P.S.R. ENGINEERING COLLEGE
Autonomous Institution Affiliated to Anna University, Chennai
Approved by AICTE, New Delhi & Accredited by National Board of Accreditation (NBA)
Accredited by NAAC listed under 12 (B) of the UGC Act, 1956.
An ISO 9001 : 2015 Certified Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

				L	T	P	C
191OE1B	**JAVA SCRIPTS**			**3**	**0**	**0**	**3**
Programme:	B.E Computer Science & Engineering	**Sem:**	**6**	**Category:**		**OE**	
Prerequisites:	**191CS43-Object Oriented Programming**						
Aim	To offer an overview of all JavaScript basics, including HTML for building web pages.						

Course Outcomes: The Students will be able to	
CO1:	Discuss the concepts of Advanced Java Script.
CO2:	Design the processes, and use mechanics for game development.
CO3:	Create interactive Games.
CO4:	Use DOM & JQuery tools and methods to develop game.
CO5:	Design & implement animated webpage on the canvas using java scripts.
CO6:	Create interactive games.

S.No	Topic Name	No. of Periods	Cumulative periods	Reference Books	Teaching Aids
UNIT I – FUNDAMENTALS					
1.	**FUNDAMENTALS** -JavaScript.	1	1	T1	BB
2.	Data types and Variables	1	2	T1	BB
3.	Variables, Strings, Booleans	1	3	T1	BB
4.	Arrays, Objects	2	5	T1	OHP
5.	Basics of HTML	1	6	T1	ABL
6.	Tags and Elements	2	8	T1	OHP
7.	HTML Hierarchy	1	9	T1	OHP
UNIT II – FUNCTIONS AND LOOPS					
8.	Anatomy of a Function	1	10	T1	BB
9.	Function Creation	2	12	T1	OHP
10.	Calling Function	1	13	T1	ABL
11.	Passing Arguments into Functions	2	15	T1	OHP
12.	Conditionals, Loops	2	17	T1	OHP
13.	Programming Challenges for Functions and Loops	1	18	T1	OHP
UNIT III – ADVANCED JAVASCRIPT					
14.	DOM and JQuery	1	19	T1	OHP
15.	Interactive Programming, Mouse Events	2	21	T1	OHP
16.	Buried Treasure - Creating the Web Page with HTML	1	23	T1	ABL
17.	Picking a Random Treasure Location- Click Handler	1	24	T1	BB
18.	Object Oriented Programming	1	25	T1	BB

19.	Adding Methods to Objects	**2**	**27**	**T1**	**PPT**
20.	Creating Objects Using Constructors	**1**	**28**	**T1**	**OHP**
21.	Customizing Objects with Prototypes	**1**	**29**	**T1**	**OHP**
UNIT IV – CANVAS					
22.	Creating a Basic Canvas,	**1**	**30**	**T1**	**BB**
23.	Drawing on the Canvas	**1**	**31**	**T1**	**BB**
24.	Changing the Drawing Color	**1**	**32**	**T1**	**PPT**
25.	Drawing Lines or Paths, Filling Paths Drawing Arcs and Circles,	**2**	**34**	**T1**	**ABL**
26.	Drawing Lots of Circles with Function	**1**	**35**	**T1**	**BB**
27.	Animating the Size of a Square	**1**	**36**	**T1**	**OHP**
28.	Bouncing a Ball, Keyboard Events	**1**	**37**	**T1**	**OHP**
29.	Moving a Ball with the Keyboard	**1**	**38**	**T1**	**OHP**
UNIT V – GAME DEVELOPMENT					
30.	Making a Snake Game	**2**	**40**	**T1**	**BB**
31.	The Structure of the Game-	**1**	**41**	**T1**	**ABL**
32.	Game Setup	**2**	**43**	**T1**	**PPT**
33.	Drawing the Border	**2**	**45**	**T1**	**BB**
34.	Displaying the Score	**1**	**46**	**T1**	**OHP**
35.	Ending the Game	**1**	**47**	**T1**	**OHP**

Text Books:	
1. Nick Morgan, "Java Script for Kids", no starch press, San Francisco, 2015.	
References:	
1. Marijn Haverbeke, "Eloquent Java Script", no starch press, San Francisco, 2014. 2. David Sawyer McFarland, "Java Script & JQuery", 3/e, USA, 2014.	

Course Outcomes	**Programme Outcomes (POs)**												**Programme Specific Outcomes (PSOs)**			
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**	**PSO4**
CO1	3			3						3			3		3	
CO2		3		2								2	2			3
CO3	2		2											2		
CO4		2			3									3	2	2
CO5	2			2						2		3	2			
CO6		2		3											3	2

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

STAFF IN-CHARGE **HOD/CSE**

UNIT 1

FUNDAMENTALS -JAVASCRIPT

JAVA SCRIPTS

Dr.S.AMUTHA M.E.,Ph.D,

Associate Professor

Department Of Computer Science And Engineering

PSR ENGINEERING COLLEGE, SIVAKASI

OUTLINE

1. What is Javascript?
2. History of Javascript
3. Features of JavaScript
4. Applications of Javascript

What is Javascript?

- JavaScript is a light-weight object-oriented programming language which is used by several websites for scripting the webpages

- Full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.

What is Javascript?

- JavaScript is used to create client-side dynamic pages.
- JavaScript is *an object-based scripting language* which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

History of JavaScript

- 1993, **Mosaic**, the first popular web browser, came into existence.
- In the **year 1994**, **Netscape** was founded by **Marc Andreessen**
- in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser.
- The company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms.

History of JavaScript

- Now, two languages were there: Java and the scripting language
- Netscape decided to give a similar name to the scripting language as Java's.
- It led to 'Javascript'.
- Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'.
- Later, the marketing team replaced the name with '**LiveScript**'.
- But, due to trademark reasons in December 1995, the language was finally renamed to 'JavaScript'.

Features of JavaScript

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.

Features of JavaScript

5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

Applications of JavaScript

- JavaScript is used to create interactive websites.
- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

JavaScript Example

```
<script>
document.write("Hello world
Application");
</script>
```

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

```
<script type="text/javascript">
document.write("JavaScript is a simple language");
</script>
```

JavaScript Example

- The **document.write()** function is used to display dynamic content through JavaScript.

JavaScript code

- Between the body tag of html
- Between the head tag of html
- In .js file (external javaScript)

Between the body tag of html

```
<script type="text/javascript">
 alert("Hello Javatpoint");
</script>
```

Code between the head tag

```
<html>
<head>
        <script type="text/javascript">
                function msg(){
                        alert(“GOOD AFTERNOON TO ALL");
                }
        </script>
</head>
<body>
        <p>Welcome to JavaScript</p>
        <form>
                <input type="button" value="click" onclick="msg()"/>
        </form>
</body>
</html>
```

External JavaScript file

- create external JavaScript file and embed it in many html page.
- It provides **code re usability** because single JavaScript file can be used in several html pages.
- An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

message.js

```
function msg(){
 alert("Hello Javatpoint");
}
```

index.html

```
<html>
    <head>
    <script type="text/javascript" src="message.js">
    </script>
</head>
```

```
<body>
    <p>Welcome to JavaScript</p>
    <form>
    <input type="button" value="click" onclick="msg()"
/>
    </form>
</body>
</html>
```

Advantages of External JavaScript

- It helps in the reusability of code in more than one HTML file.
- It allows easy code readability.
- It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
- It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
- The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

- The stealer may download the coder's code using the url of the js file.
- If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
- The web browser needs to make an additional http request to get the js code.
- A tiny to a large change in the js code may cause unexpected results in all its dependent files.

INTRODUCTION TO JAVASCRIPT

- JavaScript is a lightweight, interpreted programming language.
- It is designed for creating network-centric applications.
- It is complimentary to and integrated with Java.
- JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

WHY TO LEARN JAVASCRIPT ?

- JavaScript is the most popular programming language
- Front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.
- Javascript is everywhere, it comes installed on every modern web browser
- Do not need any special environment setup

WHY TO LEARN JAVASCRIPT ?

- For example Chrome, Mozilla Firefox , Safari and every browser you know as of today, supports JavaScript.

- JavaScript usage has now extended to mobile app development, desktop app development, and game development.

JAVASCRIPT FEATURES

- JavaScript is a case-sensitive language.

JAVASCRIPT PROGRAMMING

```
<html>
<body>
<script language = "javascript" type "text/javascript">
<!-- document.write("Hello World!") //--> </script>
</body>
</html>
```

JAVASCRIPT FRAMEWORKS

- Angular
- React
- jQuery
- Vue.js
- Ext.js
- Ember.js
- Meteor
- Mithril
- Node.js
- Polymer
- Aurelia
- Backbone.js

APPLICATIONS OF JAVASCRIPT

- Client side validation
- Manipulating HTML Pages
- User Notifications
- Back-end Data Loading
- Presentations
- Server Applications

WHAT IS JAVASCRIPT ?

- JavaScript is a dynamic computer programming language

- JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript

ADVANTAGES OF JAVASCRIPT

- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces

LIMITATIONS OF JAVASCRIPT

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

JAVASCRIPT DEVELOPMENT TOOLS

- **Microsoft FrontPage**
- **Macromedia Dreamweaver MX**
- **Macromedia HomeSite 5**

FIRST JAVASCRIPT CODE

```
<script language = "javascript ”   type = "text/javascript">
JavaScript code
</script>
```

Example :

```
<script language = "javascript“    type = "text/javascript">
<!–
var1 = 10
var2 = 20
//-->
</script>
```

JAVASCRIPT DEVELOPMENT TOOLS

```
<script language = "javascript" type = "text/javascript">
 <!--
 // This is a comment. It is similar to comments in C++
/*
This is a multi-line comment in JavaScript
It is very similar to comments in C Programming */
 //-->
</script>
```

JAVASCRIPT DATA TYPES

Numbers, eg. 123, 120.50 etc.

Strings of text e.g. "This text string" etc.

Boolean e.g. true or false.

- **JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value.**
- **JavaScript supports a composite data type known as object.**

JAVASCRIPT VARIABLES

```
<script type = "text/javascript">
<!–
var money;
var name;
//-->
</script>
```

JAVASCRIPT VARIABLES

```
<script type = "text/javascript">
<!–
var money, name;
//-->
</script>
```

JAVASCRIPT VARIABLE SCOPE

Global Variables − A global variable has global scope which means it can be defined anywhere in your JavaScript code.

Local Variables − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

JAVASCRIPT VARIABLES

```
<html>
 <body onload = scopeofvariable();>
<script type = "text/javascript">
<!-- var myVar = "global"; // Declare a global variable
function scopeofvariable( ) {
 var myVar = "local"; // Declare a local variable
document.write(myVar);
}
//-->
</script>
</body> </html>
```

JAVASCRIPT –NAMING VARIABLES

- Should not use any of the reserved keywords
- Should not start with a numeral (0-9)
- Must begin with a letter or an underscore character
- JavaScript variable names are case-sensitive

JAVASCRIPT - RESERVED WORDS

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

JAVASCRIPT - OPERATORS

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

JAVASCRIPT - OPERATORS

- ➢ Arithmetic Operators
- ➢ Comparison Operators
- ➢ Logical (or Relational) Operators
- ➢ Assignment Operators
- ➢ Conditional (or ternary) Operators

BASIC HTML

- **HTML** (HyperText Markup Language) is the language used to make web pages.
- **HyperText** refers to text that is con-nected by hyperlinks, the links on a web page.
- A **markup language** is used to annotate documents.

HTML documents in a text editor, a simple program designed for writing plaintext
files without the formatting

word processors like Microsoft Word. Word-processed documents contain formatted text (with different fonts, type colors, font sizes, etc.),

Word processors are designed to make it easy to change the formatting of the text. Word processors usually allow you to insert images and graphics as well.

TEXT EDITORS

- HTML in the cross-platform (compatible with Windows, Mac OS, and Linux) Sublime Text editor.
- To install Sublime Text, visit http://www.sublimetext.com/. Installation instructions differ for each operating system.
- Sublime Text will color-code your programs with syntax highlighting.
- This is designed to make programs easier for programmers to read by assigning different colors to different types of code.

Sublime Text has lots of color schemes to choose from. IDLE color scheme, which you can match on your screen by going to **preferences**

->**color Scheme** and selecting IDLE.

OUR FIRST HTML DOCUMENT

```
<h1>Hello world!</h1>
<p>My first web page.</p>
```

HEADING ELEMENTS

```
<h1>First-level heading</h1>
<h2>Second-level heading</h2>
<h3>Third-level heading</h3>
<h4>Fourth-level heading</h4>
<h5>Fifth-level heading</h5>
<h6>Sixth-level heading</h6>
```

THE P ELEMENT

The p element is used to define separate paragraphs of text

WHITESPACE IN HTML AND BLOCK-LEVEL ELEMENTS

- Whitespace means any character that results in blank space on the page
- for example, the space character, the tab character, and the newline character (the character that is inserted when you press enter or return).
- Any blank lines you insert between two pieces of text in an HTML document will get collapsed into a single space.

- The p and h1 elements are called block-level elements because they display their content in a separate block, starting on a new line, and with any following content on a new line.

INLINE ELEMENTS

```
<h1>Hello world!</h1>
<p>My <em>first</em> <strong>web page</strong>.</p>
<p>Let's add another
<strong><em>paragraph</em></strong>.</p>
```

The **em** element makes its content italic. The **strong** element makes its content bold. The **em and strong** elements are both inline elements, which means that they don't put their content onto a new line, as block-level elements do.

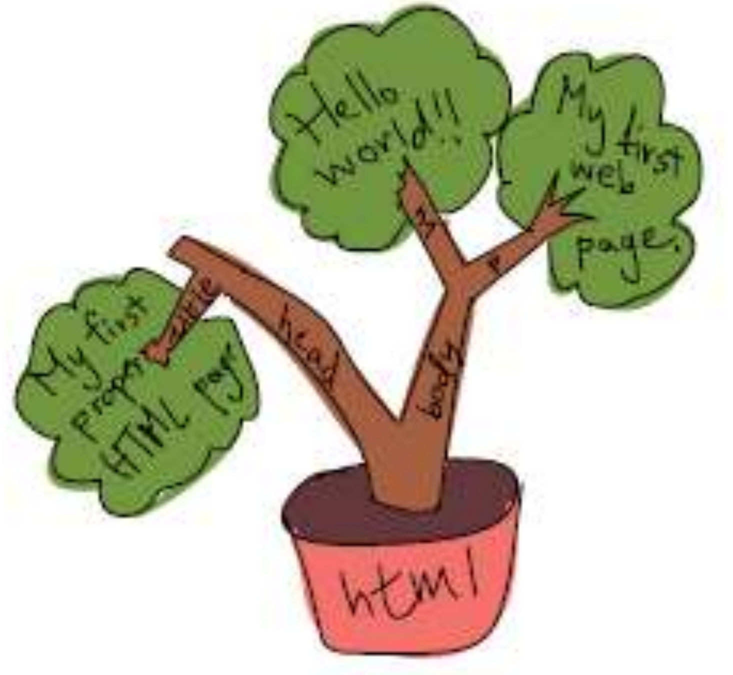
A FULL HTML DOCUMENT

A full HTML document requires some extra elements.

