

# HTML, CSS, and JavaScript Training



# Table of Contents

## Chapter 1 : Exploring JavaScript in the Console

- Installing Atom
- Chrome Developer Tools
- Our first program
- Why do we use Chrome Developer Tools?
- Why do we use Atom as the text editor?
- Exercise
- Summary

## Chapter 2 : Solving Problem Using JavaScript

- Variables
- Comments
- Arithmetic operators
- More operators and operations
- Summary

## Chapter 3 : Introducing HTML and CSS

- HTML
- CSS
- JavaScript on an HTML page
- Summary

## Chapter 4 : Diving a Bit Deeper

- Diving a Bit Deeper
- JavaScript methods
- HTML buttons and form
- If statement
- Switch-case
- Loops
- Summary

## **Chapter 5 : Ahoy! Sailing into Battle**

- The HTML part
- The CSS part
- The JavaScript part
- The final code
- Summary

## **Chapter 6 : Exploring the Benefits of jQuery**

- Exploring the Benefits of jQuery
- Installing jQuery
- Explaining the code
- Going deeper
- Summary

## **Chapter 7 : Introducing the Canvas**

- Implementing canvas
- Adding JavaScript
- Drawing a rectangle
- Drawing a line
- A quick exercise
- Drawing a circle
- Draw linear gradient
- A quick exercise
- Let's make a clock!
- Summary

## **Chapter 8 : Tidying Up Your Code Using OOP**

- Tidying up Your Code Using OOP
- Inheritance in JavaScript
- Encapsulation in JavaScript
- Dissecting Hangman
- Summary

## Chapter 1. Exploring JavaScript in the Console

[browser console](#)

Before we start talking about lines of codes, objects, variables, and so on, we need to know what JavaScript is. JavaScript is a programming language that is used to add interactivities to the web pages and build web applications. Static websites are not very popular these days, therefore, we use JavaScript to make our websites interactive.

Some people also call it a scripting language as it is an easy language and does not require compilers like other languages. JavaScript was not designed as a general purpose programming language, it was designed to manipulate web pages. You can write a desktop application using JavaScript. JavaScript can also access your machine's hardware. You can try making a desktop application with a **software development kit (SDK)** such as PhoneGap for mobile or the Microsoft app SDK for desktop. The JavaScript codes are interpreted on web pages and then run by a browser. Any modern Internet browser, for example Firefox, Safari, Google Chrome, UC Browser, Opera, and so on, supports JavaScript.

### Note

A *compiler* is a computer program that processes codes and turns them to machine language. Making a website *interactive* means adding features that are controlled by the users to the website. For example, online registration forms, online calculator, and so on. The *Static* website has fixed objects and contents and it displays the same information to all the visitors.

Basically, JavaScript is included on an HTML page or written on a separate file that has a `.js` extension. If you know nothing about HTML, don't worry as you will learn about it in [Chapter 3, Introducing HTML and CSS](#). So, where can you use JavaScript?

The answer is simple, you can do the following:

- You can create an active user interface.
- You can control web browsers.
- You can validate user inputs (if they are typed wrong).
- You can create custom web pages that can pop up on the browser, holding information or images.

- You can create dynamic pages without **Common Gateway Interface (CGI)**. CGI is used by the web servers to process a browser's information.

## Note

The thing that you should remember is JavaScript is not Java, the programming language developed by Sun Microsystem.

[Visual Studio Code](#)

Throughout this book, we will use **Google Chrome** as the default browser and **Atom** as the text editor.

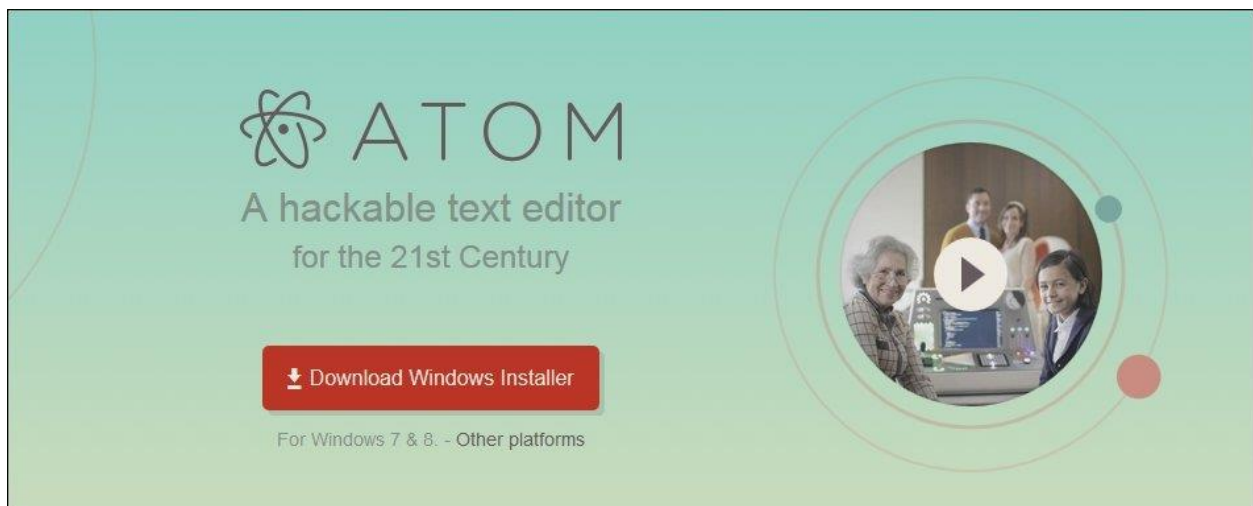
If you do not have these two software already installed on your computer, it is necessary to download and install them.

We will use the Atom text editor as it is a cross-platform editor, has a built-in package manager, does smart autocompletion, and has a lot of other advantages.

## Installing Atom [vsc](#)

---

To install the Atom text editor, follow the <https://atom.io/> link and press **Download Windows Installer**, as shown in the following screenshot:



A file called **AtomSetup.exe** will start downloading.

Click on the `AtomSetup.exe` file to get started with installing Atom.

### Tip

Make sure that you give the administrative rights while installing it for better performance.

Atom will launch automatically after the installation is completed.

If you are on another platform, use the **Other platforms** link:

- If you are a Mac user, go to the <https://github.com/atom/atom/releases/latest> link and download the `atom-X.X.X-full.nupkg` file, where `X.X.X` is the version number of Atom. Install it by double-clicking on the file.
- If you are an Ubuntu user, you can follow the <https://github.com/atom/atom/releases/latest> link and download the `atom-amd64.deb` file. After downloading it, launch your **Terminal** in the same folder, where you placed the file after downloading it. Then, write the following code:

```
• sudo dpkg --install atom-amd64.deb
```

You may need the administrative password to install it. After the installation is complete, you can run Atom from the Terminal by typing `Atom` and pressing `Enter`.

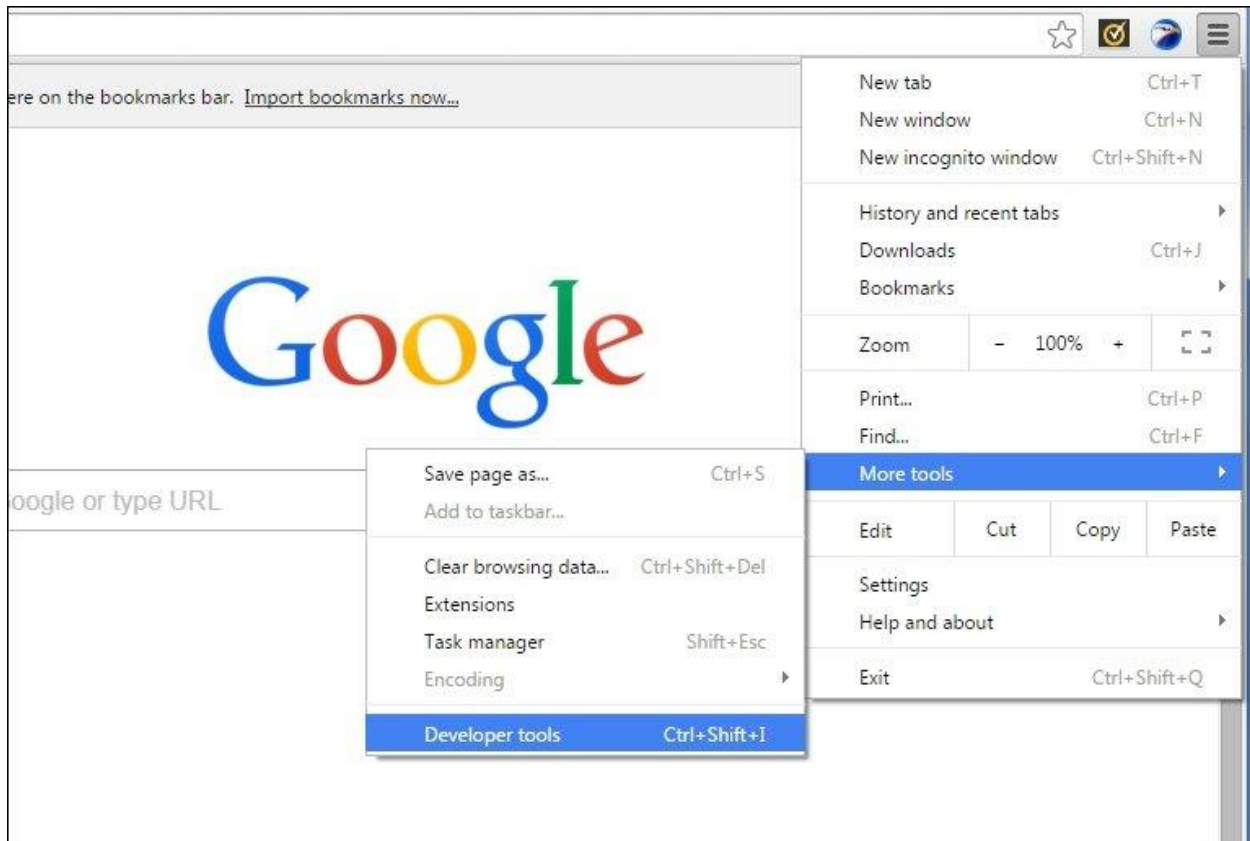
## Chrome Developer Tools

F12

Right click, inspect  
Menu burger

ctrl + U = view source code

Let's take a look at the **Chrome Developer Tools** that are used for JavaScript, specially the *console*. Since Google Chrome is downloaded and installed on your machine, open the Google Chrome browser, go to the menu (on the right-hand top corner), hover on **More tools** and select **Developer tools**, as shown in the following screenshot:



You will see the following tools:

- **Elements**
- **Network**
- **Sources**
- **Timeline**
- **Profiles**
- **Resources**
- **Audits**
- **Console**

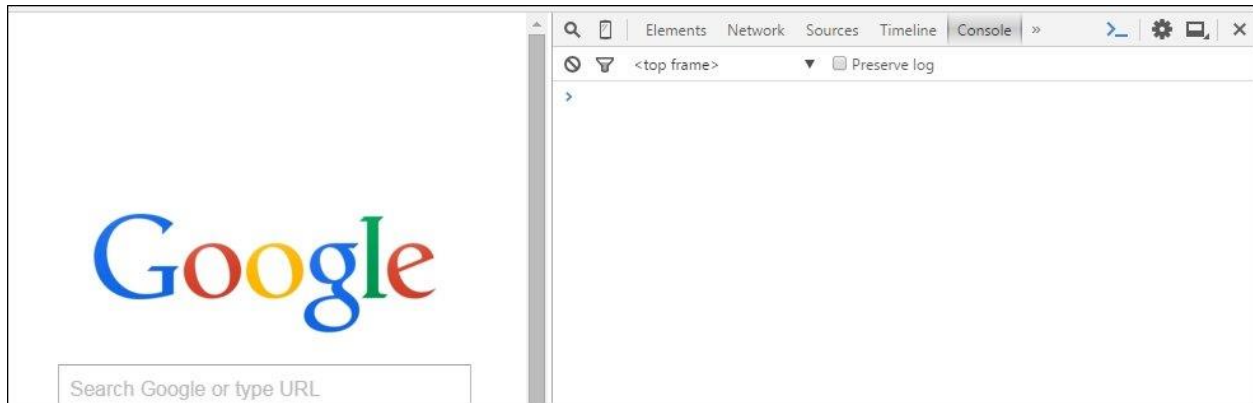
## Our first program

---

Now, let's check whether JavaScript works on your machine.

---

From the tools, select **Console**. If you cannot find **Console**, click on the **>>** symbol, as follows:

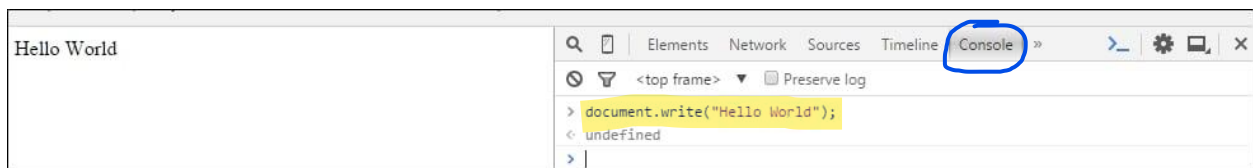


Once your console is open, type the following code and hit *Enter* on your keyboard:

```
document.write("Hello World");
console.log('Hello World');
```

```
System.out.println("Hello World"); // Java
CTR + L = clear console
```

If you can see the output on the left-hand side panel as shown in the following, then you have successfully configured JavaScript on your browser:



The output that you will see is as follows:

**Hello World**

Congratulations!

Note

### Downloading the example code

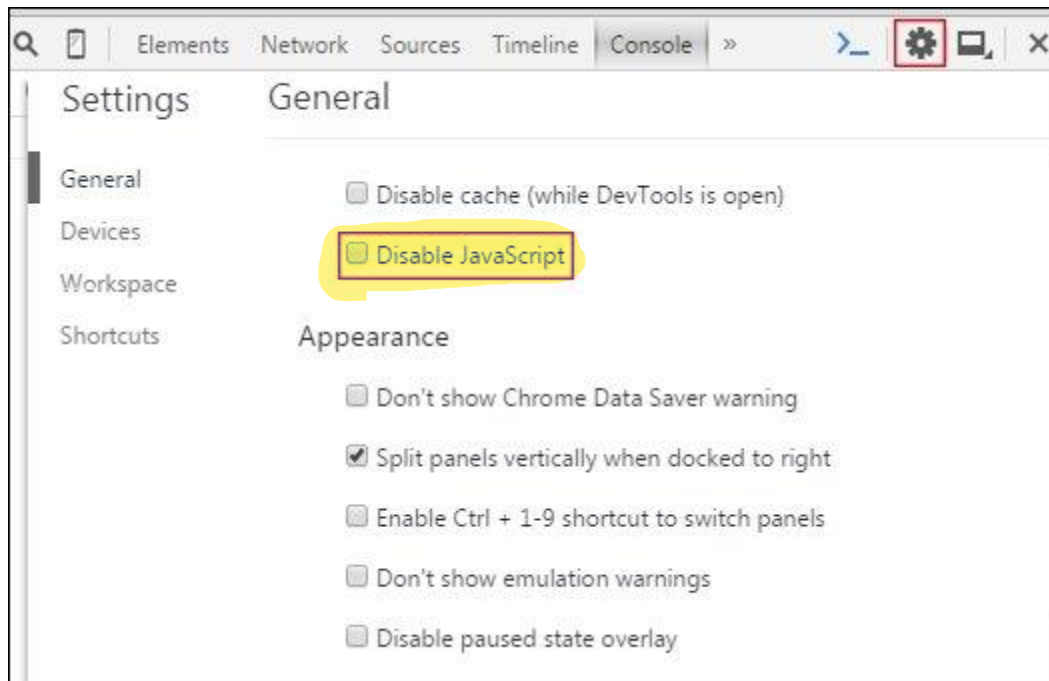
You can download the example code files for all the Packt books that you have purchased from your account at <http://www.packtpub.com>. If you purchased this book



elsewhere, you can visit <http://www.packtpub.com/support> and register in order to have the files e-mailed to you directly.

If you cannot see the text, check your code or install Google Chrome with administrative rights.

You can also click on the gear button of your console. Check whether **Disable JavaScript** is unchecked:



You can also debug your JavaScript codes using this tool.

If you type anything wrong; consider that you forgot the inverted commas of the **Hello World** string, you will get the following errors:

```
> document.write(Hello World)
✖ Uncaught SyntaxError: missing ) after argument list VM1681:2
  at Object.InjectedScript._evaluateOn (<anonymous>:904:140)
  at Object.InjectedScript._evaluateAndWrap (<anonymous>:837:34)
  at Object.InjectedScript.evaluate (<anonymous>:693:21)
  InjectedScript._evaluateOn @ VM1539:904
  InjectedScript._evaluateAndWrap @ VM1539:837
  InjectedScript.evaluate @ VM1539:693
```

To speed up writing your codes, you may learn some keyboard shortcuts for both console and Atom text editor.

Here are few keyboard shortcuts for console:

- *Ctrl + L*: Clear console
- *Tab*: Autocomplete common prefix
- Right arrow: Accept suggestion
- *Ctrl + U*: Clear console prompt
- Up/Down: Next/previous line
- *Enter*: Execute command

Here are few keyboard shortcuts for Atom text editor:

- *Ctrl + B*: Browse list of open files
- *Ctrl + Alt + R*: Reload Atom
- *Ctrl + Shift + L*: Change syntax highlighting
- *Alt + Shift + S*: Show available code snippets
- *Ctrl + Shift + M*: Markdown preview
- *Ctrl + Alt + I*: Toggle Developer Tools
- *Ctrl + N*: New file
- *Ctrl + Shift + N*: New Window
- *Ctrl + P*: Open file (type the name to perform a search)
- *Ctrl + O*: Open file
- *Ctrl + Shift + O*: Open folder
- *Ctrl + S*: Save
- *Ctrl + Shift + S*: Save as
- *Ctrl + W*: Close tab
- *Ctrl + Shift + W*: Close window
- *Ctrl + G*: Go to line
- *Ctrl + L*: Select line
- *Ctrl + Shift + D*: Duplicate line
- *Ctrl + Shift + K*: Delete line
- *Ctrl + Up/Down*: Move line up/down
- *Ctrl + /*: Toggle comment line
- *Ctrl + Enter*: New line below

- *Ctrl + [ ]*: Indent/unindent selected lines
- *Ctrl + J*: Join lines
- *Ctrl + Alt + .*: Complete bracket
- *Ctrl + M*: Go to matching bracket
- *Ctrl + Alt + M*: Select code inside matching brackets
- *Ctrl + Alt + /*: Fold/unfold code
- *Ctrl + Alt + F*: Fold selected code
- *Ctrl + Alt + [ ]*: Fold/unfold all code
- *Ctrl + F*: Find in current file
- *Ctrl + Shift + F*: Find in project
- *F3*: Find next
- *Shift + F3*: Find previous
- *Ctrl + Enter*: Replace all
- *Ctrl + Alt + /*: Use Regex in search
- *Ctrl + Shift + =/ -*: Increase/decrease text size
- *Ctrl + 0* (zero): Reset text size
- *F11*: Toggle fullscreen

## Why do we use Chrome Developer Tools?

---

The following points the use of Chrome Developer Tools:

- Easy to see the errors
- Easy to edit/debug codes using the line numbers
- Real-time output (No need to refresh the page)

## Why do we use Atom as the text editor?

---

The following points the use of Atom as the text editor:

- Zero-compromise combination of hackability and usability
- An open source text editor
- Every Atom window is essentially a locally-rendered web page

## Exercise

---

To enhance your knowledge of JavaScript, write a program that will print your name.

## Summary

---

In this chapter, we saw how to download Google Chrome and Atom, and install them.

You learned how to write your first code using Chrome Developer Tools (**Console**). You have also learned a few keyboard shortcuts for Chrome Developer Tools and Atom text editor.

You also learned what JavaScript is, why learning JavaScript is important, and how JavaScript is different from other languages.

We can now jump in the world of JavaScript.

Your journey begins from [Chapter 2](#), *Solving Problems Using JavaScript*.

## Chapter 2. Solving Problems Using JavaScript

You have learned how to print something using JavaScript on console in the previous chapter. Now, let's see the fundamentals behind JavaScript syntax, variables, arithmetic operators, and comments.

In the computer world, there is nothing but data. You can read, modify, and create new data; however, anything that isn't data simply does not exist. In JavaScript, we need to handle data to develop a website.

To understand the basic syntax of JavaScript, first of all you need to know that JavaScript is *case sensitive*. You cannot interchange lower case and upper case letters in JavaScript. Therefore, when dealing with the JavaScript syntax, you need to remember that writing the code is not the only important task, you must also watch the syntax whether it's written correctly.

Let me show you an example. In the previous chapter, you have successfully printed **Hello World** on your browser using the `document.write();` syntax.

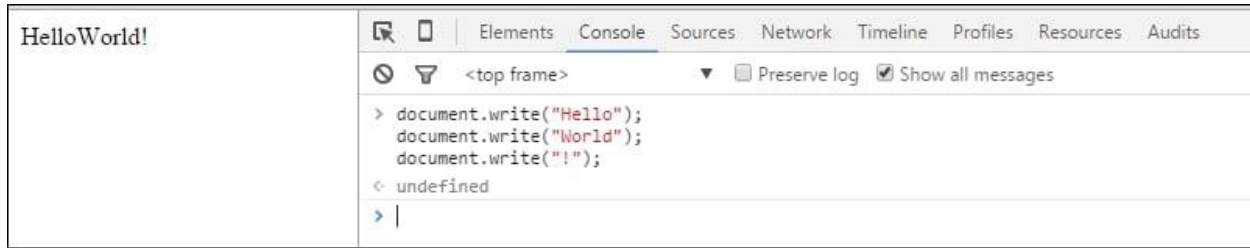
What would happen if you wrote `Document.write("Hello World");`? Yes! It won't run successfully. You will get an error message. This kind of errors is known as **Uncaught SyntaxError**.

A JavaScript statement is typically written on one line. You may finish your statement with a semicolon or not. It is not mandatory to end a statement with a semicolon. However, it is a good practice to add a semicolon after each statement.

Let's consider the following example:

```
document.write("Hello");  
document.write("World");  
document.write("!");
```

Its output will be as follows:



## Note

JavaScript keywords (such as `for`, `while`, `if`, `switch`, `case`, and so on) are always in lowercase. The build-in objects (such as `Date`, `Math`, `Number`, and so on) start with uppercase.

## Variables

We already know that the computer world has nothing but data.

There are different types of data (we call them *data types*), as follows:

- Your name is a kind of data
- Your age is data
- Your grade is also data

Yet, they all are different. What is the difference between them? Your name only contains a group of *characters* or, as some people also call it, **string**. Your age is an **integer** type data. Your grade is a **float** type data. The wonderful thing in JavaScript is that you do not have to specify the data type before writing a *variable's* name.

## Note

JavaScript allows working with three data types. Strings (for example, "This is an example of string"), numbers (for example, 2015, 3.1415, and so on), and Boolean (for example, true or false).

Did we discuss *variables*? Well, you already know the data types. You will need *something* to store your data. This *something* is called *variable*. In JavaScript, we use `var` before the variable names. Remember that `var` starts with small letter.

Let's consider the following example:

```
var x;
var y;
var sum;
var name;
```

Let's say that we have 14 apples and 6 oranges. To store them in variables we will use the following:

```
var apples = 14;
var oranges = 6;
```

The following example is not the same. Can you tell why?

```
var Apples = 14;
var apples = 14;
var APPLES = 14;
var appleS = 14;

let name or
let $name,
let _name
let my123
```

Yes, JavaScript is case sensitive. All the variables are different here, though the values of the variables are the same.

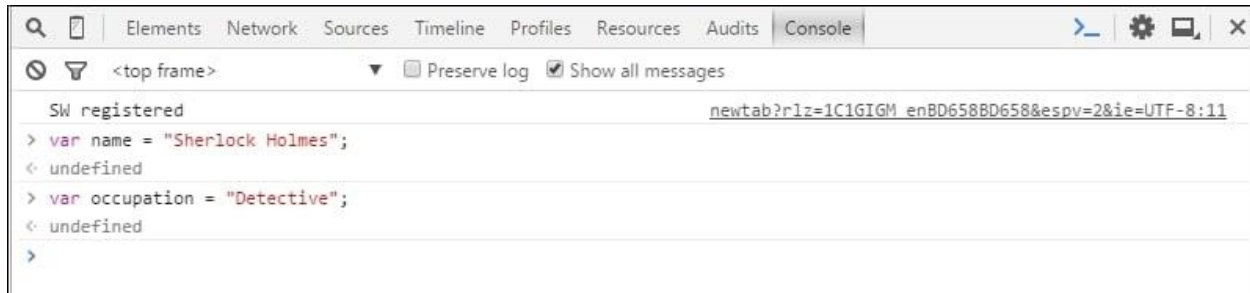
Now, let's do some coding. Previously, on console, you printed your name as homework. I hope you did it without any trouble. How about we now print your name differently using a variable? Assume that your name is `Sherlock Holmes`. What kind of data is it?

You are right, it is *string* type. Usually for string type data, we put the string between two quotes.

Let's consider the following example:

```
var name = "Sherlock Holmes";  
var occupation = "Detective"
```

To print them using console, you need to type each statement and press *Enter*. Take a look at the following image:



## Note

Do not copy and paste the codes on the console. You might get a syntax error.

You will see an extra line appearing after you hit *Enter*, stating *undefined*. Don't worry about this for now. It just returned a console log.

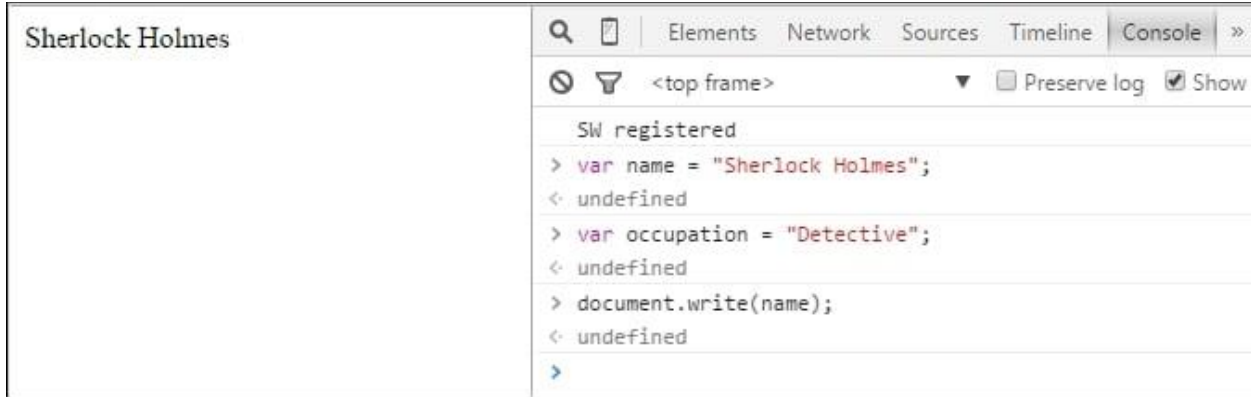
You stored the *Sherlock Holmes* string on the *name* variable and you stored *Detective* on *occupation*. Every time you access *name* or *occupation*, you can access the stated strings.

Consider that you want to print **Sherlock Holmes** on your screen. Just type the following:

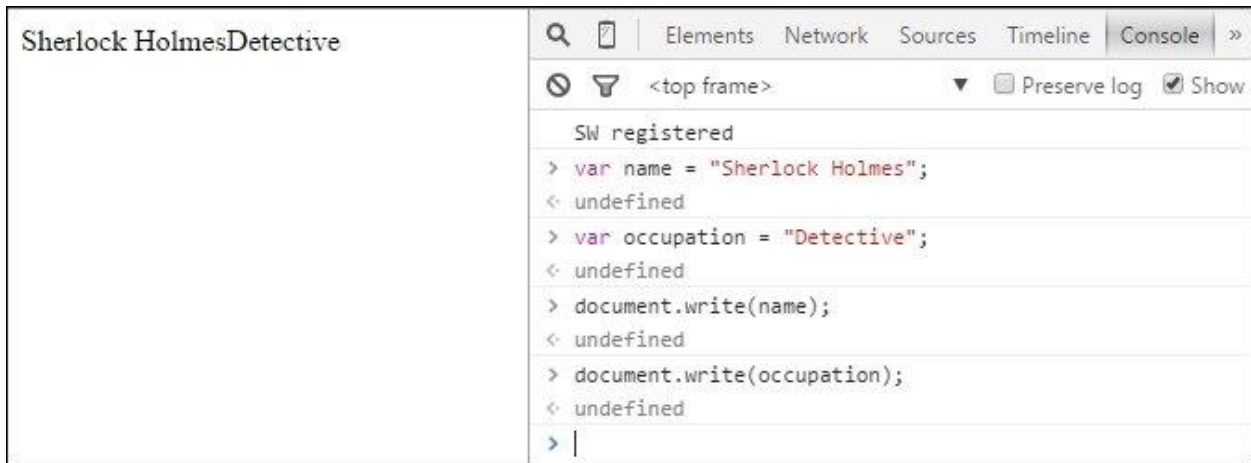
```
document.write(name);
```

After typing, hit *Enter*. You will see **Sherlock Holmes** is printed on the screen, as follows:





Type `document.write(occupation);` and hit *Enter*, as shown in the following screenshot:



You may be wondering why there is no space between **Sherlock Holmes** and **Detective**. As, on the console, the history is not automatically removed from the web page on the left-hand side and after you hit *Enter* for your second output (`occupation`), the string places itself right after the previous string. This will always happen, unless you clear your console using the *Ctrl + L* keyboard shortcut and reload the web page pressing the key *F5*.

