codecademy

Learn JavaScript: Objects

Dot Notation for Accessing Object Properties

Properties of a JavaScript object can be accessed using the dot notation in this manner: `object.propertyName`. Nested properties of an object can be accessed by chaining key names in the correct order.

```
const apple = {
  color: 'Green',
  price: {
    bulk: '$3/kg',
    smallQty: '$4/kg'
  }
};
console.log(apple.color); // 'Green'
console.log(apple.price.bulk); // '$3/kg'
```

Restrictions in Naming Properties

JavaScript object key names must adhere to some restrictions to be valid. Key names must either be strings or valid identifier or variable names (i.e. special characters such as `-` are not allowed in key names that are not strings).

```
// Example of invalid key names
const trainSchedule = {
  platform num: 10, // Invalid because of the space between words.
  40 - 10 + 2: 30, // Expressions cannot be keys.
  +compartment: 'C' // The use of a + sign is invalid unless it is enclosed in quotations.
}
```

Objects

An *object* is a built-in data type for storing key-value pairs. Data inside objects are unordered, and the values can be of any type.

Accessing non-existent JavaScript properties

When trying to access a JavaScript object property that has not been defined yet, the value of `undefined` will be returned by default.

```
const classElection = {
  date: 'January 12'
};

console.log(classElection.place); // undefined
```

JavaScript Objects are Mutable

JavaScript objects are *mutable*, meaning their contents can be changed, even when they are declared as `const`. New properties can be added, and existing property values can be changed or deleted.
It is the *reference* to the object, bound to the variable, that cannot be changed.

```
const student = {
  name: 'Sheldon',
  score: 100,
  grade: 'A',
}

console.log(student)
// { name: 'Sheldon', score: 100, grade:
'A' }

delete student.score
student.grade = 'F'
console.log(student)
// { name: 'Sheldon', grade: 'F' }

student = {}
// TypeError: Assignment to constant
variable.
```

JavaScript `for...in` loop

The JavaScript `for...in` loop can be used to iterate over the keys of an object. In each iteration, one of the properties from the object is assigned to the variable of that loop.

```
let mobile = {
  brand: 'Samsung',
  model: 'Galaxy Note 9'
};

for (let key in mobile) {
  console.log(`${key}: ${mobile[key]}`);
}
```

Properties and values of a JavaScript object

A JavaScript object literal is enclosed with curly braces `{}`. Values are mapped to keys in the object with a colon (`:`), and the key-value pairs are separated by commas. All the keys are unique, but values are not.
Key-value pairs of an object are also referred to as *properties*.

```
const classOf2018 = {
  students: 38,
  year: 2018
}
```

Delete operator

Once an object is created in JavaScript, it is possible to remove properties from the object using the `delete` operator. The `delete` keyword deletes both the value of the property and the property itself from the object.

The `delete` operator only works on properties, not on variables or functions.

```
const person = {
  firstName: "Matilda",
  age: 27,
  hobby: "knitting",
  goal: "learning JavaScript"
};

delete person.hobby; // or delete
person[hobby];

console.log(person);
/*
{
  firstName: "Matilda"
  age: 27
  goal: "learning JavaScript"
}
*/
```

javascript passing objects as arguments

When JavaScript objects are passed as arguments to functions or methods, they are passed by *reference*, not by value. This means that the object itself (not a copy) is accessible and mutable (can be changed) inside that function.

```
const origNum = 8;
const origObj = {color: 'blue'};

const changeItUp = (num, obj) => {
  num = 7;
  obj.color = 'red';
};

changeItUp(origNum, origObj);

// Will output 8 since integers are passed
by value.
console.log(origNum);

// Will output 'red' since objects are
passed
// by reference and are therefore mutable.
console.log(origObj.color);
```

JavaScript Object Methods

JavaScript objects may have property values that are *functions*. These are referred to as object *methods*. Methods may be defined using anonymous *arrow function expressions*, or with *shorthand method syntax*.

Object methods are invoked with the syntax: `objectName.methodName(arguments)` .

```
const engine = {
  // method shorthand, with one argument
  start(adverb) {
    console.log(`The engine starts up ${adverb}...`);
  },
  // anonymous arrow function expression with no arguments
  sputter: () => {
    console.log('The engine sputters...');
  },
};

engine.start('noisily');
engine.sputter();

/* Console output:
The engine starts up noisily...
The engine sputters...
*/
```