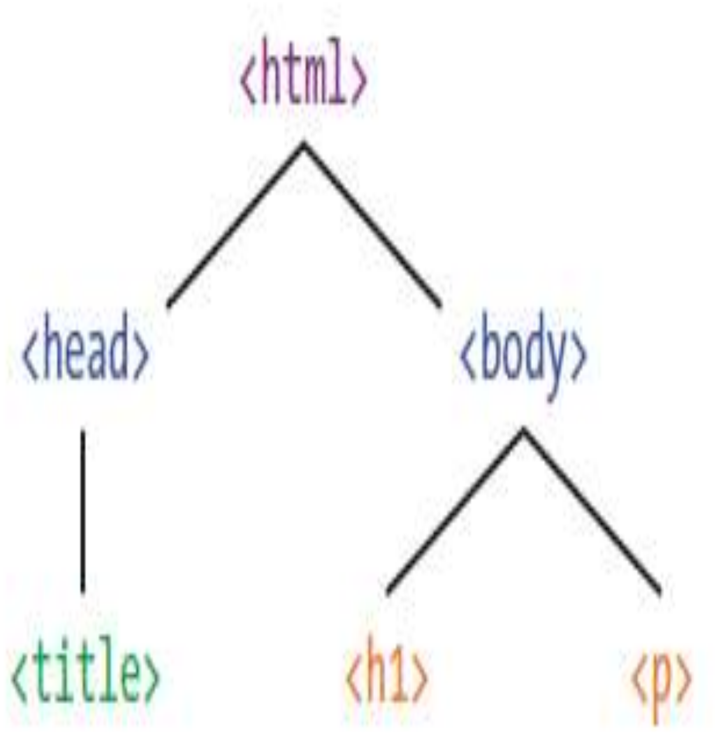
```
<!DOCTYPE html>
<html>
<head>
        <title>My first proper HTML page</title>
</head>

<body>
        <h1>Hello world!</h1>
        <p>My <em>first</em> <strong>web page</strong>.</p>
        <p>Let's add another
        <strong><em>paragraph</em></strong>.</p>
</body>
</html>
```

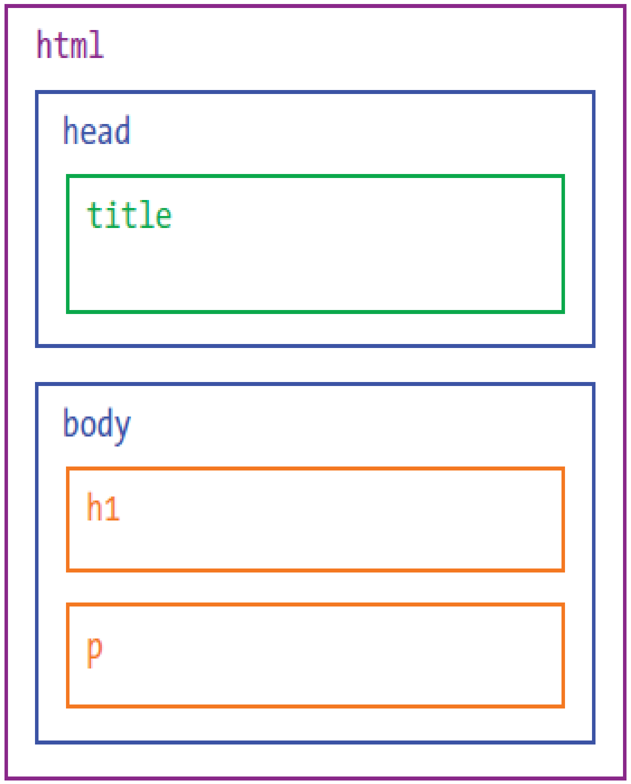
HTML HIERARCHY

HTML elements have a clear hierarchy, or order, and can be thought of as a kind of upside-down tree.



HTML HIERARCHY





ADDING LINKS TO YOUR HTML

```
<!DOCTYPE html>
<html>
<head>
<title>My first proper HTML page</title>
</head>
<body>
<h1>Hello world!</h1>
<p>My first web page.</p>
<p><a href="http://xkcd.com">Click here</a> to read some excellent
comics.</p>
</body>
</html>
```

LINK ATTRIBUTES

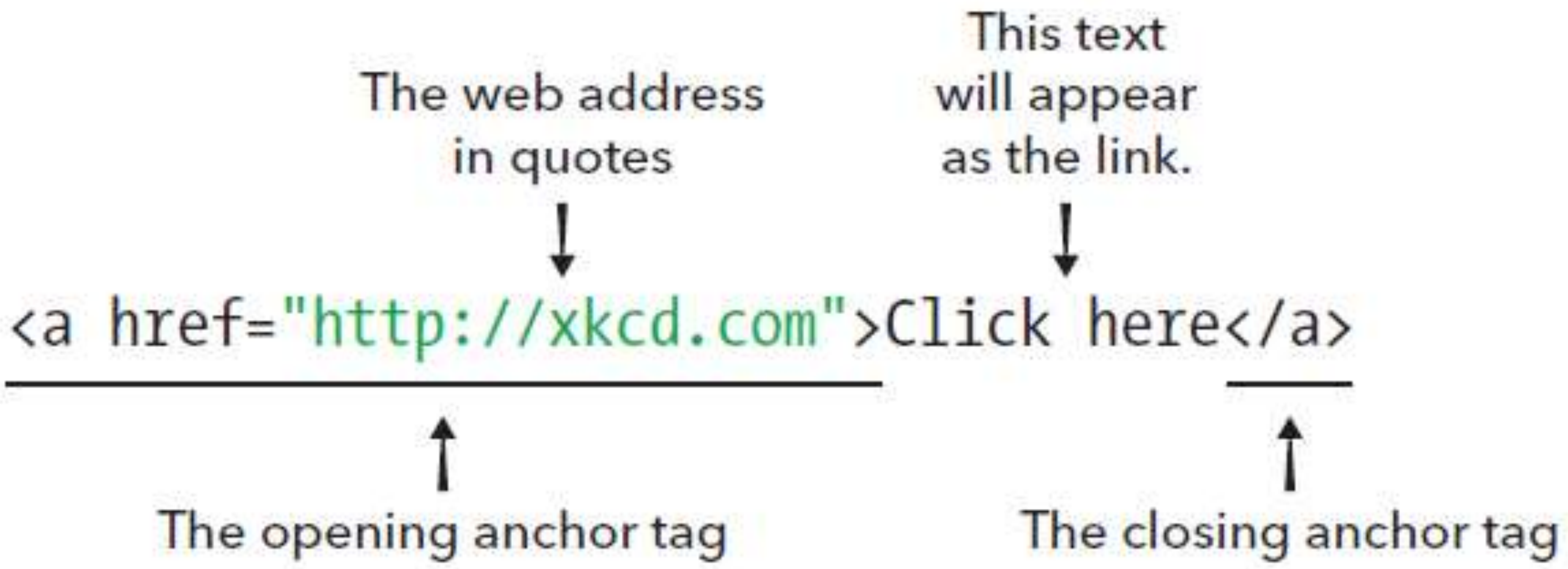
•HTML link-To tell the browser where to go when you click the a element

•Added something called an *attribute to the anchor element.*

•<a href="http://www.yahoo.com">Click here</a>

•Attributes in HTML elements are similar to key-value pairs in JavaScript objects. Every attribute has a name and a value.

The basic syntax for creating a hyperlink



```
<a href="http://xkcd.com">Click here</a>
```

TITLE ATTRIBUTES

```
<a href="http://www.google.com" title="google: Land of studying everything!"> Click here</a>
```

JAVASCRIPT VARIABLES

JAVASCRIPT –NAMING VARIABLES

JAVASCRIPT - RESERVED WORDS

JAVASCRIPT - OPERATORS

JAVASCRIPT - OPERATORS

UNIT II

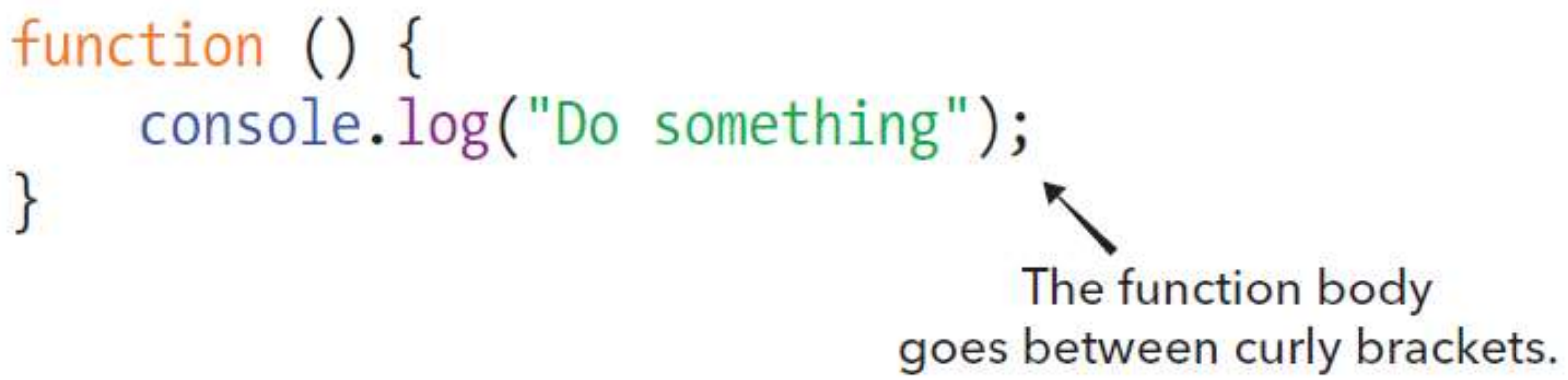
FUNCTIONS AND LOOPS

Functions

- A *function* is a way to bundle code so that it can be reused.
- Functions allow us to run the same piece of code from multiple places in a program without having to copy and paste the code repeatedly.
- Also, by hiding long bits of code in a function and giving it an easy-to-understand name.
- Splitting up your code into smaller, more manageable pieces allows you to see the bigger picture and think about how your programs are structured at a higher level.

The Basic Anatomy of a Function

- The code between the curly brackets is called the *function body*, just as the code between the curly brackets in a loop is called the *loop body*.



```
function () {
    console.log("Do something");
}
```

The syntax for creating a function

Creating a Simple Function

```
var ourFirstFunction = function () {
console.log("Hello world!");
};
```

This code creates a new function and saves it in the variable ourFirstFunction.

Calling a Function

```
ourFirstFunction();
Hello world!
undefined
```

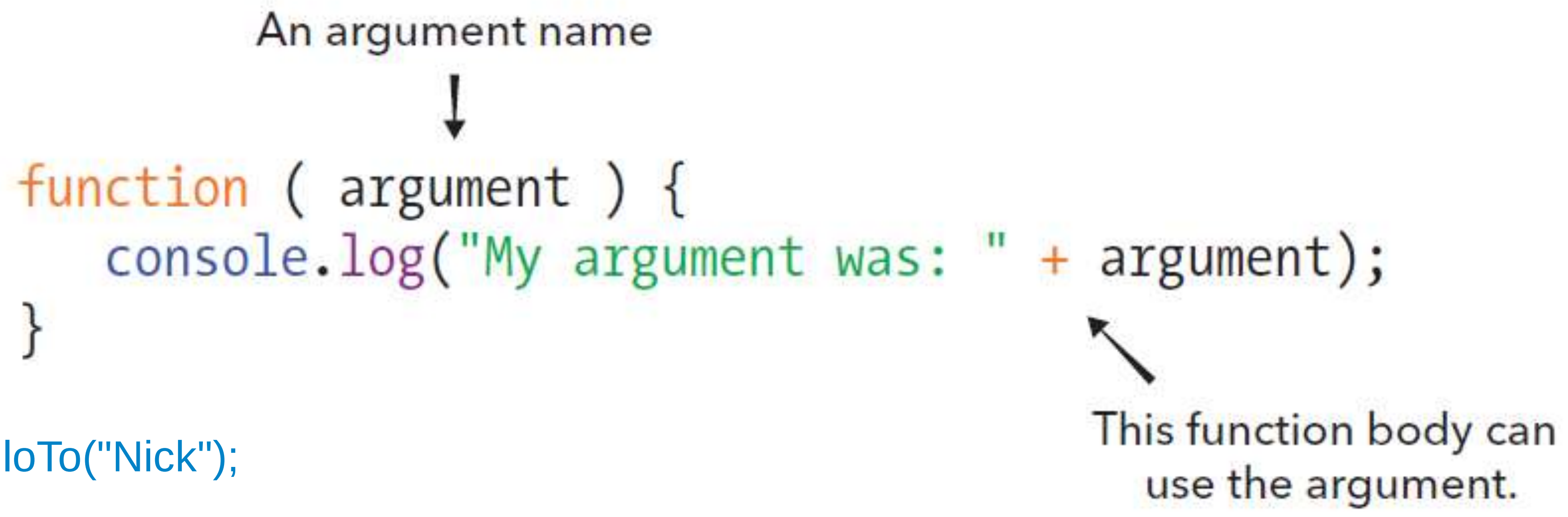
- *Calling a function with an undefined return value.*
- A *return value* is the value that a function outputs, which can then be used elsewhere in your code.
- A function always returns undefined unless there is something in the function body that tells it to return a different value.

Passing Arguments into Functions

Function *arguments* allow us to pass values into a function in order to change the function’s behavior when it’s called. Arguments always go between the function parentheses, both when you create the function and when you call it.

```
var sayHelloTo = function (name) {
console.log("Hello " + name + "!");
};
```

The syntax for creating a function with one argument



```
function ( argument ) {
   console.log("My argument was: " + argument);
}
```

sayHelloTo("Nick");

Hello Nick!

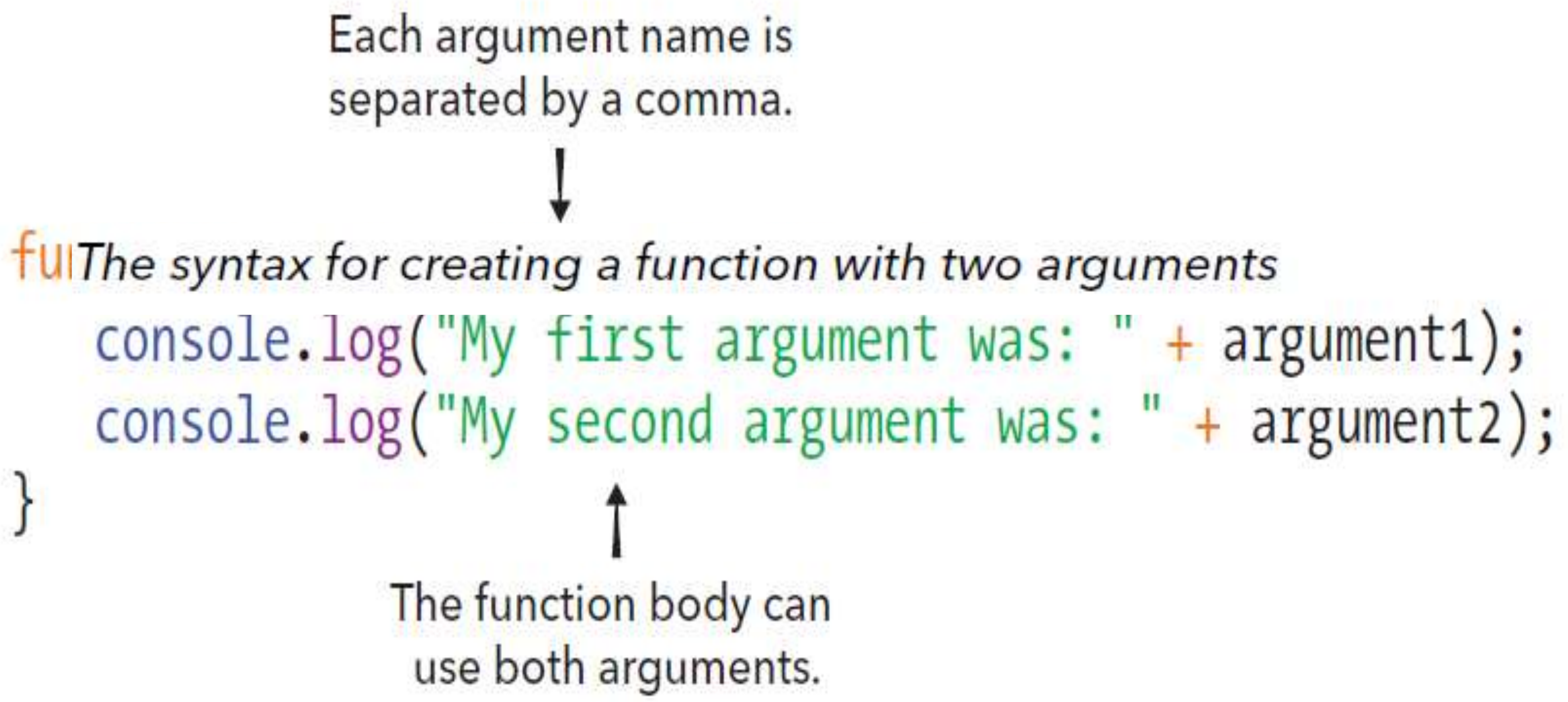
Printing Cat Faces!

```
var drawCats = function (howManyTimes) {
for (var i = 0; i < howManyTimes; i++) {
console.log(i + " =^.^=");
}
};
```

Output:

- drawCats(5);
- 0 =^.^=
- 1 =^.^=
- 2 =^.^=
- 3 =^.^=
- 4 =^.^=

Passing Multiple Arguments to a Function


Each argument name is separated by a comma.
The function body can use both arguments.

```
fuIThe syntax for creating a function with two arguments
    console.log("My first argument was: " + argument1);
    console.log("My second argument was: " + argument2);
}
```

The syntax for creating a function with two arguments

Passing Multiple Arguments to a Function

```
var printMultipleTimes = function (howManyTimes,
whatToDraw) {
for (var i = 0; i < howManyTimes; i++) {
console.log(i + " " + whatToDraw);
}
};
```

Returning Values from Functions

```
5 + Math.floor(1.2345);
6

var double = function (number) {
u return number * 2;
};

double(3);
```

Using Function Calls as Values

```
double(5) + double(6);
```

```
double(double(3));
```