#### Note

Your stored variables will also be erased from the memory when you reload the web page. Don't worry, you will be taught how to use your variables storing on a file in the next chapter.

If you want to join two (or multiple) variables, you add a plus sign (+) between the two variables, as follows:

```
document.write(name+occupation);
document.write(occupation+name);
```

Can you tell me what will be output of these commands?

Yes, you are right. The output will be as follows:

**Sherlock HolmesDetective**

**DetectiveSherlock Holmes**

Note

Your output might be in one line on the web page. If you want to split the lines, add a `<br>` HTML tag. The simplest way to add this is to type `document.write("<br>");` and hit *Enter*. Your next output will be in a new line.

If you want to add any string (for example, a space) between the two strings other than any variables, just type the following:

```
document.write(name+" "+occupation);
```

The output will be as follows:

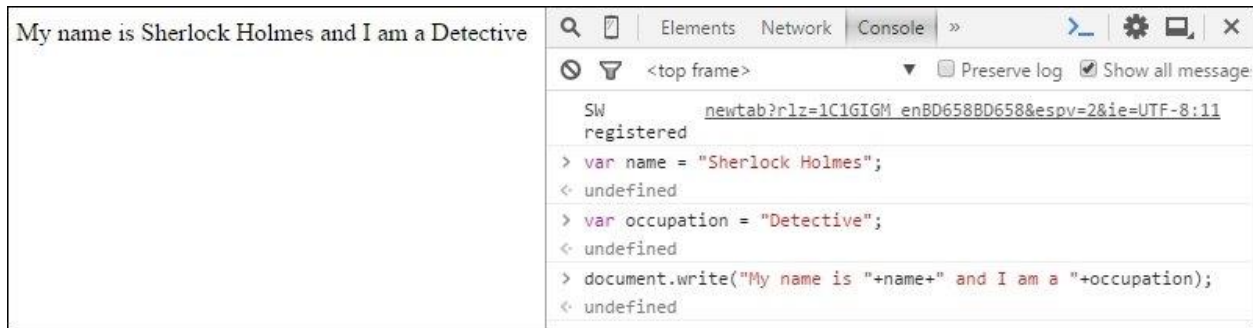
**Sherlock Holmes Detective**

What will happen when you type the following code and hit *Enter*?

```
document.write("My name is "+name+" and I am a "+occupation);
```

Yes! You are absolutely right. The output will be as shown in the following:

**My name is Sherlock Holmes and I am a Detective**



Now, add another variable on the console. Consider that **Sherlock Holmes** is 24 years old. Do you remember what kind of data age is?

Yes, it is an integer type of number. Therefore, type the following code and hit *Enter*.

```
var age = 24;
```

You have the following three variables now:

- Name
- Occupation
- Age

Let's print the following output on the web page:

**My name is Sherlock Holmes, I'm 24 years old and I am a Detective**

What will our code be on the console?

The code is as follows:

```
document.write("My name is "+name+", I\'m "+age+" years old and I am a "+occupation);
```

The output can be seen as follows:



## Tip

### Printing quotations/inverted commas

If you want to print **Shakespeare said, "To be, or not to be: that is the question!"** using the `document.write()` syntax, you will probably type the following code:

```
document.write("Shakespeare said, "To be, or not to be: that is the question!");
```

However, this will give you an error known as **SyntaxError**. To get rid of this error, you need to use a backward slash (`\`) before the two inverted commas. The correct code will be as follows:

```
document.write("Shakespeare said, \"To be, or not to be: that is the question!\");
```

The output will be as shown in the following:

**Shakespeare said, "To be, or not to be: that is the question!"**

The same rule applies for single inverted comma (`'`).

Here is a quick exercise for you:

1. Suppose **Tom** has a cat (**Lucy**). The cat, **Lucy**, is **2.4** years old. Store the name, cat's name, and its age on three different variables and print the following output using console:

**Tom's cat Lucy is 2.4 years old.**

2. Assume that you bought 4 pounds of apples. Each pound costs you \$1.2. Store the price and quantity of apples on two different variables and print the following output using console:

**I bought 4 pounds of apples. I had to pay \$1.2 for each pound.**

## Comments

---

Suppose you have done a lot of coding and some logical operations, and used a number of variables on JavaScript, and you want me to help you with the code if any errors occur. When you send me the code, I will not know what you have typed unless I have a clear knowledge of JavaScript or you have commented on the important lines.

A comment is basically a line of text or code that your browser ignores while running. You can compare comments to sticky notes or reminder.

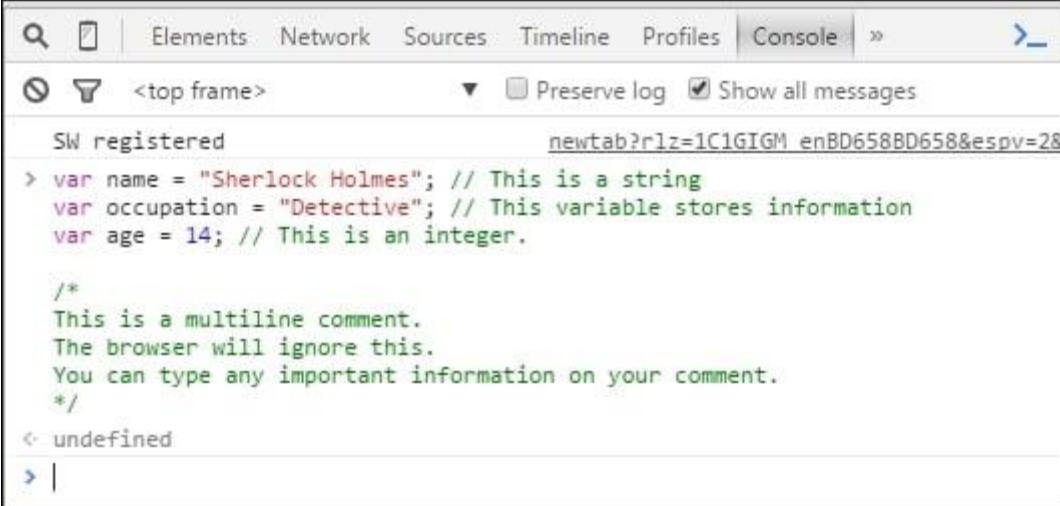
Let's consider the following example:

```
Var name = "Sherlock Holmes"; // This is a string
Var occupation = "Detective"; // This variable stores information
Var age = 14; // This is an integer type of data.
```

How do you make multiline comments? You mention the comment in the following manner:

```
/*
This is a multiline comment.
The browser will ignore this.
You can type any important information on your comment.
*/
```

Your multiline comment should be between `/*` and `*/`, as shown in the following screenshot:



```

SW registered                               newtab?rlz=1C1GIGM_enBD658BD658&espv=2&
> var name = "Sherlock Holmes"; // This is a string
  var occupation = "Detective"; // This variable stores information
  var age = 14; // This is an integer.

  /*
  This is a multiline comment.
  The browser will ignore this.
  You can type any important information on your comment.
  */
< undefined
> |

```

## Arithmetic operators

---

In JavaScript, like other programming languages, we can do some arithmetic operations. In your school, you might have already learned how to add two numbers, subtract one number from another number, multiply two numbers, and divide a number with another. You can do all these things in JavaScript with the help of a few lines of code.

In JavaScript, we use the following arithmetic symbols for the operations:

Operator	Description
+	To add
-	To subtract
*	To multiply
/	To divide

Operator	Description
%	To find the remainder (called modulus operator)

## Addition

Suppose you have two variables, `x` and `y`, with the values `3` and `4`, respectively. What should we do on the console to store the values on the variables?

Yes, we do the following:

```
var x = 3; // 3 is stored on variable x
var y = 4; // 4 is stored on variable y
```

Then, press *Enter*.

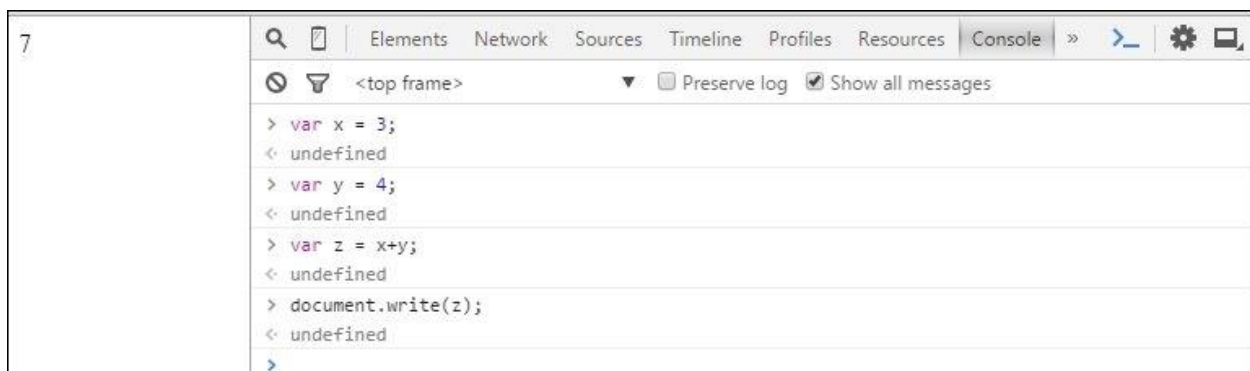
Take another variable that will hold the summation of `x` and `y`, as follows:

```
var z = x+y; // This syntax stores the sum of x and y on z
```

Can you tell me what will happen when we print `z`?

```
document.write(z);
```

Yes, you are correct, this will print `7`, as shown in the following screenshot:



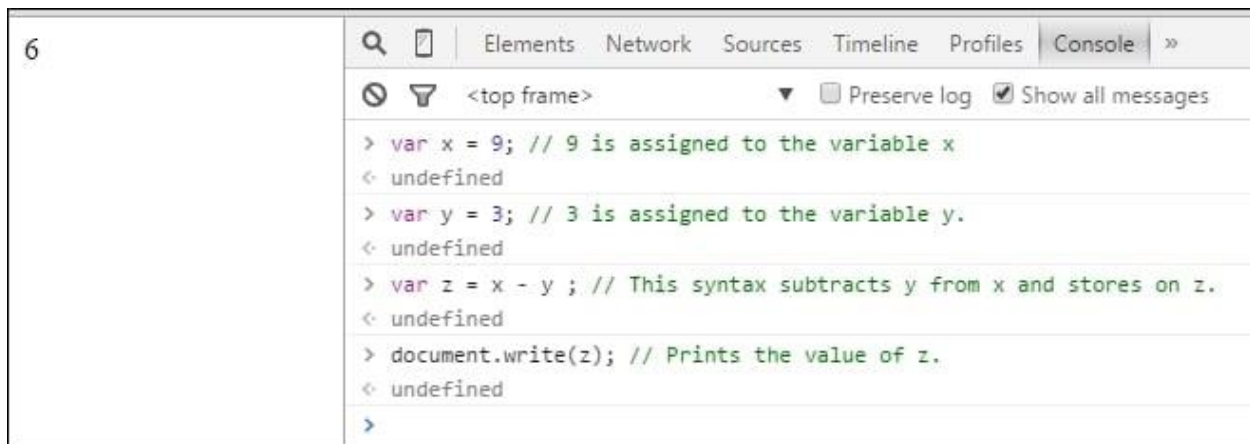
## Subtraction

To subtract a number from another, you need to put a minus sign (-) between them.

Let's consider the following example:

```
var x = 9; // 9 is assigned to the variable x.
var y = 3; // 3 is assigned to the variable y.
var z = x - y ; // This syntax subtracts y from x and stores on z.
document.write(z); // Prints the value of z.
```

The output of this code is **6**, as shown in the following screenshot:



## Multiplication

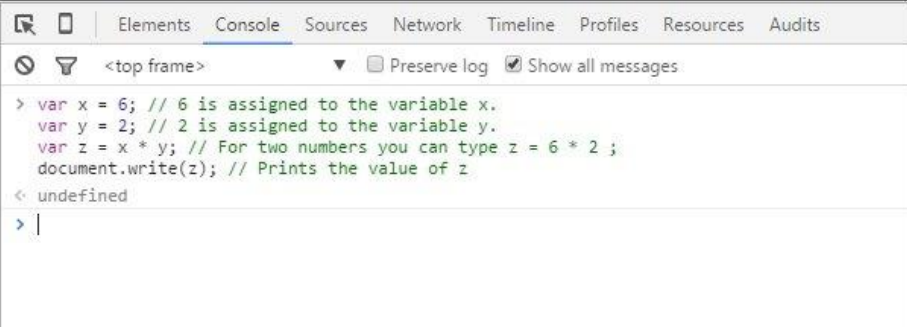
To multiply two numbers or variables that have integer or float type of data stored on them, you just put an asterisk (\*) between the variables or numbers.

Let's take a look at the following example:

```
var x = 6; // 6 is assigned to the variable x.
var y = 2; // 2 is assigned to the variable y.
var z = x * y; // For two numbers you can type z = 6 * 2 ;
document.write(z); // Prints the value of z
```

The output of this code is **12**, as shown in the following screenshot:



12	 <pre> Elements  Console  Sources  Network  Timeline  Profiles  Resources  Audits &lt;top frame&gt; &gt; var x = 6; // 6 is assigned to the variable x.   var y = 2; // 2 is assigned to the variable y.   var z = x * y; // For two numbers you can type z = 6 * 2 ;   document.write(z); // Prints the value of z &lt; undefined &gt;   </pre>
----	--

## Division

To divide a number with another, you need to put a forward slash (/) between the numbers.

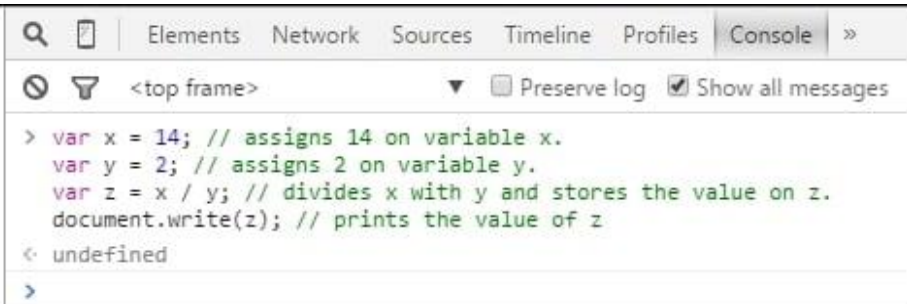
Let's take a look at the following example:

```

var x = 14; // assigns 14 on variable x.
var y = 2; // assigns 2 on variable y.
var z = x / y; // divides x with y and stores the value on z.
document.write(z); // prints the value of z.

```

The output of this code is **7**, as shown in the following screenshot:

7	 <pre> Elements  Network  Sources  Timeline  Profiles  Console  » &lt;top frame&gt; &gt; var x = 14; // assigns 14 on variable x.   var y = 2; // assigns 2 on variable y.   var z = x / y; // divides x with y and stores the value on z.   document.write(z); // prints the value of z &lt; undefined &gt; </pre>
---	---

## Modulus

If you want to find the modulus of a number with another, you need to put a percentage sign (%) between the numbers.

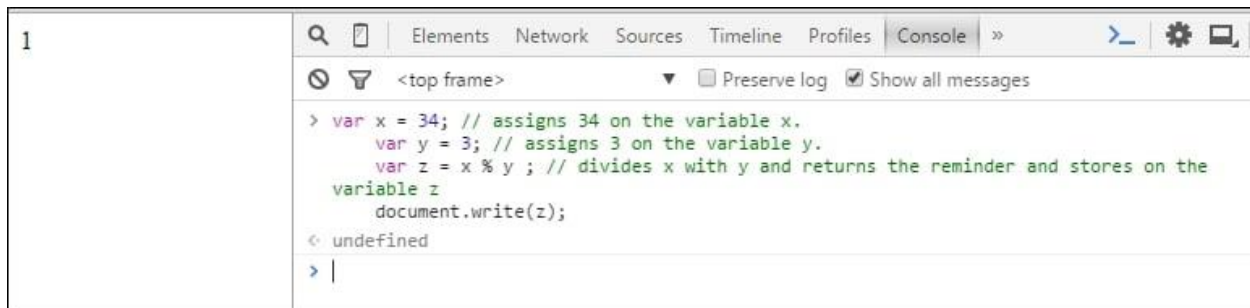
Let's consider the following example:

```

var x = 34; // assigns 34 on the variable x.
var y = 3; // assigns 3 on the variable y.
var z = x % y ; // divides x with y and returns the remainder and stores
on the variable z
document.write(z);

```

The output of this code is **1**, as shown in the following screenshot:



## Tip

### What does modulus (%) operator do?

Well, from your math class, you have already learned how to divide one number with another. Say, you divide 10 by 2. The result will be 5, which is an integer type of number. However, what will happen if you divide 10 by 3? The answer will not be an integer. The value is 3.33333333333333. You can also say that the answer is 3 and the remainder is 1. Consider the following:

$$10 = 9 + 1;$$

$$\text{That is, } (9+1)/3$$

$$= 9/3+1/3$$

$$= 3 + 1/3;$$

Therefore, the remainder is 1. What modulus does is that it finds out the remainder and returns it. Therefore,  $10\%3 = 1$ .

Now, let's summarize all the arithmetic operators that we learned so far in one single code.

Can you tell me the output of the following lines?

```
var x = 5 ;  
var y = 4 ;  
var sum = x + y ;  
var sub = x - y ;  
var mul = x * y ;  
var div = x / y ;  
var mod = x % y ;  
  
document.write("The summation of x and y is "+ sum + "<br>") ;  
document.write("The subtraction of x and y is " + sub + "<br>") ;  
document.write("The multiplication of x and y is " + mul + "<br>");  
document.write("The division of x and y is " + div + "<br>") ;  
document.write("The modulus of x and y is " + mod + "<br>") ;
```

You will get the following output:

**The summation of x and y is 9**

**The subtraction of x and y is 1**

**The multiplication of x and y is 20**

**The division of x and y is 1.25**

**The modulus of x and y is 1**

This output can be seen in the following screenshot:

The summation of x and y is 9  
 The subtraction of x and y is 1  
 The multiplication of x and y is 20  
 The division of x and y is 1.25  
 The modulus of x and y is 1

```

var x = 5 ;
var y = 4 ;
var sum = x + y ;
var sub = x - y ;
var mul = x * y ;
var div = x / y ;
var mod = x % y ;
document.write("The summation of x and y is "+ sum + "<br>");
document.write("The subtraction of x and y is " + sub + "<br>");
document.write("The multiplication of x and y is " + mul + "<br>");
document.write("The division of x and y is " + div + "<br>");
document.write("The modulus of x and y is " + mod + "<br>");

```

I guess you nailed it. Now, let's explain them in the following:

- We assigned 5 and 4 to x and y, respectively
- We assigned the summation of x and y to the sum variable, the subtraction of x and y to the sub variable, the multiplication of x and y to the mul variable, the division of x and y to the div variable, and the modulus of x and y to the mod variable
- Then, we printed them using the document.write(); syntax
- We used a <br> HTML tag to separate the output of each line

Consider the following example:

John has 56 pens. He wants to arrange them in seven rows. Each line will have an equal number of pens. Write a code that will print the number of pens in each row.

(Hint: take two variables for the number of pens and number of rows, divide the number of pens with the number of rows and store the value in a new variable.)

The sample output is as follows:

**John will have to place XX pens on each line. // XX is the number of pens**

## More operators and operations

JavaScript has more operators other than those stated earlier. Let's go little bit deeper.

## Increment or decrement operators

If you have an integer and you want to increment it by 1 or any number, you can type the following:

```
var x = 4; // assigns 4 on the variable x.  
  
x = x + 1;  
  
/* since x=4, and you are adding 1 with x, so the final value is 4 + 1 =  
5, and 5 is stored on the same variable x. */
```

You can also increment your variable by 1, typing the following:

```
var x = 4; // assigns 4 on the variable x.  
  
x++; // This is similar to x = x + 1.
```

What will you do if you want to increment your variable by more than 1? Well, you can follow this:

```
var x = 4; // assigns 4 on the variable x.  
  
x = x + 3; // Say, you want to increment x by 3.  
  
/* since x = 4, and you are adding 3 with x, so the final value is 4 + 3  
= 7, and 7 is stored on the same variable x. */
```

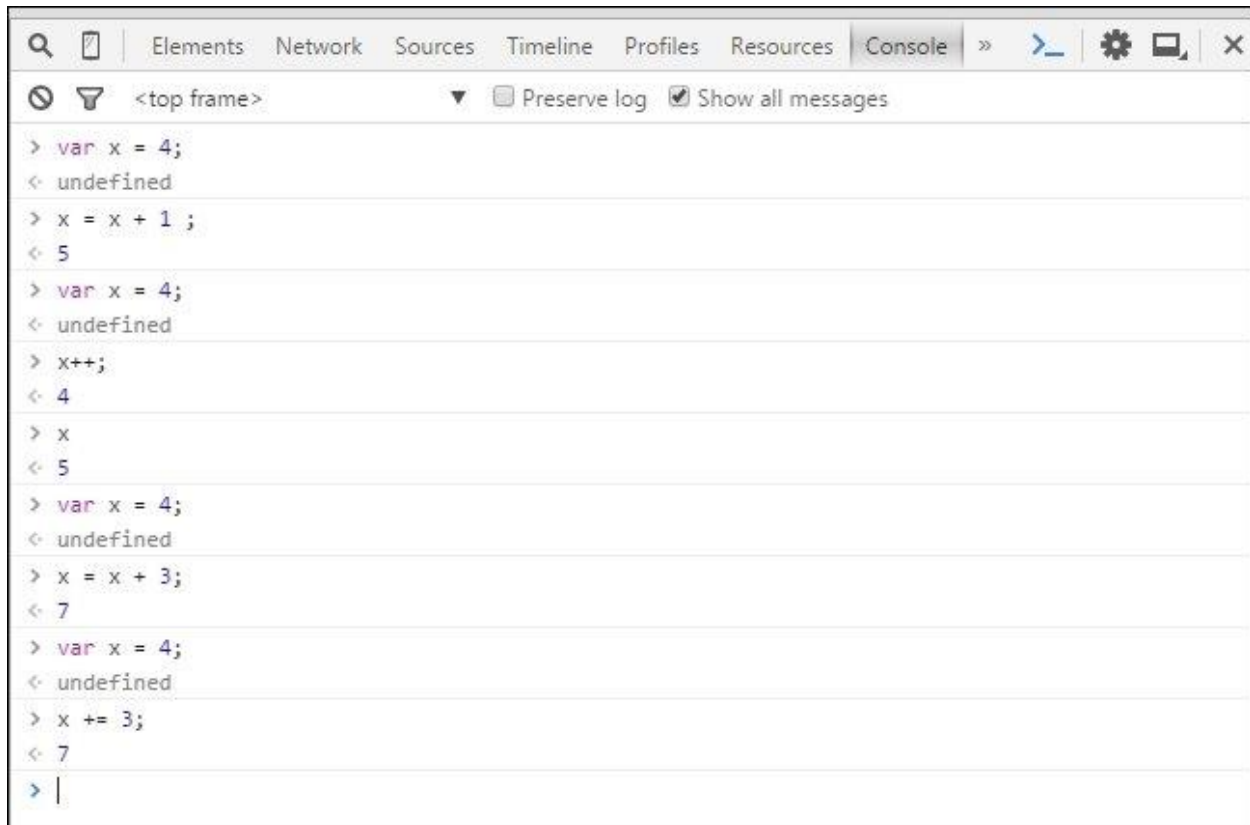
You can increment your variable by typing the following as well:

```
var x = 4; // assigns 4 on the variable x.  
  
x += 3; // This is similar to x = x + 3.
```

### Tip

Remember that you should not place a space between an operator (for example +, -, \*, /, and so on) and equal sign (=).

The output will look similar to the following screenshot on the console:



```

> var x = 4;
< undefined
> x = x + 1 ;
< 5
> var x = 4;
< undefined
> x++;
< 4
> x
< 5
> var x = 4;
< undefined
> x = x + 3;
< 7
> var x = 4;
< undefined
> x += 3;
< 7
> |

```

What about the decrement operator? Yes, you are absolutely right. Decrement operations are same as the increment operations. The only thing that changes is the sign. Your addition (+) operator will be replaced by the subtraction operator (-). Let's take a look at an example:

```

var x = 9; // assigns 9 on the variable x.
x = x - 1;
/* since x = 9, and you are subtracting 1 from x, so the final value is
9 - 1 = 8, and 8 is stored on the same variable x. */

```

You can also decrement your variable by 1 typing the following:

```

var x = 9; // assigns 9 on the variable x.
x--; // This is similar to x = x - 1.

```


What will you do if you want to decrement your variable by more than 1? Well, you can follow this:

```
var x = 9; // assigns 9 on the variable x.  
x = x - 4; // Say, you want to decrement x by 4.  
/* since x = 9, and you are subtracting 4 from x, so the final value is  
9 - 4 = 5, and 5 is stored on the same variable x. */
```

You can also decrement your variable by typing the following:

```
var x = 9; // assigns 9 on the variable x.  
x -= 4; // This is similar to x = x - 4.
```

The output of these codes can be seen in the following screenshot:



The screenshot shows a browser's developer console with the following content:

```
Elements Network Sources Timeline Profiles Resources Console »  
⊘ <top frame> ▼ Ⓞ Preserve log ☑ Show all messages  
> var x = 9;  
< undefined  
> x = x - 1;  
< 8  
> var x = 9 ;  
< undefined  
> x--;  
< 9  
> x  
< 8  
> var x = 9;  
< undefined  
> x = x - 4;  
< 5  
> var x = 9;  
< undefined  
> x -= 4;  
< 5  
> |
```

These type of operations are very important for logical operations in JavaScript. You will learn about their uses in [Chapter 4, Diving a Bit Deeper](#).

## Assignment operators

An assignment operator assigns a value to an operator. I believe that you already know about assignment operators, don't you? Well, you use an equal sign (=) between a variable and its value. By doing this, you assigned the value to the variable.

Let's take a look at the following example:

```
var name = "Sherlock Holmes"
```

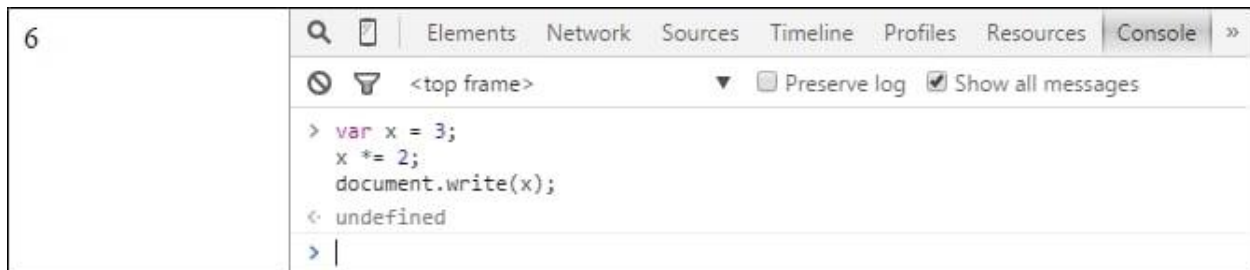
The `Sherlock Holmes` string is assigned to the `name` variable. You have already learned about increment and decrement operators. Can you tell me what will the output of the following codes be?

```
var x = 3;  
x *= 2;  
document.write(x);
```

The output will be **6**.

Do you remember why this has happened?

