Creating an object literal

This lesson teaches the different ways to create object literals in JavaScript.

WE'LL COVER THE FOLLOWING

Introduction

Just like other variables in JavaScript, an object too has to be defined and initialized in order to be created. There are various ways to create an object literal.

An *object literal* can be created:

- by using *figure brackets* `{...}` in the declaration.
- by using the `new` keyword.
- based on an existing object by using the `create()` method.

All of these approaches do exactly the same thing.

Syntax

Figure Brackets

We discussed earlier that an object contains data values and functions known as its *properties*. Let's take a look at the syntax for creating an object literal using `{..}`:

```
var objectName = {

  //properties defined
  propertyName1 : propertyValue1,
  propertyName2 : propertyValue2,
  functionName() {}

}
```

As shown above, to define a property value, we first need to write the name of the property followed by a colon and then the property value.

A property value can be anything, such as:

- *string*
- *integer*
- *boolean*
- *object*

> **Note:** All properties are separated by a *comma*.

Using `new` Operator

Now let's learn how to make an object using `new`:

```
var objectName = new Object()
```

The `new` keyword is used to create a new object from a constructor function. In the case above, we use `Object()`, which is an inbuilt constructor function used to make new objects. Since `Object()` has no arguments passed to it, it will create an *empty* object whose properties will then need to be defined.

To create an object with properties, user-defined constructor functions can be created which take arguments. We will learn how to do this in the next

chapter.

However, for the sake of simplicity and execution speed, the first approach is preferred to create an object literal.

`create()` Method

`create()` allows the creation of a new object based on an existing one.

> **Important Note:** The *new* object created will have the same properties as the object on which it was based.

Let's take a look at the syntax:

```
var newObjectName = Object.create(existingObjectName)
```

Example

Now let's try making an object using the three approaches one by one.

Suppose you want to create an `employee` object. An employee has a `name`, `age` and `designation`. These can be the employee's *properties*. A function `displayName` which displays the employee's `name` can also be added into the properties.

Creating an Object Using `{...}`

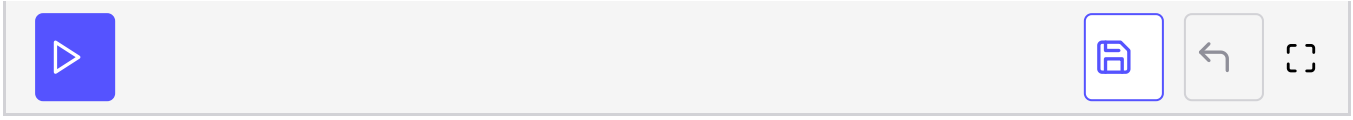
Let's create the `employee` object using the figure brackets first.

```
//creating an object named employee

var employee = {
  //defining properties of the object
  //setting data values
  name : 'Joe',
  age : 28,
  designation : 'Developer',
  //function to display name of the employee
  displayName() {
    console.log("Name is Joe")
  }
}

//displaying the properties of the object
//the method to access properties will be discussed in detail in the next lesson
```

```
employee.displayName()
console.log("Age is:",employee.age)
console.log("Designation is:",employee.designation)
```

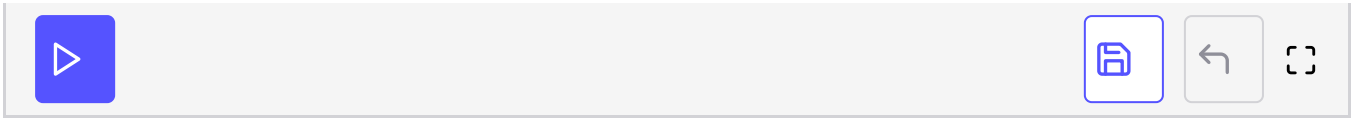


As mentioned earlier, the value of a property can also be another object. Let's look at an example of how we can implement that.

```
//creating an object named employee

var employee = {
  //defining properties of the object
  //setting data values
  name : {
    firstName: 'Joe',
    lastName: 'Adams'
    },
  age : 28,
  designation : 'Developer',
  displayName() {
    console.log("Name is Joe")
  }
}

//displaying the properties of the object
//the method to access properties will be discussed in detail in the next lesson
employee.displayName()
console.log("Age is:",employee.age)
console.log("Designation is:",employee.designation)
```



In the above example, the `name` property has another object as its property. This object contains `firstName` and `lastName` as its properties. Since we are encapsulating this object inside `name`, it doesn't need to be defined using the `var` keyword. Adding the commas in between the two properties automatically creates an object.