Using Functions to Simplify Code

A Function to Pick a Random Word

```
randomWords[Math.floor(Math.random() *
randomWords.length)];
```

```
var pickRandomWord = function (words) {
return words[Math.floor(Math.random() * words.length)];
};

var randomWords = ["Planet", "Worm", "Flower",
"Computer"];

pickRandomWord(randomWords);
"Flower“
pickRandomWord(["Charlie", "Raj", "Nicole", "Kate",
"Sandy"]);
"Raj"
```

A Random Insult Generator

- var randomBodyParts = ["Face", "Nose", "Hair"];
- var randomAdjectives = ["Smelly", "Boring", "Stupid"];
- var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];

- // Pick a random body part from the randomBodyParts array:
- var randomBodyPart = randomBodyParts[Math.floor(Math.random() * 3)];
- // Pick a random adjective from the randomAdjectives array:
- var randomAdjective = randomAdjectives[Math.floor(Math.random() * 3)];
- // Pick a random word from the randomWords array:
- var randomWord = randomWords[Math.floor(Math.random() * 5)];
- // Join all the random strings into a sentence:
- var randomString = "Your " + randomBodyPart + " is like a " + randomAdjective + " " + randomWord + "!!!";
- randomString;
- "Your Nose is like a Stupid Marmot!!!"

```
var randomBodyParts = ["Face", "Nose", "Hair"];
var randomAdjectives = ["Smelly", "Boring", "Stupid"];
var randomWords = ["Fly", "Marmot", "Stick", "Monkey",
"Rat"];
// Join all the random strings into a sentence:
var randomString = "Your " +
pickRandomWord(randomBodyParts) +
" is like a " + pickRandomWord(randomAdjectives) +
" " + pickRandomWord(randomWords) + "!!!";
randomString;
"Your Nose is like a Smelly Marmot!!!"
```

```
generateRandomInsult = function () {
var randomBodyParts = ["Face", "Nose", "Hair"];
var randomAdjectives = ["Smelly", "Boring", "Stupid"];
var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];
// Join all the random strings into a sentence:
var randomString = "Your " + pickRandomWord(randomBodyParts) +
" is like a " + pickRandomWord(randomAdjectives) +
" " + pickRandomWord(randomWords) + "!!!";
u return randomString;
};
generateRandomInsult();
"Your Face is like a Smelly Stick!!!"
generateRandomInsult();
"Your Hair is like a Boring Stick!!!"
generateRandomInsult();
"Your Face is like a Stupid Fly!!!"
```

Leaving a Function Early with return

```
var fifthLetter = function (name) {
if (name.length < 5) {
return;
}
return "The fifth letter of your name is " + name[4] + ".";
};
```

fifthLetter("Nicholas");

"The fifth letter of your name is o.“

- fifthLetter("Nick");
- undefined

Using return Multiple Times Instead of if...else Statements

```
var medalForScore = function (score) {
if (score < 3) {
 return "Bronze";
}
if (score < 7) {
return "Silver";
}
return "Gold";
};
```

Shorthand for Creating Functions

```
var double = function (number) {
return number * 2;
};

function double(number) {
return number * 2;
}
```

- the longhand version is known as a function *expression.*
- *The shorthand version is known as a* function *declaration.*

Programming Challenges

- #1: Doing Arithmetic with Functions

Create two functions, add and multiply. Each should take two arguments. The add function should sum its arguments and return the result, and multiply should multiply its Arguments. Using only these two functions, solve this simple mathematical problem:

36325 * 9824 + 777

#2: Are These Arrays the Same?

Write a function called areArraysSame that takes two arrays of numbers as arguments. It should return true if the two arrays are the same (that is, they have the same numbers in the same order) and false if they're different. Try running the following code to make sure your functions are working correctly:

- areArraysSame([1, 2, 3], [4, 5, 6]);
- false

- areArraysSame([1, 2, 3], [1, 2, 3]);
- true
- areArraysSame([1, 2, 3], [1, 2, 3, 4]);
- false

Hint 1: you'll need to use a for loop to go through each of the values in the first array to see whether they're the same in the second array. You can return false in the for loop if you find a value that's not equal.

Hint 2: you can leave the function early and skip the for loop altogether if the arrays are different lengths.

#3: Hangman, Using Functions

- Go back to your Hangman game from Chapter 7. We're going to rewrite it using functions. I've rewritten the final Hangman code here, but with certain parts of the code replaced by function calls. All you need to do is write the functions!

CONDITIONALS and LOOPS

- Conditionals and loops are two of the most important concepts in JavaScript.
- A conditional says, “If some – thing is true, do this. Otherwise, do that.”
- A loop says, “As long as something is true, keep doing this.”

- **Conditionals and loops** are powerful concepts that are key to any sophisticated program.
- They are called **control structures** because they allow you to control which parts of your code are executed when and how often they’re executed, based on certain conditions you define

Embedding JavaScript in HTML

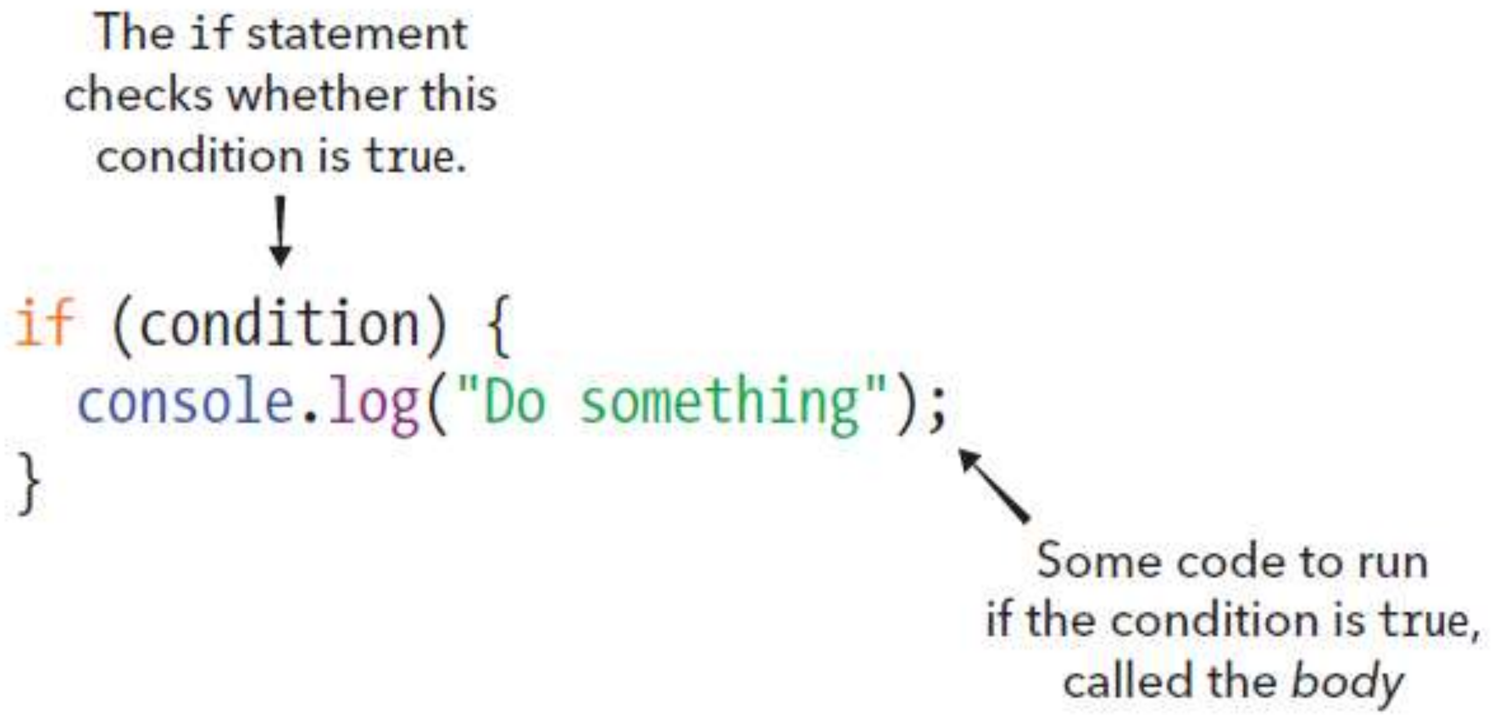
```
<!DOCTYPE html>
  <html>
  <head>
  <title>My first proper HTML page</title>
  </head>
  <body>
  <h1>Hello world!</h1>
  <p>My first web page.</p>
  <script>
  var message = "Hello world!";
  console.log(message);
  </script>
  </body>
  </html>
```

Conditionals

- If statements
- if...else statements.

```
  var name = "Nicholas";
v console.log("Hello " + name);
w if (name.length > 7) {
x console.log("Wow, you have a REALLY long
name!");
```

The general structure of an if statement



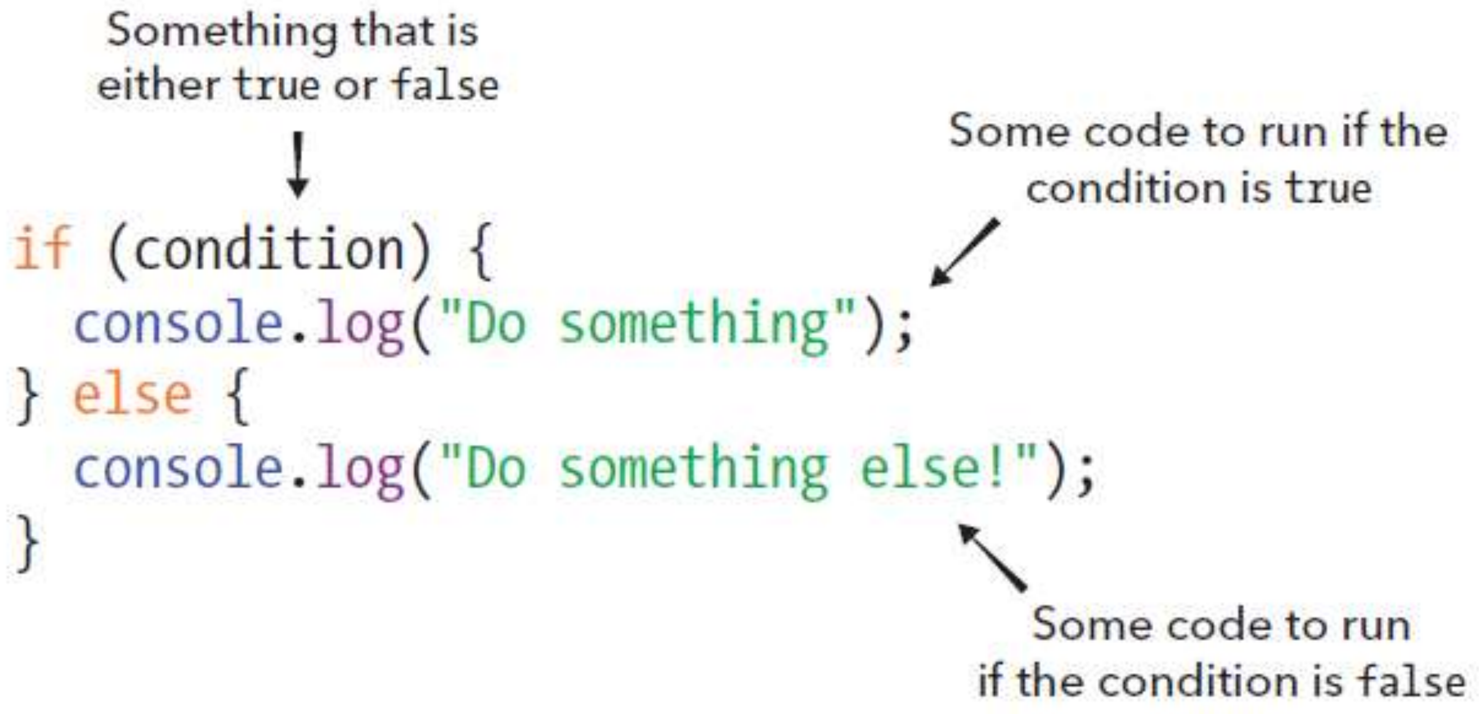
```
if (condition) {
  console.log("Do something");
}
```

The if statement checks whether this condition is true.
Some code to run if the condition is true, called the *body*

if…else Statements

```
var name = "Nicholas";
console.log("Hello " + name);
if (name.length > 7) {
console.log("Wow, you have a REALLY long
  name!");
} else {
console.log("Your name isn't very long.");
}
```

The general structure of an if...else statement

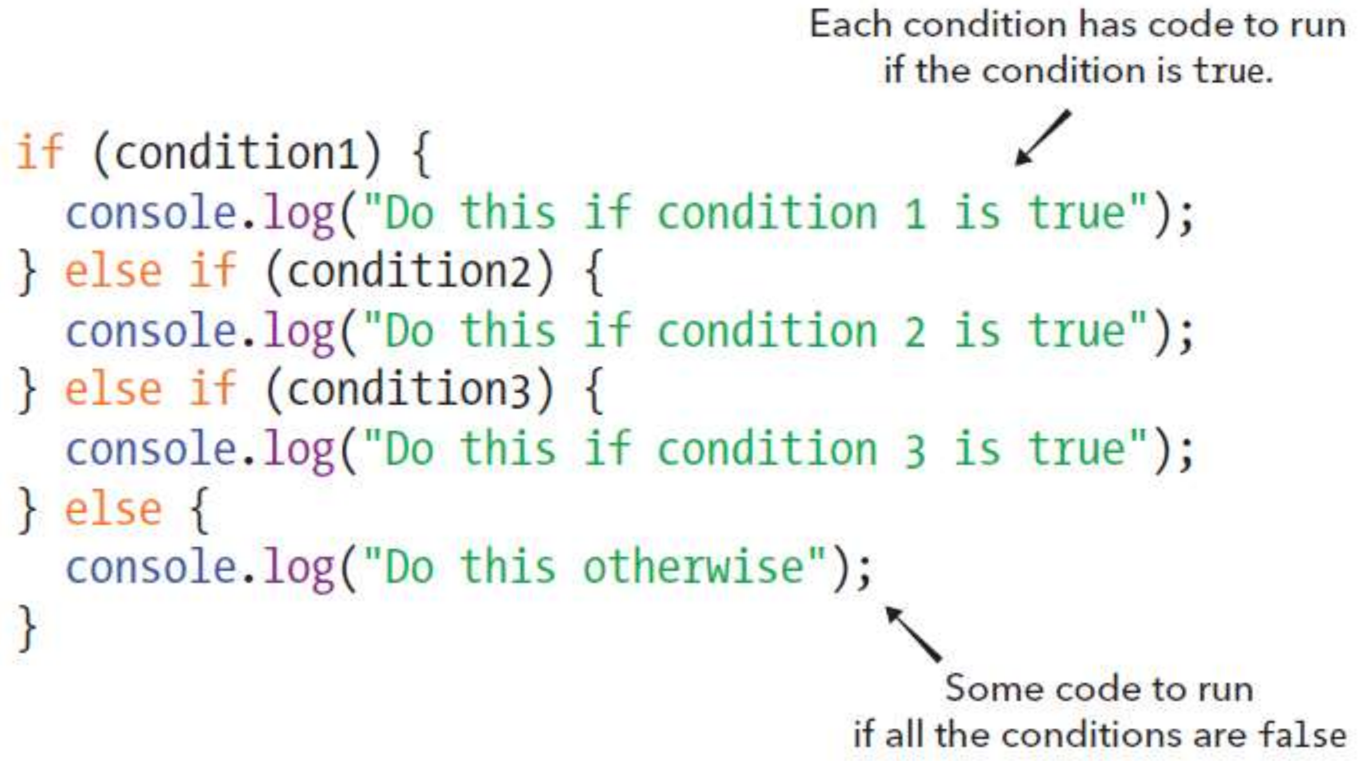

Something that is either true or false
Some code to run if the condition is true
Some code to run if the condition is false

```
if (condition) {
  console.log("Do something");
} else {
  console.log("Do something else!");
}
```

Chaining if...else Statements

```
var lemonChicken = false;
var beefWithBlackBean = true;
var sweetAndSourPork = true;
if (lemonChicken) {
console.log("Great! I'm having lemon chicken!");
} else if (beefWithBlackBean) {
console.log("I'm having the beef.");
} else if (sweetAndSourPork) {
console.log("OK, I'll have the pork.");
} else {
console.log("Well, I guess I'll have rice then.");
}
```

Chaining multiple if...else statements



```
if (condition1) {
  console.log("Do this if condition 1 is true");
} else if (condition2) {
  console.log("Do this if condition 2 is true");
} else if (condition3) {
  console.log("Do this if condition 3 is true");
} else {
  console.log("Do this otherwise");
}
```

Each condition has code to run if the condition is true.
Some code to run if all the conditions are false

```
var lemonChicken = false;
var beefWithBlackBean = false;
var sweetAndSourPork = false;
if (lemonChicken) {
console.log("Great! I'm having lemon chicken!");
} else if (beefWithBlackBean) {
console.log("I'm having the beef.");
} else if (sweetAndSourPork) {
console.log("OK, I'll have the pork.");
}
```