The `x *= 2;` equation is similar to `x = x * 2;` as `x` is equal to `3`, and later it is multiplied by `2`. The final number (`3 x 2 = 6`) is assigned to the same `x` variable. That's why we got the following output:



Let's perform the following exercise:

What is the output of the following code?



```

var w = 32;
var x = 12;
var y = 9;
var z = 5;

w++;
w--;
x*2;
y = x;
y--;
z%2;

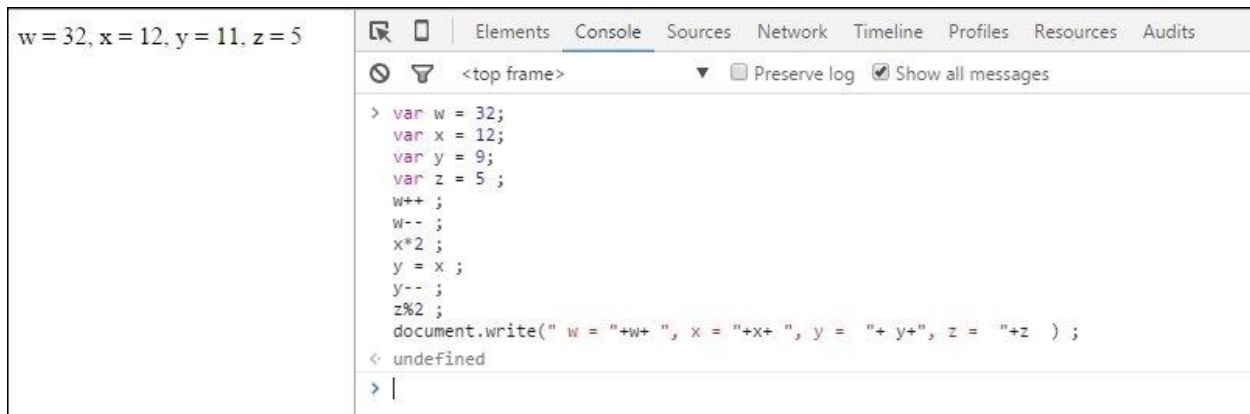
document.write(" w = "+w+ ", x = "+x+ ", y = "+ y+", z = "+z );

```

We will get the following output:

**w = 32, x = 12, y = 11, z = 5**

This output can be seen in the following screenshot:

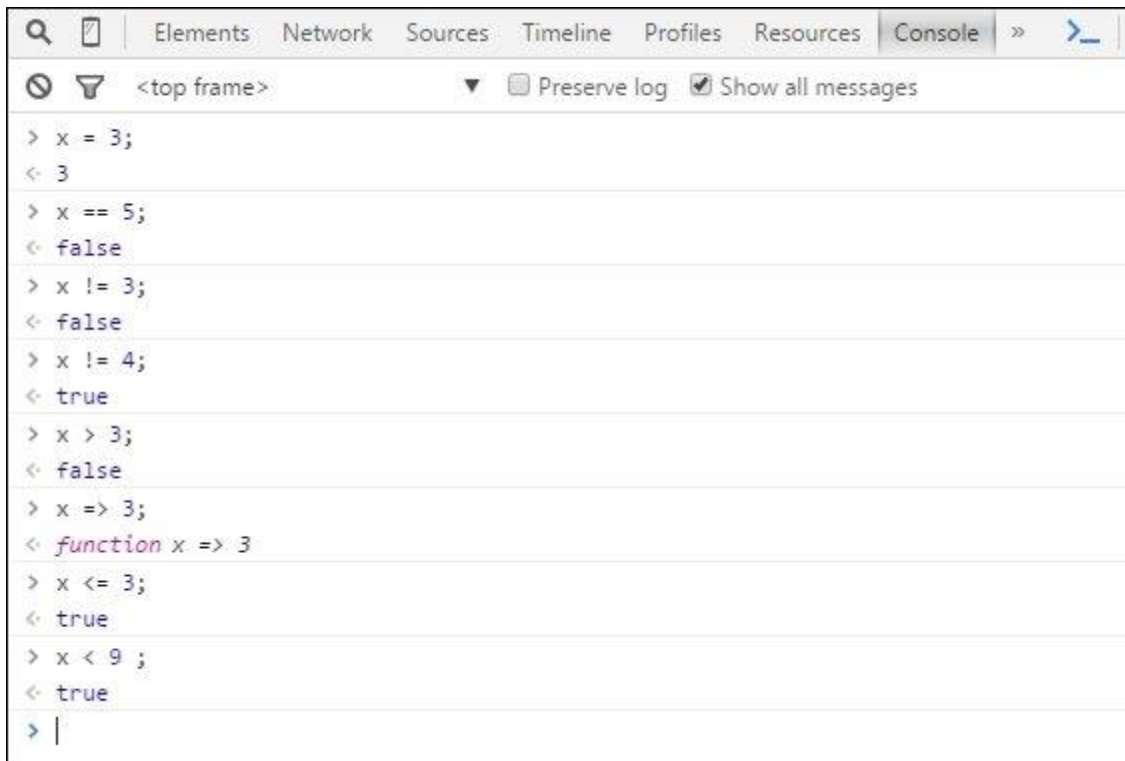


## JavaScript comparison and logical operators

If you want to do something logical and compare two numbers or variables in JavaScript, you need to use a few logical operators. The following are a few examples of the comparison operators:

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>= =>	Equal to or greater than
<=	Less than or equal to

The following are a few examples that use these operators:



```

> x = 3;
< 3
> x == 5;
< false
> x != 3;
< false
> x != 4;
< true
> x > 3;
< false
> x => 3;
< function x => 3
> x <= 3;
< true
> x < 9 ;
< true
> |

```

You will learn more about the use of these operators in the following chapters.

Let's discuss a few bitwise logical operators and bitwise operators:

Operators	Description
<code>&amp;&amp;</code>	This means the AND operator. To check whether two or more statements are true, we use this.
<code>  </code>	This means the OR operator. To check whether any of the statement is true, we use this.
<code>~</code>	This means the NOT operator.
<code>^</code>	This means the XOR operator.
<code>&gt;&gt;</code>	This means the Right Shift operator.
<code>&lt;&lt;</code>	This means the Left Shift operator.

They might be hard for you to learn right now. Don't worry, you don't have to use them now. We will use them in [Chapter 4](#), *Diving a Bit Deeper*.

## Summary

---

In this chapter, you learned about the JavaScript syntax. We discussed the JavaScript variables and how to assign a value to a variable. You learned how to comment on the code. You now know why commenting is important. You finally learned an important topic: operators and operations. JavaScript, without using operators and logical functions, will not be so rich nowadays. Therefore, learning about the logical operations is the key to gain good knowledge of JavaScript.

I would like to suggest you to practice all the code in this chapter at home. You just type them on the console, avoid copying and pasting the codes. This will hamper with your learning. As a programmer must have a good typing speed, copying and pasting the codes will not improve this skill. You may face problems in typing codes; however, you will learn.

You can solve any arithmetic problem using JavaScript. You can also check whether your logic is true or false on console. If you can do this, we can move on to the next chapter, [Chapter 3](#), *Introducing HTML and CSS*, where you will learn about HTML, CSS, and so on.

## Chapter 3. Introducing HTML and CSS

You have already learned about JavaScript syntax, arithmetic operators, and comment in the previous chapter. We used console for these purposes. Now, how about you learn something interesting, which will pave the way for you to be a good JavaScript programmer? In this chapter, we are going to study about the **HyperText Markup Language (HTML)** syntax, **Cascading Style Sheets (CSS)** syntax, and how to use JavaScript in an HTML page.

HTML is the source code of a web page. All the web pages that you load on your web browser are built with HTML. Go to any website (for example, <https://www.google.com>) and press *Ctrl + U* (on Mac, click *command + U*) on your keyboard, you will get the web page's source code. This works on all modern web browsers, such as Firefox, Google Chrome, UC, and so on.

The entire code that you will see is in HTML. You may also find a few lines with JavaScript. Therefore, in order to understand the structure of a web page (the code behind the page), you need to know about HTML. This is one of the easiest languages on the web.

### HTML

---

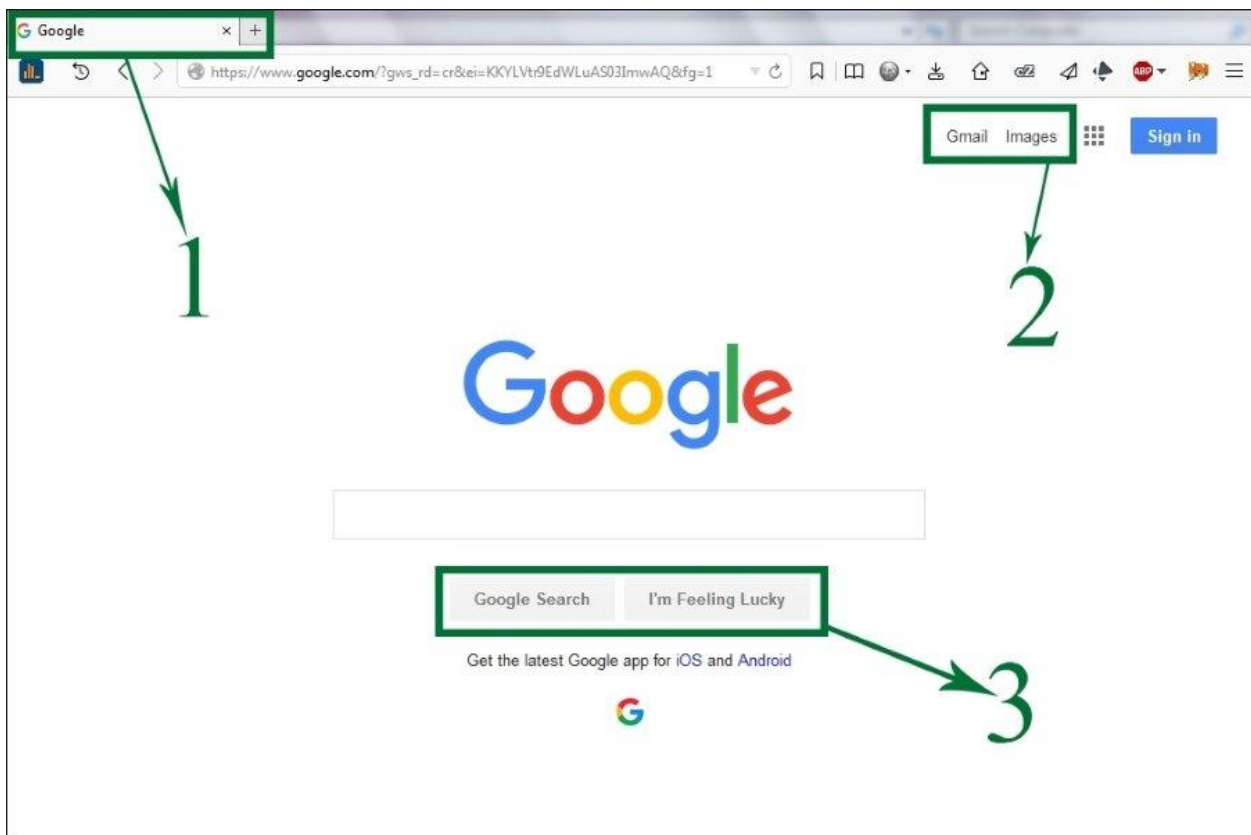
HTML is a markup language. What does it mean? Well, a markup language processes and presents texts using specific codes for formatting, styling, and layout design. There are a lot of markup languages (for example, **Business Narrative Markup Language (BNML)**, **ColdFusion Markup Language (CFML)**, **Opera Binary Markup Language (OBML)**, **Systems Biology Markup Language (SBML)**, **Virtual Human Markup Language (VHML)**, and so on); however, in modern web, we use HTML. HTML is based on **Standard Generalized Markup Language (SGML)**. SGML was basically used to design document papers.

Note

There are a number of HTML versions. HTML 5 is the latest version. Throughout this book, we will use the latest version of HTML.

Before you start learning HTML, let me ask you to think of your favorite website. What does the website contain? A few web pages? You may see some text, few images, one or two text fields, buttons, and some more elements on each of the web pages. Each of these elements are formatted by HTML.

Let me introduce you to a web page. On your Internet browser, go to <https://www.google.com>. You will see a page as shown in the following image:



The first thing you will see on the top of your browser is the title of the webpage. Let's observe the page that we just loaded:

- Here, the marked box, 1, is the title of the web page that we loaded.
- The second box, 2, indicates some links or text.
- The word **Google** in the middle of the page is an image.
- The third box, 3, consists of two buttons.

- Can you tell me what **Sign in** on the right-hand top of the page is? Yes, it is a button.

Let's demonstrate the basic structure of HTML. The term *tag* will be used frequently to demonstrate the structure.

An HTML tag is nothing but a few predefined words between the less than sign (<) and greater than sign (>). Therefore, the structure of a tag is <WORD>, where WORD is the predefined text that is recognized by the Internet browsers. This type of tag is called open tag. There is another type of tag that is known as close tag. The structure of a close tag is similar to </WORD>. You just have to put a forward slash after the less than sign.

After this section, you will be able to make your own web page with some text using HTML. The structure of an HTML page is similar to the following image. This image has eight tags. Let's introduce all these tags with their activities, as shown in the following:

1	<html>-----(1)	
2		tag :
3	<head>-----(2)	<html>
4	<title>-----(3)	<body>
5	↗	<p>
6	</title>-----(4)	
7	</head>-----(5)	<img src="">
8	↗	
9	Element	
10	<body>-----(6)	
11		
12		
13	</body>-----(7)	
14		
15	</html>-----(8)	

- 1: The tag <html> is an open tag and it closes at line 15 with the </html> tag.
  - These tags tell your Internet browser that all the texts and scripts in these two tags are HTML documents.
- 2: This is the <head> tag, which is an open tag and closes at line 7 with the </head> tag.
  - These tags contain the title, script, style, and metadata of a web page.

- **3:** This is the `<title>` tag, and closes at line **4** with the `</title>` tag.
  - This tag contains the title of the web page. The previous image had the title **Google**. To see this on the web browser, you need to type the following:

```
<title> Google </title>
```

- **4:** This is the close tag of the `<title>` tag.
- **5:** This is the closing tag of the `<head>` tag.
- **6:** This is the `<body>` tag, and closes at line **13** with the `</body>` tag.

Everything you can see on a webpage is written between these two tags. Every element, image, link and so on are formatted here. To see This is a web page. on your browser, you need to type the following:

```
<body>
```

```
This is a web page.
```

```
</body>
```

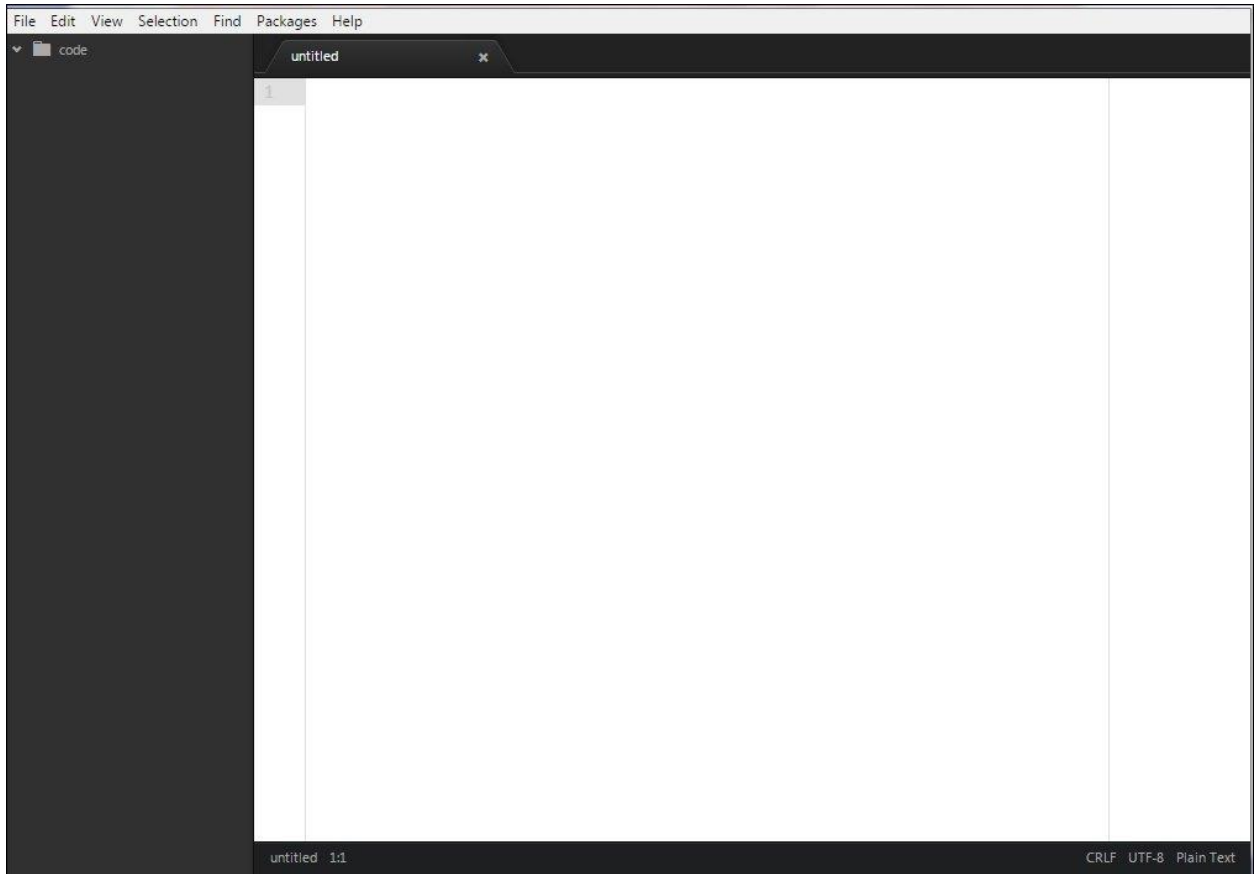
- **7:** The `</body>` tag closes here.
- **8:** The `</html>` tag is closes here.

## Your first webpage

You just learned the eight basic tags of an HTML page. You can now make your own web page. How? Why not try with me?

1. Open your text editor (You have already installed Atom in [Chapter 1](#), *Exploring JavaScript in the Console* of this book).
2. Press `Ctrl + N`, which will open a new `untitled` file as shown in the following image:

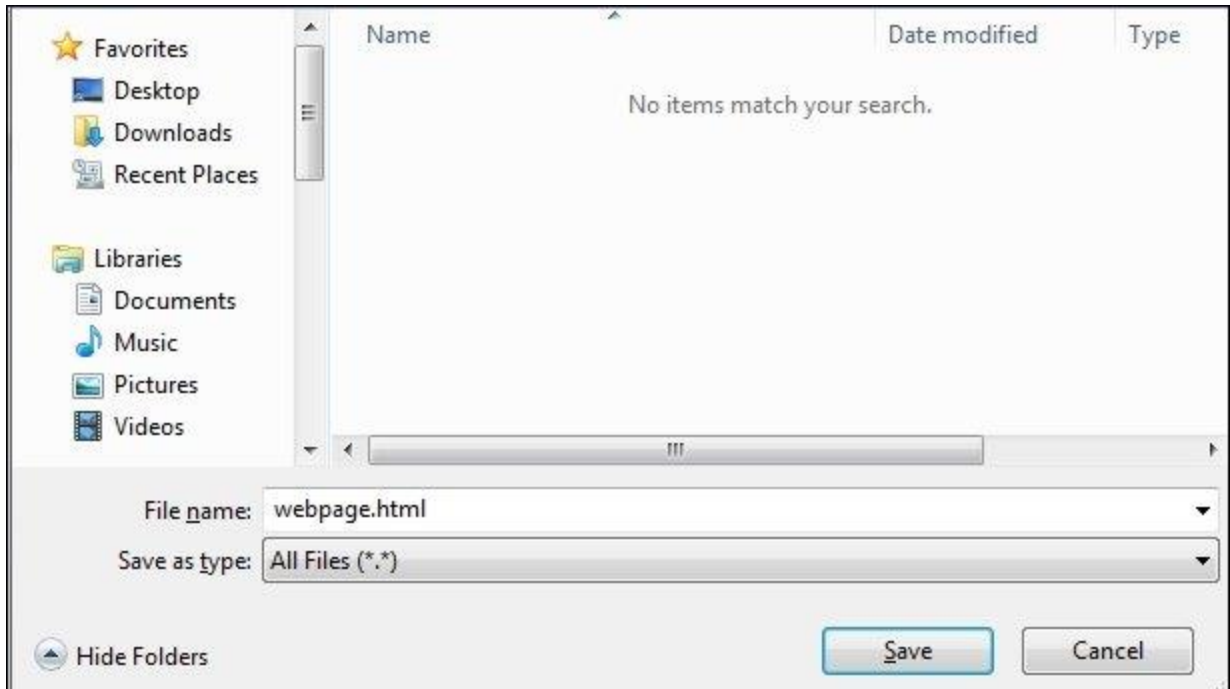




3. Type the following HTML codes on a blank page:

4. `<html>`
5. `<head>`
6. `<title>`
7. `My Webpage!`
8. `</title>`
9. `</head>`
10. `<body>`
11. `This is my webpage :)`
12. `</body>`
- `</html>`

13. Then, press *Ctrl + Shift + S*, which will tell you to save your code somewhere on your computer:



14. Type a suitable name on the **File name:** field. I would like to name my HTML file `webpage`, therefore, I typed `webpage.html`. You may be wondering why I added an extension (`.html`).

#### Note

As this is an HTML document, you need to add `.html` or `.htm` after the name that you give your webpage. The `.htm` extension is an old form of `.html`. It was limited to keep the file extension in three characters, therefore, people used `.htm` instead of `.html`. You can also use `.htm`.

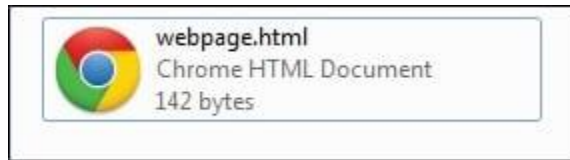
15. Press the **Save** button. This will create an HTML document on your computer. Go to the directory, where you just have saved your HTML file.

#### Note

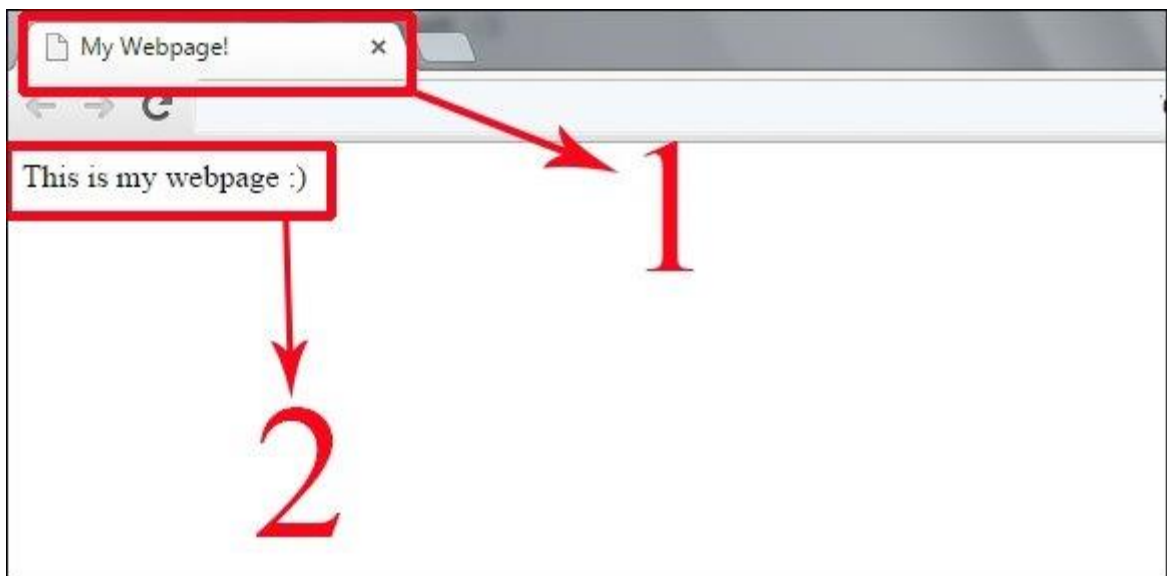
Remember that you can give your web page any name. However, this name will not be visible on your browser. It is not the title of your web page. It is good practice not to keep a blank space in your webpage's name. For example, you want to name your HTML file `This is my first webpage.html`. Your computer will face no problem showing the result on the Internet

browsers; however, when your website will be on a server, this name might face a problem. Therefore, I would suggest you to keep an underscore ( \_ ) where you need to add a space, such as `This_is_my_first_webpage.html`.

16. You will find a file similar to the following image:



17. Now, double-click on the file. You will see your first web page on the Internet browser!



You typed `My Webpage!` between the `<title>` and `</title>` tags, which is why your browser shows this in the first selection box, **1**. You typed `This is my webpage :)` between the `<body>` and `</body>` tags. Therefore, you can see the text on your browser in the second selection box, **2**.

Congratulations! You created your first web page!

Note

You can edit your codes and other texts of the `webpage.html` file by right-clicking on the file and select **Open with Atom**. You must save (`Ctrl + S`) your codes and text before reopening the file in your browser.

### More HTML tags

There are a number of HTML tags to format text and objects of your web page. How about we study a few of them now?

Description	Syntax with example	Result on browser
Bold Text	<code>&lt;b&gt; This is bold &lt;/b&gt;</code>	<b>This is bold</b>
Italic Text	<code>&lt;i&gt; This is italic &lt;/i&gt;</code>	<i>This is italic</i>
Underlined Text	<code>&lt;u&gt; Underline Text &lt;/u&gt;</code>	<u>Underline Text</u>
Deleted Text	<code>&lt;del&gt; Delete me &lt;/del&gt;</code>	<del>Delete me</del>
Subscript Text	<code>CO&lt;sub&gt;2&lt;/sub&gt;</code>	CO <sub>2</sub>
Superscript	<code>3x10&lt;sup&gt;8&lt;/sup&gt;</code>	3x10 <sup>8</sup>
Largest headline	<code>&lt;h1&gt; Hi Kids! &lt;/h1&gt;</code>	<b>Hi Kids!</b>
Smallest headline	<code>&lt;h6&gt; Hi Kids &lt;/h6&gt;</code>	<b>Hi Kids</b>
Paragraph Text	<code>&lt;p&gt;This is a paragraph &lt;/p&gt;</code>	This is a paragraph
Break Tag	<code>This &lt;br&gt;is &lt;br&gt;a break;</code>	This is a break;

## Note

There are six headline tags (<h1> to <h6>). You can add more than one tag for a text if required. For example: <b><i><u> JavaScript </b></i></u> will have the following output: ***JavaScript***. There is no specific order in which you should close the tags. The best practice is to follow the sequence of open tags.

## Coloring HTML text

To color an HTML text, we can type the following:

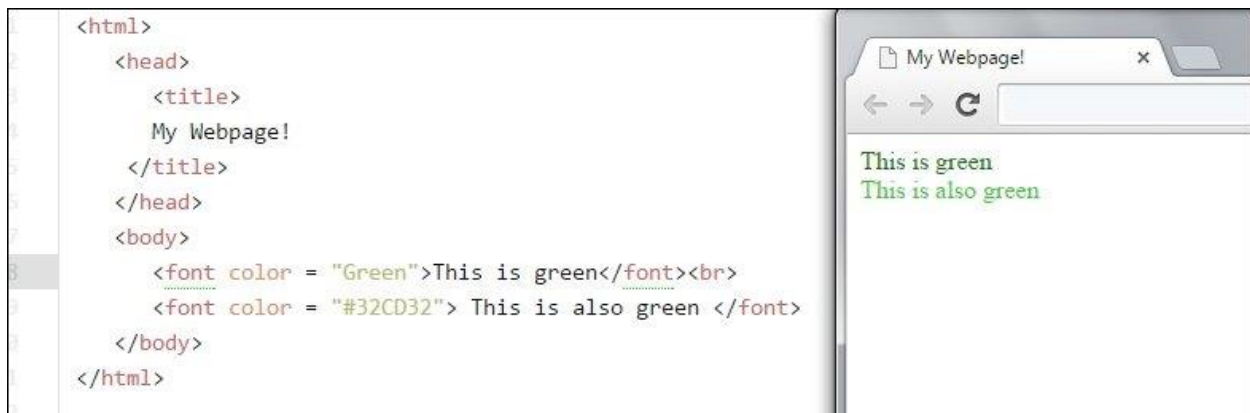
```
<font color = "Green"> I am green </font>
```

attribute / property

You can type any standard color name between the two inverted commas (" "). You can also use hex color code, as follows:

```
<font color = "#32CD32"> I am green </font>
```

Here, **32CD32** is the hex code of green. Look at the following image. The left-hand side is the code, where we used both color name and hex code. On the right-hand side, we got the output of our browser:



## Note

A hex color code consists of six digits (it is a hexadecimal number). It starts with a pound sign or hash sign (#) and we place the six digit hexadecimal number after it. The

hexadecimal number represents red, blue, and green colors' amount. Each two digits represents `00` to `FF` (hexadecimal number). In the example, we used `#32CD32` for green. `32`, `CD`, and `32` are the amount of red, blue, and green; respectively; in hexadecimal.

If you don't know what a hexadecimal number is, remember that we use decimal number where 10 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) are used. However, in hexadecimal numbers, we use 16 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F).

I would recommend you to use this website (<http://html-color-codes.info/>) to get your favorite color's hex code without thinking about the hex code.

