Unit5: Objects: Grouping your data

At the end of this unit, the learner should be able to:

- ➢ Define object oriented programming related concepts
- ➢ Organize information with JavaScript objects
- ➢ Create objects
- ➢ Add properties to objects
- ➢ Access properties using dot notation

I. Object-oriented programming

Before diving into JavaScript, let's take a moment to review what people mean when they say object-oriented, and what the main features of this programming style are. Here's a list of concepts that are most often used when talking about **object-oriented programming** (**OOP**):

- Object, method, and property
- Class
- Encapsulation
- Aggregation
- Reusability/inheritance
- Polymorphism

1. Objects

As the name object-oriented suggests, objects are important. An object is a representation of a thing (someone or something), and this representation is expressed with the help of a programming language. The thing can be anything, a real-life object, or a more convoluted concept. Taking a common object, a cat, for example, you can see that it has certain characteristics–color, name, weight, and so on and can perform some actions–meow, sleep, hide, escape, and so on. The characteristics of the object are called properties in OOP-speak, and the actions are called methods.

The analogy with the spoken language are as follows:

- Objects are most often named using nouns, such as book, person, and so on
- Methods are verbs, for example, read, run, and so on
- Values of the properties are adjectives

Take the sentence “The black cat sleeps on the mat” as an example. “The cat” (a noun) is the object, “black” (adjective) is the value of the color property, and “sleep” (a verb) is an action or a method in OOP. For the sake of the analogy, we can go a step further and say that “on the mat” specifies something about the action “sleep”, so it's acting as a parameter passed to the sleep method.

2. Classes

In real life, similar objects can be grouped based on some criteria. A hummingbird and an eagle are both birds, so they can be classified as belonging to some made-up Birds class. In OOP, a class is a blueprint or a recipe for an object. Another name for object is instance, so we can say that the eagle is one concrete instance of the general Birds class. You can create different objects using the same class because a class is just a template, while the objects are concrete instances based on the template.

There's a difference between JavaScript and the classic OO languages such as C++ and Java. You should be aware right from the start that in JavaScript, there are no classes; everything is based on objects. JavaScript has the notion of prototypes, which are also objects.

3. Encapsulation

Encapsulation is another OOP related concept, which illustrates the fact that an object contains (encapsulates) the following:

- Data (stored in properties)
- The means to do something with the data (using methods)

One other term that goes together with encapsulation is information hiding. This is a rather broad term and can mean different things, but let's see what people usually mean when they use it in the context of OOP.

Imagine an object, say, an MP3 player. You, as the user of the object, are given some interface to work with, such as buttons, display, and so on. You use the interface in order to get the object to do something useful for you, like play a song. How exactly the device is working on the inside, you don't know, and, most often, don't care. In other words, the implementation of the interface is hidden from you. The same thing happens in OOP when your code uses an object by calling its methods. It doesn't matter if you coded the object yourself or it came from some third-party library; your code doesn't need to know how the methods work internally. In compiled languages, you can't actually read the code that makes an object work. In JavaScript, because it's an interpreted language, you can see the source code, but the concept is still the same–you work with the object's interface without worrying about its implementation.

Another aspect of information hiding is the visibility of methods and properties. In some languages, objects can have public, private, and protected methods and properties. This categorization defines the level of access the users of the object have. For example, only the methods of the same object have access to the private methods, while anyone has access to the public ones. In JavaScript, all methods and properties are public, but we'll see that there are ways to protect the data inside an object and achieve privacy.

4. **Aggregation**

Combining several objects into a new one is known as aggregation or composition. It's a powerful way to separate a problem into smaller and more manageable parts (divide and conquer). When a problem scope is so complex that it's impossible to think about it at a detailed level in its entirety, you can separate the problem into several smaller areas, and possibly then separate each of these into even smaller chunks. This allows you to think about the problem on several levels of abstraction.

Take, for example, a personal computer. It's a complex object. You cannot think about all the things that need to happen when you start your computer. But, you can abstract the problem saying that you need to initialize all the separate objects that your Computer object consists of the Monitor object, the Mouse object, the Keyboard object, and so on. Then, you can dive deeper into each of the subobjects. This way, you're composing complex objects by assembling reusable parts.

To use another analogy, a Book object can contain (aggregate) one or more Author objects, a Publisher object, several Chapter objects, a TOC (table of contents), and so on.