Loops

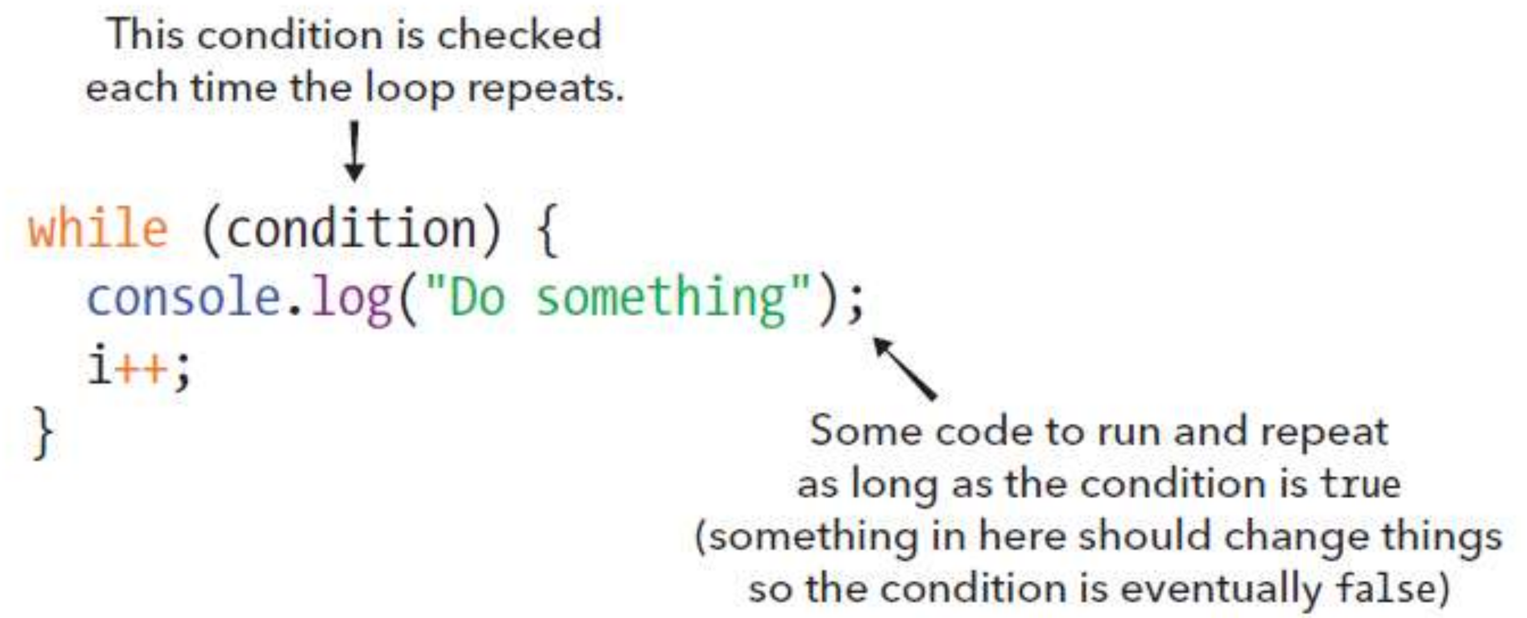
- while Loops

  A while loop repeatedly executes its body until a particular condition stops being true.

- for Loops

  for loops make it easier to write loops that create a variable, loop until a condition is true, and update the variable at the end of each turn around the loop.

The general structure of a while loop



This condition is checked
each time the loop repeats.

```
while (condition) {
  console.log("Do something");
  i++;
}
```

Some code to run and repeat
as long as the condition is true
(something in here should change things
so the condition is eventually false)

Counting Sheep with a while Loop

```
var sheepCounted = 0;
u while (sheepCounted < 10) {
v console.log("I have counted " +
  sheepCounted + " sheep!");
sheepCounted++;
}
console.log("Zzzzzzzzzzz");
```

Preventing Infinite Loops

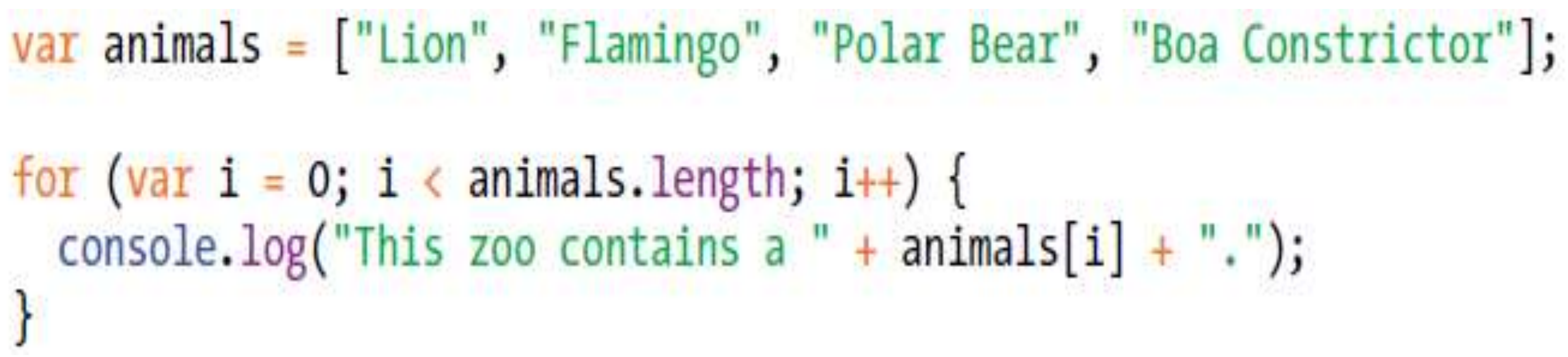
- if the condition set never becomes false, your loop will loop forever (or at least until you quit your browser).
- The program would keep doing this forever! This is called an *infinite loop.*

for Loops

```
for (var sheepCounted = 0; sheepCounted <
  10; sheepCounted++) {
console.log("I have counted " + sheepCounted
  + " sheep!");
}
console.log("Zzzzzzzzzzz");
```

Using for Loops with Arrays and Strings

Use of for loops is to do something with every element in an array or every character in a string.



```
var animals = ["Lion", "Flamingo", "Polar Bear", "Boa Constrictor"];

for (var i = 0; i < animals.length; i++) {
  console.log("This zoo contains a " + animals[i] + ".");
}
```

Access individual characters

```
var name = "Nick";
for (var i = 0; i < name.length; i++) {
console.log("My name contains the letter " +
  name[i] + ".");
}
```

Other Ways to Use for Loops

```
for (var x = 2; x < 10000; x = x * 2) {
console.log(x);
}
```

- Write a loop to print the powers of 3 under 10,000 (it should print 3, 9, 27, etc.).
- Rewrite this loop with a while loop. (Hint: Provide the setup *before the loop.)*

Programming Challenges

- Programming Challenges
- var animals = ["Cat", "Fish", "Lemur", "Komodo Dragon"];
- Output:
- ["Awesome Cat", "Awesome Fish", "Awesome Lemur", "Awesome Komodo Dragon"]
- animals[0] = "Awesome " + animals[0];

#2: Random String Generator

var alphabet = "abcdefghijklmnopqrstuvwxyz";

- var input = "javascript is awesome";
- var output = "";

#3: h4ck3r sp34k

ALGORITHM

- Replace certain letters with numbers
- 4 for A, 3 for E, 1 for I, and 0 for O
- to use a for loop to go through all the
- letters of the input string.
- If the letter is "a", add a "4" to the output string.
- If it’s "e", add a "3".
- If it’s "i", add a "1",
- if it’s "o", add a "0"

- Otherwise, just add the original letter
- to the new string.
- As before, you can use += to add each new
letter to the output string.
- After the loop, log the output string to the console
- If it works correctly, you should see it log "j4v4scr1pt 1s 4w3s0m3".

Creating a Hangman Game

- How To Use Dialogs To Make The Game Interactive And Take Input From Someone Playing The Game.
- Hangman is a word-guessing game.
- One player picks a secret word, and the other player tries to guess it.

UNIT III

ADVANCED JAVASCRIPT

jQuery

- jQuery is a lightweight, "write less, do more", JavaScript library.
- The purpose of jQuery is to make it much easier to use JavaScript on your website.
- jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.
- jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.
- The jQuery library contains the following features:

  HTML/DOM manipulation
  CSS manipulation
  HTML event methods
  Effects and animations
  AJAX
- Utilities

jQuery Syntax

- The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).
- Basic syntax is: **$(*selector*).*action*()**
- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)
- Examples:
- $(this).hide() - hides the current element.
- $("p").hide() - hides all <p> elements.
- $(".test").hide() - hides all elements with class="test".
- $("#test").hide() - hides the element with id="test".

jQuery Selectors

- jQuery selectors allow you to select and manipulate HTML element(s).
- jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.
- All selectors in jQuery start with the dollar sign and parentheses: $().

The element Selector

- The jQuery element selector selects elements based on the element name.
- You can select all <p> elements on a page like this:
- $("p")

When a user clicks on a button, all <p> elements will be hidden:

-
  ```
  $(document).ready(function(){
    $("button").click(function(){
      $("p").hide();
    });
  });
  ```

The #id Selector

- The jQuery *#id* selector uses the id attribute of an HTML tag to find the specific element.
- An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.
- To find an element with a specific id, write a hash character, followed by the id of the HTML element:
- $("#test")

When a user clicks on a button, the element with id="test" will be hidden:

-
  ```
  $(document).ready(function(){
    $("button").click(function(){
      $("#test").hide();
    });
  });
  ```

jQuery

How to execute jQuery Code ?

- Download the jQuery library from the official website.
- Use online the jQuery CDN links.

Unit-3 The DOM and jQuery Chapter 9

1. Selecting DOM Elements
2. Using id to Identify Elements
3. Selecting an Element Using getElementById
4. Replacing the Heading Text Using the
5. Using jQuery to Work with the DOM Tree
6. DOM
7. Loading jQuery on Your HTML Page
8. Replacing the Heading Text Using jQuery
9. Creating New Elements with jQuery
10. Animating Elements with jQuery
11. Chaining jQuery Animations

JavaScript

- Print text to the browser console or display an alert or prompt dialog
- JavaScript to manipulate (control or modify) and interact with the HTML you write in web pages

Tools Required…

- To write much more powerful JavaScript:

The DOM and jQuery.

The DOM and jQuery

- The *DOM, or document object model, is what allows JavaScript to* access the content of a web page.
- Web browsers use the DOM to keep track of the elements on a page (such as paragraphs, headings, and other HTML elements), and JavaScript can manipulate DOM elements in various ways.
- Use JavaScript to replace the main heading of the HTML document with input from a prompt dialog.

jQuery

jQuery- which makes it much easier to work with the DOM.

jQuery gives us a set of functions that we can use to choose which elements to work with and to make changes to those elements.

Selecting DOM Elements

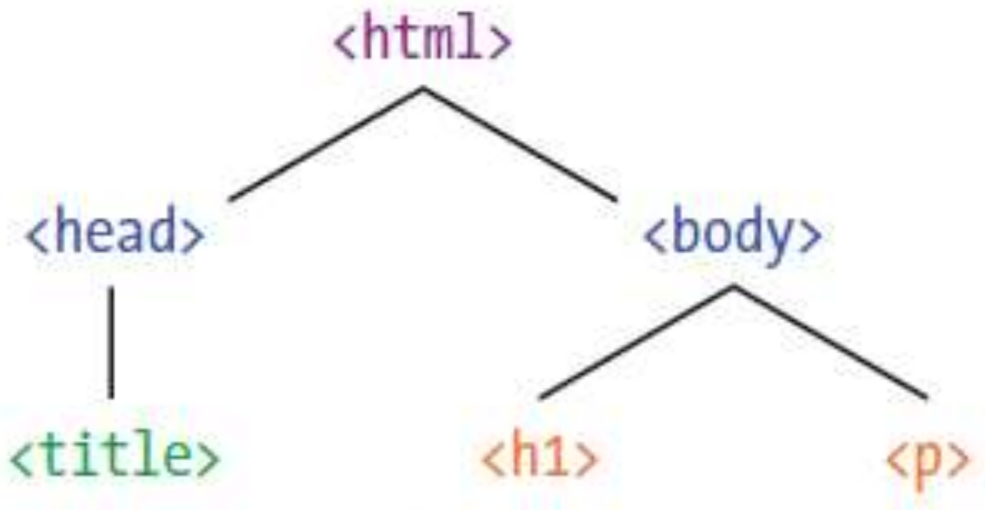


Figure 9-1: The DOM tree for a simple HTML

Using id to Identify Elements

```
<h1 id="main-heading">Hello world!</h1>
```

Selecting an Element Using getElementById

```
var headingElement =
  document.getElementById("main-heading");
```

Selecting an Element Using getElementById

Use the innerHTML property to retrieve and replace the text inside the selected element:

```
headingElement.innerHTML;
```

- This code returns the HTML contents of headingElement
- The element selected using getElementById

Replacing the Heading Text Using the DOM

```
<!DOCTYPE html>
<html>   <head>
<title>Playing with the DOM</title>
</head>
<body>
<h1 id="main-heading">Hello world!</h1>
<script>
var headingElement = document.getElementById("main-heading");
console.log(headingElement.innerHTML);
 var newHeadingText = prompt("Please provide a new heading:");
headingElement.innerHTML = newHeadingText;
</script></body></html>
```

Using jQuery to Work with the DOM Tree

- The built-in DOM methods are great, but they're not very easy to use.
- Because of this, many developers use a set of tools called jQuery to access and manipulate the DOM tree.
- jQuery is a JavaScript *library—a collection of related tools (mostly functions)* that gives us, in this case, a simpler way to work with DOM elements.
- Once we load a library onto our page, we can use its
- functions and methods in addition to those built into JavaScript and those provided by the browser.

Loading jQuery on Your HTML Page

```
<script src="https://code.jquery.com/jquery-
   2.1.0.js">
</script>
```

Replacing the Heading Text Using jQuery

```
<!DOCTYPE html>
<html>
<head>
<title>Playing with the DOM</title>
</head>
<body>
<h1 id="main-heading">Hello world!</h1>
  <script src="https://code.jquery.com/jquery-2.1.0.js"></script>
<script>
var newHeadingText = prompt("Please provide a new heading:");
$("#main-heading").text(newHeadingText);
</script>
</body>
</html>
```

Creating New Elements with jQuery

```
$("body").append("<p>This is a new
   paragraph</p>");
```

Append to add multiple elements in a for loop

```
for (var i = 0; i < 3; i++) {
var hobby = prompt("Tell me one of your
   hobbies!");
$("body").append("<p>" + hobby + "</p>");
}
```

Animating Elements with jQuery

```
$("h1").fadeOut(3000);
```

Chaining jQuery Animations

```
$("h1").text("This will fade out").fadeOut(3000);
```

- Calling multiple methods in a row like this is known as *chaining*.

```
$("h1").fadeOut(3000).fadeIn(2000);

$("h1").slideUp(1000).slideDown(1000);
```

Animation Methods

- jQuery provides two additional animation methods similar to fadeOut and fadeIn, called slideUp and slideDown.
- The slideUp method makes elements disappear by sliding them up, and slideDown makes them reappear by sliding them down.

1. We use fadeIn to make invisible elements visible. But what happens if you call fadeIn on an element that's already visible or an element that comes *after the element you're* animating?
2. For example, say you add a new p element to your *dom.html document after the heading. Try using slideUp* and slideDown to hide and show the h1 element, and see what happens to the p element. What if you use fadeOut and fadeIn?
3. What happens if you call fadeOut and fadeIn on the same element without chaining the calls?

For example:

```
$("h1").fadeOut(1000);
$("h1").fadeIn(1000);
```

Try adding the preceding code inside a for loop set to
run five times. What happens?

What do you think the show and hide jQuery methods do?
Try them out to see if you're right. How could you use hide to
fade in an element that's already visible?

Programming Challenges

1. #1: Listing Your Friends with jQuery
   (And Making Them Smell!)
1. #2: Making a Heading Flash
2. #3: Delaying Animations
3. #4: Using fadeTo

#1: Listing Your Friends with jQuery (And Making Them Smell!)

1. Create an array containing the names of a few friends.
2. Using a for loop, create a p element for each of your friends and add it to the end of the body element using the jQuery append method.
3. Use jQuery to change the h1 element so it says My friends instead of Hello world!.
4. Use the hide method followed by the fadeIn method to fade in each name as it's provided.
5. Now, modify the p elements you created to add the text smells! after each friend.
6. Hint: If you select the p elements using $("p"), the append method will apply to all the p elements.