### Linking HTML text

To hyperlink a text, we use an anchor tag as follows:

```
<a href = "http://www.google.com"> Go to Google </a>
```

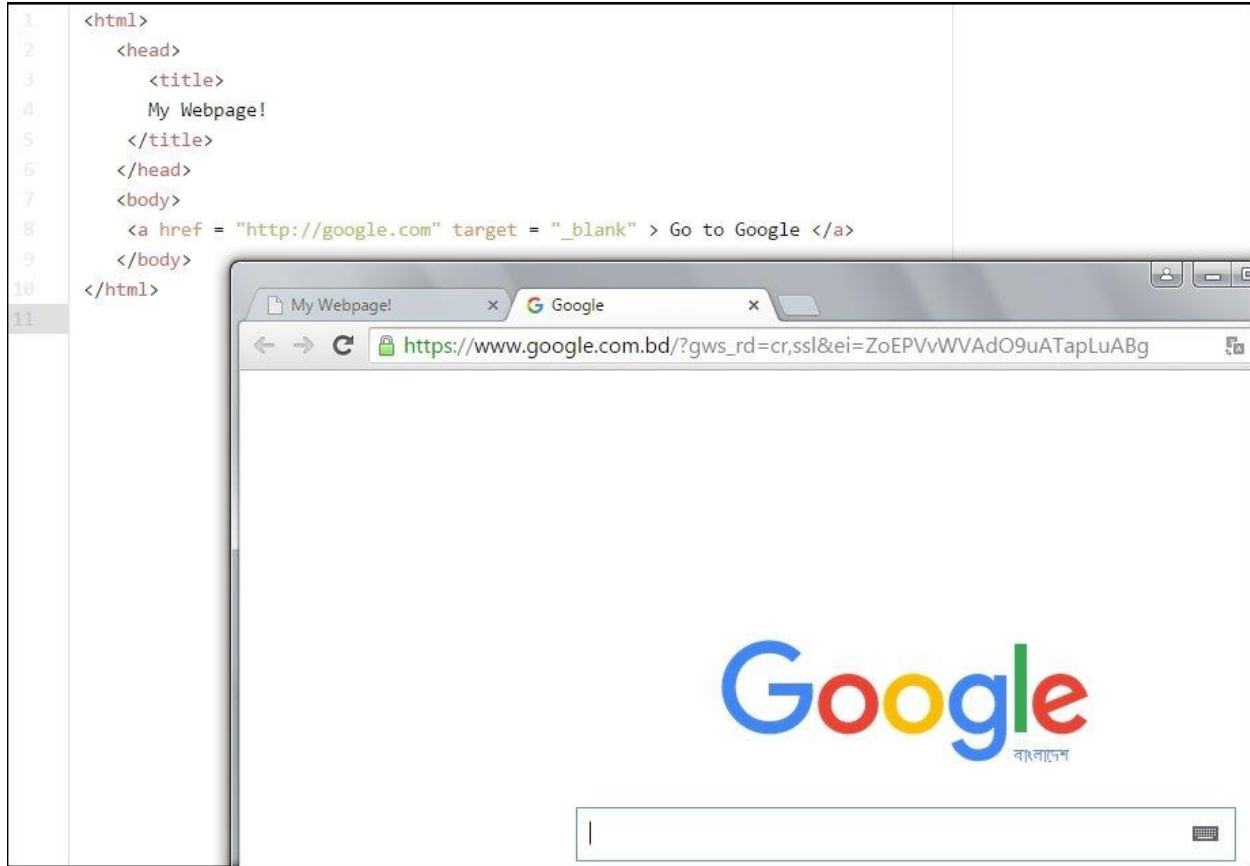
The output of this code will be a link. If you click on the link, it will send you to the URL that we used between the inverted commas (here, <http://www.google.com>).

If you want to open your link in a new tab of your browser, you need to add a target as shown in the following:

```
<a href = "http://google.com" target = "_blank" > Go to Google </a>
```

Here, `target = "_blank"` is an attribute that tells your browser to open the link in a new tab. There are few more attributes. You can try them at home and let us know what you see on your browser.

The other attributes are `_parent`, `_self`, and `_top`. The following image has the code that has the `_blank` attribute. It opens <http://google.com> in a new tab. I would suggest you to find what the other attributes do:



## Inserting an image

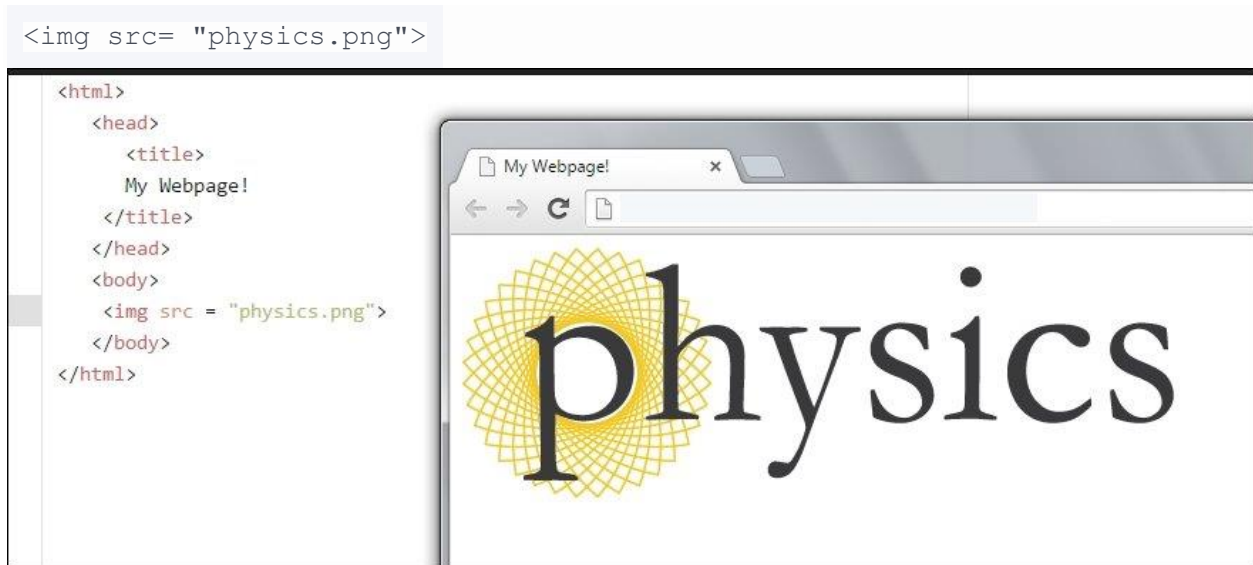
Inserting an image on an HTML document is super easy. You just have to find the image file extensions. The tag that we use to insert an image is as shown in the following:

```
<img src = "Image_name.extension">
```

The **src** attribute is the source of your image. If your image is placed on the same directory of the HTML file, you don't have to write the whole file source. Throughout this book, we will keep our image files on the same directory, where we save our HTML files.

[.gif](#), [png](#), [jpg](#), [jpeg](#), [svg](#)

Let's say that I have an image in the same folder where I saved the HTML document. The name of the image is **physics** and its extension is **.png**. Now, to add this on the HTML document, I need to add the following code:



### Note

We use three types of images on an HTML document. **Portable Network Graphics (PNG)**, **Graphics Interchange Format (GIF)** and **Joint Photographic Experts Group (JPG or JPEG)**. To find your image's extension, right-click on your image, go to **Properties**, and then, click on the **Details** tab to scroll down until you find the **Name** field. You will find the image name with the extension. The procedure might be different on your machine, depending on your operating system.

If you want to set the height and width of the image, you need to use two attributes, as shown in the following:

unit in pixel

```
< img src = "physics.png" width="100" height="40">
```

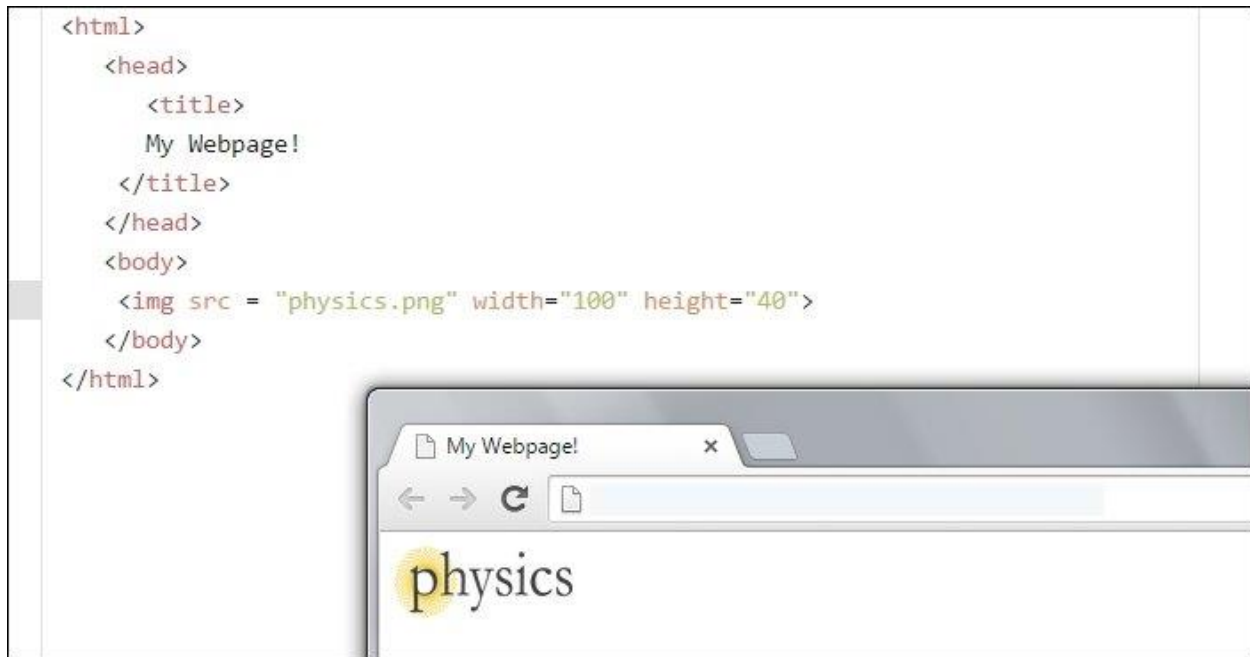
Here, **100** and **40** are the pixel of the image. In the previous versions of HTML, it was defined as pixels or percentage.

### Note

A pixel is the smallest unit of an image. Using percentage (%) is better if you want to see the same ratio of the image on different screen sizes, otherwise, you can use the pixel (px) unit.

The output will look similar to the following:





There are more HTML tags; however, we have covered most of the tags that we use to build a web page. Can you imagine the output of the following codes?

```
<html>
  <head>
    <title>
      Example
    </title>
  </head>
  <body>
    <h1> This is a headline </h1>
    <h2> This is a headline </h2>
    <h3> This is a headline </h3>
    <h4> This is a headline </h4>
    <h5> This is a headline </h5>
    <h6> This is a headline </h6>
```

```
<b>This is a bold text</b>. But <i>This is an italic text</i>. We  
can <u> underline</u> our text. <a href = "http://www.google.com">Go to  
Google </a> <br>  
<font color = "#AA2FF">This is colorful text</font>  
<br>  
  
</body>  
</html>
```

The output will look similar to the following image:



## CSS

If you want to make your web page beautiful, you must know CSS. CSS is a language that allows you to describe your web pages, color your texts, change the font of the text, and modify the layout of the web page.

There are two parts of a CSS syntax:

- Selector

- Decorator

Before proceeding with learning CSS, you need to introduce yourself with an HTML tag:

```
<style>
```

```
</style>
```

This tag should be kept between the `<head></head>` tags. Therefore, the structure of the code will be as shown in the following:

```
<html>
  <head>
    <title>
    </title>
    <style>
      // your codes will be typed here
    </style>
  </head>
  <body>
  </body>
</html>
```

The CSS codes will be written in between the `<style></style>` tags.

To format your text, you need to remember the tag that you used for the text. Consider that you have a text in the `<h1></h1>` tag in the body of the HTML document, as follows:

```
<h1> This is an example of HTML text. </h1>
```

To apply CSS, you need to type the following between the `<style> </style>` tags:

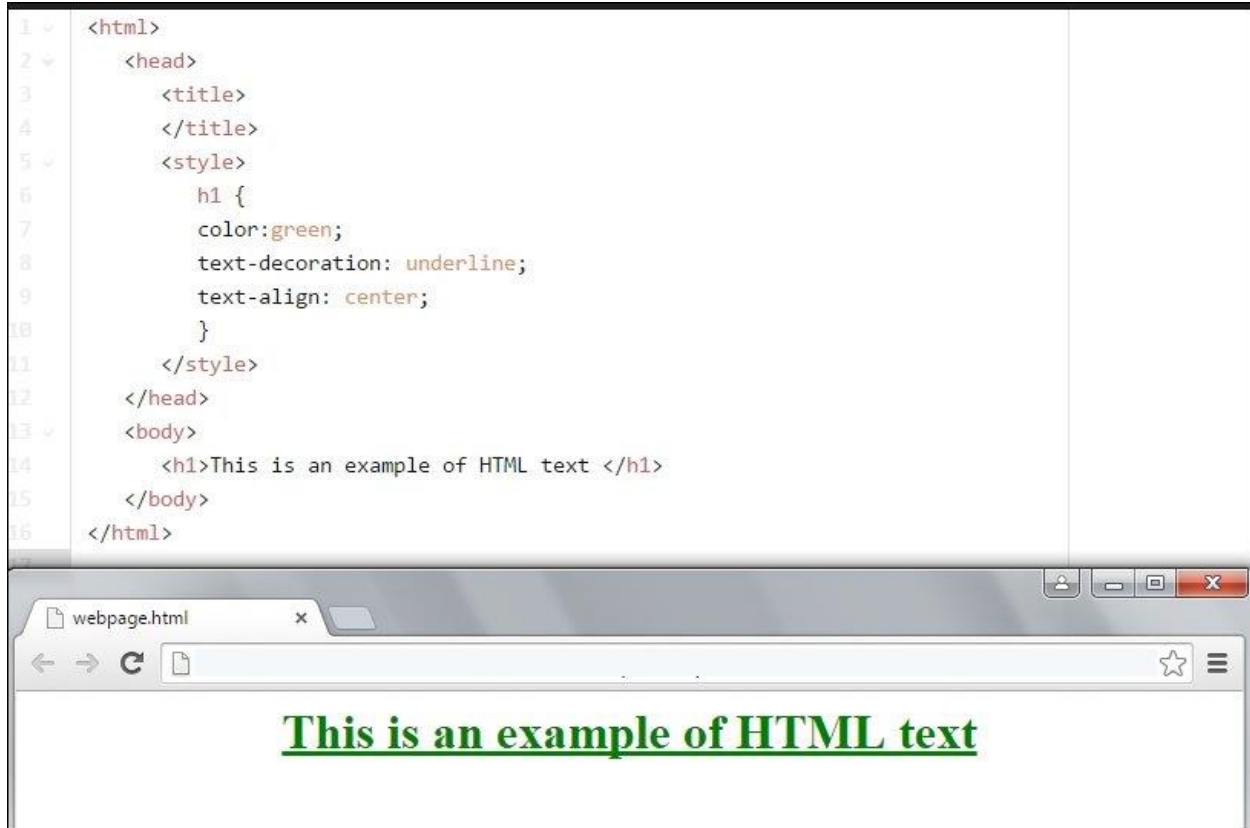
```
<html>
```

```
<head>
  <title>
</title>
  <style>
    h1 {
      color:green;
      text-decoration: underline;
      text-align: center;
    }
  </style>
</head>
<body>
  <h1>This is an example of HTML text </h1>
</body>
</html>
```

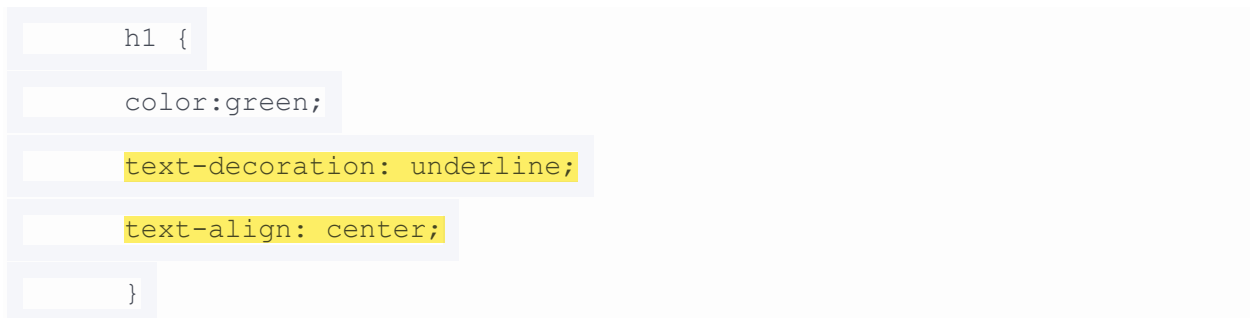
selector. ada 10 rules

Decorator

The output of the code will be as follows:



Look at the code carefully. We used the following CSS for the text in the `<h1></h1>` tags:



Here, we used a few CSS syntaxes (`color`, `text-decoration`, and so on). There are a number of CSS syntaxes, also called property (and every property may contain more than one value).

## JavaScript on an HTML page

---

You have already learned how to print something using JavaScript on console. How about we do it on an HTML page? Before doing this, let's introduce an HTML tag, `<script></script>`. Our JavaScript code will be between these tags.

As there are lots of scripting languages, we need to define what kind of language we are using between these tags. Therefore, we type the following:

`this is optional`

```
<script type = "text/javascript">
  // Our JavaScript Codes will be here.
</script>
```

Let's see an example. In the previous chapter, you learned how to do basic operations using JavaScript on console. Now, we are going to perform a few operations between the `<script></script>` tags in an HTML page. Look at the following code carefully:

```
<html>
  <head>
    <title>
      JavaScript Example
    </title>
  </head>
  <body>
    <script type="text/javascript">
      var x = 34;
      var y = 93;
      var sum = x+y;
      document.write("The sum of "+x+" and "+y+" is "+sum);
    </script>
  </body>
</html>
```

The output of the code will be as follows:

```
<html>
  <head>
    <title>
      JavaScript Example
    </title>
  </head>
  <body>
    <script type="text/javascript">
      var x = 34;
      var y = 93;
      var sum = x+y;
      document.write("The sum of "+x+" and "+y+" is "+sum);
    </script>
  </body>
</html>
```



I hope that you could guess the output of the codes by yourself.

## Summary

---

In this chapter, you learned HTML, CSS, and their syntaxes and usages. We also covered how to implement JavaScript on an HTML document. You are now able to build your own web page and make it wonderful using JavaScript. I would suggest you not to skip any part of this chapter in order to have a better understanding of the next chapter, [Chapter 4](#), *Diving a Bit Deeper*.



## Chapter 4. Diving a Bit Deeper

In most of the JavaScript programs, which we learned so far, the lines of code were executed in the same order in which they appeared in the program. Each code was executed only once. Thus, the code did not include tests to determine if the conditions were true or false or we did not perform any logical statements.

In this chapter, you are going to learn some logical programming. You will learn about the following topics:

- **Loops**                      **function** / method / subroutine
- **If statement**
- **Switch case**

You already know how to embed JavaScript codes on an HTML document. Before starting this chapter, you will learn a few HTML tags and JavaScript methods. These methods and tags will be used throughout the book.

### Note

In object-oriented programming, we don't directly perform any operations on the data from outside an object; we ask an object to perform the operation by passing one or multiple parameters. This task is called an object's method.

## JavaScript methods

---

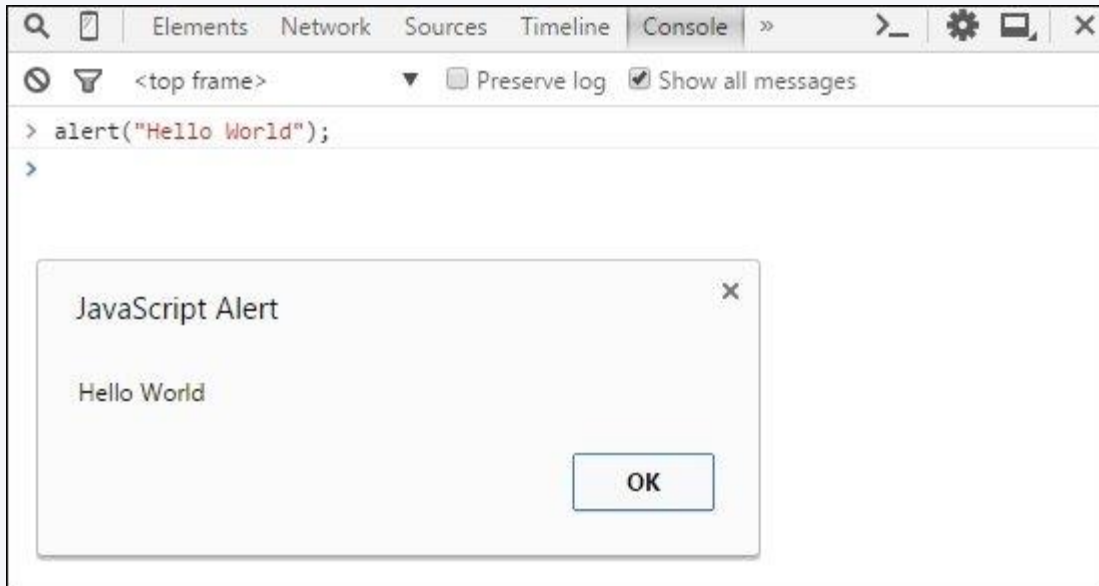
In the previous chapters, you learned how to print something using `document.write()`. Now, you will learn something more.

We will check the methods on both console and HTML document, as follows:

- To show an alert or a pop-up box using JavaScript, we use the following method:

```
alert("Hello World");
```

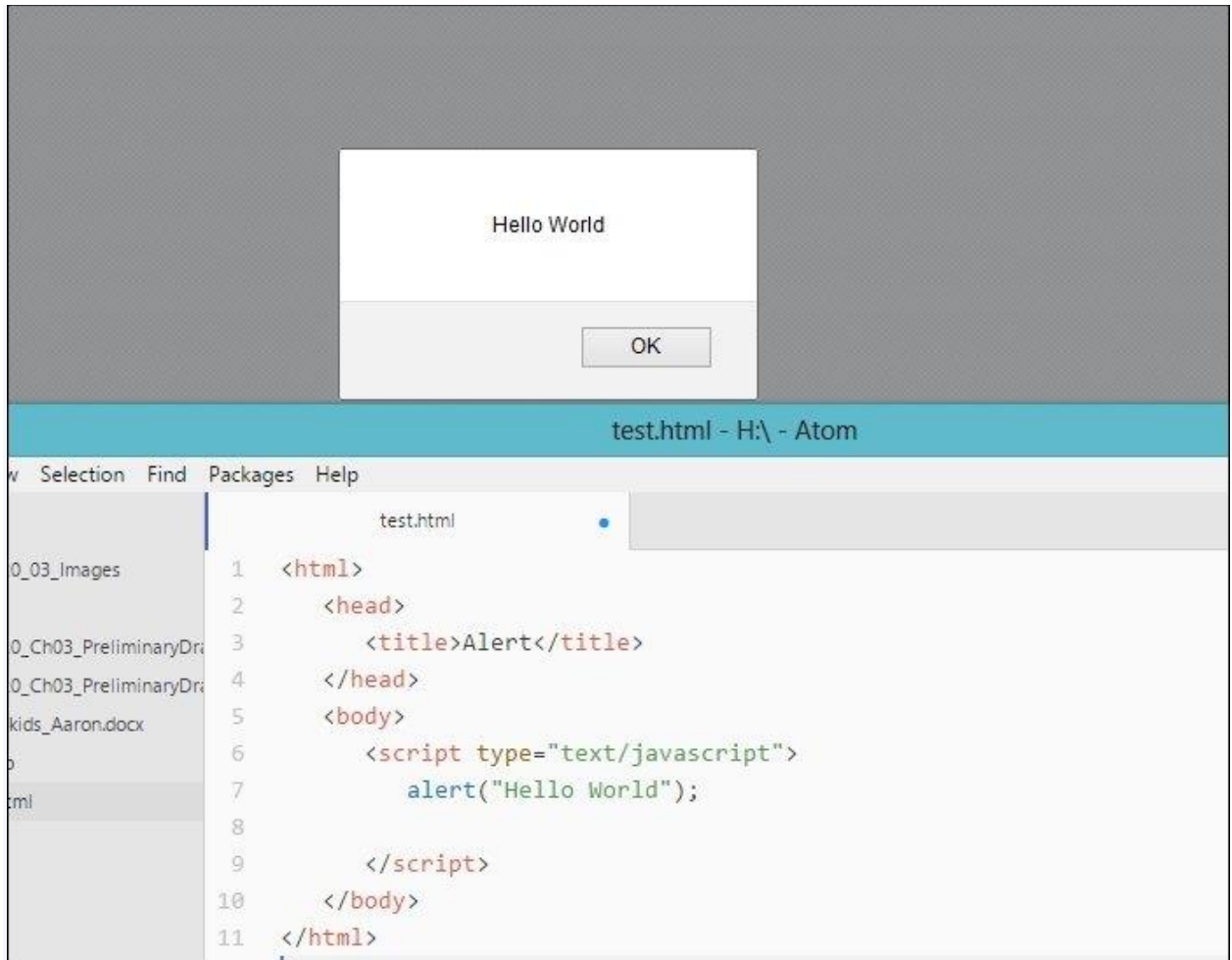
Type this on the console and press *Enter*, you will see a pop-up box saying **Hello World**:



You can write your code to show a pop-up box similar to the following on an HTML document:

```
<html>
  <head>
    <title>Alert</title>
  </head>
  <body>
    <script type="text/javascript">
      alert("Hello World");
    </script>
  </body>
</html>
```

The output will be as follows:



- If you want to take information from users, you need to use a prompt box to do this. Consider the following for example:
  - You want to take input of the username and print it on the main web page.
  - You can do this using the `window.prompt()` method.
  - The structure of `window.prompt()` is similar to the following:

```

window.prompt("What is your name?"); // You can type anything
between the inverted commas.

```

- Now, you need to store the information on a variable. You already know how to do this from the previous chapters. Type the following and press *Enter*:

```

var name = window.prompt("what is your name?");

```

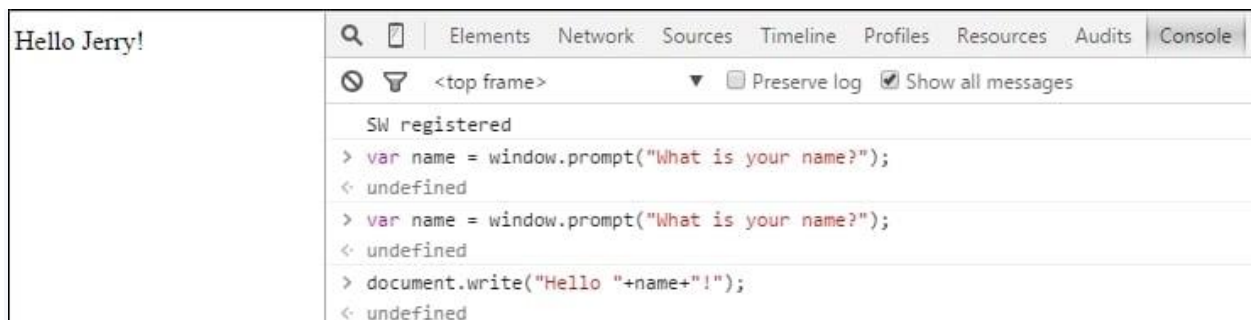
- After running this code on console, you will be asked to input something on a textbox. After typing your information, you need to press the **OK** button. Your information is now stored in the `name` variable:



- If you want to print the variable on your web page, you can use the `document.write()` method, as follows:

```
document.write("Hello "+name+"!");
```

- The output of these steps can be seen in the following screenshot:



- The codes on an HTML document will be as shown in the following:

```
<html>
<head>
  <title>Prompt</title>
</head>
<body>
  <script type="text/javascript">
    var name = window.prompt("What is your name?");
```

```
o document.write("Hello "+name+"!");  
o </script>  
o </body>  
  </html>
```

## HTML buttons and form

---

In the last chapter, you learned about a few HTML tags. Now, we will study a few tags that will make learning HTML more interesting.

### Buttons

If you want to add buttons to your HTML web page, you can use the `<button></button>` tags. The structure of the tags is as follows:

```
<button type="button">Click Here </button>
```

If you want to make your button do something, for example, open an URL; you can consider the following code:

```
<a href="http://google.com/"><button type="button">Click Me </button>  
</a>
```

The output of the code will be as follows:

```

1 <html>
2   <head>
3     <title>Button</title>
4   </head>
5   <body>
6     <a href="http://google.com"> <button type = "button">Click Me </button></a>
7   </body>
8 </html>

```



The screenshot shows a web browser window with a single tab titled 'Button'. The address bar displays 'file:///l:/button.html'. The main content area of the browser shows a single button with the text 'Click Me'.

