

JavaScript Essentials

Objects



Table of Contents

1. Overview - Object basics
2. Dot notation
3. Bracket notation
4. Setting object members
5. Q&A

- Understand the fundamental JavaScript object syntax
- Understand the basic theory behind object-oriented programming, how this relates to JavaScript ("most things are objects"), and how to start working with JavaScript objects
- Able to access and setting member in Object using Dot notation and Bracket notation

Section 1

Overview – Object basics

- An object is a collection of related data and/or functionality (which usually consists of several variables and functions — which are called properties and methods when they are inside objects.)
- Let's work through an example to understand what they look like.

- Syntax to create an Object:

opening brace

```
var p = { };  
p;  
► {}
```

closing brace

- Syntax to add member to Object

```
1  const objectName = {  
2      member1Name: member1Value,  
3      member2Name: member2Value,  
4      member3Name: member3Value  
5  };
```

- To represent a Fresher Angular

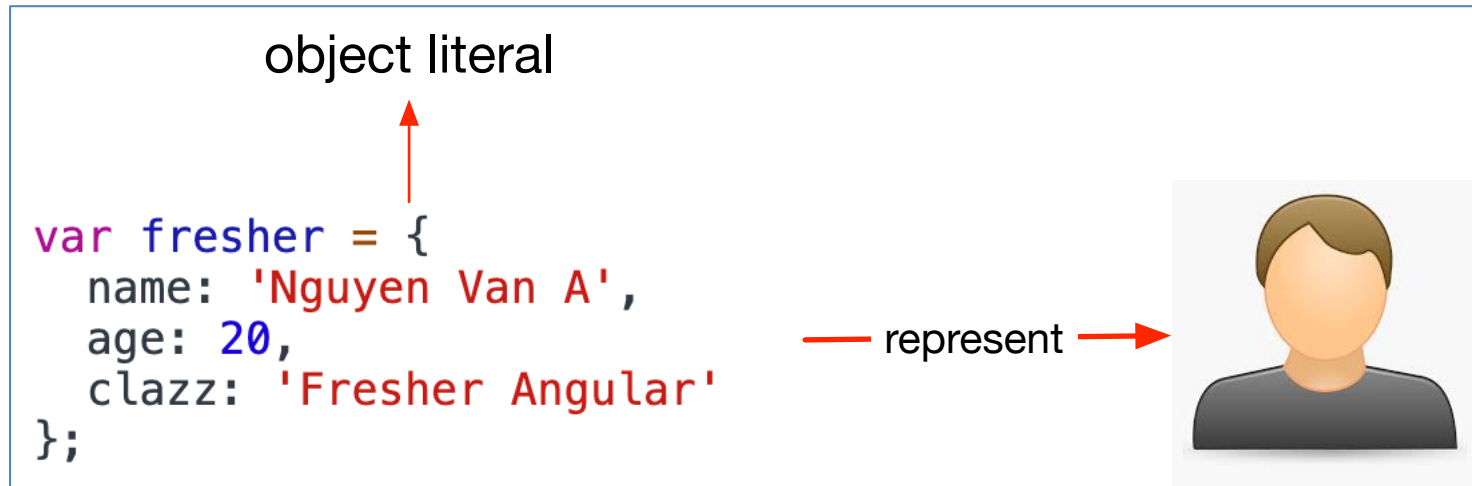
```
var fresher = {  
  name: 'Nguyen Van A',  
  age: 20,  
  clazz: 'Fresher Angular'  
};
```

— represent →



- The value of an object member can be pretty much anything — in our person object we've got a string, a number, two arrays, and two functions.
- The first four items are data items, and are referred to as the object's **properties**.
- The last two items are functions that allow the object to do something with that data, and are referred to as the object's **methods**

- An object like this is referred to as an **object literal** — we've literally written out the object contents as we've come to create it.



- An object is a collection of related data and/or functionality (which usually consists of several variables and functions — which are called properties and methods when they are inside objects.)
- The value of an object member can be pretty much anything
- If it's a value we refer to it as a **property**
- If it's a function we refer to it as a **method**

- **Object literal** - we've literally written out the object contents as we've come to create it
- Object is much more efficient than sending several items individually, and it is easier to work with than an array, when you want to identify individual items by name