#2: Making a Heading Flash

How could you use fadeOut and fadeIn to cause the heading to flash five times, once a second? How could you do this using a for loop? Try modifying your loop so it fades out and fades in over 1 second the first time, over 2 seconds the second time, over 3 seconds the third time, and so on.

#3: Delaying Animations

The delay method can be used to delay animations. Using delay, fadeOut, and fadeIn, make an element on your page fade out and then fade back in again after 5 seconds

#4: Using fadeTo

- Try using the fadeTo method. Its first argument is a number of milliseconds, as in all the other animation methods. Its second argument is a number between 0 and 1. What happens when you run the following code?

fadeTo

```
$("h1").fadeTo(2000, 0.5);
```

- What do you think the second argument means?
- Try using different values between 0 and 1 to figure out what the second argument is used for.

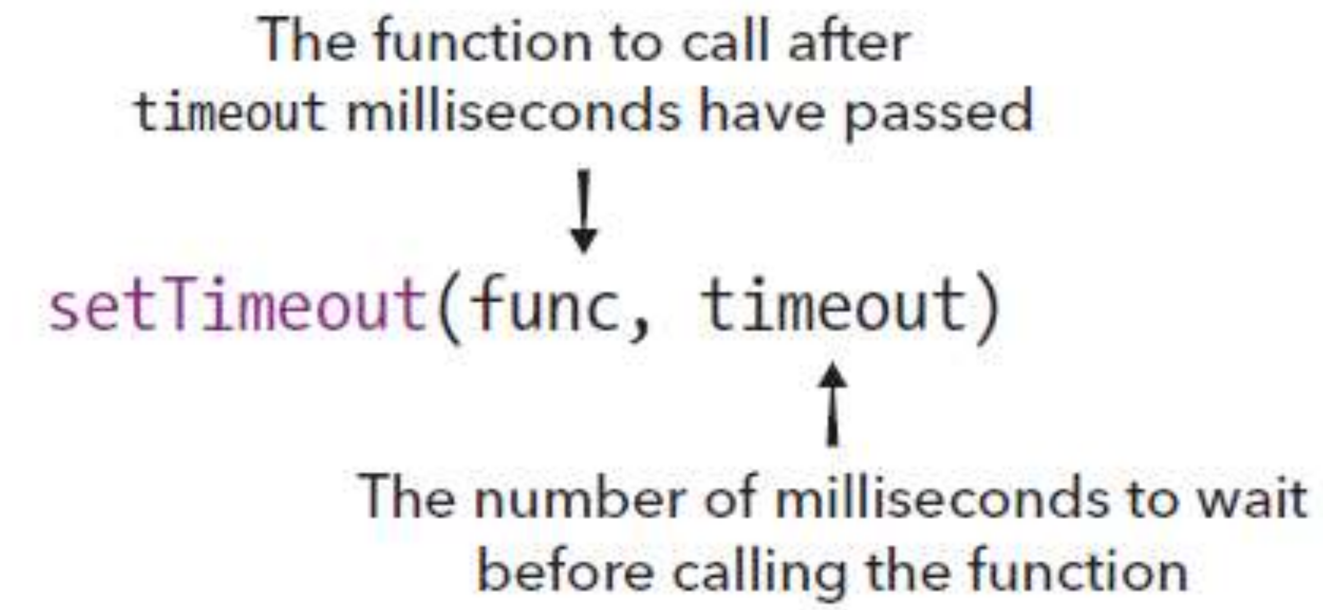
Chapter 10 - Interactive Programming

- Create interactive web pages that change over time and respond to actions by the user.
- Programming in this way is called *interactive programming*.

Delaying Code with setTimeout

- Function-to execute a function after a certain period of time.
- To set a timeout in JavaScript, we use the function setTimeout.
- This function takes two arguments : the function to call after the time has elapsed and the amount of time to wait (in milliseconds).

The arguments for setTimeout()



The function to call after timeout milliseconds have passed

```
setTimeout(func, timeout)
```

The number of milliseconds to wait before calling the function

SETTIMEOUT TO DISPLAY AN ALERT DIALOG

```
var timeUp = function () {
alert("Time's up!");
};
setTimeout(timeUp, 3000);
```

SETTIMEOUT()

- create the function timeUp,
  - which opens an alert dialog that displays the text "Time's up!".
- Call setTimeout with two arguments: the function we want to call (timeUp) and the number of milliseconds (3000) to wait before calling that function.
- “Wait 3 seconds and then call timeUp.” When
- setTimeout(timeUp, 3000) is first called, nothing happens, but after 3 seconds timeUp is called and the alert dialog pops up.

SETTIMEOUT()

- setTimeout returns 1.
- This return value is called the *timeout ID. The timeout ID is a number that's used to* identify this particular timeout (that is, this particular delayed function call).
- The actual number returned could be any number, since it's just an identifier. Call setTimeout again, and it should return a different timeout ID.

SETTIMEOUT();

- setTimeout(timeUp, 5000);
- use this timeout ID with the clearTimeout function to cancel that specific timeout.

Canceling a Timeout

- To cancel a timeout, use the function clearTimeout on the timeout ID returned by setTimeout.

```
var doHomeworkAlarm = function () {
alert("Hey! You need to do your
  homework!");
};

var timeoutId =
  setTimeout(doHomeworkAlarm, 60000);
```

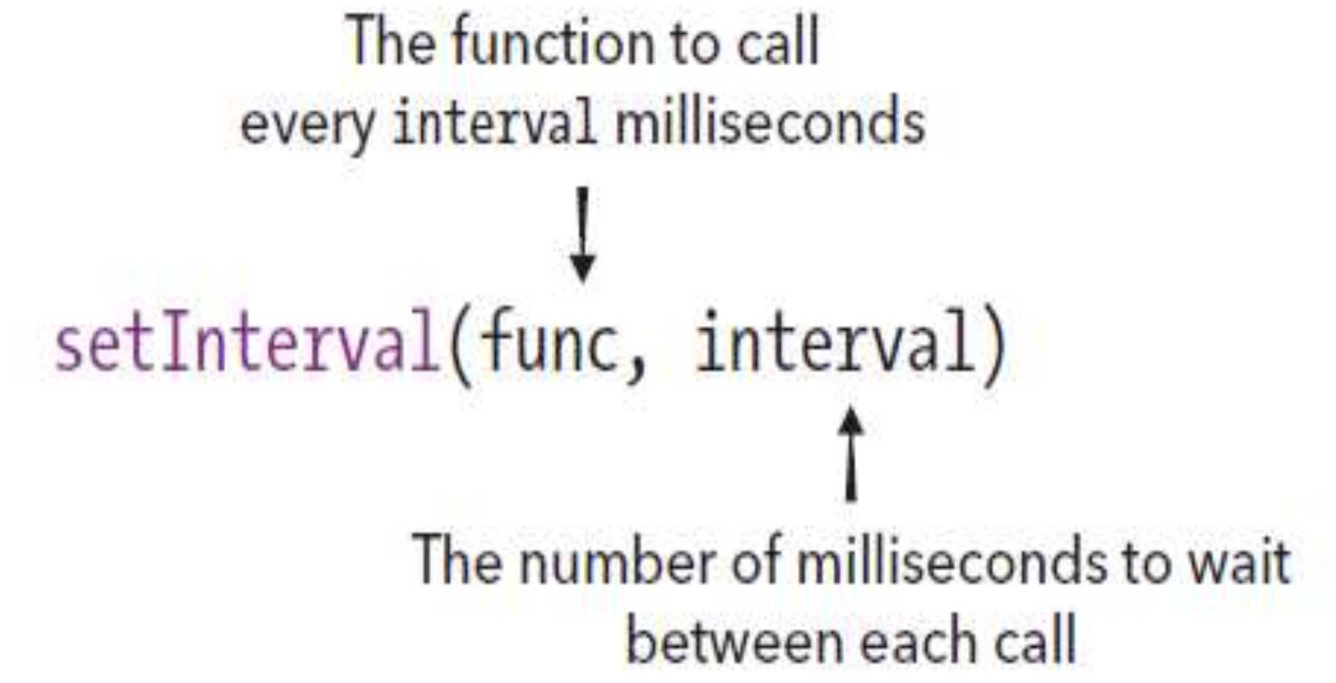
setTimeout() with two arguments

- When we call setTimeout(doHomeworkAlarm, 60000)
- JavaScript to execute that function after 60,000 milliseconds (or 60 seconds) has passed.
- Make this call to setTimeout and save the timeout ID in a new variable called timeoutId.
- To cancel the timeout, pass the timeout ID to the clearTimeout function like this:
- clearTimeout(timeoutId);

Calling Code Multiple Times with setInterval

- The setInterval function is like setTimeout, except that it repeatedly calls the supplied function after regular pauses, or *intervals*.
- To update a clock display using JavaScript, you could use setInterval to call an update function every second.
- call setInterval with two arguments:
  - the function you want to call and the length of the interval (in milliseconds).

The arguments for setInterval()


The function to call
every interval milliseconds
setInterval(func, interval)
The number of milliseconds to wait
between each call

WRITE A MESSAGE TO THE CONSOLE EVERY SECOND

```
var counter = 1;
 var printMessage = function () {
console.log("You have been staring at your console
   for " + counter + " seconds");
counter++;
};
 var intervalId = setInterval(printMessage, 1000);

clearInterval(intervalId);
```

Try It Out!

- Modify the preceding example to print the message every five seconds instead of every second.

SETINTERVAL()

- we create a new variable called counter and set it to 1.
- create a function called printMessage.
- This function does two things.
- First, it prints out a message telling you how long you have been staring at your console. Then, at it increments the counter variable.

SETINTERVAL()

- call setInterval(), passing the printMessage function and the number 1000.
- Calling setInterval like this means “call printMessage every 1,000 milliseconds.”
- setTimeout returns a timeout ID,
- setInterval returns an *interval ID, which is to save in the variable* intervalId.
- use this interval ID to tell JavaScript to stop executing the printMessage function using the clearInterval function.

Try It Out!

- Modify the preceding example to print the message every five seconds instead of every second

Animating Elements with setInterval

```
!DOCTYPE html>
<html>
<head>
<title>Interactive programming</title>
</head>
<body>
<h1 id="heading">Hello world!</h1>
<script src="https://code.jquery.com/jquery-
   2.1.0.js"></script>
<script>
// We'll fill this in next
</script>
</body>
</html>
```

Put your code inside the <script> tags of the HTML document

```
var leftOffset = 0;
var moveHeading = function () {
                $("#heading").offset({ left: leftOffset });
                leftOffset++;
if (leftOffset > 200) {
                  leftOffset = 0;
}
};

setInterval(moveHeading, 30);
```

Animating Elements with setInterval

- heading element gradually move across the screen until it travels 200 pixels, at that point, it will jump back to the beginning and start again.
- we create the variable leftOffset, which we'll use later to position our Hello world! heading. It starts with a value of 0, which means the heading will start on the far left side of the page.
- create the function moveHeading, which we'll
- call later with setInterval. Inside the moveHeading function, at, weuse $("#heading") to select the element with the id of "heading" (our h1 element) and use the offset method to set the left offset of the heading—that is, how far it is from the left side of the screen.

Responding to User Actions

- fixed amount of time- setTimeout(),setTimeInterval()
- when a user performs certain actions - clicking, typing, or even just moving the mouse.
- perform an action such as clicking, typing, or moving your mouse, something called an *event is* triggered
- events by adding an *event handler* to the element where the event happened

Responding to Clicks

- When a user clicks an element in the browser, this triggers a *click event.*
- *jQuery makes it easy to add a handler for a click* Event.

```
var clickHandler = function (event) {
console.log("Click! " + event.pageX + " " +
   event.pageY);
};

 $("h1").click(clickHandler);
```

The mousemove Event

```
<!DOCTYPE html>
<html>
<head>
<title>Mousemove</title>
</head>
<body>
<h1 id="heading">Hello world!</h1>
<script src="https://code.jquery.com/jquery-2.1.0.js"></script>
<script>
    $("html").mousemove(function (event) {
    $("#heading").offset({
    left: event.pageX,
    top: event.pageY
});
});
</script>
</body>
</html>
```

Programming Challenges

- #1: Follow the Clicks

  Modify the previous mousemove program so that instead of following your mouse, the heading will follow just your clicks. Whenever you click the page, the heading should move to the click location.

#2: Create Your Own Animation

- Use setInterval to animate an h1 heading element around the page, in a square.
- It should move 200 pixels to the right, 200 pixels down, 200 pixels to the left, 200 pixels up, and then start again.
- Hint: You'll need to keep track of your current direction (right, down, left, or up) so that you know whether to increase or decrease the left or top offset of the heading. You'll also need to change the direction when you reach a corner of the square.

#3: Cancel an Animation with a Click

- Building upon Challenge #2, add a click handler to the moving h1 element that cancels the animation.
- Hint: You can cancel intervals with the clearInterval function.

#4: Make a "Click the Header" Game!

Modify Challenge #3 so that every time a player clicks the heading, instead of stopping, the heading speeds up, making it harder and harder to click. Keep track of the number of times the heading has been clicked and update the heading text so it shows this number. When the player has reached 10 clicks, stop the animation and change the text of the heading to "You Win."

Hint:

To speed up, you'll have to cancel the current interval and then start a new one with a shorter interval time.

Chapter 11

Find the Buried Treasure

The buried treasure game

- Designing the Game

  - list of steps we need to take to set up the game so it can respond accordingly when a player clicks the treasure map.

Steps

- 1. Create a web page with an image (the treasure map) and a place to display messages to the player.
- 2. Pick a random spot on the map picture to hide the treasure.

- 3. Create a click handler. Each time the player clicks the map, the click handler will do the following:

  a. Add 1 to a click counter.

  b. Calculate how far the click location is from the treasure location.

  c. Display a message on the web page to tell the player whether they're hot or cold.

  d. Congratulate the player if they click on the treasure or very close to it, and say how many clicks it took to find the treasure.

Creating the Web Page with HTML

- use a new element called img for the treasure map and add a p element where we can display messages to the player. Enter the following code into a new file called *treasure.html.*

```
<!DOCTYPE html>
<html>
<head>
<title>Find the buried treasure!</title>
</head>
<body>
<h1 id="heading">Find the buried treasure!</h1>
 <img id="map" width=400 height=400
    src="http://nostarch.com/images/treasuremap.png">
 <p id="distance"></p>
<script src="https://code.jquery.com/jquery-2.1.0.js"></script>
<script>
// Game code goes here
</script>
</body>
</html>
```

Picking a Random Treasure Location

- Picking Random Numbers

```
var getRandomNumber = function (size)
{
return Math.floor(Math.random() * size);
};
```

Setting the Treasure Coordinates

```
var width = 400;
var height = 400;
v var target = {
            x: getRandomNumber(width),
            y: getRandomNumber(height)
};
```

The Click Handler

```
$("#map").click(function (event) {
// Click handler code goes here
});
```

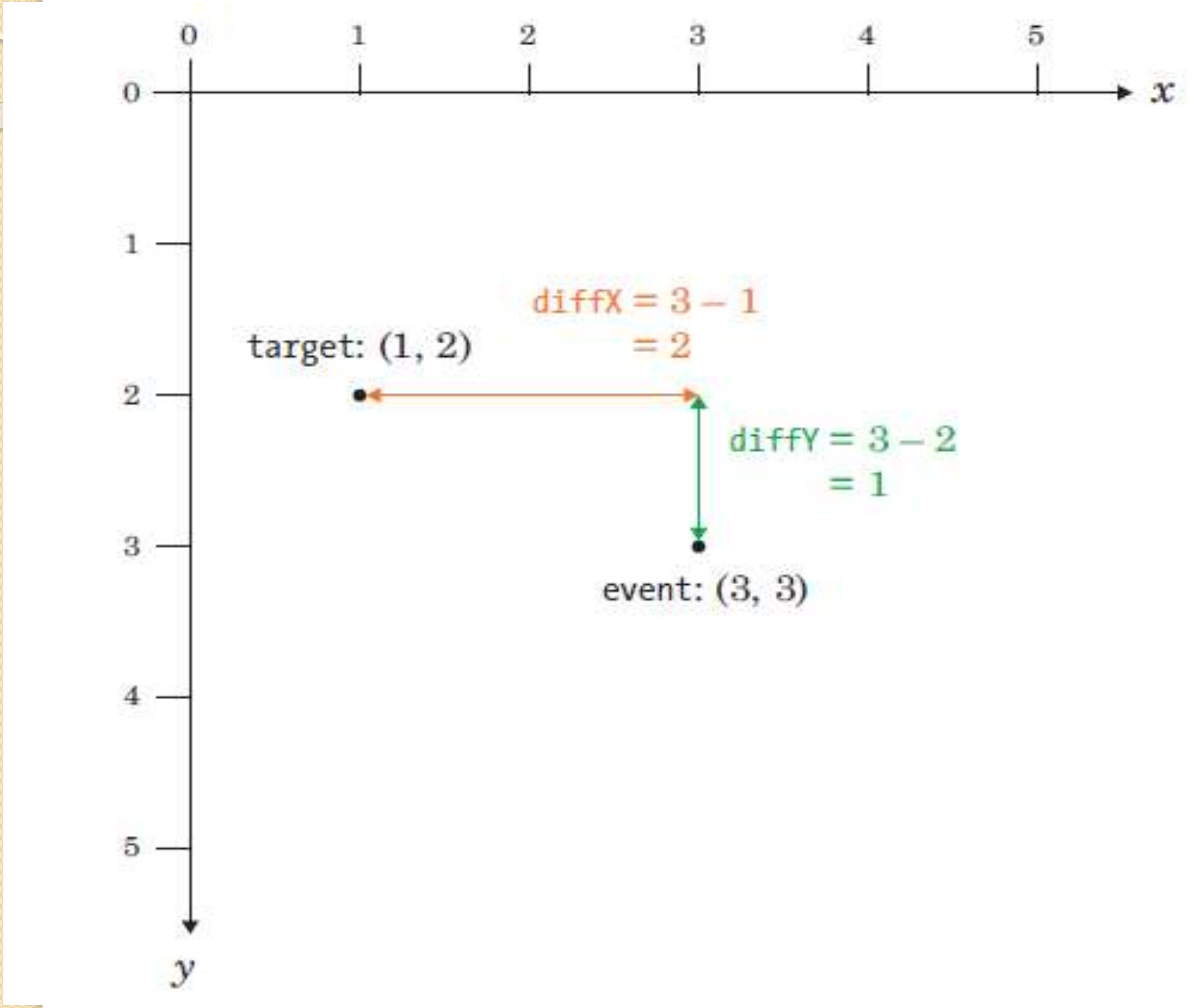
Counting Clicks

- var clicks = 0;