## Form

In HTML, we use form to represent a document section that contains interactive controls to submit information to a web server. The basic structure of HTML form is as shown in the following:

```

<form>
  User ID: <input type = "text"><br>
  Password: <input type = "password"><br>
</form>

```

Js Validation (optional)  
Back-end Validation (compulsory)

The output of the code will be as follows:

```
1 <html>
2   <head>
3     <title>Form</title>
4   </head>
5   <body>
6     <form>
7       User ID: <input type = "text"><br>
8       Password: <input type ="password"><br>
9     </form>
10  </body>
11 </html>
12
```



Let's dive little bit deeper now!

## If statement

---

Let's say John has 23 apples and Tom has 45 apples. We want to check who has more apples using JavaScript programming. We need to make our browser understand the **if statement**.

### Note

The if statement compares two variables.

To check our condition, we need to declare the two variables containing the number of apples, as follows:

```
var john = 23;
```

```
var tom = 45;
```

To check which number is bigger, we can apply the if statement as shown in the following:

```
if(john > tom)
{
    alert("John has more apples than tom");
}
```

Let's say that we do not know which variable is bigger. Then, we need to check both the variables. Therefore, we need to include the following codes to our program:

```
if(tom > john )
{
    alert("Tom has more apples than John");
}
```

The whole code in an HTML page will be as follows:

```
<html>
<head>
<title>
    If statement
</title>
</head>
<body>
<script type="text/javascript">
    var john = 23;
    var tom = 45;
    if(john > tom){
        alert("John has more apples than Tom");
    }
</script>
</body>
</html>
```

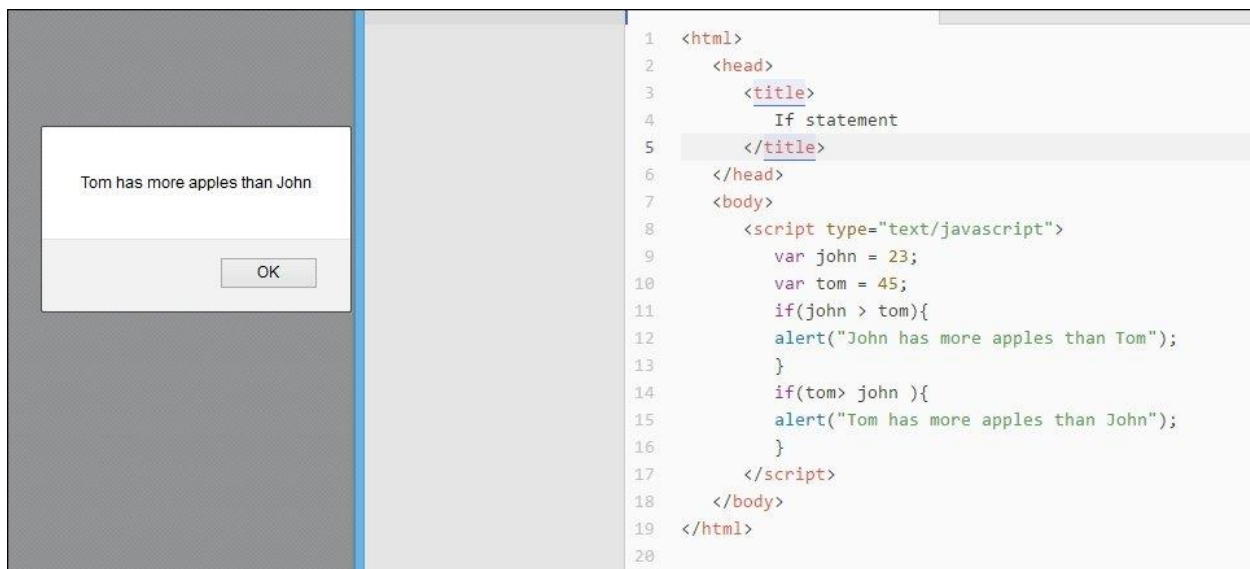


```

    }
    if(tom> john ){
        alert("Tom has more apples than John");
    }
</script>
</body>
</html>

```

The output will be as follows:



You learned about the conditional operators in the previous chapters. In if statement, you can use all of them. Here are a few examples with comments:

```

    >=
    If(tom => john){
        //This will check if the number of apples are equal or greater.
    }
    If(tom <= john)
    {
        //This will check if the number of apples are equal or less.
    }

```

```
}  
If(tom == john)  
{  
  //This will check if the number of apples are equal.  
}
```

To check multiple conditions, you need to use OR (||) or AND (&&).

Consider the following examples:

```
If(john == 23 || john => tom)  
{  
  /* This will check if John has 23 apples or the number of John's apple  
  is equal to or greater than Tom's. This condition will be full filled if  
  any of these two conditions are true.  
  */  
}  
If(tom == 23 && john <= tom)  
{  
  /* This will check if Tom has 23 apples or the number of john's apple is  
  less than Tom's or equal. This condition will be full filled if both of  
  these two conditions are true.  
  */  
}
```

## Switch-case

---

If you have more than three conditions, it is good practice to use the **switch-case** statement. The basic structure of switch-case is as shown in the following:

```
switch (expression) {  
    case expression1:  
        break;  
    case expression2:  
        break;  
    case expression3:  
        break;  
    //-----  
    //-----  
    // More case  
    //-----  
    // -----  
    default:  
}
```

Every **case** has a **break**. However, the **default** does not need a **break**.

Consider that Tom has 35 pens. His friends John, Cindy, Laura, and Terry have 25, 35, 15, and 18 pens, respectively. Now, John wants to check who has 35 pens. We need to compare the number of Tom's pens with everyone's pens. We can use switch-case for this type of case. The code will be as follows:

```
<html>  
    <head>  
        <title>  
            Switch-Case  
        </title>
```

```
</head>
<body>
  <script type="text/javascript">
    var Tom = 35;
    switch (Tom) {
      case 25: //Number of John's pens
        document.write("John has equal number of pens as Tom");
        break;
      case 35: //Number of Cindy's pens
        document.write("Cindy has equal number of pens as Tom");
        break;
      case 15: //Number of Laura's pens
        document.write("Laura has equal number of pens as Tom");
        break;
      case 18: //Number of Terry's pens
        document.write("Terry has equal number of pens as Tom");
        break;
      default:
        document.write("No one has equal pens as Tom");
    }
  </script>
</body>
</html>
```

The output will be as follows:

Cindy has equal number of pens as Tom	<pre> 1 &lt;html&gt; 2   &lt;head&gt; 3     &lt;title&gt; 4       Switch-Case 5     &lt;/title&gt; 6   &lt;/head&gt; 7   &lt;body&gt; 8     &lt;script type="text/javascript"&gt; 9     var Tom = 35; 10    switch (Tom) { 11      case 25: //Number of John's pens 12        document.write("John has equal number of pens as Tom"); 13        break; 14      case 35: //Number of Cindy's pens 15        document.write("Cindy has equal number of pens as Tom"); 16        break; 17      case 15: //Number of Laura's pens 18        document.write("Laura has equal number of pens as Tom"); 19        break; 20      case 18: //Number of Terry's pens 21        document.write("Terry has equal number of pens as Tom"); 22        break; 23      default: 24        document.write("No one has equal pens as Tom"); 25    } 26  &lt;/script&gt; 27  &lt;/body&gt; 28 &lt;/html&gt; </pre>
---------------------------------------	---

## Note

Now, change the value of second case (35) to other and check your result.

## Exercise

1. Suppose you need to go to school every day except Saturday and Sunday. Write a code, where you will input today's date number and the web page will show you whether you need to go to school or not. (Hint: use a switch case.)
2. Consider that you have a garden and you water all the plants on even days of the month. Write a code that will show you whether you will water your plants on that day. (Hint: use the if condition and modulus operator (%).)

## Loops

---

In this paragraph, we will learn something interesting called **loop**.

Consider that you need to print a line 100 times using JavaScript. What you will do?

You can type `document.write("The line I want You to Write");` 100 times in your program or you can use loop.

The basic use of loop is to do something more than one time. Say, you need to print all the integers of  $1 + 2 + 4 + 6 + \dots + 100$  series upto 100. The calculation is the same, you only need to do it multiple times. In these cases, we use loop.

We will discuss two types of loops, namely **for loop** and **while loop**.

## The for loop

The basic structure of the for loop is as follows:

```
for(starting ; condition ; increment/decrement)
{
    statement
}
```

The **starting** parameter is the initialization of your loop. You need to initialize the loop in order to start it. The **condition** parameter is the key element to control the loop.

The **increment/decrement** parameter defines how your loop will increase/decrease.

Let's see an example. You want to print **javascript is fun** 10 times. The code will be as shown in the following:

```
<html>
<head>
    <title>For Loop</title>
</head>
<body>
    <script type="text/javascript">
        var java;
        for (java=0; java<10; java++) {
            for(int java=0; java<10; java++)

```

```

        document.write("javascript is fun"+"<br>");
    }
</script>
</body>
</html>

```

The output will be similar to the following:



Yes! You printed the line 10 times. If you look at the code carefully, you will see the following:

- We declared a variable named `java`
- In the `for` loop, we initialized `0` to its value
- We added a `java<10` condition that made the browser count from `0` to `10`
- We incremented the variable by `1`; that's why we added `java++`

### Exercise

1. Write a code using JavaScript that will print the following output:

- ```

2.   I have 2 apples.
3.   I have 4 apples.

```

```

4.   I have 6 apples.
5.   I have 8 apples.
6.   I have 10 apples.
7.   I have 12 apples.
8.   I have 14 apples.
9.   I have 16 apples.
10.  I have 18 apples.

    I have 20 apples.

```

11. Write a code that will print all the even numbers from 2 to 500.

### The while loop

You have already have learned how to execute something multiple times using the for loop. Now, we will learn another loop known as the while loop. The structure of while loop is as follows:

```

initialize;
while (condition) {
    statement;
    increment/decrement;
}

```

The codes for the previous example will be like the following:

```

<html>
  <head>
    <title>For Loop</title>
  </head>
  <body>

```



```
<script type="text/javascript">  
    var java = 0;  
    while(java < 10){  
        document.write("javascript is fun"+"<br>");  
        java++;  
    }  
</script>  
</body>  
</html>
```

The output will be the same as the `for` loop.

### Exercise

1. Write a code that will print all the odd values from 1 to 600 using the while loop.  
(Hint: use the modulus operator.)
2. Write a code that will print the following output:

```
3.  5 x 1 = 5  
4.  5 x 2 = 10  
5.  5 x 3 = 15  
6.  5 x 4 = 20  
7.  5 x 5 = 25  
8.  5 x 6 = 30  
9.  5 x 7 = 35  
10. 5 x 8 = 40  
11. 5 x 9 = 45  
    5 x 10 = 50
```

### Summary

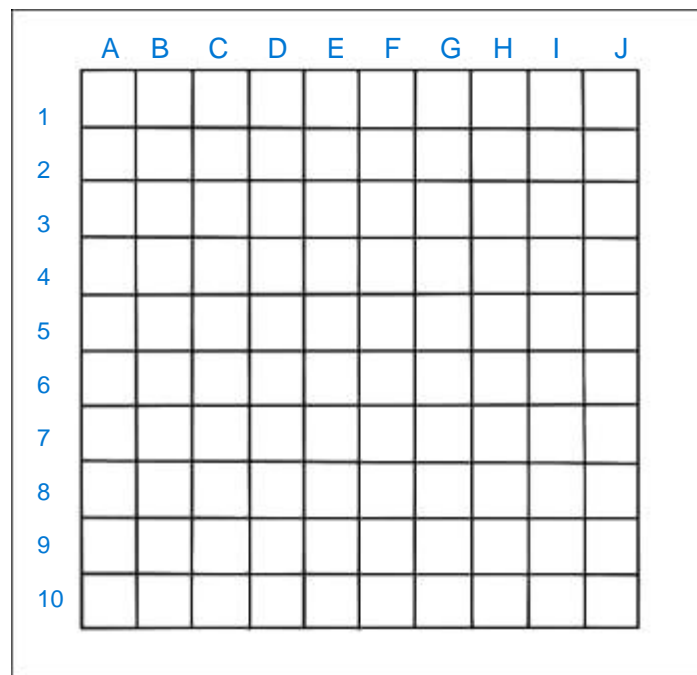
In this chapter, you learned logical operations using JavaScript. You learned loops, conditional operation, and other HTML tags.

We need to focus on this chapter as we have discussed the most important attributes in JavaScript here. You can become a JavaScript master if you practice this chapter and the last three chapters. I recommend you not to go further unless you have a good knowledge all the four chapters. If you have already learned about all the topics that we discussed earlier, let's move on to [Chapter 5](#), *Ahoy! Sailing into Battle*.

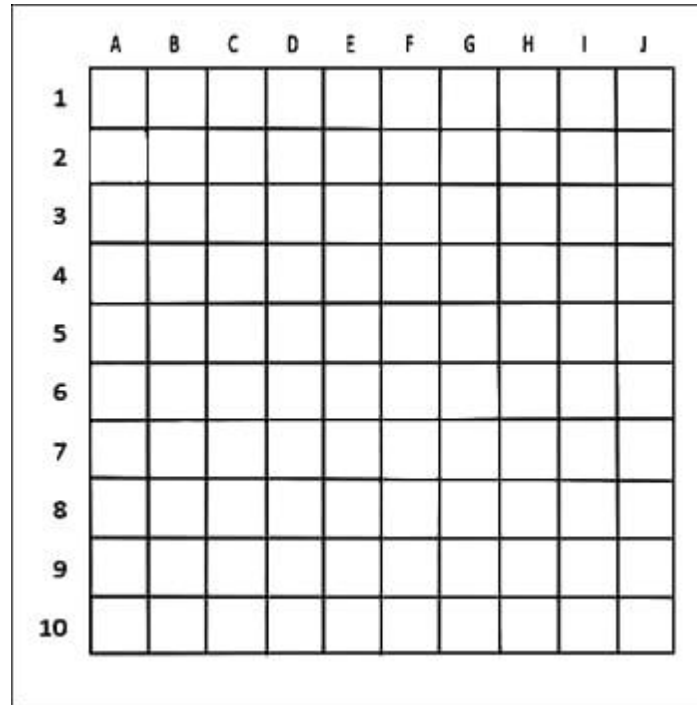
## Chapter 5. Ahoy! Sailing into Battle

In this chapter, we are going to develop a full game using HTML, CSS, and JavaScript. We will focus on the JavaScript coding, therefore, we will not care about the graphics of the game. We will code a game named **Battleship**. Many of you have heard of it before. This is a memory game. Your imagination and intuition will help you to win the game. There are a few variations for playing the game.

Let's discuss how the game looks. There are a few square-shaped geometrical objects connected to each other as shown in the following. The number of rows and columns need to be equal:



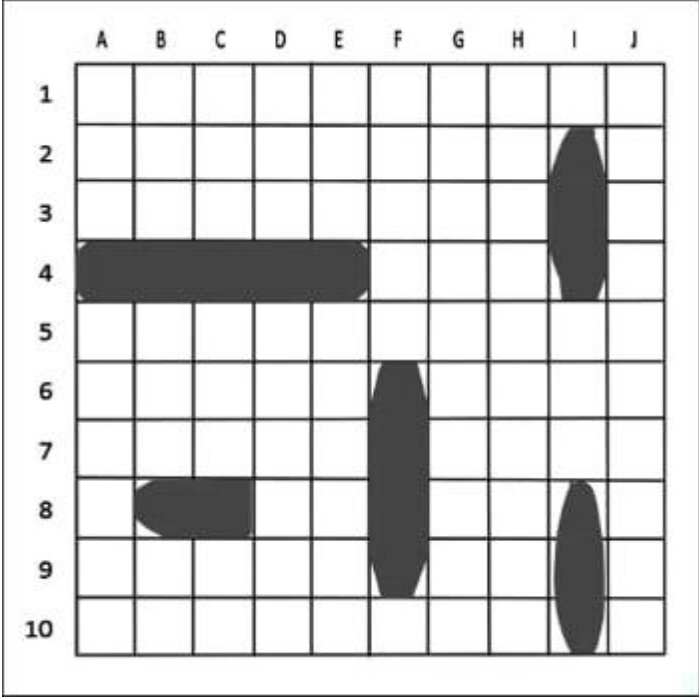
The rows and columns are usually named with the help of number system or alphabets. Let's say that the rows are **1, 2, 3, 4, 5, 6, 7, 8, 9, and 10**. The columns are **A, B, C, D, E, F, G, H, I, and J**. We can name them by either numbers or alphabets:



It is a two player game. The following are its rules:

- Both the players will secretly place their ships (there can be different types of boats or water vehicles) on their matrices/grids.
- The players can put their ships vertically or horizontally; however, not diagonally.
- The players must place all their ships on the grid before they start playing.
- Their ships cannot overlap each other's.
- When all the ships are placed, the players cannot move their ships from the grid.
- After placing all the ships, the first player will state a coordinate of the second player and if there is a ship belonging to the second player, the ship will blow.
- Then, the second player will state a coordinate of the first player. If there is a ship belonging to the first player, it will blow.
- The coordinate may look similar to **A2**, **B2**, **D5**, and so on. The first alphabet will be the x axis of the grids and the number will represent y axis of the grid.
- The player that blows all the ships of the opponent will win.

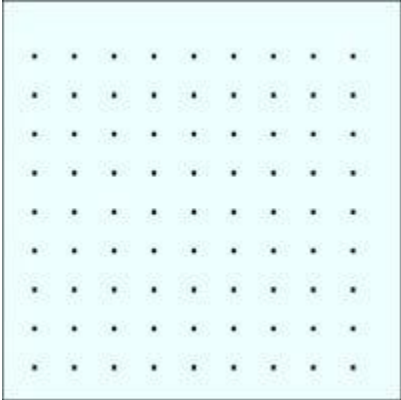
The following figure shows few ships placed on the grid:



Now, we will head to the programming part of the game.

We will stick to the following rules so that our game does not become difficult to code:

- 1. There will be one ship belonging to both the players.
- 2. The ship will occupy four parts of the grid.
- 3. A player will have to input both the x and y axes coordinates at the prompt.
- 4. The grid will be 9 x 9.
- 5. The player will have to put **h** or **v** for the horizontal or vertical position of the ship.
- 6. To simplify the drawing, we will put dots (.) on the position of the grids. The grids will look similar to the following image:



7. We will need a **Fire** button to start the game.

## The HTML part

---

The HTML part will look similar to the following code:

```
<html>
  <head>
  </head>
  <body>
    <h1> Battleship Game </h1>
  </body>
  <style>
    // We will code in CSS here
  </style>
  <script type = "text/javascript">
    //We will code in JavaScript here
  </script>
</html>
```

The output of the code will be as shown in the following image:

```

1  <html>
2    <head>
3  </head>
4    <body>
5      <h1> Battleship Game </h1>
6    </body>
7    <style>
8  //We will code in CSS here
9    </style>
10   <script type = "text/javascript">
11 //We will code in JavaScript here
12   </script>
13 </html>

```

## Battleship Game

### The CSS part

---

We use a CSS coding in the `<style></style>` tags for the body. As we will heed on the coding in JavaScript only, we will not bother about the visual part of the game. To make the body of the game colorful, we will use the following code:

```

<style>
  body {
    background-color: #eff;
  }
</style>

```

### The JavaScript part

---

This part is the main part of our game, we will pay attention to this part the most. We will write all our codes in the `<script></script>` tags.

For the grids, we will need a two dimensional array. We will take a `game` variable to store the data as follows:





```

var t=document.createTextNode("Fire!");
document.body.appendChild(button);
button.appendChild(t);

```

Let's make a function to draw the board:

```

function drawBoard() {
    var boardContents = "";
    var i;
    var j;
    for (i=0; i<9; i++) {
        for (j=0; j<9; j++) {
            boardContents = boardContents + game[i][j]+" ";
            // Append array contents for each board square
        }
        boardContents = boardContents + "<br>";
        // Append a line break at the end of each horizontal line
    }
    return boardContents;
    // Return string representing board in HTML
}

```

Now, put draw the board on the HTML page by writing the following code:

```
board.innerHTML = drawBoard();
```

We will ask the player where he wants to place his ship using the `prompt()` function:

```

var x=prompt("Where would you like to place your ship? Enter an X
coordinate: (0-8)");

```

```

var y=prompt("Where would you like to place your ship? Enter a Y
coordinate: (0-8)");

var direction=prompt("Place (h)orizontally, (v)ertically");

x = Number(x); // Convert the string returned by "prompt" into a
number

y = Number(y); // Convert the string returned by "prompt" into a
number

```

If the player chooses the horizontal orientation for their ship, we need to replace the dots by writing the following code:

```

if (direction[0] == "h") {
    var c;
    for (c = x; c < (x + 4); c++)
    {
        game[y][c] = '#';
    }
}

```

If the player chooses the vertical orientation for their ship, we need to replace the dots by writing the following code:

```

if (direction[0] == "v") {
    var c;
    for (c = y; c < (y + 4); c++)
    {
        game[c][x] = '#';
    }
}

```

We need to redraw the board after placing the ship, as follows:

```
board.innerHTML = drawBoard();
```

Lets create the `fire()` function.

Our `fire()` function will be as follows:

```
function fire() {
  //We will write codes here.
}
```

When the `fire()` function is called, we need to take input from the player as shown in the following:

```
var fireX=prompt("Where would you like to fire? Enter an X coordinate:
(0-8)");
var fireY=prompt("Where would you like to fire? Enter a Y coordinate:
(0-8)");
```

Convert the inputs into numbers, as follows:

```
fireX = Number(fireX);
// Convert the string returned by "prompt" into a number
fireY = Number(fireY);
// Convert the string returned by "prompt" into a number
```

If the input does not match with the `#` character, we will print `You Missed`. using the following code:

```
if (game[fireY][fireX] == ".") {
  // Check if the specified coordinate is occupied by the cruiser
  alert("You Missed.");
```

```
}

```

If the input hits the ship, we will print few messages and draw the board again:

```
else if (game[fireY][fireX] == "*") {
    alert("You already hit the ship there.");
} else {
    alert("Kaboom! You hit a ship");
    game[fireY][fireX] = "*";
    board.innerHTML = drawBoard();
    // Redraw board with hit marker at specified coordinate
}

```

Now, we need to check whether there is any ship remaining on the board. We will use the following code:

```
var shipfound;
var i;
var j;
// Check if there are any ships remaining on the board
for (i=0; i<9; i++) {
    for (j=0; j<9; j++) {
        if (game[i][j] != "." && game[i][j] != "*") {
            shipfound = true;
            // Taking a boolean data type to set it if a ship is found
        }
    }
}

```

If no ship is left, we will end the game:

```

if (!shipfound) {
    // If no ships are found end the game
    alert("All ships have been sunk. Well done Captain! Game over");
    document.body.removeChild(button);
    // Remove the fire button from the page after game over
}

```

## The final code

---

Our final codes will look similar to the following:

```

<html>
  <head>
  </head>
  <body>
    <h1> Battleship Game </h1>
  </body>
  <style>
    body {
      background-color: #eff;
    }
  </style>
  <script>
    var game = [
      [".", ".", ".", ".", ".", ".", ".", ".", "."],
      [".", ".", ".", ".", ".", ".", ".", ".", "."],
      [".", ".", ".", ".", ".", ".", ".", ".", "."],
      [".", ".", ".", ".", ".", ".", ".", ".", "."],

```

```

        [".", ".", ".", ".", ".", ".", ".", ".", "."],
        [".", ".", ".", ".", ".", ".", ".", ".", "."],
        [".", ".", ".", ".", ".", ".", ".", ".", "."],
        [".", ".", ".", ".", ".", ".", ".", ".", "."],
        [".", ".", ".", ".", ".", ".", ".", ".", "."],
    ];

    var board = document.createElement("PRE");

    // preparing the HTML <pre> element to display the board on the page
    document.body.appendChild(board);

    var button=document.createElement("BUTTON");

    // Preparing the "Fire! button to allow the player to fire at the
    ship

    button.onclick = fire; // Clicking the button calls the fire()
    function
    var t=document.createTextNode("Fire!");
    document.body.appendChild(button);
    button.appendChild(t);
    function drawBoard() {
        var boardContents = "";
        var i; var j;
        for (i=0; i<9; i++) {
            for (j=0; j<9; j++) {
                boardContents = boardContents + game[i][j]+" ";
                // Append array contents for each board square
            }
            boardContents = boardContents + "<br>";
            // Append a line break at the end of each horizontal line

```

```

    } return boardContents;

    // Return string representing board in HTML
}

board.innerHTML = drawBoard();

// Display the board on the page using the above function

var x=prompt("Where would you like to place your cruiser? Enter an X
coordinate: (0-8)");

var y=prompt("Where would you like to place your cruiser? Enter a Y
coordinate: (0-8)");

var direction=prompt("Place (h)orizontally, (v)ertically");

x = Number(x); // Convert the string returned by "prompt" into a
number

y = Number(y); // Convert the string returned by "prompt" into a
number

if (direction[0] == "h") {

    var c;

    for (c = x; c < (x + 4); c++)

    {

        game[y][c] = '4';

    }

}

// Draw cruiser vertically

if (direction[0] == "v") {

    var c;

    for (c = y; c < (y + 4); c++)

    {

        game[c][x] = '4';

```

```

    }
}

board.innerHTML = drawBoard(); // Redraw board with cruiser added

// Function for firing a shot when the "Fire! button is pressed

function fire() {

    var fireX=prompt("Where would you like to fire? Enter an X
coordinate: (0-8)");

    var fireY=prompt("Where would you like to fire? Enter a Y
coordinate: (0-8)");

    fireX = Number(fireX);

    // Convert the string returned by "prompt" into a number

    fireY = Number(fireY);

    // Convert the string returned by "prompt" into a number

    if (game[fireY][fireX] == ".") {

        // Check if the specified coordinate is occupied by the cruiser

        alert("Missed.");

    }

    else if (game[fireY][fireX] == "*") {

        alert("You already hit the ship there.");

    } else {

        alert("Kaboom! You hit a ship");

        game[fireY][fireX] = "*";

        board.innerHTML = drawBoard();

        // Redraw board with hit marker at specified coordinate

    }

    var shipfound;

    var i;

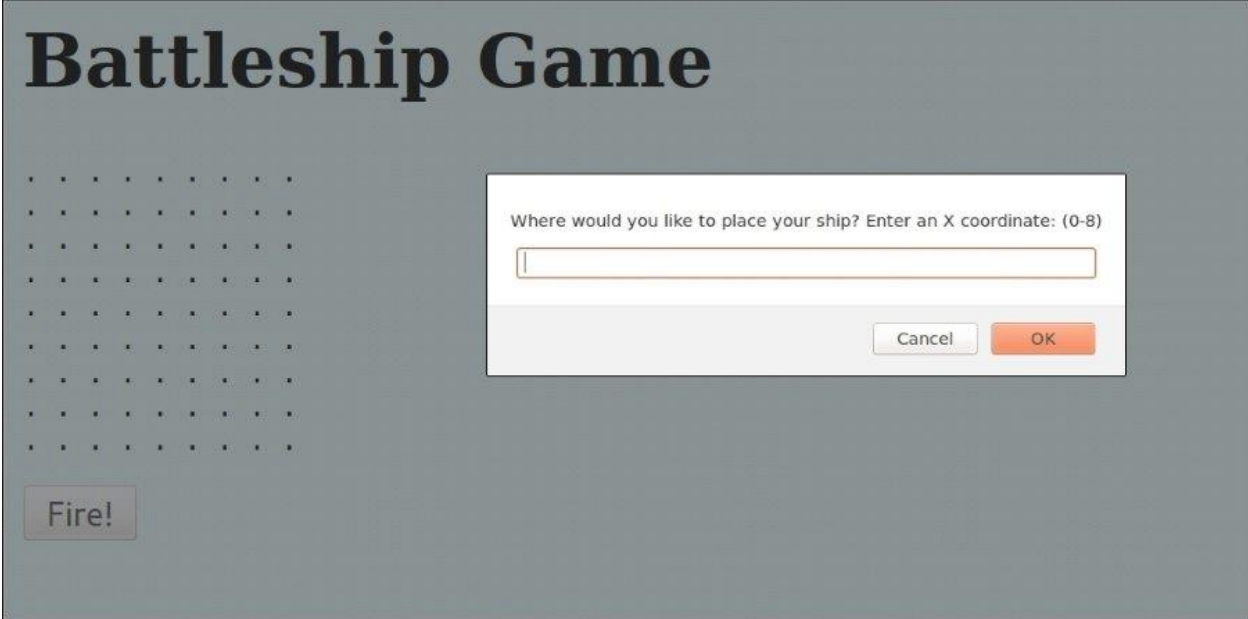
```



```
var j;

// Check if there are any ships remaining on the board
for (i=0; i<9; i++) {
    for (j=0; j<9; j++) {
        if (game[i][j] != "." && game[i][j] != "*") {
            shipfound = true;
            // Set to true if a ship is found
        }
    }
}if (!shipfound) {
    // If no ships are found end the game
    alert("All ships have been sunk. Well done Captain! Game over");
    document.body.removeChild(button);
    // Remove the fire button from the page after game over
}
}
</script>
</html>
```

If you run the preceding code, you will see the following prompt:



Let's play the game that we created. The first player has to place his ship. He has to input the coordinates of the ship.