5. **Inheritance**

Inheritance is an elegant way to reuse existing code. For example, you can have a generic object, Person, which has properties such as name and date_of_birth, and which also implements the walk, talk, sleep, and eat functionality. Then, you figure out that you need another object called Programmer. You can reimplement all the methods and properties that a Person object has, but it will be smarter to just say that the Programmer object inherits a Person object, and save yourself some work. The Programmer object only needs to implement more specific functionality, such as

the writeCode method, while reusing all of the Person object's functionality. In classical OOP, classes inherit from other classes, but in JavaScript, as there are no classes, objects inherit from other objects.
When an object inherits from another object, it usually adds new methods to the inherited ones, thus extending the old object. Often, the following phrases can be used interchangeably–B inherits from A and B extends A. Also, the object that inherits can pick
one or more methods and redefine them, customizing them for its own needs. This way, the interface stays the same and the method name is the same, but when called on the new
object, the method behaves differently. This way of redefining how an inherited method works is known as **overriding**.

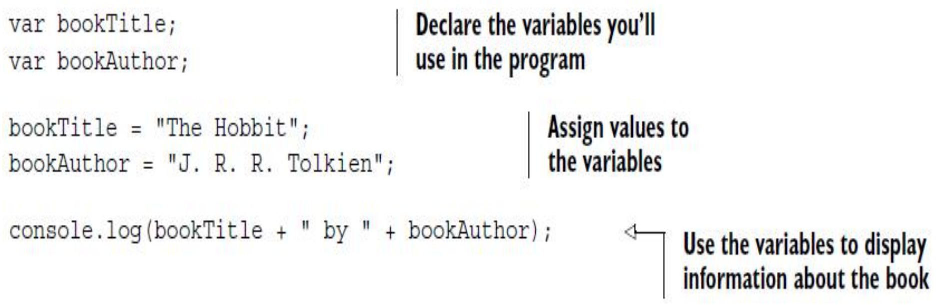
6. **Polymorphism**

In the preceding example, a Programmer object inherited all of the methods of the parent
Person object. This means that both objects provide a talk method, among others. Now imagine that somewhere in your code, there's a variable called Bob, and it just so happens
that you don't know if Bob is a Person object or a Programmer object. You can still call the
talk method on the Bob object and the code will work. This ability to call the same method
on different objects, and have each of them respond in their own way, is called polymorphism.

II. Creating objects

In unit 6 you saw how to declare variables and assign them values. As your programs grow, so does the number of variables you use; you need ways to organize all this data, to make your programs easier to understand and easier to update and add to in the future. Sometimes it makes sense to group items and see them as a whole. JavaScript objects provide a simple and efficient way to collect variables together so that you can pass them around as a group rather than individually. The following listing shows variables you use to generate this test output on the console:

```
> The Hobbit by J. R. R. Tolkien
```



```
var bookTitle;
var bookAuthor;

bookTitle = "The Hobbit";
bookAuthor = "J. R. R. Tolkien";

console.log(bookTitle + " by " + bookAuthor);
```

Declare the variables you'll use in the program

Assign values to the variables

Use the variables to display information about the book

Figure 8.1: Using variables to represent a book

NB: you can replace console.log() by alert().

First, you declare two variables, bookTitle and bookAuthor, using the var keyword. You're going to use those two names to store and access values in the program. You then assign strings (text) to your freshly created variables. You wrap the strings in quotation marks so JavaScript doesn't try to interpret them as keywords or variable names. Finally, you log a message to the console. You build the message by using the concatenation operator (the + symbol) to join three strings.
It may be early days but you certainly have more than one book. How can you cope with the variables needed as you buy more? You could have a different prefix for each book. The next listing ups the number of books to three, printing these messages to the console:

```
> There are three books so far ...
> The Hobbit by J. R. R. Tolkien
> Northern Lights by Philip Pullman
> The Adventures of Tom Sawyer by Mark Twain
```

```
var book1Title = "The Hobbit";
var book1Author = "J. R. R. Tolkien";

var book2Title = "Northern Lights";
var book2Author = "Philip Pullman";

var book3Title = "The Adventures of Tom Sawyer";
var book3Author = "Mark Twain";
console.log("There are three books so far...");
console.log(book1Title + " by " + book1Author);
console.log(book2Title + " by " + book2Author);
console.log(book3Title + " by " + book3Author);
```