Calculating the Distance Between the Click and the Treasure

```
var getDistance = function (event, target) {
var diffX = event.offsetX - target.x;
var diffY = event.offsetY - target.y;
return Math.sqrt((diffX * diffX) + (diffY * diffY));
};
```

Calculating the horizontal and vertical distances between event and target

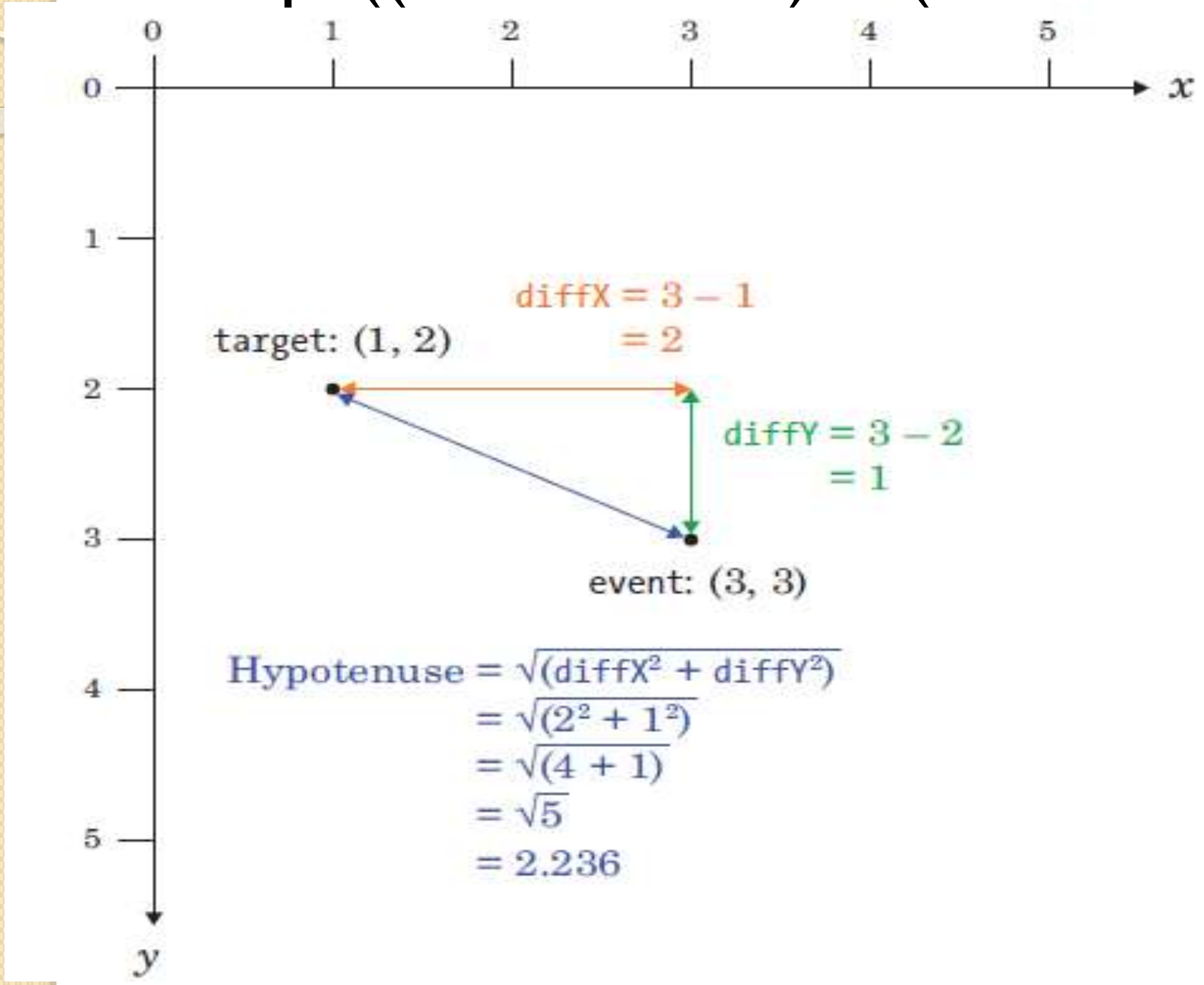


Using the Pythagorean Theorem

- To calculate the distance between the click and the treasure,we need to calculate the length of the hypotenuse, based on the lengths diffX and diffY.

Sample calculation

- Math.sqrt((diffX * diffX) + (diffY * diffY))



Calculating the hypotenuse to find out the distance between event and target

Telling the Player How Close They Are

```
var getDistanceHint = function (distance) {
if (distance < 10) {
return "Boiling hot!";
} else if (distance < 20) {
return "Really hot";
} else if (distance < 40) {
return "Hot";
} else if (distance < 80) {
return "Warm";
} else if (distance < 160) {
return "Cold";
} else if (distance < 320) {
return "Really cold";
} else {
return "Freezing!“;  }
};
```

click handler to calculate the distance, pick the appropriate string and display that string to the player

```
var distance = getDistance(event, target);
var distanceHint = getDistanceHint(distance);
$("#distance").text(distanceHint);
```

Checking If the Player Won

```
if (distance < 8) {
alert("Found the treasure in " + clicks + "
  clicks!");
}
```

Putting It All Together

Programming Challenges

#1: Increasing the Playing Area

- You could make the game harder by increasing the size of the playing area. How would you make it 800 pixels wide by 800 pixels tall?

#2: Adding More Messages

- Try adding some extra messages to display to the player (like "Really really cold!"), and modify the distances to make the game your own.

#3: Adding a Click Limit

- Add a limit to the number of clicks and show the message "GAME OVER" if the player exceeds this limit.

#4: Displaying the Number of Remaining Clicks

- Show the number of remaining clicks as an extra piece of text after the distance display so the player knows if they're about to lose.

CHAPTER 12

OBJECT ORIENTED PROGRAMMING

Object-oriented programming

- It is a way to design and write programs so that all of the program’s important parts are represented by objects.

For example, when building a racing game

- car as an object
- create multiple car objects that share the same properties and functionality.

A Simple Object

```
var dog = {
name: "Pancake",
legs: 4,
isAwesome: true
};
```

“Accessing Values in Objects”

- access its properties using dot notation
- dog.name;
- "Pancake“

Add properties to a JavaScript

```
dog.age = 6;
dog;
```

OUTPUT:

```
Object {name: "Pancake", legs: 4, isAwesome: true,
    age: 6}
```

Adding Methods to Objects

created several properties with different

- kinds of values
- a string ("Pancake")
- Numbers (4 and 6)
- a Boolean (true).

function as a property inside object

- save a function as a property in an object, that

property is called a *method.*

Accessing Bark method

```
dog.bark = function () {
 console.log("Woof woof! My name is " +
   this.name + "!");
};
```

- dog.bark();

Using the this Keyword

- this keyword inside a method to refer to the object on which the method is currently being called
- For example, when call the bark method on the dog object, this refers to the dog object.
- this.name refers to dog.name.

this keyword

- The this keyword makes methods more versatile, allowing you to add the same method to multiple objects and have it access the properties of whatever object it’s currently being called on.

Sharing a Method Between Multiple Objects

```
var speak = function () {
console.log(this.sound + "! My name is " +
   this.name + "!");
};
```

Sharing a Method Between Multiple Objects

- create a new function called speak that we can use as a method in multiple objects that represent different animals.
- When speak is called on an object, it will use the object’s name (this.name) and the sound the animal makes (this.sound) to log a message.

Multiple Objects

```
var cat = {
sound: "Miaow",
name: "Mittens",
speak: speak
};

cat.speak();
```

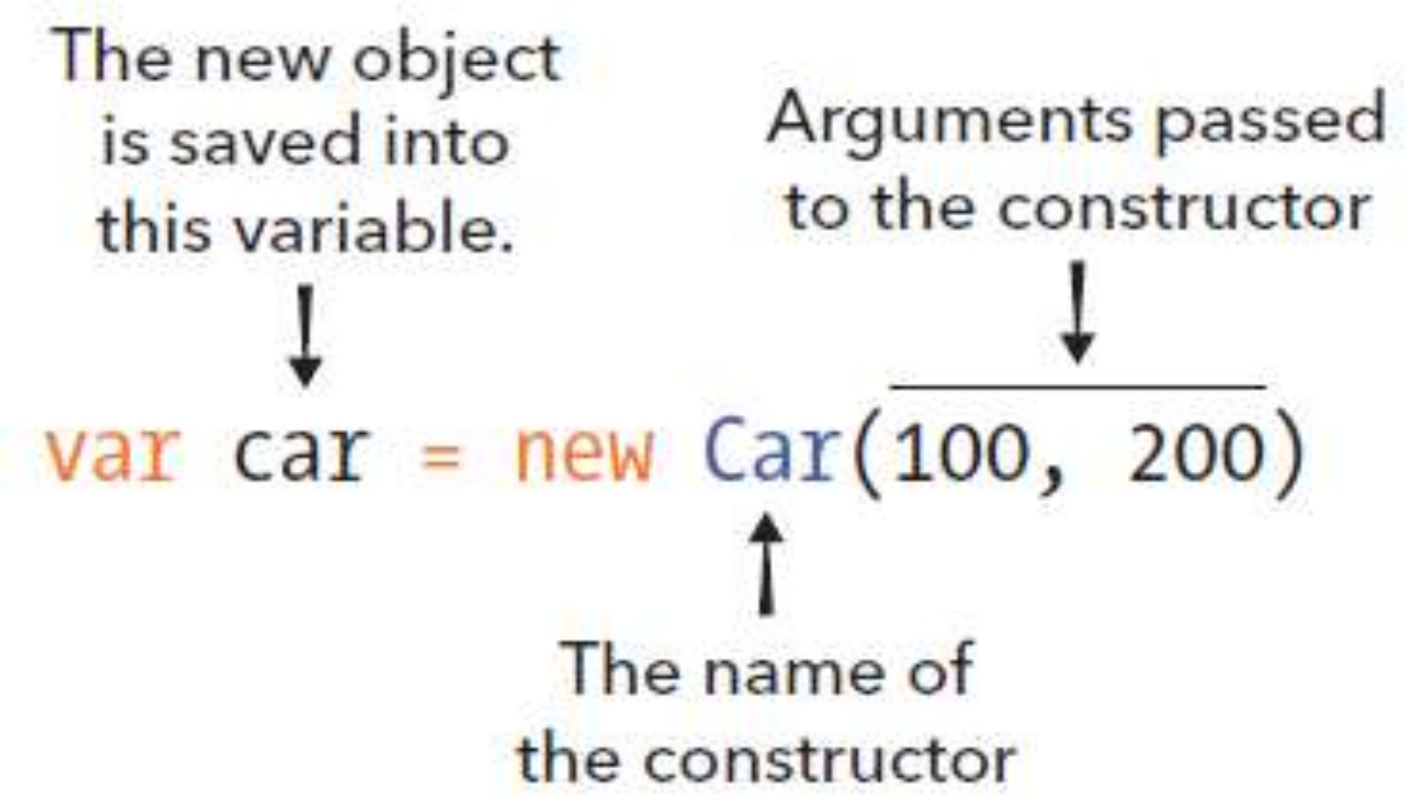
speak function as a method in other objects too:

```
var pig = {
sound: "Oink",
name: "Charlie",
speak: speak
};

var horse = {
sound: "Neigh",
name: "Marie",
speak: speak
};

pig.speak();
horse.speak();
```

Creating Objects Using Constructors

- *constructor is a function that creates objects and* gives them a set of built-in properties and methods.

Anatomy of the Constructor



```
var car = new Car(100, 200)
```

The new object is saved into this variable.
Arguments passed to the constructor
The name of the constructor

Creating a Car Constructor

create a Car constructor that will add an x and y property to each new object it creates. These properties will be used to set each car’s onscreen position when we draw it.

cars.html

- Creating the HTML Document Before we can build our constructor, we need to create a new HTML document.
- Make a new file called *cars.html*

cars.html

```
<!DOCTYPE html>
<html>
<head>
<title>Cars</title>
</head>
<body>
<script src="https://code.jquery.com/jquery-
    2.1.0.js"></script>
<script>
// Code goes here
</script>
</body>
</html>
```

The Car Constructor Function

- Now add this code to the empty <script> tags in *cars.html (replacing* the comment // Code goes here) to create the Car constructor that gives each car a set of coordinates.

```
 <script>
var Car = function (x, y) {
this.x = x;
this.y = y;
};
</script>
```

- Our new constructor Car takes the arguments x and y. We’ve
- added the properties this.x and this.y to store the x and y values
- passed to Car in our new object. This way, each time we call Car as
- a constructor, a new object is created with its x and y properties set
- to the arguments we specify.

Calling the Car Constructor

- calling a constructor to create a new object. For example, to create a car object named tesla, open *cars.html in a web browser.*

```
var tesla = new Car(10, 20);
tesla;
Car {x: 10, y: 20}
```

- The code new Car(10, 20) tells JavaScript to create an object
- using Car as a constructor, pass in the arguments 10 and 20 for its
- x and y properties, and return that object. We assign the returned
- object to the tesla variable with var tesla.
- Then when we enter tesla, the Chrome console returns the
- name of the constructor and its x and y values: Car {x: 10, y: 20}.

Drawing the Cars

- To show the objects created by the Car constructor, create a function called drawCar to place an image of a car at each car object's (*x, y) position in a browser window. how* this function works, rewrite it in a more object-oriented way in "Adding a draw Method to the Car Prototype".
- Add this code between the <script> tags in *cars.html*

- create a string containing HTML that points to an image of a car. (Using single quotes to create this string lets us use double quotes in the HTML.)
- pass carHTML to the $function, which converts it from a string to a jQuery element. That means the carElement variable now holds a jQuery element with the information for our <img> tag, and we can tweak this element before adding it to the page.

