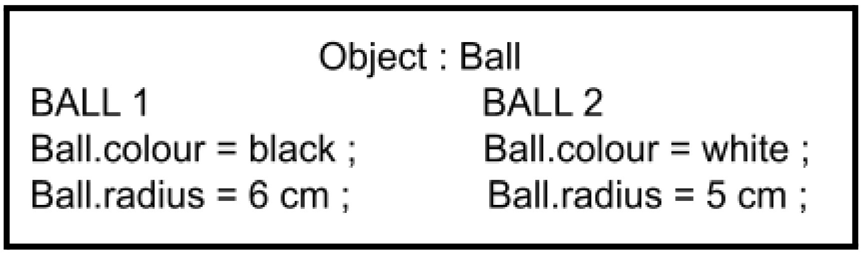
Objects

Overview

JavaScript objects are a collection of properties in a **key-value pair**. These objects can be understood with real-life objects, like similar objects have the same type of properties, but they differ from each other.

Example: Let's say a ball is an object and have properties like 'colour' and 'radius'. So **every ball will have the same properties, but different balls will have different values** to them.



Object : Ball

BALL 1	BALL 2
Ball.colour = black ;	Ball.colour = white ;
Ball.radius = 6 cm ;	Ball.radius = 5 cm ;

Some important points about objects are -

- Object contains properties separated with a comma(,).
- Each property is represented in a **key-value pair.**
- Key and value are separated using a colon(:).
- The key can be a string or variable name that does not contain special characters, except underscore(_).
- The value can contain any type of data - primitive, non-primitive and even a function.
- The objects are **passed by reference** to a function.

```
Example :    var obj = {
                 key1: "value1",
                 key2: 12345,
                 "key3": true,
                 key4: function( ) {
                     //code
                   }
               }
```

Creating an Object

1. **Using curly brackets -**

   - Create empty object as - `var obj = { } ;`
   - Object with some initial properties as -
     `var obj = { key1: value1, … , keyN:valueN }`

2. **Using new operator -**

   - Create empty object as - `var obj = new Object( );`
   - Object with properties as -
     `var obj = new Object( { key1: value1, …, keyN: valueN } )`

The properties can be created at the time of creating an object and also after that. **Both creating and accessing the properties share similar syntax**.

Creating and Accessing Properties

The properties are created in a key-value pair, but some restrictions exist in the way some keys are created. There are two ways to create and access properties -

1. **Using a dot operator -** You can use dot operator only when the property name starts with a character. Property can be **accessed like** - `obj.propertyName.` Similarly, you can **create property** like - `obj.propertyName = value`

2. **Using a square bracket -** You need to use a square bracket when the key name starts with a number. If the name contains a special character, then it will be stored as a string. Property is **accessed like** - obj["propertyName"]. Similarly, you **create property** like - obj["propertyName"] = value

NOTE: If you access a property that has not been defined, then **undefined** is returned.

You can also **set function as the value to the key.** So the key then becomes the method name and **need parentheses to execute**. So you can execute methods like - obj.methodName( ) and obj["methodName"]( ).

Example :

```
var ball = {
            sport : "Cricket",
            colour : "Yellow" ,
            radius : 3 ,
            print : function( ){
                        console.log("Coding Ninjas");
                    }
  }

console.log( ball.sport );  // Cricket
console.log( ball["radius"] );  // 3
console.log( ball.size );  // undefined
ball.print( ); // Coding Ninjas
```

Deleting Property

You can remove property of object using **delete** operator followed by the property name. You can either use **dot operator** or **square bracket** notation.

Syntax :

```
delete obj.objectName ;
        OR
delete obj["objectName"] ;
```

Example:

```
delete ball.radius ;
```

How are Objects Stored

There are two things that are very important in objects -

- Objects are **stored in a heap.**
- Objects are **reference types.**

These two are important in regard that **object variables point to the location** where they are stored. This means that **more than one variable can point to the same location.**

Until now, you are **creating new objects** every time like -

```
var item1 = { name: “Coding Ninjas” } ;
var item2 = { name: “Coding Ninjas” } ;
```

The **above two lines will create two different objects** are not therefore equal -

```
item1 == item2;    // Returns - false
item1 === item2;  // Returns - false
```

But, if you assign **one object to another,** then the value of **'item1' gets assigned to 'item2',** and therefore, they both will point to the same location -

Example :

```
var item1 = { name: "Coding Ninjas" } ;
var item2 =  { name: "Coding Ninjas" } ;
item1 = item2;

console.log(item1 == item2) ;   // Returns true
console.log(item1 === item2) ; // Returns true
```

Iterating Objects

JavaScript provides a special form of loop to traverse all the keys of an object. This loop is called **'for...in'** loop.

Syntax :

```
for (variable in object) {
            // Statements
}
```

Here the **'variable' gets assigned the property name** on each iteration, and 'object' is the object you want to iterate. Use the **square bracket notation with variables to access the property values.**
The **iteration may not be in a similar order as to how you see properties in objects** or how you have added them because the objects are ordered specially.
The **property names as integers are iterated first** in ascending order. Then the other names are iterated in the order they were added.

Example :

```
for( key in ball) {
        console.log( key , ":", ball[key] );
}
```

Output :

```
sport : Cricket
colour : Yellow
radius : 3
print : function(){
         console.log("Coding Ninjas");
 }
```

ARRAY AS OBJECT

Arrays are actually objects. If you use the **typeOf( )** method on an array, you will see that it will return an **object**. If you see an array on a console, they are **key-value pairs**, with the **positive integers as the keys.**
Arrays can also store properties just like objects.

Example :

```
array["one"] = 1;
array.one ; // 1
array["one"] ; // 1
```

Arrays vs Object

- Arrays have a **length** property that objects does not have.
- You can access the values of the arrays like - array[0]; or array["0"]; whereas in objects, you have to use **double quotes ("")** only.
- Only when you use an integer as a key, it will change the 'length' property.
- Adding a non-integer key will not have any effect on the length' property.

NOTE: Length property will be set according to the maximum integer key of the array.

Using for...in loop to Iterate

Since **arrays are also objects**, you can use the 'for-in' loop to traverse it. Traversing the array using 'for-in' loop is the same like traversing an object.

There is something interesting about arrays you need to know.

```
var arr = [10, 20, 30] ;
arr["four"] = 40 ;
console.log(arr) ;
```

Output : Array(3) [10, 20, 30]

- But, it also **contains the property "four: 40"**, but it **does not show** in the array. But if you use the **for-in** loop to traverse it, you can traverse all the properties.

```
for(var i in arr) {
    console.log( i, ":", arr[i]);
}
```

Output :

```
0 : 10
1 : 20
2 : 30
four: 40
```

this keyword

Define a function to get the full name of a person in the object-person

```
var person = {
        firstName: "Tony",
        lastName : "Stark",
        age : 40 ,
        getname: function( ) {
                 return this.firstName + " " + this.lastName;
                }
        };

        console.log(person.getname( )) ; //Tony Stark
```

- In a function definition, this refers to the "owner" of the function.
- In the example above, **this** is the person object that owns the getname function.
- In other words, **this.firstName** means the firstName property of this object and **this.lastName** means the firstName property of this object