Declare variables and assign them values in one step

Figure 8.2: Using prefixes to tell book variables apart

This works up to a point. But as the number of books and the number of facts about each book increase, the number of variables is harder to manage. It would be helpful to be able to group all of the information about a book together, using a single variable. It can be easier to ask for book1 rather than book1Title, book1Author, book1ISBN, and so on separately. JavaScript provides us with the ability to create *objects* to group variables. Very specific notation, or *syntax*, is used to define a new object. Let's look at a full example and then break it down into stages. Figure 8.3 shows how to create a book as an object rather than as separate variables.

```
var book;

book = {
    title : "The Hobbit",
    author : "J. R. R. Tolkien",
    published : 1937
};

console.log(book);
```

Figure 8.3: A book as an object

1. **Creating an empty object**

In unit 6 you saw that variables can be declared but not assigned a value until later in a program. You might have to wait for some user input or a response from a server or a reading from a sensor before you know the value you want to assign to the variable. In the same way, you can create an object with no properties, knowing that properties will be added at some point in the future. To create an object, use curly braces, as in the following listing.

```
var book;

book = {};
```

Declare a variable by using the var keyword

Create an object by using curly braces; assign it to the variable

Figure 8.4: creating an empty object

You create an empty object, one with no properties, and assign it to the variable book. It's not much use without any properties, and you'll see how to add new properties to an existing object in the next sections. But how would you create your book object with properties in place?

2. **Properties as key-value pairs**

The book in figure 8.3 includes three properties: its title, its author, and its year of publication. The values of those properties are "The Hobbit", "J. R. R. Tolkien", and 1937. In a JavaScript object, the names of the properties are called *keys*. For book the keys are title, author, and published. When creating an object, you add a property by including its key and value, separated by a colon, between the curly braces. Figure 8.4 shows a property definition.

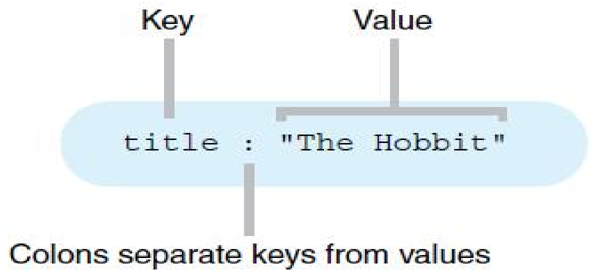


Figure 8.4: Set properties by using key-value pairs

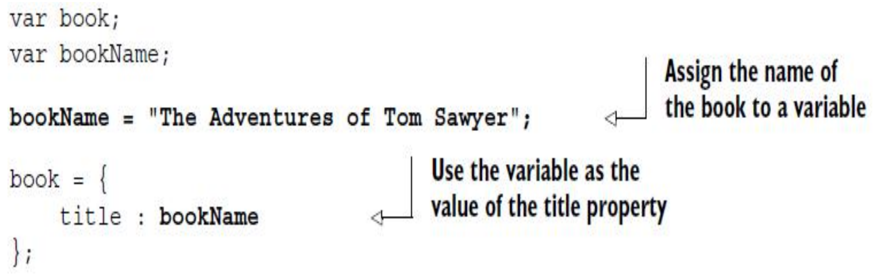
Another name for a key-value pair is a name-value pair, but we'll stick with keys and values in this book. In the next figure you create an object with a single property.

```
var book;

book = {
    title : "The Hobbit"
};
```

Figure 8.5: An object with a single property

You declare a variable and then create an object and assign it to the variable. The object has a single property. The key of the property is title and its value is "The Hobbit". We usually simply say that the title property of book is "The Hobbit".

Property values aren't restricted to number and string literals, like 50 or "The Hobbit". You can also use previously declared variables as values. The following figure assigns the name of a book to a variable and then uses that variable as the value of an object's property.

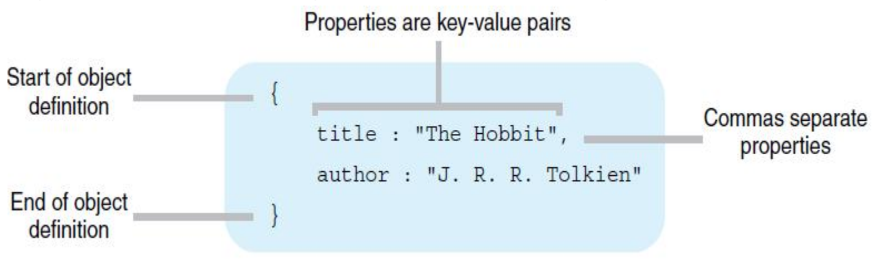


```
var book;
var bookName;

bookName = "The Adventures of Tom Sawyer";

book = {
    title : bookName
};
```

Assign the name of the book to a variable
Use the variable as the value of the title property

Figure 8.6: using a variable as a property value.