- use the css method on carElement to set the position of the car image. This code sets the left position of the image to the car object’s x value and its top position to the y value.
- In other words, the left edge of the image will be x pixels from the left edge of the browser window, and the top edge of the image will be y pixels down from the top edge of the window

Testing the drawCar Function

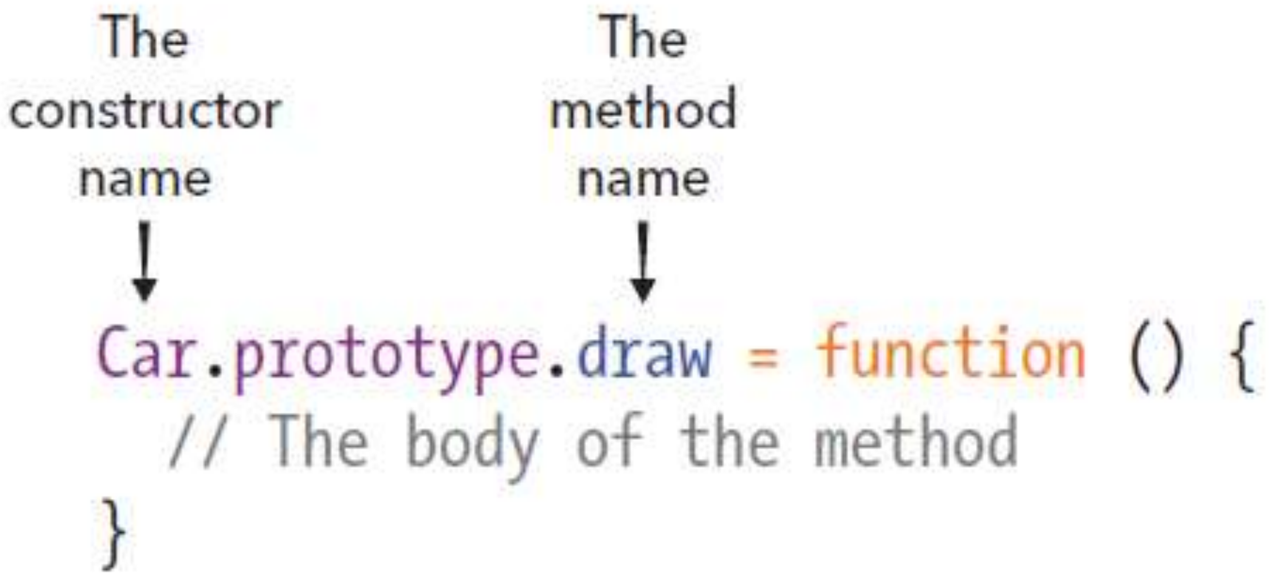
- Let’s test the drawCar function to make sure it works. Add this code to your *cars.html file (after the other JavaScript code) to create* two cars.
- Here, we use the Car constructor to create two car objects, one at the coordinates (20, 20) and the other at (100, 200), and then we use drawCar to draw each of them in the browser. Now when you open *cars.html,*

Customizing Objects with Prototypes

- A more object-oriented way to draw our cars would be to give each car object a draw method.
- Then, instead of writing drawCar(tesla), write tesla.draw().
- In object-oriented programming, want objects to have their own functionality built in as methods

- In this case, the drawCar function is always meant to be used on car objects, so instead of saving drawCar as a separate function, we should include it as part of each car object.
- JavaScript *prototypes make it easy to share functionality* (as methods) between different objects.
- All constructors have a prototype property, and we can add methods to it.
- Any method that we add to a constructor’s prototype property will be available as a method to all objects created by that constructor.

syntax for adding a method to a prototype property.



```
Car.prototype.draw = function () {
  // The body of the method
}
```

Adding a draw Method to the Car Prototype

- add a draw method to Car.prototype so that all objects we create
- using Car will have the draw method. Using **File->Save As,**
- Save your *cars.html file as cars2.html. Then replace all of the JavaScript*
- in your second set of <script> tags in *cars2.html*

Adding a draw Method to the Car Prototype

- After creating our Car constructor, we add a new method called draw to Car.prototype.
- This makes the draw method part of all of the objects created by the Car constructor.
- The contents of the draw method are a modified version of our drawCar function.
- First, we create an HTML string and save it as carHTML.
- create a jQuery element representing this HTML, but this time we save it as a property of the object by

  assigning it to this.carElement.

Adding a draw Method to the Car Prototype

- Then at x, we use this.x and this.y to set the coordinates of the top-left corner of the current car image. (Inside a constructor, this refers to the new object currently being created.)
- Haven’t changed the code’s functionality, only its organization. The advantage to this approach is that the code for drawing the car is part of the car, instead of a separate function.

Adding a moveRight Method

- add some methods to move the cars around, beginning with a moveRight method to move the car 5 pixels to the right of its current position. Add the following code after your definition of Car.prototype.draw:

moveRight method

- Save the moveRight method in Car.prototype to share it with all objects created by the Car constructor.
- With this.x += 5 we add 5 to the car’s x value, which moves the car 5 pixels to the right. Then we use the css method on this.carElement to update the car’s position in the browser.
- Try the moveRight method in the browser onsole. First, refresh *cars2.html, and then open the console and enter these lines*

```
tesla.moveRight();
tesla.moveRight();
tesla.moveRight();
```

tesla.moveRight

- Each time you enter tesla.moveRight, the top car should move 5 pixels to the right.
- You could use this method in a racing game to show the car moving down the racetrack.

Try It Out!

- Try moving nissan to the right. How many times do you need to call moveRight on nissan to make it line up with tesla?
- Use setInterval and moveRight to animate nissan so that it drives across the browser window.

Adding the Left, Up, and Down move Methods

- Now we’ll add the remaining directions to our code so that we can move our cars around the screen in any direction.
- These methodsare basically the same as moveRight, so we’ll write them all at once.
- Add the following methods to *cars2.html just after the code for* moveRight:

Programming Challenges

- #1: **Drawing in the Car Constructor**

  Add a call to the draw method from inside the Car constructor so that car objects automatically appear in the browser as soon as you create them.

#2: Adding a speed Property

- Modify the Car constructor to add a new speed property with a value of 5 to the constructed objects.
- Then use this property instead of the value 5 inside the movement methods.
- Now try out different values for speed to make the cars move faster or slower.

#3: Racing Cars

- Modify the moveLeft, moveRight, moveUp, and moveDown methods so they take a single distance argument, the number of pixels to move, instead of always moving 5 pixels. For example, to move the nissan car 10 pixels to the right, you would call nissan.moveRight(10).
- Now, use setInterval to move the two cars (nissan and tesla) to the right every 30 milliseconds by a different random distance between 0 and 5. You should see the two cars animate across the screen, jumping along at varying speeds. Can you guess which car will make it to the edge of the window first?

UNIT IV

CANVAS

UNIT – IV

CHAPTER 13

The canvas Element

CANVAS

- use JavaScript to draw pictures with the HTML canvas element, which you can think of as a blank canvas or sheet of paper
- Lines
- Shapes
- Text

Creating a Basic Canvas

- create a new HTML document for the canvas element.

canvas.html

```
<!DOCTYPE html>
<html>
<head>
<title>Canvas</title>
</head>
<body>
 <canvas id="canvas" width="200"
    height="200"></canvas>
<script>
// We'll fill this in next
</script>
</body>
```

- Create a canvas element and give it an id property of "canvas“
- The width and height properties set the dimensions of the canvas element in pixels.
- Set both dimensions to 200.

Drawing on the Canvas

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
ctx.fillRect(0, 0, 10, 10);
```

Selecting and Saving the canvas Element

1. Select the canvas element using document.getElementById("canvas")
2. The getElementById method returns a DOM object representing the element with the supplied id.
3. This object is assigned to the canvas variable with the code

```
var canvas = document.getElementById("canvas")
```

Getting the Drawing Context

- Get the *drawing context from the canvas element*
- A drawing context is a JavaScript object that includes all the methods and properties for drawing on a canvas.
- To get this object, we call getContext on canvas and pass it the string "2d" as an argument.
- This argument says that we want to draw a two-dimensional image on our canvas.
- We save this drawing context object in the variable ctx using the code

```
var ctx = canvas.getContext("2d").
```

Drawing a Square

- draw a rectangle on the canvas by calling the method fillRect on the drawing context.
- The fillRect method takes four arguments.
- *x- and y-coordinates of the top-left corner of the rectangle (0, 0) and the* width and height of the rectangle (10, 10).
- Draw a 10-pixel-by-10-pixel rectangle at coordinates (0, 0), which are at the top-left corner of the canvas.

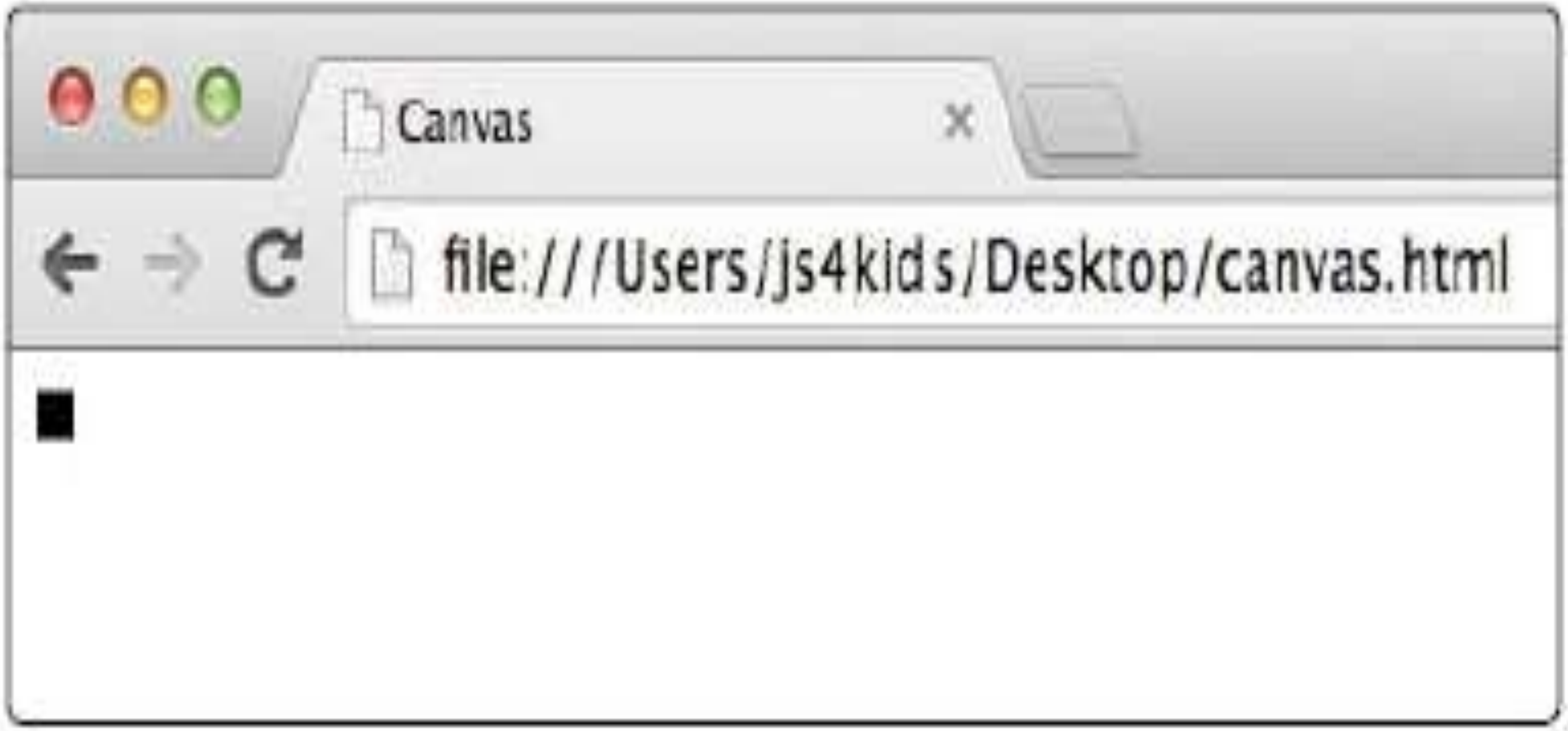


Figure 13-1: Our first canvas drawing

Drawing Multiple Squares

- use a loop to draw multiple squares running diagonally down the screen. Replace the code in the <script> tags with the following.
- set of eight black squares:

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
for (var i = 0; i < 8; i++) {
ctx.fillRect(i * 10, i * 10, 10, 10);
}
```

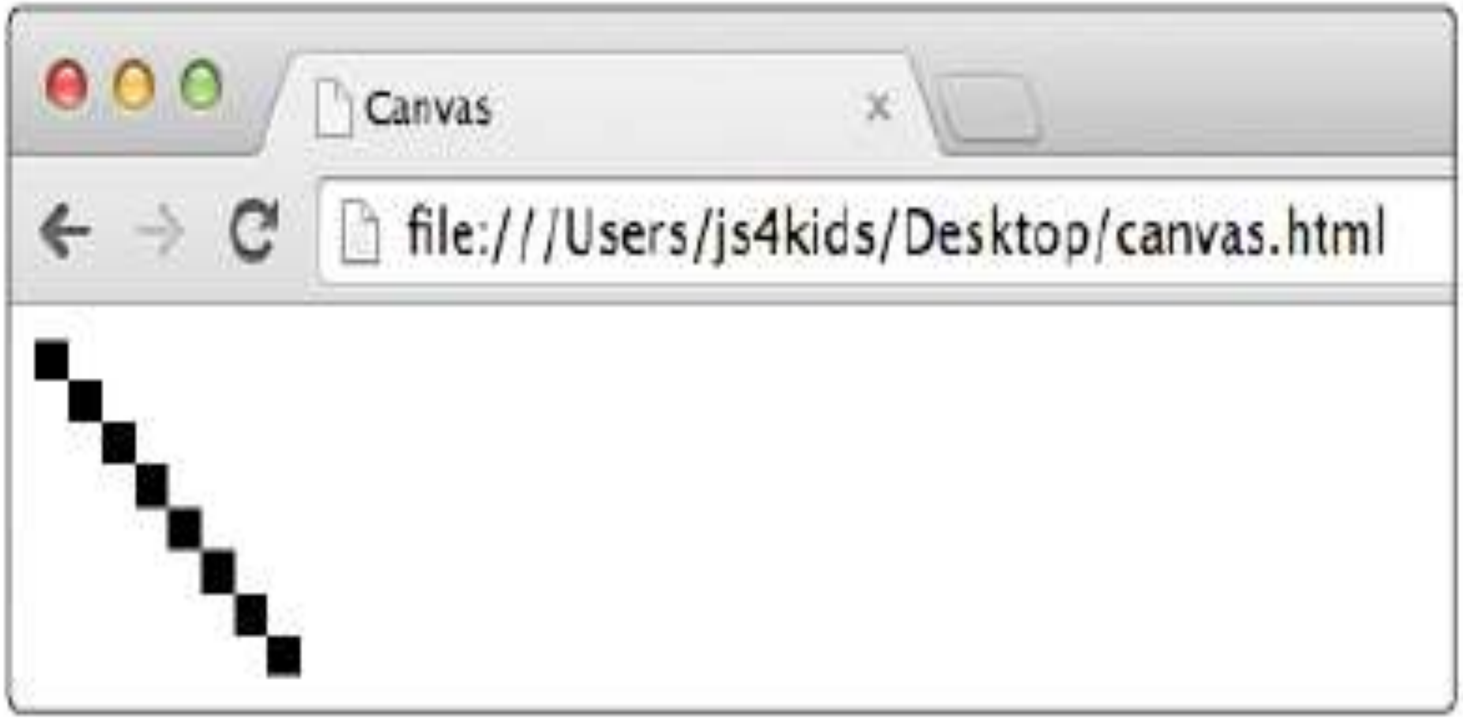


Figure 13-2: Drawing multiple squares using a for loop

Try It Out!

- Now that you know how to draw squares and rectangles on the canvas, try drawing this little robot using the fillRect method.

Hint: You’ll need to draw six separate rectangles. I made the head using a 50-pixel-by-50-pixel rectangle. The neck, arms, and legs are all 10 pixels wide.

Changing the Drawing Color

- By default, when you call fillRect, JavaScript draws a black rectangle.
- To use a different color, you can change the fillStyle property of the drawing context.
- When you set fillStyle to a new color, everything you draw will be drawn in that color until you change fillStyle again.
- The easiest way to set a color for fillStyle is to give it the name of a color as a string.