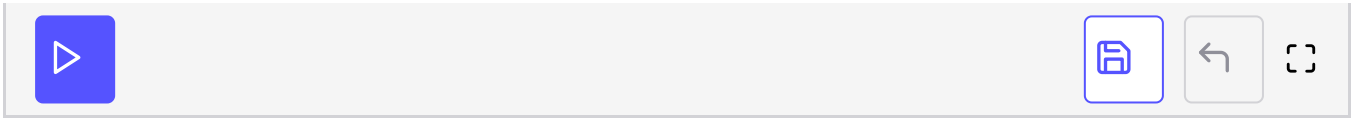
Creating an Object Using `Object()`

Let's create the `employee` object using the *object constructor*.

```
//an empty employee object created
var employee = new Object()
//adding properties to the object
employee.name =  'Joe'
employee.age = 28
```

```
employee.designation = 'Developer'
//adding a function called display to the object
//using the function keyword
employee.display = function() {
  console.log("Name is Joe")
}

//displaying the properties of the object
//the method to access properties will be discussed in detail the next lesson
console.log("Age is:",employee.age)
console.log("Designation is:",employee.designation)
employee.display()
```



In the above example, an empty `employee` object is created in **line 2**. Properties are then added to it as seen in **lines 4-6**. In **line 9**, a function called `display` is added as a property. The *keyword* `function` is used in order to define it.

Creating an Object Using `create`

Let's make the `employee` object using the `create` method.

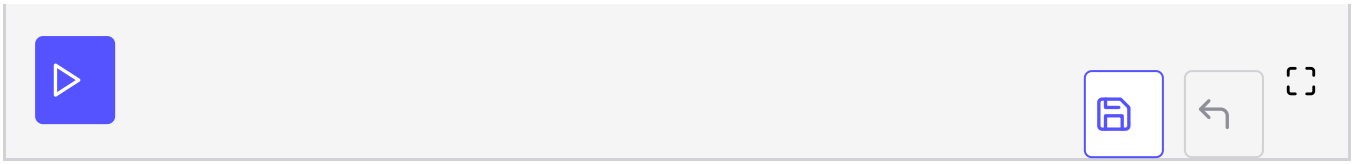
```
//creating an object named employee1

var assistantManager = {
  //defining properties of the object
  //setting data values
  name : 'Joe',
  age : 28,
  designation : 'Developer',
  //function to display name of the employee
  displayName() {
    console.log("Name is Joe")
  }
}

//Example: we have an "assistantManager" who gets promoted to "manager" position
//so we create a "manager" object based on "assistantManager"
//it will have same properties as "assistantManager"
//however these properties can be changed or added to
var manager = Object.create(assistantManager)

//displaying the properties of the object assistantManager
//the method to access properties will be discussed in detail the next lesson
assistantManager.displayName()
console.log("Age is:",assistantManager.age)
console.log("Designation is:",assistantManager.designation)
//displaying the properties of the object employee2
//this will show the same values as that of object employee1
manager.displayName()
console.log("Age is:",manager.age)
```

```
console.log( 'Designation is: ',manager.designation)
```



When the above code runs, the properties of `manager` will be exactly the same as that of `assistantManager` since it is based on it. However, these properties can be modified and additional properties can also be added to it.

`const`

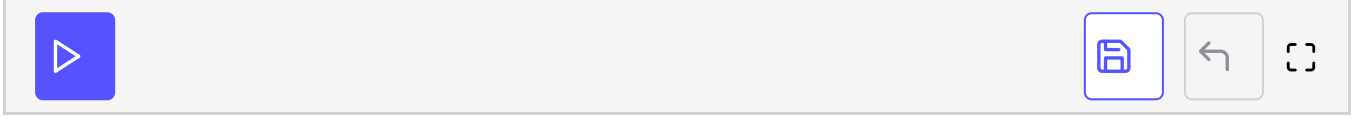
In some places, you might notice the *keyword* `const` is used to declare an object. Using `const` doesn't allow the object to have another object's binding, meaning you cannot assign a new object to this variable; hence, it cannot be assigned new content.

```
//creating an object named employee using const

const employee = {
    name : 'Joe',
    age : 28,
    designation : 'Developer'
}

//value of name, age or designaton can be changed
employee.name = 'Amy'
console.log("New name is:",employee.name)

//object cannot be assigned new object
//You will get an error when you uncomment and run the line below
//employee = {sex : 'male', status : 'single'}
```



If you uncomment **line 15** and run the code, the error `employee is read-only`, shows up. This is because `const` doesn't allow the object identity to be changed.

Now that you know how to create object literals, in the next lesson, we will learn about the different ways to access an object's properties.