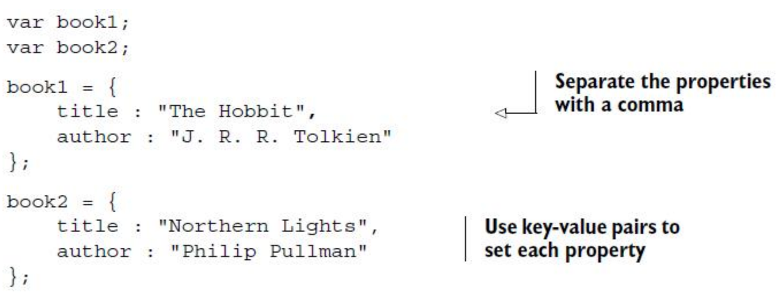
Having an object with a single property is a little extravagant; you might as well stick with a variable. Let's see how to create an object with more than one property. When you need multiple properties, commas separate the key-value pairs. Figure 8.7 shows two properties as part of an object definition, and figure 8.8 creates two objects, each with two properties.



```
{
    title : "The Hobbit",
    author : "J. R. R. Tolkien"
}
```

Properties are key-value pairs
Start of object definition
Commas separate properties
End of object definition

Figure 8.7: An object definition with two properties.



```
var book1;
var book2;

book1 = {
    title : "The Hobbit",
    author : "J. R. R. Tolkien"
};

book2 = {
    title : "Northern Lights",
    author : "Philip Pullman"
};
```

Separate the properties with a comma
Use key-value pairs to set each property

Figure 8.9: Objects with multiple properties.

Now that you've created an object, you need to be able to access its properties.

3. **Accessing object properties**

For JavaScript objects, to access the values of an object's properties you can use *dot notation*. Join the name of the variable to the name of the property, its key, with a period or dot. For books, to access the author property of the object assigned to the variable called book, you write book.author (figure 8.10).

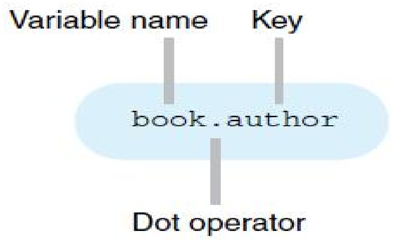
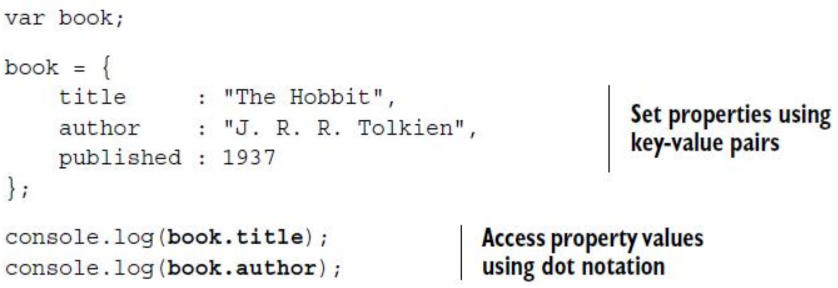


Figure 8.10: Accessing object properties using dot notation

In the next figure, we print the title and author properties of the book object to the console to give the following output:

```
> The Hobbit
> J. R. R. Tolkien
```



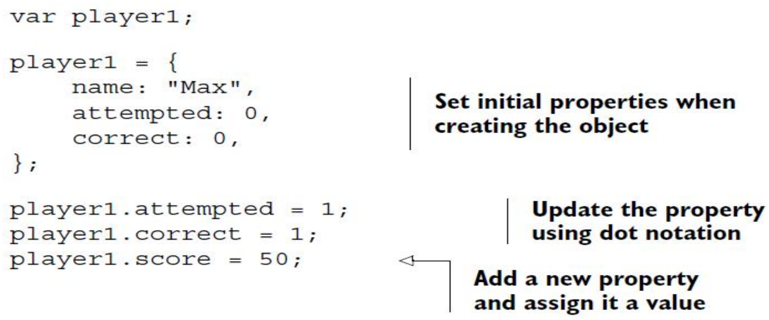
```
var book;

book = {
    title     : "The Hobbit",
    author    : "J. R. R. Tolkien",
    published : 1937
};

console.log(book.title);
console.log(book.author);
```

Set properties using key-value pairs
Access property values using dot notation

Figure 8.11: using dot notation to access property values.