Section 2

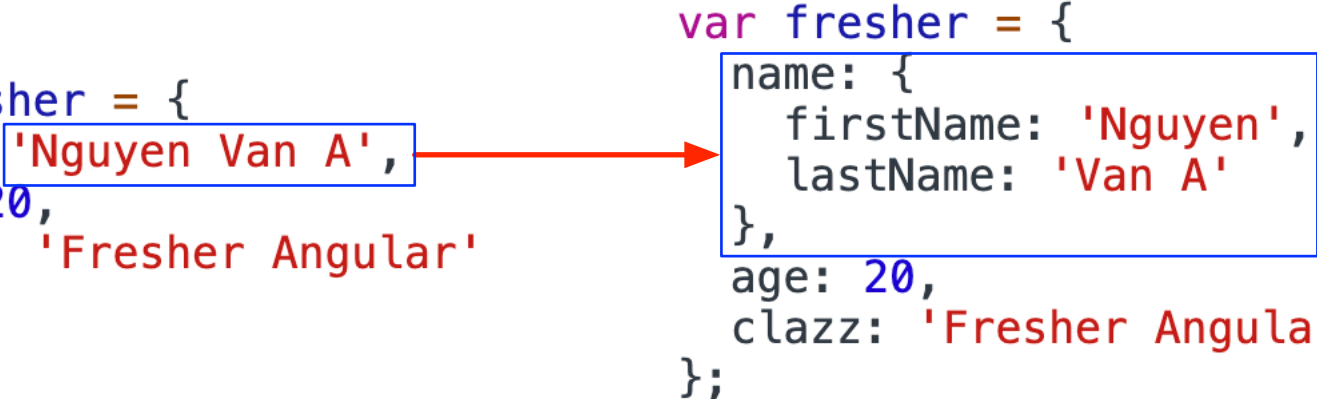
Dot notation

- Above, you accessed the object's properties and methods using **dot notation**. The object name (person) acts as the **namespace** — it must be entered first to access anything **encapsulated** inside the object
- Next you write a **dot**, then the item you want to access — this can be the name of a simple property, an item of an array property, or a call to one of the object's methods, for example:

```
1 | person.age  
2 | person.interests[1]  
3 | person.bio()
```

- It is even possible to make the value of an object member another object. For example, try changing the name member:

```
var fresher = {  
  name: 'Nguyen Van A',  
  age: 20,  
  clazz: 'Fresher Angular'  
};
```



The diagram illustrates the transformation of the 'name' property. On the left, the original code has 'name: 'Nguyen Van A'', where 'Nguyen Van A' is highlighted with a blue box. A red arrow points from this box to a new, more complex object structure on the right. This new structure is enclosed in a blue box and shows 'name' as an object with 'firstName: 'Nguyen'' and 'lastName: 'Van A''.

```
var fresher = {  
  name: {  
    firstName: 'Nguyen',  
    lastName: 'Van A'  
  },  
  age: 20,  
  clazz: 'Fresher Angular'  
};
```

- Here we are effectively creating a **sub-namespace**. This sounds complex, but really it's not — to access these items you just need to chain the extra step onto the end with another dot. Try these in the JS console:

```
fresher.name.firstName  
"Nguyen"  
fresher.name.lastName  
"Van A"
```


- **Note:** when using Dot notation the keyword on the right of Dot must be a valid identifier

```
fresher.name; // valid  
fresher.name-; // invalid  
fresher.0name; // invalid  
fresher.nam+e; // invalid
```

- To access member of an object use Dot notation
- The object name acts as the **namespace** — it must be entered first to access anything **encapsulated** inside the object
- We can create **sub-namespace** by setting value of a member to another object

Section 3

Bracket notation

- There is another way to access object properties — using bracket notation. Instead of using these:

```
fresher.name;  
fresher.age;  
fresher.clazz
```

- You can use

```
fresher['name'];  
fresher['age'];  
fresher['clazz'];
```

- With this notation, you can use **variable** to access a member
- This looks very similar to how you access the items in an array, and it is basically the same thing — instead of using an index number to select an item, you are using the name associated with each member's value.
- It is no wonder that objects are sometimes called **associative arrays** — they map strings to values in the same way that arrays map numbers to values.

- Access member using variable:

```
var n = 'name';  
var l = 'lastName';  
  
fresher[n][l];  
"Van A"
```

- Use Bracket notation to access a member using either 'string' or a variable
- If it's a variable inside Bracket, then JavaScript will resolve the value of that variable and access the member of corresponding key
- Objects are sometimes called **associative arrays**

Section 4

Setting object members

- So far we've only looked at retrieving (or getting) object members — you can also set (update) the value of object members by simply declaring the member you want to set (using dot or bracket notation), like this:

```
fresher.age = 24;  
fresher['name']['firstName'] = 'Binh';
```

- Setting members doesn't just stop at updating the values of existing properties and methods; you can also create completely new members. Try these in the JS console:

```
fresher.eyes = 'brow';  
fresher;
```

```
▼ {name: {...}, age: 24, clazz: "Fresher Angular", eyes: "brow"} ⓘ  
  age: 24  
  clazz: "Fresher Angular"  
  eyes: "brow"  
  ► name: {firstName: "Binh", lastName: "Van A"}  
  ► __proto__: Object
```

- One useful aspect of bracket notation is that it can be used to set not only member values dynamically, but member names too.
- Let's say we wanted users to be able to store custom value types in their people data, by typing the member name and value into two text inputs. We could get those values like this:

```
1 | let myDataName = nameInput.value;  
2 | let myDataValue = nameValue.value;
```

- We could then add this new member name and value to the person object like this:

```
1 | person[myDataName] = myDataValue;
```

- To test this, try adding the following lines into your code, just below the closing curly brace of the person object:

```
1 | let myDataName = 'height';  
2 | let myDataValue = '1.75m';  
3 | person[myDataName] = myDataValue;
```

Practice 1: Setting object members

- To setting object members you can use either dot notation or bracket notation
- Then use assignment (=) to set value
- If the **memberKey** does exists then the value is updated
- If the **memberKey** doesn't exists, then a new member will be created

Thank you

Q&A