Consider that we input 3 on the x axis and 2 on the y axis. Place our ship on the vertical orientation. The game screen will look as shown in the following:



You can see that your ship is placed. Now, you can shoot your opponent (computer) by pressing the **Fire** button. You will be asked to input the coordinates of the grid, where

you want to shoot. If you miss a shot, you will see a message that we coded, **You Missed**.

I hope that you are able to play the game that you built.

Congratulations!

If you want to develop your game more (such as enhance the graphics, number of ships, and so on), you only need to develop CSS and JavaScript.

Now, we will see a better code for the Battleship game, as shown in the following:

1. Make a `js` folder anywhere in your computer.
2. In the `js` folder, place the three files that are included in this chapter: `battleship.js`, `functions.js`, and `jquery.min.js`.
3. Outside the `js` folder, place the `battleship.css` and `index.html` files.

Open the `index.html` file in a Notepad, you will see the following code:

```
<html>
  <head>
    <title>Battleship</title>
    <meta name="viewport" content="width=device-width" />
    <link href="battleship.css" rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <h1>BATTLESHIP</h1>
    <div class="game-types">
      <h2 class='game-choice'>Choose a game type</h2>
      <dl class="game-description">
        <dt>Standard</dt>
        <dd>Classic Battleship with randomly placed ships</dd>
      </dl>
    </div>
  </body>
</html>
```

```

    <dt>Custom</dt>
    <dd>Choose any 5 ships and place them where you like. The
computer will have the same 5 ships, randomly placed</dd>
</dl>
<div class='button-wrapper'>
    <button class="standard">Standard</button>
    <button class="custom">Custom</button>
</div>
</div>
<div class='ship-picker'>
    <h2>Pick 5 Ships</h2>
    <h3>Selected ships</h3>
    <ul class="ship-list">
        <li>
            <p></p>
            <div class='remove'>X</div>
        </li>
        <li>
            <p></p>
            <div class='remove'>X</div>
        </li>
        <li>
            <p></p>
            <div class='remove'>X</div>
        </li>
        <li>
            <p></p>

```

```
        <div class='remove'>X</div>
    </li>
    <li>
        <p></p>
        <div class='remove'>X</div>
    </li>
</ul>
<ul class='ship-choices button-wrapper'>
    <li class="ship-choice">Carrier</li>
    <li class="ship-choice">Battleship</li>
    <li class="ship-choice">Submarine</li>
    <li class="ship-choice">Cruiser</li>
    <li class="ship-choice">Destroyer</li>
</ul>
<div class='button-wrapper'>
    <button class='build-fleet inactive'>Build Fleet</button>
</div>
</div>
<div class="ship-placer">
    <div class="board placer-board">
        <div class="labels">
            <div class="row-label">
            </div>
            <div class="column-label">
            </div>
        </div>
        <div class="playable-area">
```

```
    </div>
  </div>
  <div class='ships-to-place'>
    <h3>Ships to place</h3>
    <ul>
    </ul>
  </div>

  <div class="clear"></div>
  <div class="instructions">
    <p>Use 'WASD' keys to rotate pieces</p>
  </div>

  <div class='button-wrapper'>
    <button class="start inactive">Start game</button>
  </div>
</div>

<div class="game-area">
  <div class="board-wrap">
    <h1 class="hidden">BATTLESHIP</h1>
    <div class="single-board-wrap">
      <div class="board human-board">
        <div class="labels">
          <div class="row-label">
          </div>
          <div class="column-label">
          </div>
        </div>
      </div>
    </div>
  </div>

```

```
    </div>
    <div class="playable-area">
    </div>
  </div>
  <h2>Human Board</h2>
</div>
<div class="single-board-wrap">
  <div class="board ai-board">
    <div class="labels">
      <div class="row-label">
      </div>
      <div class="column-label">
      </div>
    </div>
    <div class="playable-area">
    </div>
  </div>
  <h2>Opponent Board</h2>
</div>
<div class="button-wrapper">
  <button class="new-game">New Game</button>
  <button class="stats hidden">Show Stats</button>
</div>
</div>
<div class="info-area">
  <h2>Enemy ships remaining</h2>
  <div class="scoreboard">
```

```

        <div class="ships-left">
    </div>
</div>
    <div class="gamelog-container">
        <h2>GAME LOG</h2>
    </div>
</div>
</div>
</div>
<script src="js/jquery.min.js"></script>
<script src="js/functions.js"></script>
<script src="js/battleship.js"></script>
</body>
</html>

```

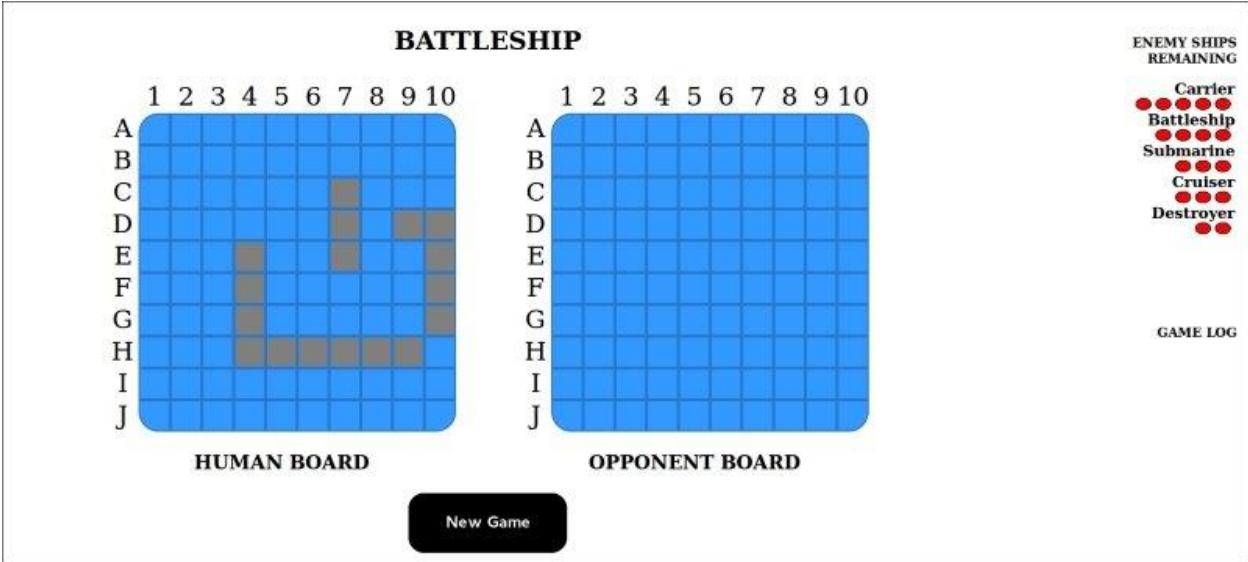
We included the three JavaScript files in the HTML file. We added a jQuery file, which we will discuss in the next chapter. The output of the preceding code will show you the following screen:



You can click the **Standard** button to play the standard Battlefield or **Custom** button to play a non-standard Battlefield.

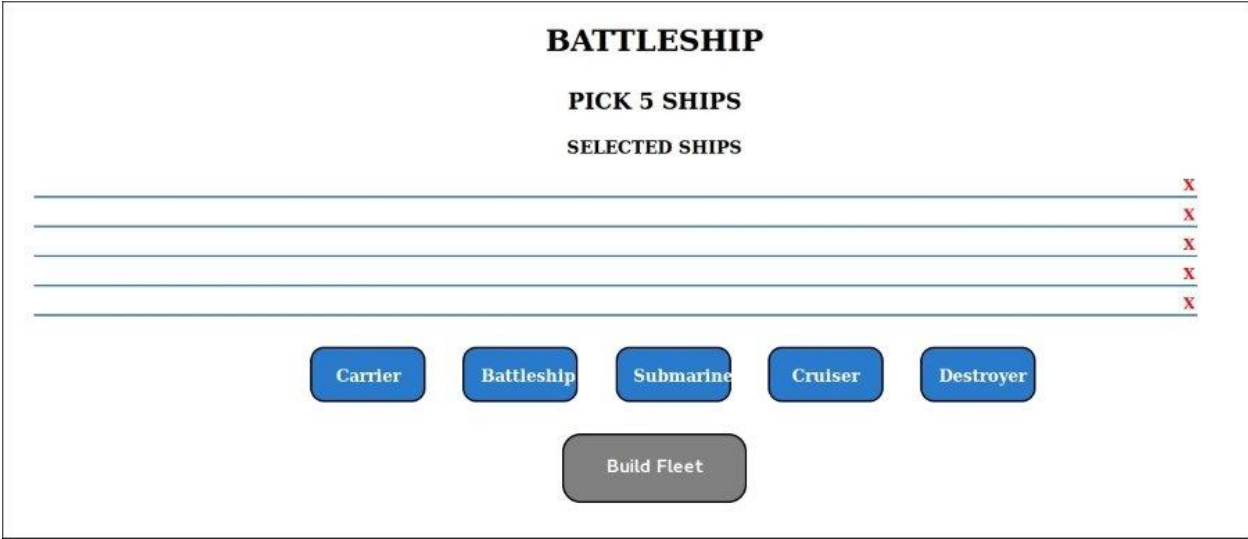


If you select the **Standard** button, you will get the following screen:



Now, you can guess the position of the opponent's ship and click on the grid. There will be a log panel on the right-hand side of the screen. You can also see how many and which ships you have destroyed from the preceding panel of the game log panel.

If you select the **Custom** play, you will see the following screen:



After adding the five ships, you can play the game. You can add the same ship twice or more, if required.

You can place your ships vertically or horizontally and click on the tiles to blow the opponent's ship. You can click one tile at a time.

## Summary

---

In this chapter, we built a complete game and played it. We also played a better version of the game we have built. All you need to remember is that you must know the logic behind all the code that we previously discussed. You are given the source code of the better version of the game with this chapter. I hope that you will study the code and write your own Battleship game. We used a `jquery.js` JavaScript file on our improved version of the Battleship. The `jquery.js` file has a lot of lines of code (We will discuss this in [Chapter 6](#), *Exploring the Benefits of jQuery*).

If you master all the code that we discussed in this chapter, we can now move to the next chapter.

## Chapter 6. Exploring the Benefits of jQuery

If you have gone through the previous chapter, you probably have implemented **jQuery** in your **Battleship** game. In this chapter, we will discuss about jQuery in detail.

The jQuery library is a JavaScript framework. It was released in 2006. People used to call it **jSelect**. We use jQuery in our websites so that we can work with JavaScript easily and add effects to our web pages. You may think jQuery is different from JavaScript. No! jQuery is just another JavaScript file. It is a very lightweight library that helps you to decorate your web pages more easily with less coding.

We use jQuery due to the following advantages:

- It is open source; you can edit or modify its code if required
- It is a small library (about 150 KB file)
- The community support for jQuery is very strong; you can get help from the users easily
- It is user-friendly and popular
- It supports cross-browsers
- It is openly developed; you can fix any bug or add features to it by editing the codes
- It helps the developers to build responsive sites by using AJAX
- It has built-in animation functions that help a developer to create animations in their website

[Asynchronous JavaScript and XML](#)

### Installing jQuery

---

The question is where to find jQuery. Well, you can find it at <http://jquery.com/>. I have also attached the file with this book. You can download it from there.

If you go to <http://jquery.com/>, you will see the following screen:

Download API Documentation Blog Plugins Browser Support

Search

**Download jQuery**  
v1.11.3 or v2.1.4

View Source on GitHub →  
How jQuery Works →

**Lightweight Footprint**  
Only 32KB minified and gzipped. Can also be included as an AMD module.

**CSS3 Compliant**  
Supports CSS3 selectors to find elements as well as in style property manipulation.

**Cross-Browser**  
IE, Firefox, Safari, Opera, Chrome, and more.

**What is jQuery?**

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

**Corporate Members**

famo.us WORDPRESS (mt) mediatemple maxCDN

**Resources**

- jQuery Core API Documentation
- jQuery Learning Center
- jQuery Blog
- Contribute to jQuery
- About the jQuery Foundation
- Browse or Submit jQuery Bugs

**TRY jQuery**  
Learn jQuery in your browser!

Click the **Download jQuery** button. You will be redirected to the following page:

Download API Documentation Blog Plugins Browser Support

Search

**Downloading jQuery**

Compressed and uncompressed copies of jQuery files are available. The uncompressed file is best used during development or debugging; the compressed file saves bandwidth and improves performance in production. You can also download a [sourcemap file](#) for use when debugging with a compressed file. The map file is *not* required for users to run jQuery, it just improves the developer's debugger experience. As of jQuery 1.11.0/2.1.0 the `///  
# sourceMappingURL` comment is *not included* in the compressed file.

To locally download these files, right-click the link and select "Save as..." from the menu.

**jQuery 1.x**

The jQuery 1.x line had major changes as of jQuery 1.9.0. We *strongly* recommend that you also use the jQuery Migrate plugin if you are upgrading from pre-1.9 versions of jQuery or need to use plugins that haven't yet been updated. Read the [jQuery 1.9 Upgrade Guide](#) and the [jQuery 1.9 release blog post](#) for more information.

[Download the compressed, production jQuery 1.11.3](#)

[Download the uncompressed, development jQuery 1.11.3](#)

[Download the map file for jQuery 1.11.3](#)

[jQuery 1.11.3 release notes](#)

**jQuery 2.x**

jQuery 2.x has the same API as jQuery 1.x, but *does not support Internet Explorer 6, 7, or 8*. All the notes in the [jQuery 1.9 Upgrade Guide](#) apply here as well. Since IE 8 is still relatively common, we recommend using the 1.x version unless you are certain no IE 6/7/8 users are visiting the site. Please read the [2.0 release notes](#) carefully.

[Download the compressed, production jQuery 2.1.4](#)

[Download the uncompressed, development jQuery 2.1.4](#)

[Download the map file for jQuery 2.1.4](#)

[jQuery 2.1.4 release notes](#)

There are two versions of jQuery: **1.x.x** and **2.x.x**. There are just a few differences between these versions. The compressed version's code is not readable as the version does not have blank spaces and comments; however, the uncompressed version is clearly coded and formatted, it also has important comments to understand the code and functions' work. If you want to learn how a function of jQuery works, I would suggest you to go through the jQuery uncompressed version.

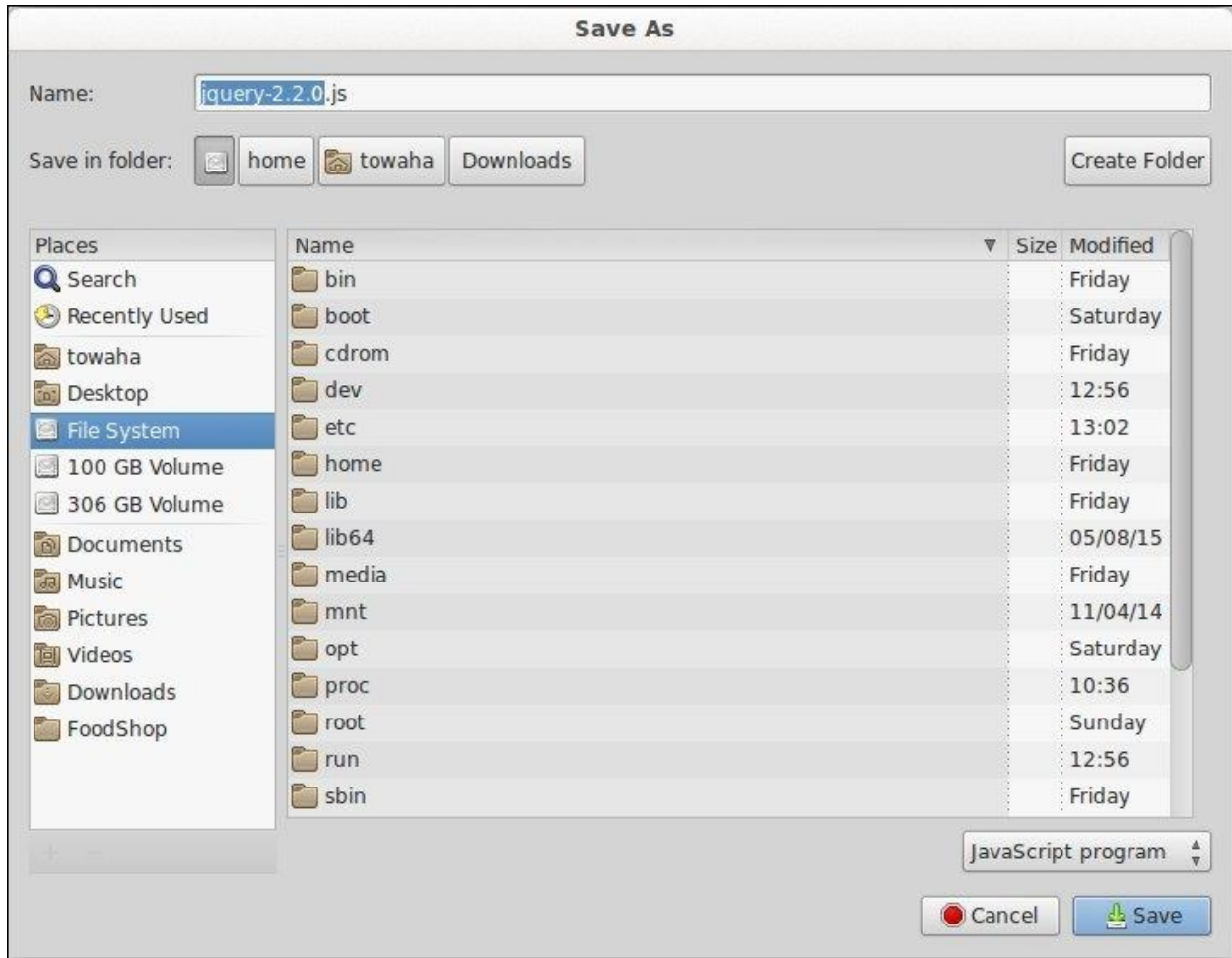
Throughout this chapter, we will use the **2.x.x** version. The latest version of **2.x.x** is **2.2.0**.

#### Note

You can download the compressed or uncompressed version of jQuery.

I'll advice you to use the compressed version as it is lightweight.

We will use the uncompressed version for this chapter so that you can study the **jquery.js** and get a clear concept of how it works. After clicking **Download the uncompressed, development jQuery 2.2.0**, you will see the jQuery library on your browser. Click *Ctrl + S* on your keyboard to save the file, as shown in the following screenshot:



After downloading the jQuery, place it in your computer. For simplicity, rename it to `jquery`.

Create a new HTML file in the same folder and include the `jquery.js` in your HTML document by typing the following code in the `<head></head>` tags:

```
<script src="jquery.js"></script> Copy
```

To check whether your imported `jquery.js` is working, type the following code. I will explain the code later:

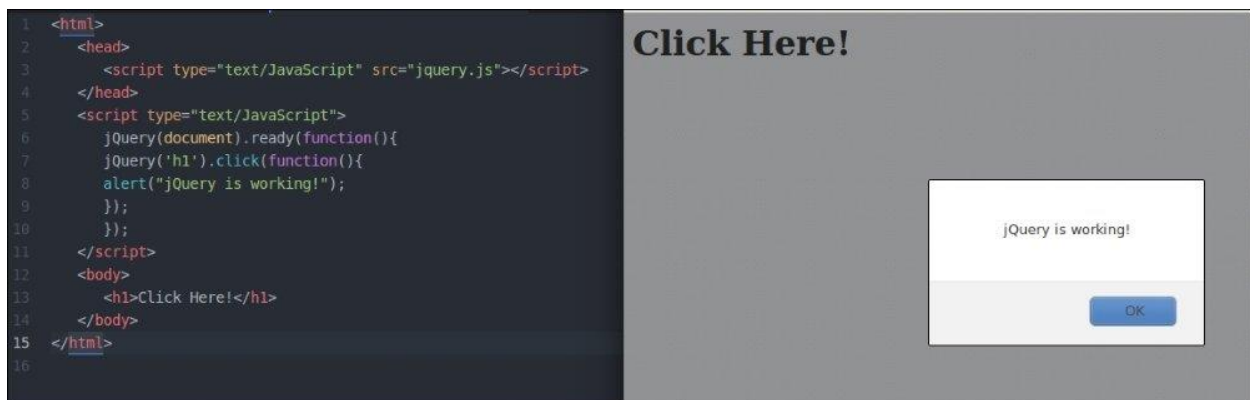
```
<html>
<head>
<script type="text/JavaScript" src="jquery.js"></script>
```

```

</head>
<script type="text/JavaScript">
    jQuery(document).ready(function()
    {
        jQuery('h1').click(function()
        {
            alert("jQuery is working!");
        } //click function ends here.
    );
    } // ready function ends here.
);
</script>
<body>
    <h1>Click Here!</h1>
</body>
</html> Copy

```

After opening the HTML file, click on **Click Here!** You will see the following screen:



It means your jQuery is working.

Let's discuss the code that we have written.

## Note

You can also install jQuery without downloading it. This kind of installation is known as **content delivery network (CDN)** installation.

You need to add the following line to your HTML document and if you're connected online, your browser will automatically load jQuery.

```
<script type = "text/javascript" src =  
"http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></scri  
pt> Copy
```

## Explaining the code

---

Now, let's discuss the code that we previously used. We used the following function in our code:

```
jQuery(document).ready(function() {  
//our codes.  
}); Copy
```

This is a jQuery function that allows you to set your jQuery ready to be used. You can replace **jQuery** with a dollar sign (\$) as shown in the following:

```
$(document).ready(function() {  
//our codes.  
}); Copy
```

You need to think where you want to apply jQuery. We have written `<h1>Click Here!</h1>` in our body tags. We wanted our **Click Here!** to do something when clicked and that's why we added a **click** function similar to the following format:



```
jQuery('h1').click(function() {  
    //our codes.  
});
```

 Copy

The `jQuery` can be replaced with `$` as earlier mentioned.

We added an `alert` function so that when we click on our text, there appears an alert box.

## Going deeper

---

Let's discuss jQuery functions/methods that we use frequently in detail.

All the methods should be written in the `ready()` function. Some of the commonly used methods are as follows:

- Load
- Keyup
- Keydown
- Change
- Focus
- Blur
- Resize
- Scroll

### The load() method

Using this method, you can load a file on your browser. Consider that you want to fetch some text from a `.txt` file on your browser. You can do the following coding:

```
<html>  
<head>  
<script type="text/JavaScript" src="jquery.js"></script>
```

```

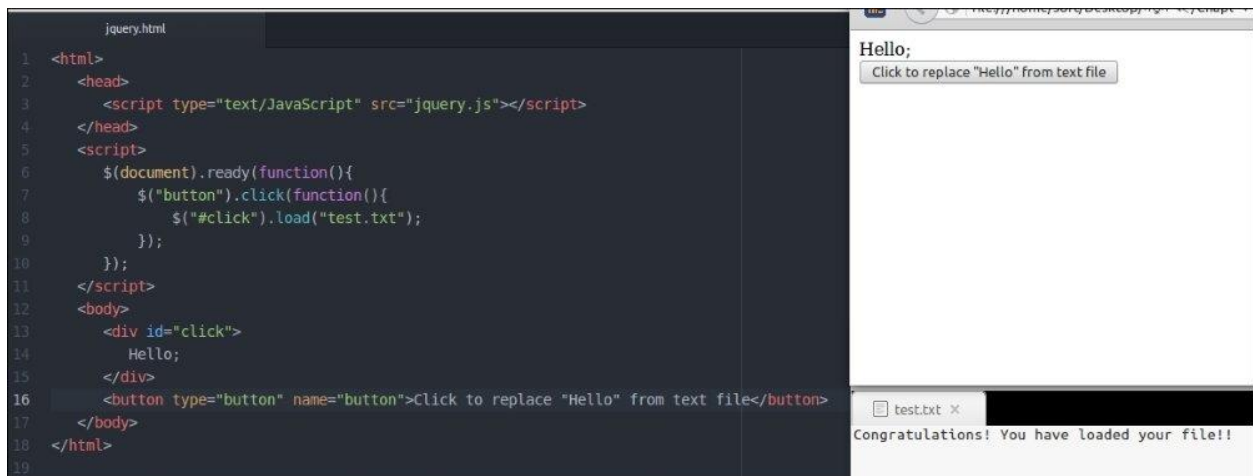
</head>
<script>
    $(document).ready(function() {
        $("button").click(function() {
            $("#click").load("test.txt");
        });
    });
</script>
<body>
    <div id="click">
        Hello;
    </div>
    <button type="button" name="button">Click to replace "Hello" from
text file</button>
</body>
</html>

```

AJAX function  
request back-end server

Copy

After clicking the button, the text in the `click` div will be changed to **Congratulations!** **You have loaded your file!!**, as shown in the following:



## The keyup() and keydown() methods


Using this method, you can control your keyboard buttons' key-pressing. You can make your browser do something when a key is pressed or not pressed. Consider that you have a textbox and you want to take an input from there. When the keys are pressed, you want your textbox to change to red color; otherwise the color should remain green. You can do this by implementing/writing the following code:

```
<html>
  <head>
    <script type="text/JavaScript" src="jquery.js"></script>
  </head>
  <script>
    $(document).ready(function() {
      $("input").keydown(function() {
        $("input").css("background-color", "green");
      });
      $("input").keyup(function() {
        $("input").css("background-color", "red");
      });
    });
  </script>
  <body>
    Type Something: <input type="text">
  </body>
</html> Copy
```

```

jquery.html
1 <html>
2   <head>
3     <script type="text/JavaScript" src="jquery.js"></script>
4   </head>
5   <script>
6     $(document).ready(function(){
7       $("input").keydown(function(){
8         $("input").css("background-color", "green");
9       });
10      $("input").keyup(function(){
11        $("input").css("background-color", "red");
12      });
13    });
14  </script>
15  <body>
16    Type Something: <input type="text">
17  </body>
18 </html>

```



### The change() method

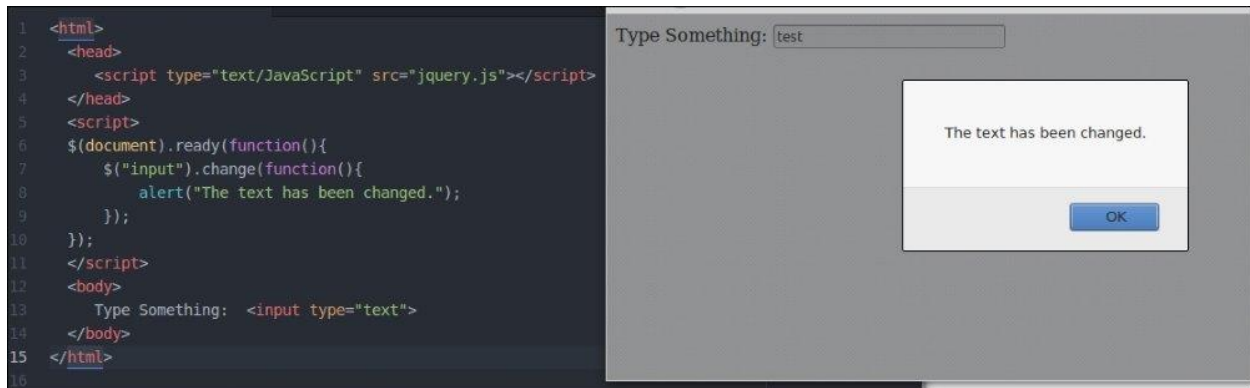
To change some text, you can use this method by implementing the following code:

```

<html>
  <head>
    <script type="text/JavaScript" src="jquery.js"></script>
  </head>
  <script>
    $(document).ready(function() {
      $("input").change(function() {
        alert("The text has been changed.");
      });
    });
  </script>
  <body>
    Type Something: <input type="text">
  </body>
</html> Copy

```

Your output will look similar to the following image:



The blur() and focus() methods

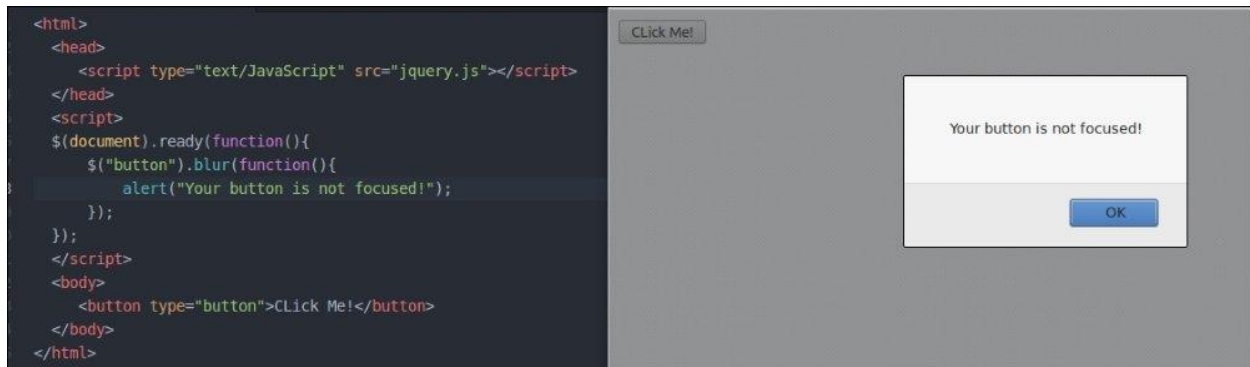
To make your text or button blurred or focused, you can implement the following code:

```

<html>
  <head>
    <script type="text/JavaScript" src="jquery.js"></script>
  </head>
  <script>
    $(document).ready(function() {
      $("button").blur(function() {
        alert("Your button is not focused!");
      });
    });
  </script>
  <body>
    <button type="button">Click Me!</button>
  </body>
</html> Copy

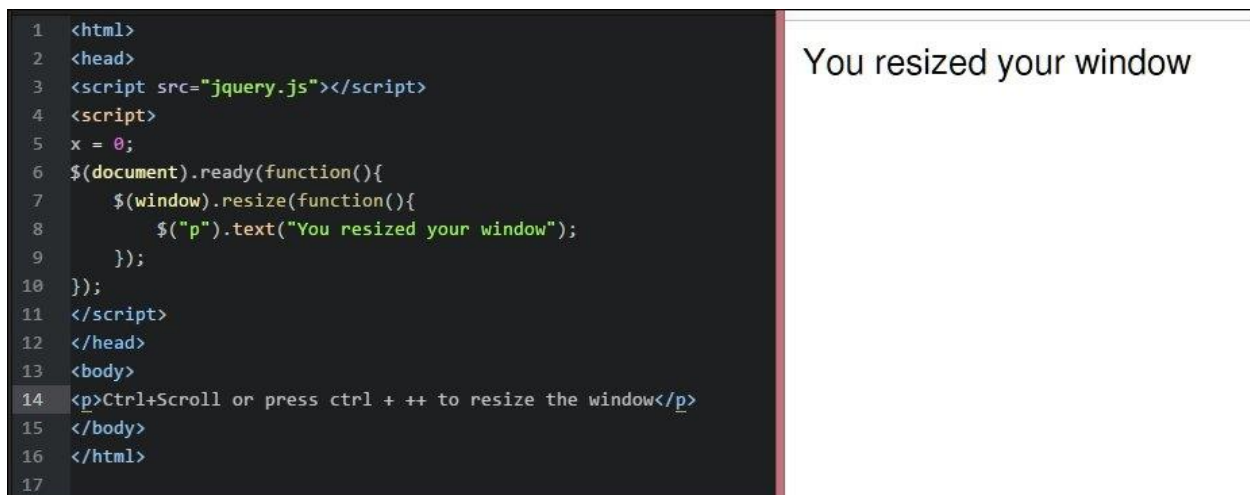
```

You can do this for the `focus()` method too, as follows:



The `resize()` method

If you want to see how many times your browser is resized, you can do the following on your HTML document:



The `scroll()` method

You can add actions to the mouse scrolling using the following code:



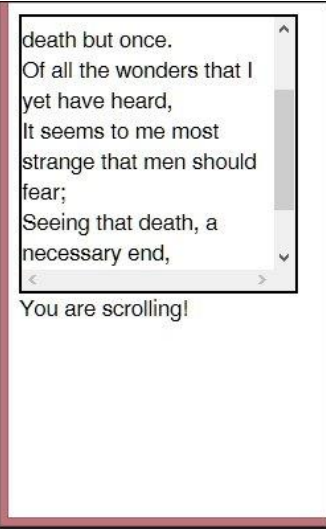
```
$(document).ready(function() {  
    $("div").scroll(function() {  
        $("span").text("You are scrolling!");  
    });  
});  
  
</script>  
</head>  
<body>  
    <div style="border:2px solid black;width:200px;  
height:200px;overflow:scroll;">  
        Cowards die many times before their deaths;<br>  
        The valiant never taste of death but once.<br>  
        Of all the wonders that I yet have heard,<br>  
        It seems to me most strange that men should fear;<br>  
        Seeing that death, a necessary end,<br>  
        Will come when it will come.<br>  
    </div>  
    <span></span>  
</body>  
</html> Copy
```

When you scroll with your mouse, you can see the event that you created in the `scroll()` function, as follows:

```

1 <html>
2 <head>
3 <script src="jquery.js"></script>
4 <script>
5 $(document).ready(function(){
6     $("div").scroll(function(){
7         $("span").text("You are scrolling!");
8     });
9 });
10 </script>
11 </head>
12 <body>
13 <div style="border:2px solid black;width:200px;height:200px;overflow:scroll;">
14     Cowards die many times before their deaths;<br>
15     The valiant never taste of death but once.<br>
16     Of all the wonders that I yet have heard,<br>
17     It seems to me most strange that men should fear;<br>
18     Seeing that death, a necessary end,<br>
19     Will come when it will come.<br>
20 </div>
21 <span></span>
22 </body>
23 </html>

```



The screenshot shows a browser window with a scrollable div containing the following text:

death but once.  
Of all the wonders that I  
yet have heard,  
It seems to me most  
strange that men should  
fear;  
Seeing that death, a  
necessary end,  
Will come when it will come.

Below the scrollable div, the text "You are scrolling!" is displayed, indicating that the jQuery scroll event has been triggered.

## Summary

---

The jQuery library is super fun to use and easy for new learners. All you have to do is practice the methods and functions of jQuery. There are a lot of jQuery plugins online. You can also download and install them to your web page. Using jQuery and its plugins, you can beautifully decorate and code your site easily. The most interesting part of jQuery, for me, is animation. I will explain how to animate things using jQuery in the next chapter.



## Chapter 7. Introducing the Canvas

In this chapter, we are going to learn about HTML canvas. An HTML canvas helps you to draw, especially the graphics (for example, circles, squares, rectangles, and so on) on your HTML page. The `<canvas></canvas>` tags are usually controlled by JavaScript. Canvas can draw texts, which can also be animated. Let's see what we can do using the HTML canvas.

### Implementing canvas

---

To add canvas on your HTML page, you need to define the height and width of your canvas in the `<canvas></canvas>` tags as shown in the following:

```
<html>
  <head>
    <title>Canvas</title>
  </head>
  <body>
    <canvas id="canvasTest" width="200" height="100" style="border:2px
solid #000;">

  </canvas>
</body>
</html>
```

We have defined the canvas ID as `canvasTest`, which will be used to play with the canvas. We used inline CSS on our canvas. The 2 px solid border is used to have a better view of the canvas.

### Adding JavaScript

---

Now, we are going to add few lines of JavaScript for our canvas. We need to add our JavaScript just after the `<canvas></canvas>` tags in the `<script></script>` tags.

## Drawing a rectangle

---

To test our canvas, let's draw a rectangle in the canvas by typing the following code:

```
<script type="text/javascript">
  var canvas = document.getElementById("canvasTest"); //called our
  canvas by id
  var canvasElement = canvas.getContext("2d"); // made our canvas 2D
  canvasElement.fillStyle = "black"; //Filled the canvas black
  canvasElement.fillRect(10, 10, 50, 50); //created a rectangle
</script>
```

In the script, we declared two JavaScript variables. The `canvas` variable is used to hold the content of our canvas using the canvas ID, which we used in our `<canvas></canvas>` tags. The `canvasElement` variable is used to hold the context of the canvas. We assign `black` to `fillstyle` so that the rectangle that we want to draw turns black when filled. We used `canvasElement.fillRect(x, y, w, h)`; for the shape of the rectangle. Where `x` is the distance of the rectangle from the x axis; `y` is the distance of the rectangle from the y axis; and `w` and `h` are the width and height of the rectangle, respectively.

The full code is as shown in the following:

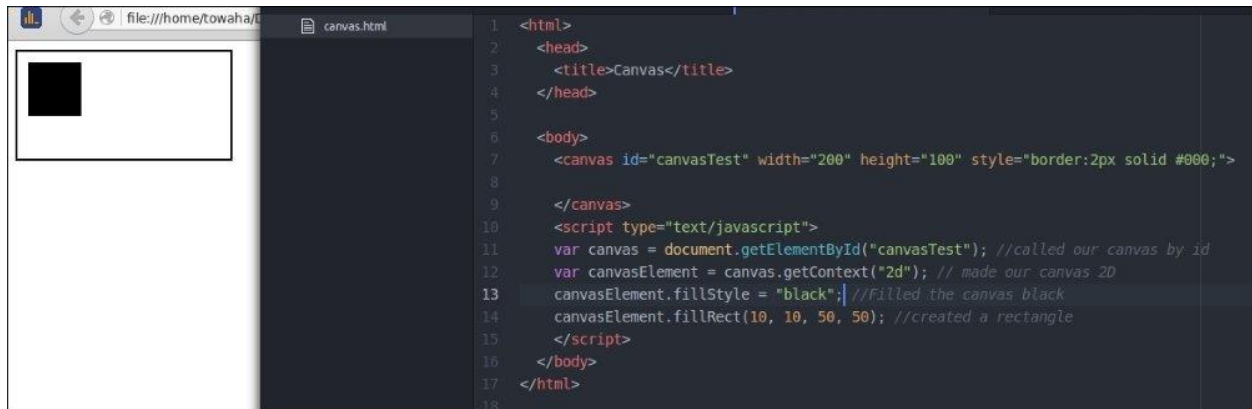
```
<html>
  <head>
    <title>Canvas</title>
  </head>
```

```

<body>
  <canvas id="canvasTest" width="200" height="100" style="border:2px
solid #000;">
  </canvas>
  <script type="text/javascript">
    var canvas = document.getElementById("canvasTest"); //called our
canvas by id
    var canvasElement = canvas.getContext("2d"); // made our canvas 2D
    canvasElement.fillStyle = "black"; //Filled the canvas black
    canvasElement.fillRect(10, 10, 50, 50); //created a rectangle
  </script>
</body>
</html>

```

The output of the code is as follows:



## Drawing a line

---

To draw a line in the canvas that you need to insert the following code in your `<script></script>` tags:

---

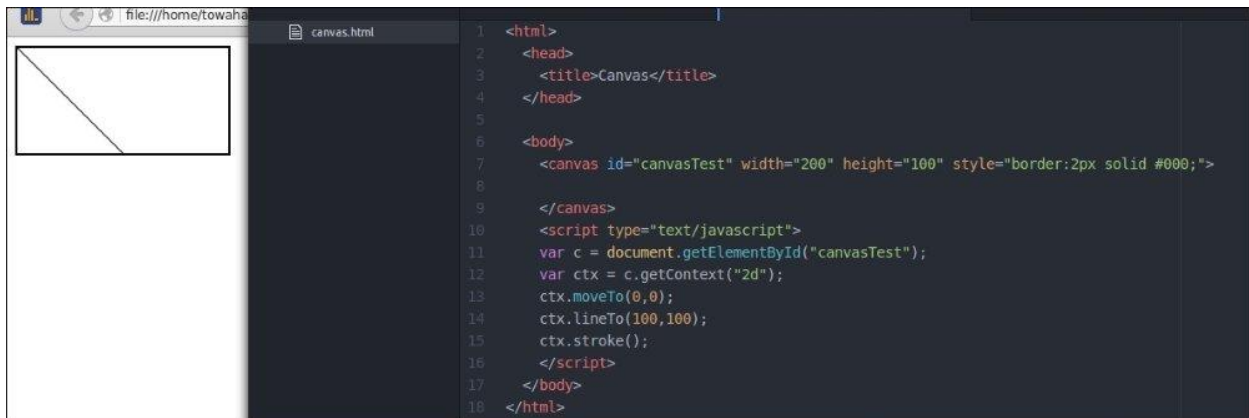
```

<script type="text/javascript">
  var c = document.getElementById("canvasTest");
  var canvasElement = c.getContext("2d");
  canvasElement.moveTo(0,0);
  canvasElement.lineTo(100,100);
  canvasElement.stroke();
</script>

```

Here, `canvasElement.moveTo(0,0);` is used to have our line start from the (0,0) co-ordinate of our canvas. The `canvasElement.lineTo(100,100);` statement is used to make the line diagonal. The `canvasElement.stroke();` statement is used to make the line visible. I would suggest you to change the numbers in `canvasElement.lineTo(100,100);` and `canvasElement.moveTo(0,0);` and see the changes to your line drawn by canvas.

The following is the output of the code:



## A quick exercise

1. Draw a line using canvas and JavaScript, which will be parallel to the y axis of the canvas.
2. Draw a rectangle having 300 px height and 200 px width. Draw a line on the same canvas, touching the rectangle.

## Drawing a circle

---

To draw a circle in the canvas, you need to add the following code in your `<script></script>` tags:

```
<script type="text/javascript">
  var c = document.getElementById("canvasTest");
  var canvasElement = c.getContext("2d");
  canvasElement.beginPath();
  canvasElement.arc(95,50,40,0,2*Math.PI);
  canvasElement.stroke();
</script>
```

Here, we used `canvasElement.beginPath();` to start drawing the circle, `canvasElement.arc(95,50,40,0,2*Math.PI);` for the shape of the circle, and `canvasElement.stroke();` to set the circle visible.

### Note

The `canvasElement.arc(95,50,40,0,2*Math.PI);` statement is similar to `canvasElement.arc(x, y, r, sA, eA, ac);`,

where `x` is the starting coordinate from x axis, `y` is the starting coordinate from y axis, `r` is the radius of the circle, `sA` is the starting angle of the circle, `eA` is the ending angle of the circle, and `ac` is the direction of the circle. Here, `ac` denotes anticlockwise.

The output of our code will be the following image:



## # Draw linear gradient

Let's draw something new. We will draw a rectangle and make its color fade gradually. Type the following code in your `<script></script>` tags:

```

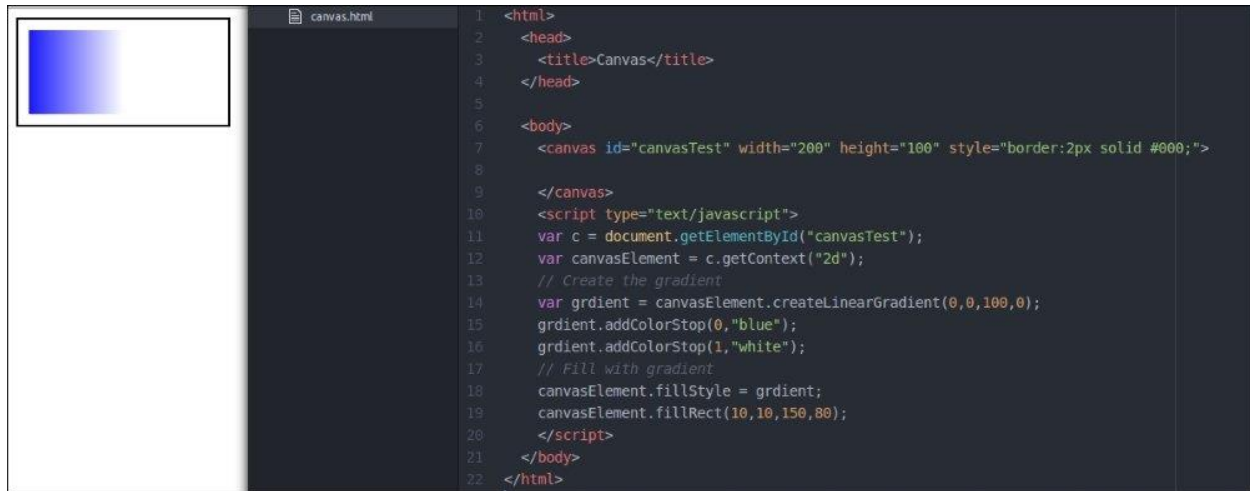
<script type="text/javascript">
  var c = document.getElementById("canvasTest");
  var canvasElement = c.getContext("2d");
  // Create the gradient
  var gradient = canvasElement.createLinearGradient(0,0,100,0);
  gradient.addColorStop(0,"blue"); // here we added blue as our primary
  color
  gradient.addColorStop(1,"white"); //here we used white as our secondary
  color.
  // Fill with gradient
  canvasElement.fillStyle = gradient;
  canvasElement.fillRect(10,10,150,80);
</script>

```

We added `canvasElement.createLinearGradient(0,0,100,0);` to create the gradient or fading. We

added `gradient.addColorStop(0,"blue");` and `gradient.addColorStop(1,"white");` to color the rectangle.

The output of the code is as shown in the following image:



## A quick exercise

---

1. Draw the following smiley face using HTML canvas. (**Hint:** you will have to draw three full circles and a half circle. The trick is that you can draw the figure by playing with the code of circle for the canvas.):



2. Draw a circle with a color gradient.

## Let's make a clock!

---

We are going to draw an analog clock and make it work as a real clock. In your body part of the HTML document, type the following code:

```
<canvas id="myclock" height="500" width="500"></canvas>
```

In your `<script></script>` tags, take the following variables:

```
Var canvas; // the clock canvas
var canvasElement; // canvas's elements
```

```
// clock settings
```

```
var cX = 0;
```

```
var cY = 0;
```

```
var radius = 150;
```

Here, `cX` and `cY` are the center coordinates of our clock. We took 150 px as the clock's radius. You can increase or decrease it.

Then, we need to initialize the variables. Make an `init();` function after defining the preceding variables.

The function should look similar to the following:

```
function init() {
    canvas = document.getElementById("myclock");
    //Called the element to work on.
    canvasElement = canvas.getContext("2d");
    //Made the context 2d.

    cX = canvas.width / 2;
    // we divided by two to get the middle point of X-axis
    cY = canvas.height / 2;
    // we divided by two to get the middle point of Y-axis
    initTime(); //called the initTime() function.
```



```

drawClock(); //Called the drawClock() function to draw the graphics.

setInterval("animateClock()", 1000); // Made the animation for each
second. Here 1000 is equal to 1 second.

}

```

Let's initialize the second, minute, and hour hands of our clock:

```

function initTime() {

    date = new Date();

    hours = date.getHours() % 12; // Divided by 12 to make our clock 12
hours.

    minutes = date.getMinutes();

    seconds = date.getSeconds();

}

```

Here, `date.getHours()`, `date.getMinutes()`, and `date.getSeconds()` will return your computer's time and save them on our variables.

Make another function that will animate our clock:

```

function animateClock() {

    //This function will help our 'second' hand to move after an interval.

    clearCanvas(); // This will clear the canvas

    refreshTime(); // This will refresh time after 1 second.

    drawClock(); // This will draw the clock.

}

```

We will write `clearCanvas()`, `refreshTime()`, and `drawClock()` now:

```
function clearCanvas() {
    canvasElement.clearRect(0, 0, canvas.width, canvas.height);
}
```

Here, `canvasElement.clearRect(0, 0, canvas.width, canvas.height);` will reset our canvas after a definite time interval.

Our `refreshTime()` function should look as shown in the following:

```
function refreshTime() {
    seconds += 1;
    if (Math.floor((seconds / 60)) != 0) { //we divide seconds by 60 until
second is equal to zero.
        minutes += 1; // If 60 second is passed we increment minute by 1.
        seconds %= 60;
    }
    if (Math.floor((minutes / 60)) != 0) {
        hours += 1; //We increment hour by 1 after 60 minutes.
        minutes %= 60;
    }
}
```

We incremented our `seconds` variable in the `refreshTime()` function. Therefore, whenever this function is called, our variable will be incremented by `1`. Then, we have done two conditional operations for our `hours` and `minutes`.

Now, let's draw the clock:

```
function drawClock() {
    drawClockBackground(); //This draws clock background.
    drawSecondsHand(); //This draws clock's second hand.
```

```

    drawMinutesHand(); //This draws clock's minute hand.
    drawHoursHand(); //This draws clock's hour hand.
}

```

We will write the `drawClockBackground()`, `drawSecondsHand()`, `drawMinutesHand()`, and `drawHoursHand()` functions:

```

function drawClockBackground() {
    //this function will draw the background of our clock. We are
    declaring few variables for mathematical purposes.

    var correction = 1/300;
    var shift_unit = 1/170;
    var shift_factor = 1/30;
    var angle_initial_position = 2;
    var angle_current_position_begin = 0;
    var angle_current_position_end = 0;
    var repeat = 60;
    var lineWidth = 10;

    for (var i=0; i < repeat; i+=1) {
        //These lines are written for making our clock error free with the
        angle of the hands (hands' positions)
        angle_current_position_begin = angle_initial_position - (i *
        shift_factor) - correction;
        angle_current_position_end = angle_current_position_begin +
        shift_unit;

        if (i % 5 === 0)
            lineWidth = 20;
    }
}

```

```

else
    lineWidth = 10;

    drawArcAtPosition(cX, cY, radius,
angle_current_position_begin*Math.PI,
angle_current_position_end*Math.PI, false, lineWidth);
}
drawLittleCircle(cX, cY);
}

```

We performed some mathematical things in this function and wrote the `drawLittleCircle(cX, cY)` function for a little circle on the center of our clock.

The function should look similar to the following:

```

function drawLittleCircle(cX, cY) {
    drawArcAtPosition(cX, cY, 4, 0*Math.PI, 2*Math.PI, false, 4);
}

```

Write the `drawSecondsHand()` function. This function will draw the second hand, as follows:

```

function drawSecondsHand() {
    /* Simple mathematics to find the co ordinate of the second hand;
    You may know this:  $x = r\cos(\theta)$ ,  $y = r\sin(\theta)$ . We used these
    here.
    We divided the values n=by 30 because after 5 seconds the second
    hand moves 30 degree.
    */
    endX = cX + radius*Math.sin(seconds*Math.PI / 30);
}

```

```

endY = cY - radius*Math.cos(seconds*Math.PI / 30);
drawHand(cX, cY, endX, endY);
}

```

Our `drawMinutesHand()` function should look as shown in the following. This function will draw the minute hand of our clock, as follows:

```

function drawMinutesHand() {
    var rotationUnit = minutes + seconds / 60;
    var rotationFactor = Math.PI / 30;
    var rotation = rotationUnit*rotationFactor;
    var handLength = 0.8*radius;
    endX = cX + handLength*Math.sin(rotation);
    endY = cY - handLength*Math.cos(rotation);
    drawHand(cX, cY, endX, endY);
}

```

Now, let's see our `drawHoursHand()` function. This function will draw the hour hand:

```

function drawHoursHand() {
    var rotationUnit = 5 * hours + minutes / 12;
    var rotationFactor = Math.PI / 30;
    var rotation = rotationUnit*rotationFactor;
    var handLength = 0.4*radius;

    endX = cX + handLength*Math.sin(rotation);
    endY = cY - handLength*Math.cos(rotation);
    drawHand(cX, cY, endX, endY);
}

```

We used a `drawHand()` function in the preceding functions. Let's write the function, as follows:

```
function drawHand(beginX, beginY, endX, endY) {  
    canvasElement.beginPath();  
    canvasElement.moveTo(beginX, beginY);  
    canvasElement.lineTo(endX, endY);  
    canvasElement.stroke();  
    canvasElement.closePath();  
}
```

Now, we are going to write the last function for our clock, as shown in the following snippet:

```
function drawArcAtPosition(cX, cY, radius, start_angle, end_angle,  
    counterclockwise, lineWidth) {  
    canvasElement.beginPath();  
    canvasElement.arc(cX, cY, radius, start_angle, end_angle,  
        counterclockwise);  
    canvasElement.lineWidth = lineWidth;  
    canvasElement.strokeStyle = "black";  
    canvasElement.stroke();  
    canvasElement.closePath();  
}
```

The full code of our clock should look similar to the following code:

```
<html>  
  <head>  
    <script type="text/javascript">
```

```
var canvas;
var canvasElement;

// clock settings
var cX = 0;

var cY = 0;
var radius = 150;

// time settings
var date;
var hours;
var minutes;
var seconds;

function init() {
    canvas = document.getElementById("myclock");
    canvasElement = canvas.getContext("2d");

    cX = canvas.width / 2;
    cY = canvas.height / 2;

    initTime();
    drawClock();
    setInterval("animateClock()", 1000);
}

// get your system time
function initTime() {
```

```
        date = new Date();
        hours = date.getHours() % 12;
        minutes = date.getMinutes();
        seconds = date.getSeconds();
    }

    // animate the clock
    function animateClock() {
        clearCanvas();
        refreshTime();
        drawClock();
    }

    // clear the canvas
    function clearCanvas() {
        canvasElement.clearRect(0, 0, canvas.width, canvas.height);
    }

    // refresh time after 1 second
    function refreshTime() {
        seconds += 1;
        if (Math.floor((seconds / 60)) != 0) { minutes += 1; seconds %=
60; }
        if (Math.floor((minutes / 60)) != 0) { hours += 1; minutes %=
60; }
    }

    // draw or redraw Clock after time refresh function is called
```



```
function drawClock() {  
    drawClockBackground();  
    drawSecondsHand();  
    drawMinutesHand();  
    drawHoursHand();  
}  
  
function drawHand(beginX, beginY, endX, endY) {  
    canvasElement.beginPath();  
    canvasElement.moveTo(beginX, beginY);  
    canvasElement.lineTo(endX, endY);  
    canvasElement.stroke();  
    canvasElement.closePath();  
}  
  
// draw Hand for seconds  
function drawSecondsHand() {  
    endX = cX + radius*Math.sin(seconds*Math.PI / 30);  
    endY = cY - radius*Math.cos(seconds*Math.PI / 30);  
    drawHand(cX, cY, endX, endY);  
}  
  
// draw Hand for minutes  
function drawMinutesHand() {  
    var rotationUnit = minutes + seconds / 60;  
    var rotationFactor = Math.PI / 30;  
    var rotation = rotationUnit*rotationFactor;  
    var handLength = 0.8*radius;
```

```
    endX = cX + handLength*Math.sin(rotation);  
    endY = cY - handLength*Math.cos(rotation);  
    drawHand(cX, cY, endX, endY);  
}
```

```
// draw Hand for hours  
function drawHoursHand() {  
    var rotationUnit = 5 * hours + minutes / 12;  
    var rotationFactor = Math.PI / 30;  
    var rotation = rotationUnit*rotationFactor;  
    var handLength = 0.4*radius;
```

```
    endX = cX + handLength*Math.sin(rotation);  
    endY = cY - handLength*Math.cos(rotation);  
    drawHand(cX, cY, endX, endY);  
}
```

```
function drawClockBackground() {  
    var correction = 1/300;  
    var shift_unit = 1/170;  
    var shift_factor = 1/30;  
    var angle_initial_position = 2;  
    var angle_current_position_begin = 0;  
    var angle_current_position_end = 0;  
    var repeat = 60;  
    var lineWidth = 10;  
  
    for (var i=0; i < repeat; i+=1) {
```

```

        angle_current_position_begin = angle_initial_position - (i *
shift_factor) - correction;

        angle_current_position_end = angle_current_position_begin +
shift_unit;

```

```

        if (i % 5 == 0) lineWidth = 20;

```

```

        else lineWidth = 10;

```

```

        drawArcAtPosition(cX, cY, radius,
angle_current_position_begin*Math.PI,
angle_current_position_end*Math.PI, false, lineWidth);
    }
    drawLittleCircle(cX, cY);
}

```

```

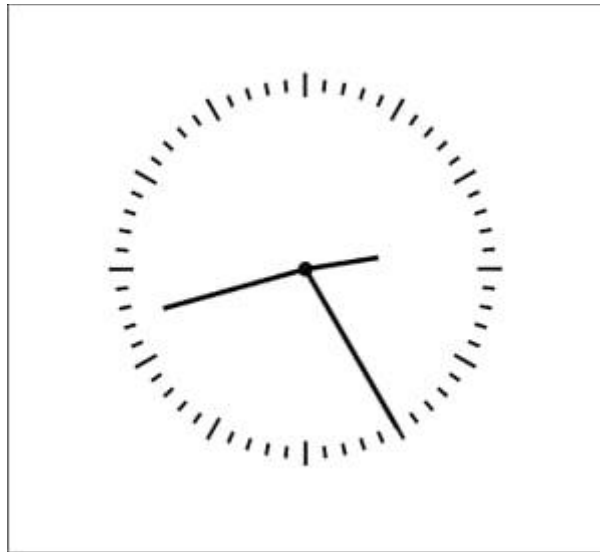
function drawArcAtPosition(cX, cY, radius, start_angle, end_angle,
counterclockwise, lineWidth) {
    canvasElement.beginPath();
    canvasElement.arc(cX, cY, radius, start_angle, end_angle,
counterclockwise);
    canvasElement.lineWidth = lineWidth;
    canvasElement.strokeStyle = "black";
    canvasElement.stroke();
    canvasElement.closePath();
}

function drawLittleCircle(cX, cY) {
    drawArcAtPosition(cX, cY, 4, 0*Math.PI, 2*Math.PI, false, 4);
}

```

```
</script>  
</head>  
<body onload="init()">  
  <canvas id="myclock" height="500" width="500"></canvas>  
</body>  
</html>
```

If you can see the output of your code as the following image, then congratulations! You successfully created your HTML clock using canvas:



## Summary

---

In this chapter, we have learned the basics of HTML canvas. I hope that you can now draw anything using the HTML canvas. You may have played online games; the components of most of them are drawn using HTML canvas. Therefore, if you want to develop your own web applications or games, you need to learn about canvas. You can easily code to draw and animate the shapes using JavaScript.

