JavaScript Objects: the basics pt1 Cheat Sheet

by kuriositty via cheatography.com/146616/cs/31746/

Objects

There are 8 data types in JavaScript. 7 of them are called **primitive**, because their values contain only a single thing (string, number, bigInt, boolean, null, undefined, symbol).

In contrast, **objects** are used to store keyed collections of various data and more complex entities.

An object can be created with *figure brackets* `{…}` with an optional list of properties. A property is a "`key: value`" pair, where key is a *string* (also called a *property name*), and value can be anything.

An empty object can be created using one of two syntaxes: `let user = new Object();` *object constructor* syntax and `let user = {};` *object literal* syntax which is usually used

Property values are accessible using the dot notation: `delete user.age;`

The last property in the list may end with a *trailing* or *hanging* comma which makes it easier to add/remove/move around properties, bcs all ines become alike:

```
   age: 30,
};
```

Property value shorthand

```
function makeUser(name, age) {
    return {
        name: name,
       age: age,
       // ...other properties
   };
}
let user = makeUs er( " Joh n", 30);
alert( use r.n ame); // John
// or
function makeUs er( name, age) {
> return {
   name, // same as name: name
   age, // same as age: age
   // ...
 };
}
```

Property value shorthand (cont)

The "for…in" loop

```
let user = {
    name: " Joh n",
   age: 30,
    isA dmin: true
};
for (let key in user) {
   // keys
    alert( key ); // name, age, isAdmin
   // values for the keys
    alert( user[key] ); // John, 30, true
}
```

To walk over all keys of an object, `for..in` loop can be used. `key` and `prop` are widely used variable names.

[Square brackets]

We can also use **multiword property names**, but then they must be quoted:

```
" likes birds": true
```

For multiword properties, the dot access doesn't work.
The **dot requires the key to be a valid variable** identifier.

```
user.likes birds = true // a syntax error
```

[Square brackets] (cont)

There's an alternative **square bracket notation that works with strings**

```
user["likes birds"] = true; // set
alert( use r["likes birds"]); // g
et
```

or:

```
let key = " likes birds";
user[key] = true;
```

The **dot notation cannot be used in a similar way**:

```
let user = {
    name: " Joh n",
   age: 30
};
let key = " nam e";
alert( user.key ); // undefined
```

`[]` are much more powerful than dot notation. They allow any property names and variables. But they are also more cumbersome to write. Most of the time, when property names are known and simple, the dot is used. And if we need something more complex, then we switch to square brackets.

Property names limitations

A variable cannot have a name equal to one of the language-reserved words like *for*, *let*, *return* etc. But for an object property, there's no such restriction. there are no limitations on property names.

They can be any strings or symbols (a special type for identifiers). Other types are automatically converted to strings. For instance, a number 0 becomes a string "0" when used as a property key: `0: " tes t " // same as " 0": " tes t"`

Ordered like an object

```
let codes = {
    " +49 ": " Ger man y",
    " +41 ": " Swi tze rla nd",
    " +44 ": " Great Britai n",
   // ..,
```

Ordered like an object (cont)

```
>  "+1": "USA"
};
for (let code in codes) {
  alert( +code ); // 49, 41, 44, 1
}
```

If we loop over an object, do we get all properties in the same order they were added? Integer properties are sorted, others appear in creation order.

[Square brackets: Computed properties]

```
let fruit = prompt("Which fruit to buy?", "apple");
let bag = {
    [fr uit]: 5,
};
alert( bag.apple ); // 5 if fruit= " app le"
// We can use more complex expres sions
// inside square brackets
let fruit = 'apple';
let bag = {
    [fruit + 'Juice']: 5 // bag.ap ple Juice = 5
};
// Essent ially, that works the same as:
let fruit = prompt ("Which fruit to buy?", " app le");
let bag = {};
// take property name from the fruit variable
bag[fruit] = 5;
```

The meaning of a computed property is simple: `[fruit]` means that the property name should be taken from *fruit*. So, if a visitor enters "apple", bag will become `{app le: 5}`

Property existence test, “in” operator

A notable feature of objects in JavaScript, is that it’s possible to access any property, because **reading a non-existing prooperty returns `undefined`**.

```
let user = {}
alert( user.n oSu chP roperty === undefined ); // t rue
```

There’s also a special operator `in` for that.

```
let user = { name: " Joh n", age: 30 };
alert( " age " in user ); // true
alert( " bla bla " in user ); // false
```

On the left side of `in` there must be a property name. That’s usually a quoted string. If we omit quotes, that means a variable should contain the actual name to be tested.

Despite `undefined`, `in` operator exists because it will recognize that a property actuallty exists when it's storing `undefined`. Situations like this happen very rarely, because undefined should not be explicitly assigned.

Long story short

Property keys must be strings or symbols (usually strings). Values can be of any type.

To access a property, we can use:
The *dot notation*: `obj.pr operty`
Square brackets notation `obj["pr ope r ty "]`. Square brackets allow taking the key from a variable, like `obj[va rWi thK ey]`.

Additional operators:
To delete a property: `delete obj.prop`
To check if a property with the given key exists: `" key " in obj`
To iterate over an object: `for (let key in obj)` loop.