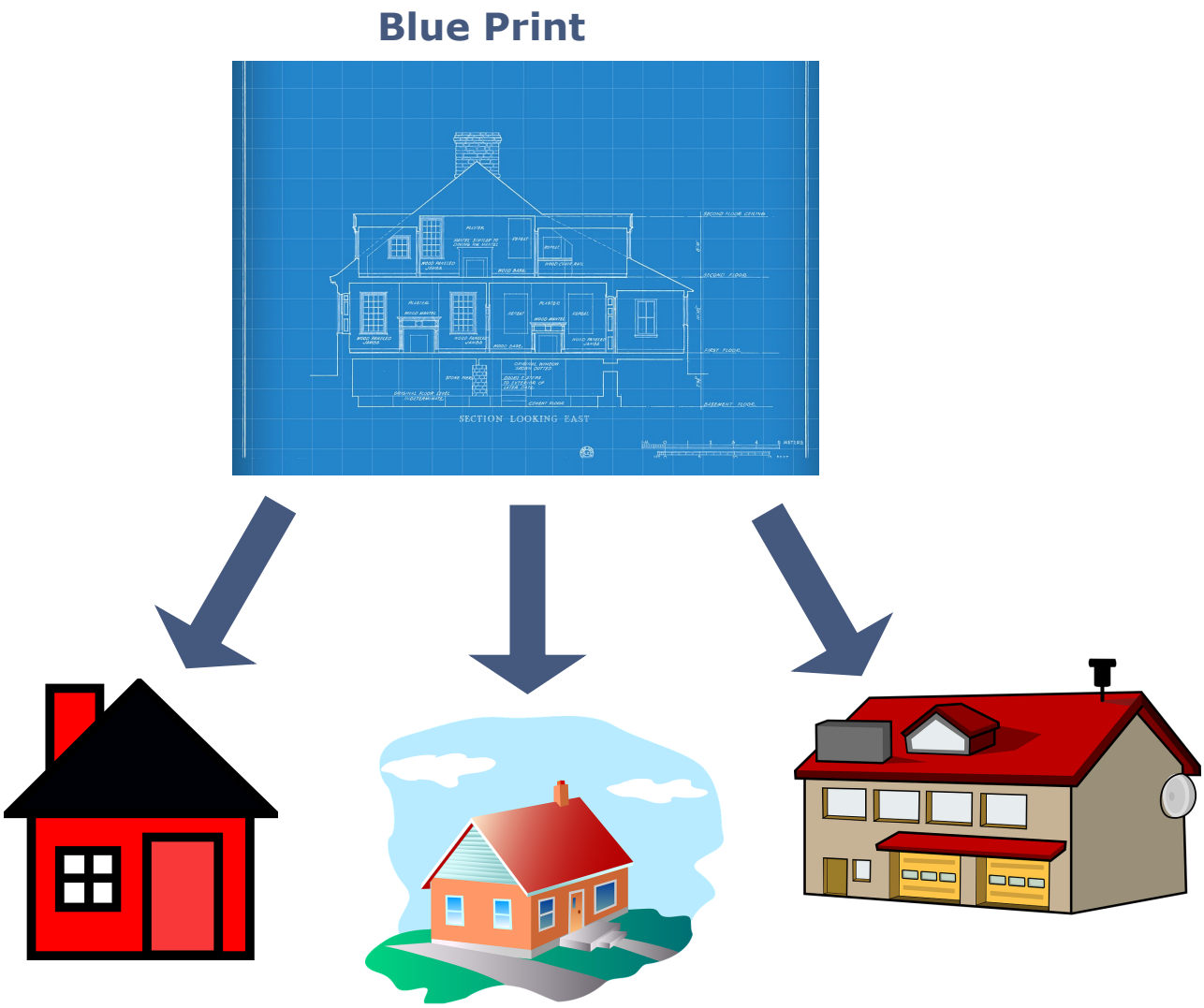
## Introduction #

The question from the previous lesson still stands: what is a better approach for creating *multiple objects*?

Let's think about it.

We discussed that all employees had all *properties* in common, i.e., `name`, `age` and `designation`. So wouldn't it be nice to have one generic template with all these properties for the object `employee` from which all future objects can then just be created? In short, wouldn't it be better to have a blueprint for the object `employee`?

The way to create an object type from a blueprint is by using **constructor functions**.




Building houses using blueprint of house

## Syntax #

The concept discussed above is basically that of *classes*. In other languages like **Java** or **C#**, classes form the blueprints for objects. However, in JavaScript until the version ES6 - which will be discussed in later chapters - there was no concept of using classes. Hence, in older versions like ES5, *constructor functions* were used to implement the functionality of classes.

Let's take a look at the syntax for *constructor functions* in the ES5 version of JavaScript.

```
function FunctionName(parameter1, parameter2,...){
    //all the properties of the object are initialized here
    //functions to be provided by objects are defined here
}
```

Constructor Function Syntax

## Explanation

As can be seen from above:

- The *keyword* `function` is used to define the function.
- The *constructor function* name should be *capitalized* just like `FunctionName` in the above snippet.
- The body of this function is basically the *constructor* part as it initializes the properties by setting them as equal to the respective parameters being passed into the function.

> **Note:** The parameters are optional.

- Additional *functions* that will be available in the objects can also be defined inside the constructor function's body.

# Example

Let's take a look at the implementation of a *constructor function* in order to understand the concept better.

```
function Employee(_name, _age, _designation){
  this.name = _name
  this.age = _age
  this.designation = _designation
}
```

Constructor Function for Employee

## Explanation

- The name of the *constructor function* is `Employee`.
- It is passed the parameters `_name`, `_age` and `designation`.
- From **lines 2-4**, the *properties* of the objects are being initialized as their values are set to the *parameters* passed. We discussed how to add properties in the last chapter. The *properties* being added are:
    - `name`

- `age`
- `designation`

All the objects created from `Employee` will contain these *properties*.

Whenever a new object is created, `this` is used to refer to this new object and set its property values. This is why, even though each object shares the same properties, they are assigned their own specific values, so the functionalities don't get mixed up.

---

Let's learn how to create these objects and look into their details in the next lesson!