In the next chapter, we will develop a game called **Rat-man** using the HTML canvas. Before starting [Chapter 8](#), *Building Rat-man!*, I hope that you've learned a lot about HTML canvas through this chapter. If you've, let's develop our game now.

## Chapter 8. Tidying up Your Code Using OOP

In this chapter, we are going to learn about **object-oriented programming (OOP)** and discuss the code of the famous game, **Hangman**.

"OOP is a programming paradigm that uses abstraction to create models based on the real world. OOP uses several techniques from previously established paradigms, including modularity, polymorphism, and encapsulation." or "OOP languages typically are identified through their use of classes to create multiple objects that have the same properties and methods."

You probably have assumed that JavaScript is an object-oriented programming language. Yes, you are absolutely correct. Let's see why it is an OOP language. If a computer programming language has the following few features, we call it object oriented:

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

Before going any further, let's discuss **objects**. We create objects in JavaScript in the following manner:

```
var person = new Object();  
person.name = "Harry Potter";  
person.age = 22;  
person.job = "Magician";
```

We created an object for a person. We added few properties of person.

If we want to access any of the property of the object, we need to call the property.

Consider that you want to have a popup of the **name** property of the preceding **person** object. You can do this with the following method:

```
person.callName = function(){  
    alert(this.name);  
};
```

We can write the preceding code as shown in the following:

```
var person = {  
    name: "Harry Potter",  
    age: 22,  
    job: "Magician",  
    callName: function(){  
        alert(this.name);  
    }  
};
```

## Inheritance in JavaScript

---

To inherit means derive something (such as, characteristics, quality, and so on) from one's parents or ancestors. In programming languages, when a class or object is based on another class or object in order to maintain the same behavior of the parent class or object is known as **inheritance**.

We can also say that this is a concept of gaining properties or behaviors of something else.

Suppose, X inherits something from Y; it is like X is a type of Y.

JavaScript occupies the inheritance capability. Let's take a look at an example. A bird inherits from an animal as a bird is a type of animal. Therefore, a bird can do the same thing that an animal does.

This kind of relationship in JavaScript is a little complex and needs a syntax. We need to use a special object called **prototype**, which assigns the properties to a type. We need to remember that only function has prototypes. Our **Animal** function should look similar to the following:

```
function Animal(){  
  //We can code here.  
};
```

To add a few properties of the function, we need to add a prototype as shown in the following:

```
Animal.prototype.eat = function(){  
  alert("Animal can eat.");  
};
```

Let's create prototypes for our **Bird** function. Our function and prototypes should look similar to the following:

```
function Bird(){  
};  
Bird.prototype = new Animal();  
Bird.prototype.fly = function(){  
  alert("Birds can fly.");  
};  
Bird.prototype.sing = function(){  
  alert("Bird can sing.");  
};
```

The result of the prototypes and function is that any **Bird** that you create will have the properties of **Animal** and **Bird**. However, if you create **Animal**, this will only have the properties of **Animal**. The properties of **Animal** are inherited by **Bird**.



Therefore, we can say that JavaScript has inheritance property.

## Encapsulation in JavaScript

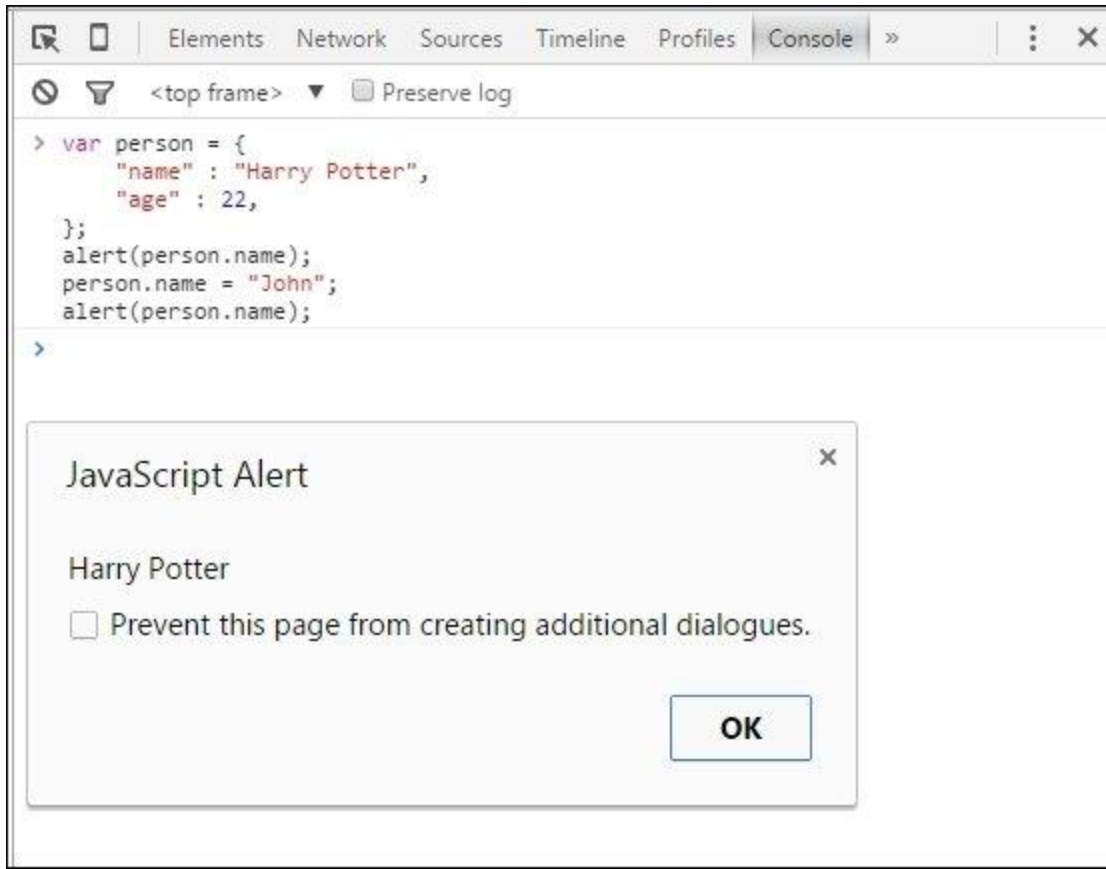
---

In OOP, **encapsulation** is one of the most important concepts that allows an object to group the members of public and private classes under a single name. We use encapsulation to protect our classes against accidental or willful folly. Encapsulation means to enclose something in or as if something is in a capsule.

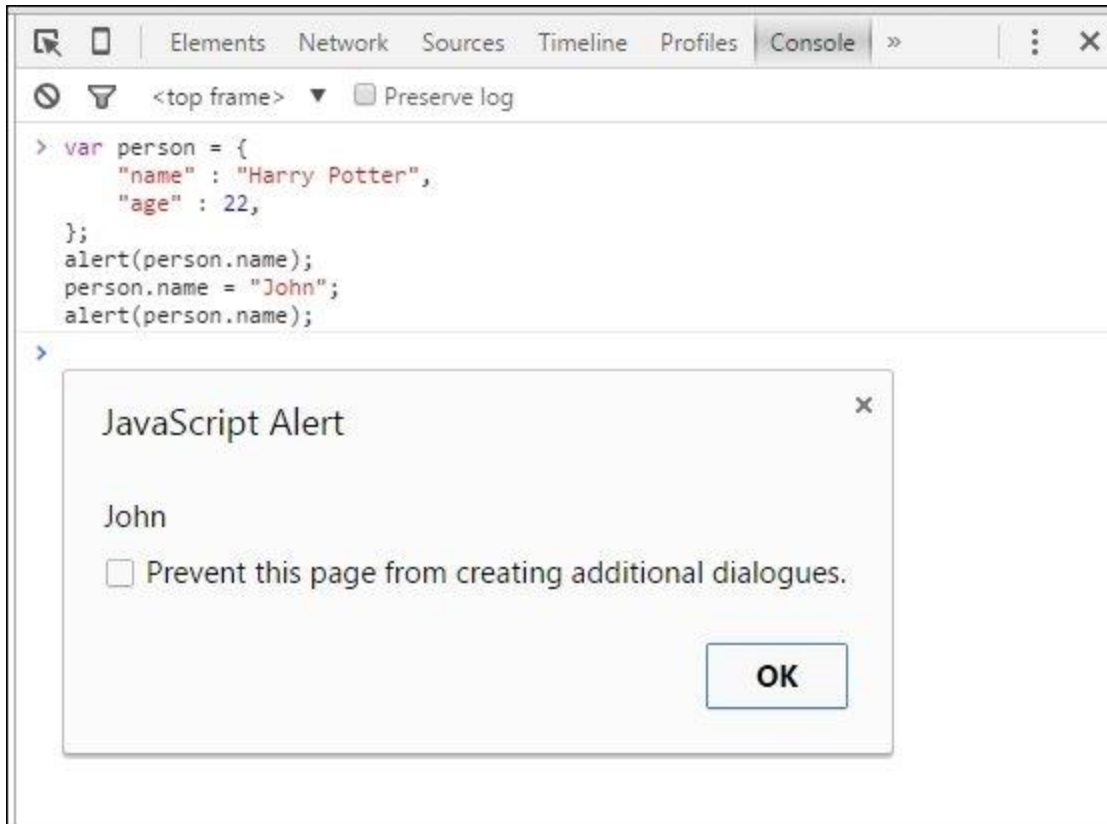
Now, we will see whether JavaScript supports encapsulation. If it does, we can say that JavaScript is an OOP language. Let's take a look at the following example:

```
var person = {  
  "name" : "Harry Potter",  
  "age" : 22,  
};  
alert (person.name) ;  
person.name = "John";  
alert (person.name) ;
```

If we run this on the console. The first alert box will print the following image:



We changed the variable `name` to `John`. Therefore, the second alert box will be similar to the following image:



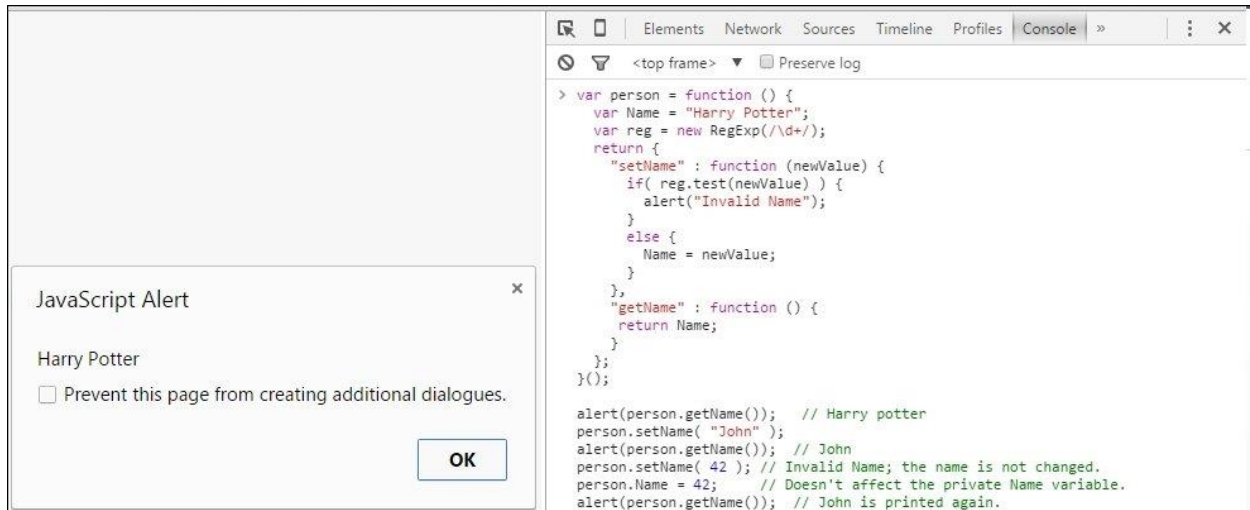
What would happen if we accidentally assigned a number to the `name` variable?

Assigning a number to the `name` variable is perfectly acceptable. As far as JavaScript is concerned, a variable can accept any type of data as its value. However, we don't want a number in the place of a name. What do we do? We can use JavaScript's encapsulation property, as follows:

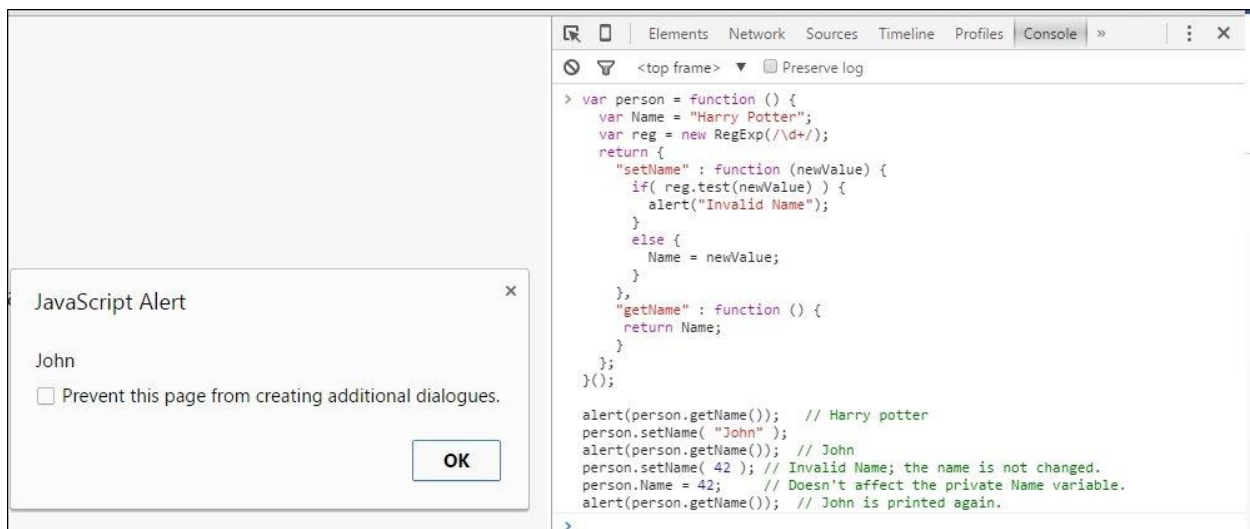
```
var person = function () {
    var Name = "Harry Potter";
    var reg = new RegExp(/\d+/);
    return {
        "setName" : function (newValue) {
            if( reg.test(newValue) ) {
                alert("Invalid Name");
            }
        }
    }
}
```

```
        else {  
            Name = newValue;  
        }  
    },  
    "getName" : function () {  
        return Name;  
    }  
};  
}();  
  
alert(person.getName()); // Harry potter  
person.setName( "John" );  
alert(person.getName()); // John  
person.setName( 42 ); // Invalid Name; the name is not changed.  
person.Name = 42; // Doesn't affect the private Name variable.  
alert(person.getName()); // John is printed again.
```

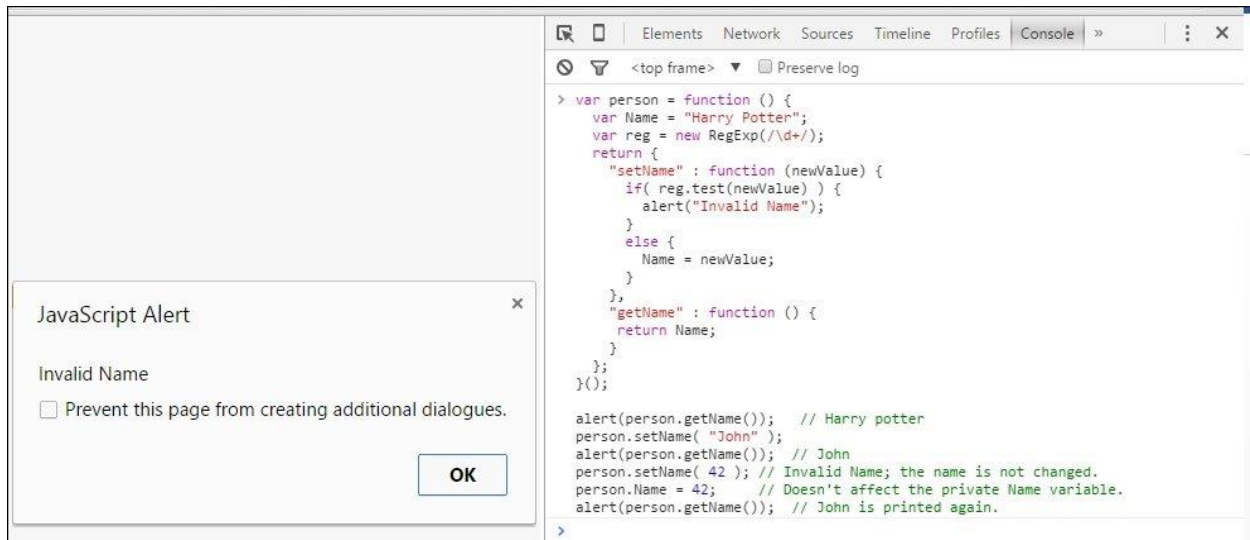
Now, if we run the above code on console, the first output will show a popup with **Harry Potter** as we only called the `getName()` function. The `getName()` function has an initial value, which is **Harry Potter**:



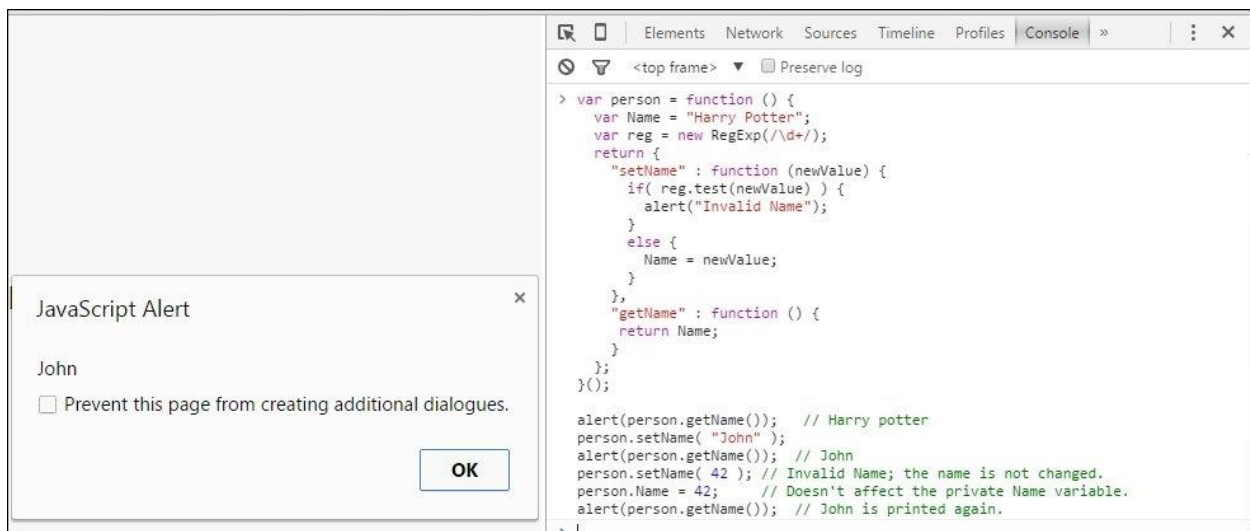
The second output will be as follows as we changed the **Name** property of **person** to **John** and again called the **getName()** function:



The third output will be as shown in the following as we tried to push a number to a string variable. A name cannot be an integer, therefore, **Invalid Name** popped up as we had a condition under the **if** statement:



The fourth output will be as shown in the following as the number was not added to the person's **Name** property. Therefore, we will get the last data that we pushed to the **Name** property:



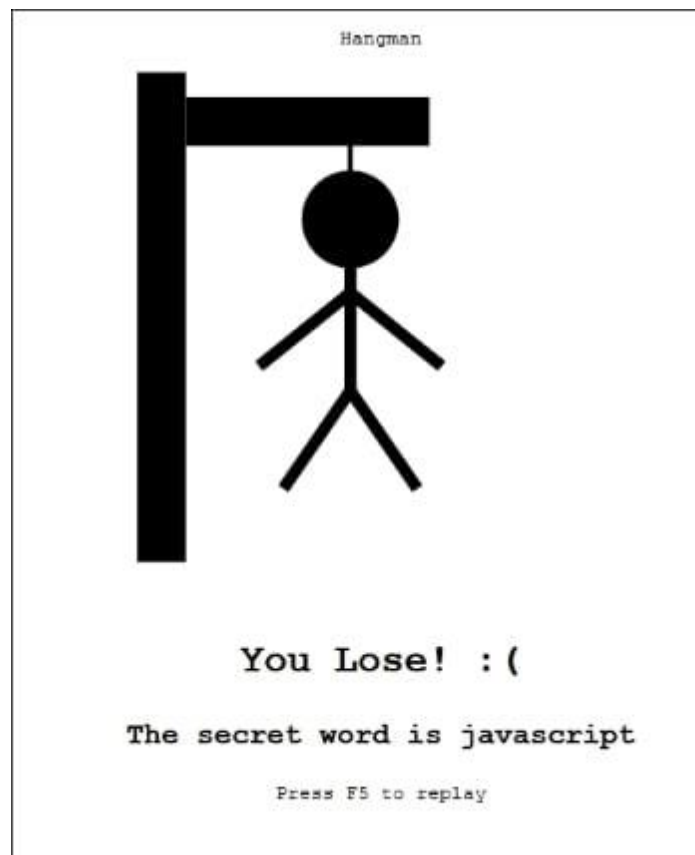
We can now confirm that JavaScript supports encapsulation.

JavaScript also supports **polymorphism** and **abstraction**. If you would like to read about them, you can refer to the following link:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction\\_to\\_Object-Oriented\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript)

Let's do something fun. You may have heard of the game called Hangman. We'll discuss the OOP in that game. First, let's introduce you to the game.

The player needs to guess a word. If he can guess the word correctly, he is safe; otherwise, he will be hanged. Take a look at the following image to get the clear idea about the game, as follows:



## Dissecting Hangman

---

There are two folders and an HTML file for the Hangman game. The two folders are named as `css` and `js`. The `index.html` HTML file should contain the following code:

```
<html lang="en" ng-app="hangman">  
  
<head>  
  
<title>Hangman</title>
```

```

<link rel="stylesheet" href="css/styles.css">
<script src="js/angular.min.js"></script>
</head>
<body ng-controller="StartHangman">
  <p>Hangman</p>
  <svg width="400" height="400">
    <rect ng-show="failedGuess.length >= 1" x="0" y="0" width="40"
height="400"></rect>
    <rect ng-show="failedGuess.length >= 2" x="40" y="20" width="200"
height="40"></rect>
    <rect ng-show="failedGuess.length >= 3" x="173" y="50" width="4"
height="100"></rect>
    <circle ng-show="failedGuess.length >= 3" cx="175" cy="120"
r="40"></circle>
    <line ng-show="failedGuess.length >= 4" x1="175" y1="150" x2="175"
y2="185" style="stroke:rgb(0,0,0)" stroke-width="10"></line>
    <line ng-show="failedGuess.length >= 4" x1="175" y1="180" x2="100"
y2="240" style="stroke:rgb(0,0,0)" stroke-width="10"></line>
    <line ng-show="failedGuess.length >= 5" x1="175" y1="180" x2="250"
y2="240" style="stroke:rgb(0,0,0)" stroke-width="10"></line>
    <line ng-show="failedGuess.length >= 6" x1="175" y1="180" x2="175"
y2="265" style="stroke:rgb(0,0,0)" stroke-width="10"></line>
    <line ng-show="failedGuess.length >= 7" x1="175" y1="260" x2="120"
y2="340" style="stroke:rgb(0,0,0)" stroke-width="10"></line>
    <line ng-show="failedGuess.length >= 8" x1="175" y1="260" x2="230"
y2="340" style="stroke:rgb(0,0,0)" stroke-width="10"></line>
  </svg>

```



```

<div ng-show="stage == 'initial'">
  <h2>Please enter your secret words:</h2>
  <input type="text" ng-model="secretWords" autofocus ng-
keyup="$event.keyCode == 13 ? startGame() : null">
  <button ng-click="startGame()">Enter</button>
</div>

<div ng-show="stage == 'play'">
  <h1>{{ answer }}</h1>
  <h2>Failed guess ({{ failedGuess.length }}) = {{
failedGuess}}</h2>

  <input type="text" ng-model="charGuess" id="char-guess" ng-
keyup="$event.keyCode == 13 ? guess(charGuess) : null"
placeholder="Guess a letter">
  <button ng-click="guess(charGuess)">Enter</button>
</div>

<div ng-show="stage == 'won'">
  <h1>You Win! :)</h1>
  <h2>That's right, the secret words is {{ secretWords }}</h2>
  <p>Press F5 to replay</p>
</div>

<div ng-show="stage == 'lost'">
  <h1>You Lose! :(</h1>
  <h2>The secret word is {{ secretWords }}</h2>

```

```
<p>Press F5 to replay</p>
</div>

<script src="js/hangman.js"></script>
</body>
</html>
```

The `css` folder should have a `styles.css` file. The `styles.css` file should contain the following code:

```
body {
  font-family: monospace;
  text-align: center;
  font-size: 16px;
  line-height: 1.40;
}

input[type="text"] {
  padding: 5px;
  font-family: monospace;
  height: 30px;
  font-size: 1.8em;
  background-color: #fff;
  border: 2px solid #000;
  vertical-align: bottom;
}

svg {
  margin: 0 0 30px;
```

```

}

button {
  cursor: pointer;
  margin: 0;
  height: 44px;
  background-color: #fff;
  border: 2px solid #000;
}

```

There should be two JavaScript files in the `js` folder, `angular.min.js` and `hangman.js`.

The `angular.min.js` file is a framework. You can download it from <https://angularjs.org/> or it is available with the code bundle of the book.

The `hangman.js` file should have the following code:

```

var hangman = angular.module('hangman', []).controller('StartHangman',
StartHangman);

function StartHangman($scope, $document) {
  $scope.stage = "initial";
  $scope.secretWords = "";
  $scope.answer = "";
  $scope.failedGuess = [];
  var hasWon = function() {
    var foundDash = $scope.answer.search(/-/);
    return (foundDash == -1);
  }
  var hasLost = function() {
    return ($scope.failedGuess.length >= 8);
  }
}

```

```

    }

    $scope.startGame = function() {

        $scope.secretWords = $scope.secretWords.toLowerCase();

        for(i in $scope.secretWords) {

            $scope.answer += $scope.secretWords[i] == ' ' ? ' ' : '-';

        }

        $scope.stage = "play"

    }

    $scope.guess = function(ch) {

        ch = ch.toLowerCase();

        $scope.charGuess = "";

        if(ch.length != 1) {

            if(ch.length > 1) {

                alert("Please only enter one character at a time");

            }

            return ;

        }

        /* If ch is already in the failed guess list */

        for(i in $scope.failedGuess) {

            if(ch == $scope.failedGuess[i]) return ;

        }

        /* Check if it's part of the answer */

        var found = false;

        $scope.answer = $scope.answer.split(""); /* convert to array of char
        */

        for(i in $scope.secretWords) {

            if($scope.secretWords[i] === ch) {

```

```

        found = true;
        $scope.answer[i] = ch;
    }
}

$scope.answer = $scope.answer.join(""); /* convert back to string */

if(!found) {
    $scope.failedGuess.push(ch);
}

if(hasWon()) {
    $scope.stage = "won";
}

if(hasLost()) {
    $scope.stage = "lost";
}
}
}
}

```

Let's discuss the code.

We used `var hangman = angular.module('hangman', []).controller('StartHangman', StartHangman);` to import our `angular.min.js` file and start controlling the rest of our game's code.

We wrote a `StartHangman($scope, $document) {}` function, where we will write our code. We passed two variables, `$scope` and `$document`, from our `angular.min.js` file.

We initialized few variables, as follows:

```

$scope.stage = "initial";
$scope.secretWords = "";

```

```
$scope.answer = "";
$scope.failedGuess = [];
```

We wrote two functions for winning and losing the game, as follows:

```
var hasWon = function() {
    var foundDash = $scope.answer.search(/-/);
    return (foundDash == -1);
}
var hasLost = function() {
    return ($scope.failedGuess.length >= 8);
}
```

We have fixed our number of guesses here. Then, we wrote a function to start our game. We made an object and used the inheritance property of JavaScript, as shown in the following:

```
$scope.startGame = function() {
    $scope.secretWords = $scope.secretWords.toLowerCase();
    for(i in $scope.secretWords) {
        $scope.answer += $scope.secretWords[i] == ' ' ? ' ' : '-';
    }
    $scope.stage = "play"
}
```

We took an input from the player in order to store as our secret word.

The prompt page of the game will look similar to the following image:

**Please enter your secret word:**

Then, our most important function, `$scope.guess = function(ch){}`, was introduced. We passed a character to the function and checked whether it matches any letters entered by the player with the secret word.

## Summary

---

In this chapter, you learned the characteristics of an OOP language. We also saw the uses of the OOP characteristics in the famous game, Hangman! I hope you enjoyed creating and playing Hangman. We will see the possibilities of JavaScript in the next and final chapter of this book.