4. **Updating object properties**

In a quiz app, players attempt questions one after another. The number of questions attempted, number of questions correct, and score will change over time. You can create a player object with initial values set and then update them whenever a question is attempted. Use dot notation to change a property that already exists or to add a new property to an object, as in the following figure 8.12.



```
var player1;

player1 = {
    name: "Max",
    attempted: 0,
    correct: 0,
};

player1.attempted = 1;
player1.correct = 1;
player1.score = 50;
```

Set initial properties when creating the object
Update the property using dot notation
Add a new property and assign it a value

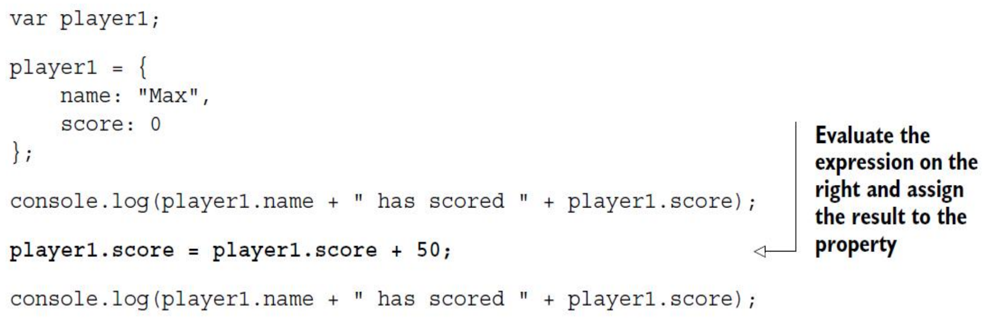
Figure 8.12: Using dot notation to update a property.

Our code in figure 8.12 sets the attempted and correct properties to an initial value when the object is created but then updates them to a new value. It uses the assignment operator, =, to assign the value, 1, on the right of the operator, to the property, player1.attempted, on its left. We set the attempted and correct properties and then immediately update them; in the actual quiz app, the change would be in response to the player answering a question. We can add new properties to an object after creating it. In Figure 3.12, we assign the value 50 to the score property of the player1 object.

```
player1.score = 50;
```

We didn't set the score property when creating the object; assigning a value automatically creates the property if it does not yet exist. Just like using variables, you can use properties in a calculation and assign the result back to the property. The next figure shows code updating a player's properties:

```
> Max has scored 0
> Max has scored 50
```



```
var player1;

player1 = {
    name: "Max",
    score: 0
};

console.log(player1.name + " has scored " + player1.score);

player1.score = player1.score + 50;

console.log(player1.name + " has scored " + player1.score);
```

Evaluate the expression on the right and assign the result to the property

Figure 8.13: Using a property in a calculation.

When you update the score property (in bold in the listing), JavaScript evaluates the right side of the assignment first. Because player1.score is 0, the expression becomes 0 + 50, which is 50. JavaScript then assigns that value to the left side, that is, back to the score property. So, we update player1.score from 0 to 50.

Assignment

1. After providing a comprehensive definition of OOP, state the six key concepts related to OOP and write shot notes on each of them.
2. A blog is made up of blog posts. It would be good to have more information about each author, to be able to tag posts with keywords, and to add comments to each post. Write the code creating a minimal object to represent a single post.
3. Write a JavaScript code to display the title, the author, the body and the date of creation of the above mention post.
4. Create a javascript object representing a quiz with the following features:

- A question
- Four proposed answers to the question
- The number of marks allocated

5. Write a complete quiz application with at least 10 questions and integrate your application in a web page. Add some functionalities that can enable the user to visualize the final mark at the end of the session.
6. Create a form collecting relevant information that can identify a student and display your output using object and functions.
7. Write a program in JavaScript that reads the radius of 10 circles and calculate the area of each. The program should print the radius as well as the area of each circle. Using oop.

Write a program in JavaScript oop that solves the following equations:

8. $Ax+B=0$, where A and B are real numbers and x the unknown value.
9. $Ax^2+Bx+C=0$, where A, B and C are real numbers and x the unknown value.
10. $\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$ Where a, b, c, d ,e and f are real numbers, but x and y the unknown values.