- var canvas = document.getElementById("canvas");
- var ctx = canvas.getContext("2d");
- ctx.fillStyle = "Red";
- ctx.fillRect(0, 0, 100, 100);

OUTPUT

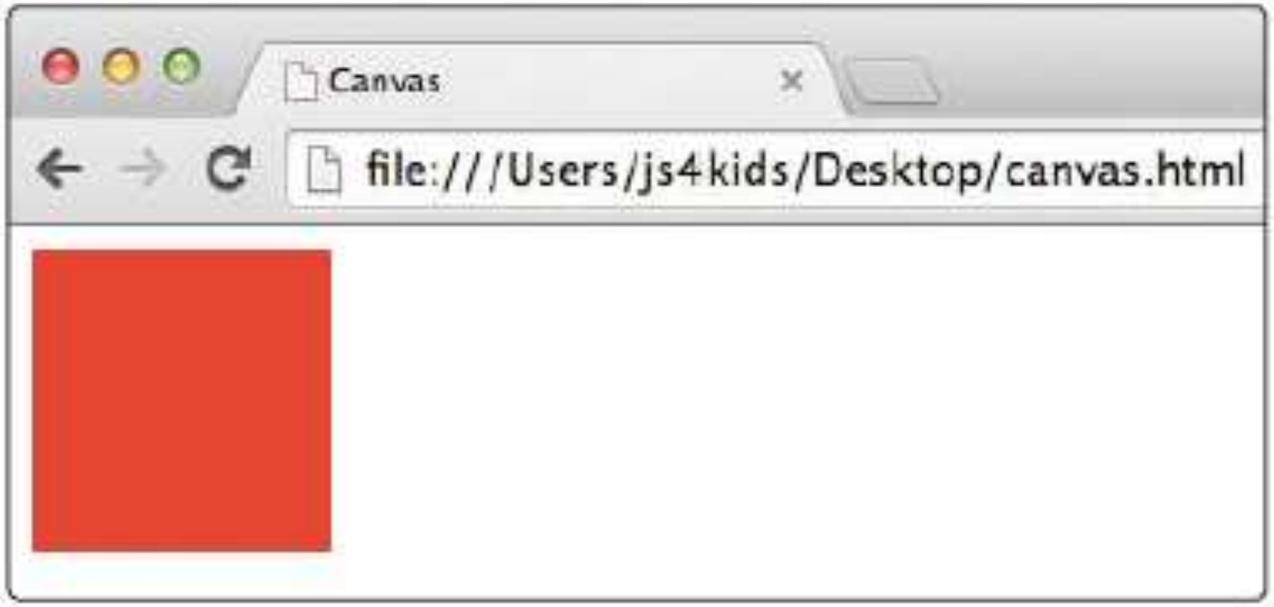


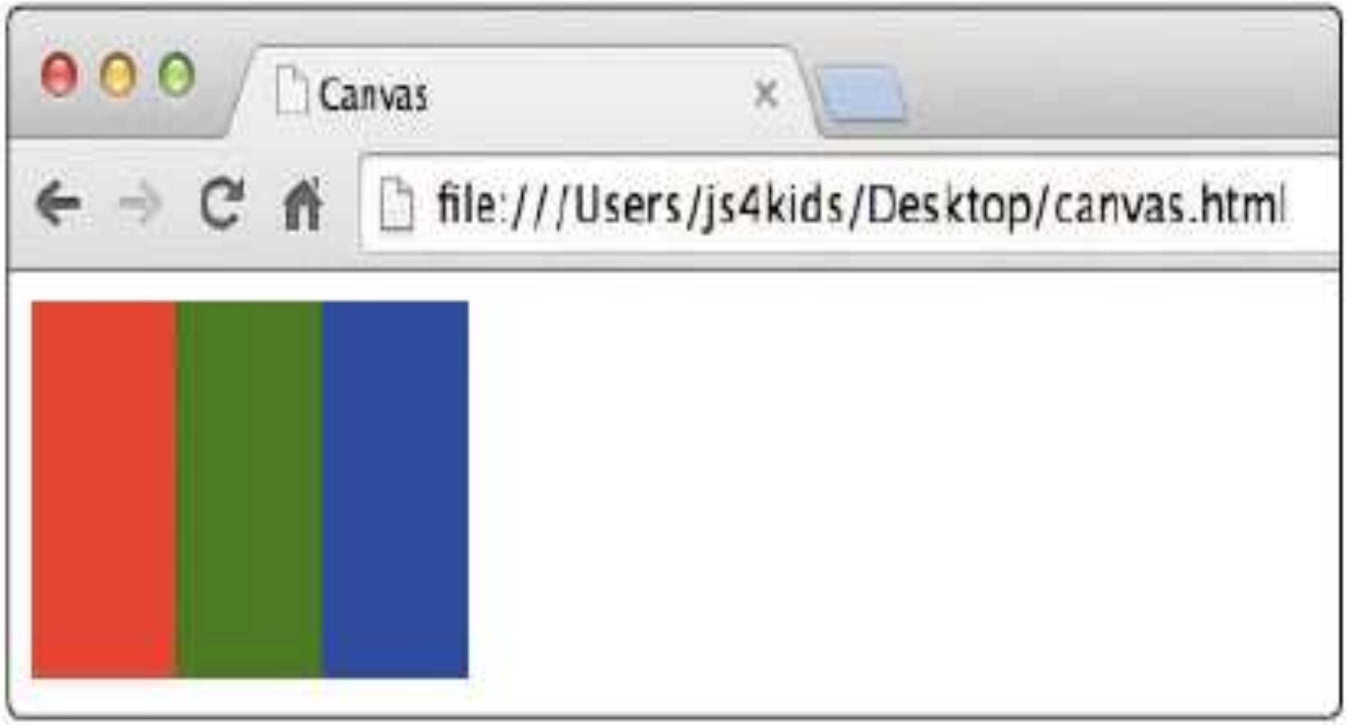
Figure 13-3: A red square

- *JavaScript understands more than 100 color names, including Green, Blue, Orange, Red, Yellow, Purple, White, Black, Pink, Turquoise, Violet, SkyBlue, PaleGreen, Lime, Fuchsia, DeepPink, Cyan, and Chocolate. You'll find a full list on the CSS-Tricks*
- *website: http://css-tricks.com/snippets/*css/ named-colors-and-hex-equivalents/.

Try It Out!

- Look at the CSS-Tricks website (*http://css-tricks.com/ snippets/css/named-colors-and-hex-equivalents/) and* choose three colors you like. Draw three rectangles using these colors. Each rectangle should be 50 pixels wide and 100 pixels tall. Don't include any space between them. You should end up with something like this:

Draw three rectangles using these colors



Drawing Rectangle Outlines

- fillRect method draws a filled-in rectangle.
- use the strokeRect method
- Running this code should draw the outline of small rectangle
- var canvas = document.getElementById("canvas");
- var ctx = canvas.getContext("2d");
- ctx.strokeRect(10, 10, 100, 20);

STROKE OUTPUT

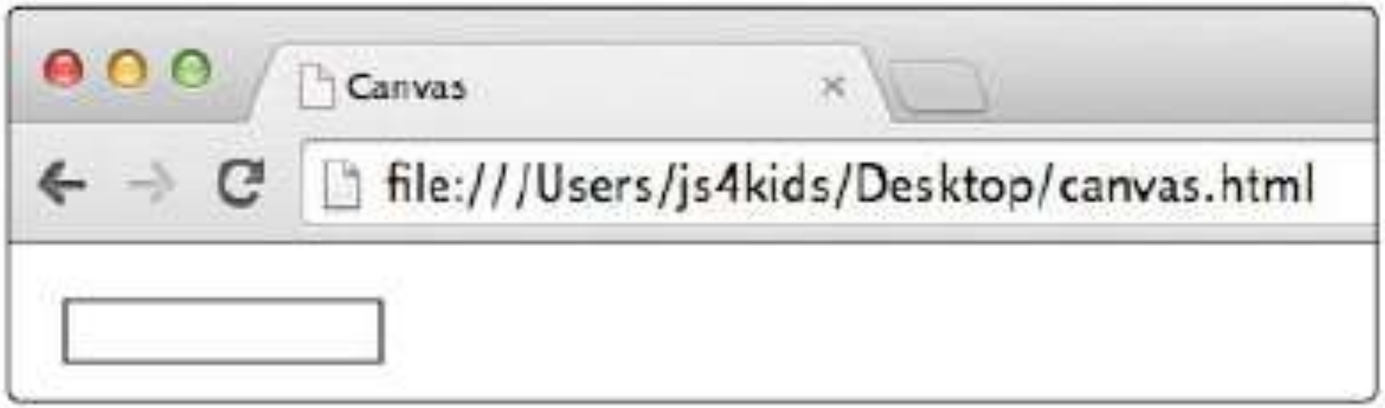


Figure 13-4: Using `strokeRect` *to draw the outline of a rectangle*

- The strokeRect method takes the same arguments as

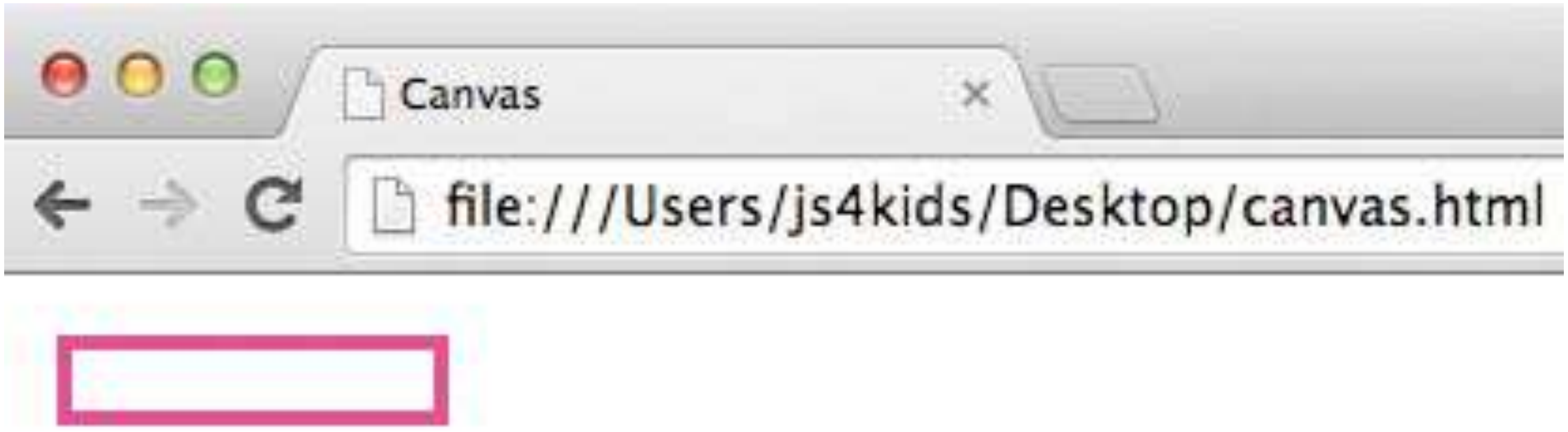
fillRect:

first the x- *and y-coordinates of the top-left corner, followed by the* width and height of the rectangle. In this example, we see that a rectangle is drawn starting at 10 pixels from the top left of the canvas, and it is 100 pixels wide by 20 pixels tall.

- Use the strokeStyle property to change the color of the rectangle's outline.
- To change the thickness of the line, use the lineWidth property.

CODE AND OUTPUT

- var canvas = document.getElementById("canvas");
- var ctx = canvas.getContext("2d");
- u ctx.strokeStyle = "DeepPink";
- v ctx.lineWidth = 4;
- ctx.strokeRect(10, 10, 100, 20);



Drawing Lines or Paths

- Lines on the canvas are called *paths. To draw a path with the* canvas, you use *x- and y-coordinates to set where each line should* begin and end. By using a careful combination of starting and stopping coordinates, you can draw specific shapes on the canvas.

CODE SNIPPETS

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
ctx.strokeStyle = "Turquoise";
ctx.lineWidth = 4;
ctx.beginPath();
ctx.moveTo(10, 10);
ctx.lineTo(60, 60);
ctx.moveTo(60, 10);
ctx.lineTo(10, 60);
ctx.stroke();
```

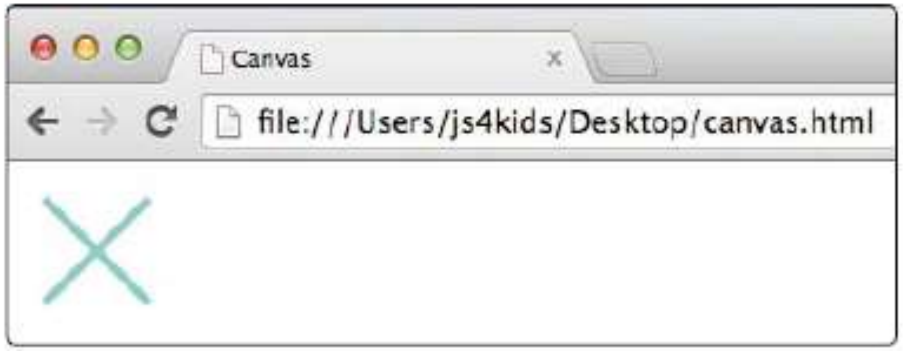
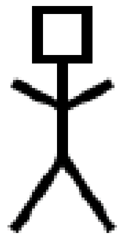


Figure 13-6: A turquoise X, drawn with `moveTo` *and* `lineTo`

Try It Out!

- Try drawing this happy stickman using the beginPath, moveTo, lineTo, and stroke methods.

- You can use the strokeRect method for the head. The head is a 20-pixel-by-20-pixel square, and the line width is 4 pixels.



Filling Paths

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
ctx.fillStyle = "SkyBlue";
ctx.beginPath();
ctx.moveTo(100, 100);
ctx.lineTo(100, 60);
ctx.lineTo(130, 30);
ctx.lineTo(160, 60);
ctx.lineTo(160, 100);
ctx.lineTo(100, 100);
ctx.fill();
```

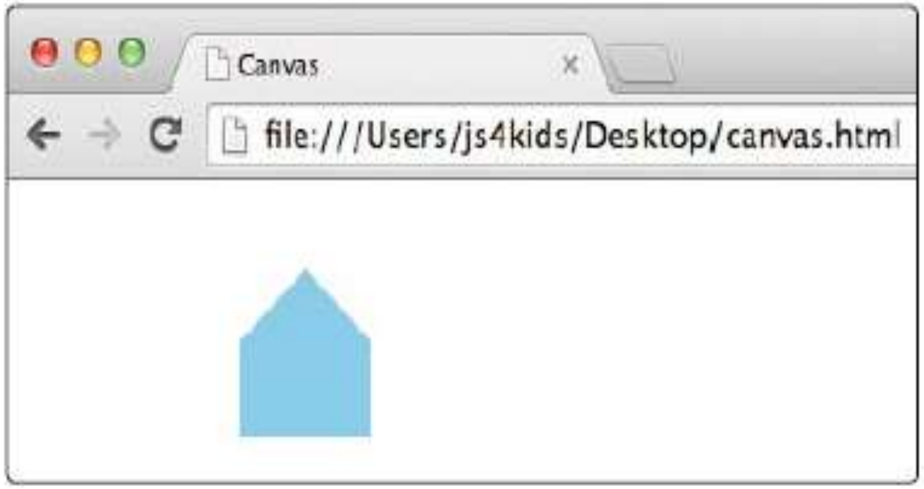
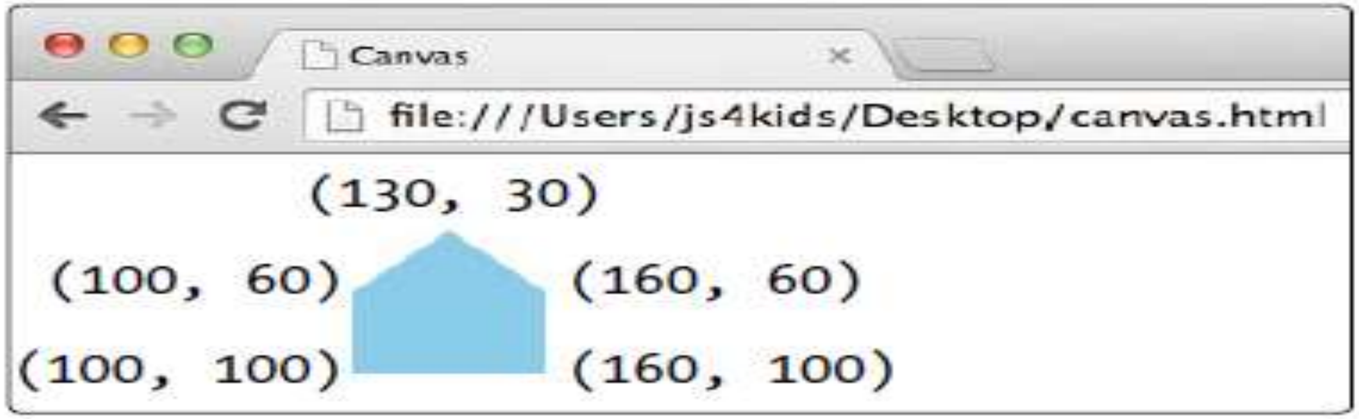


Figure 13-7: A sky blue house, drawn with a path and filled with the `fill` *method*

House with each coordinate labeled

- After setting our drawing color
- to SkyBlue, we begin our path with beginPath and then move to ourstarting point of (100, 100) using moveTo.
- Next we call lineTo five times for each corner of the house, using five sets of coordinates.
- The final call to lineTo completes the path by going back to the starting point of (100, 100).



- call the fill method, which fills our path with the chosen fill color, SkyBlue.

Drawing Arcs and Circles

- In addition to drawing straight lines on the canvas, you can
- use the arc method to draw arcs and circles. To draw a circle,
- you set the circle’s center coordinates and *radius (the distance*
- between the circle’s center and outer edge) and tell JavaScript
- how much of the circle to draw by providing a starting angle and
- ending angle as arguments. You can draw a full circle, or just a
- portion of a circle to create an arc.

- The starting and ending angles are measured in *radians. When*
- measured in radians, a full circle starts at 0 (at the right side of the
- circle) and goes up to π × 2 radians. So to draw a full circle, you tell
- arc to draw from 0 radians to π × 2 radians. Figure 13-9 shows a
- circle labeled with radians and their equivalent in degrees. The
- values 360° and π × 2 radians both mean a full circle.

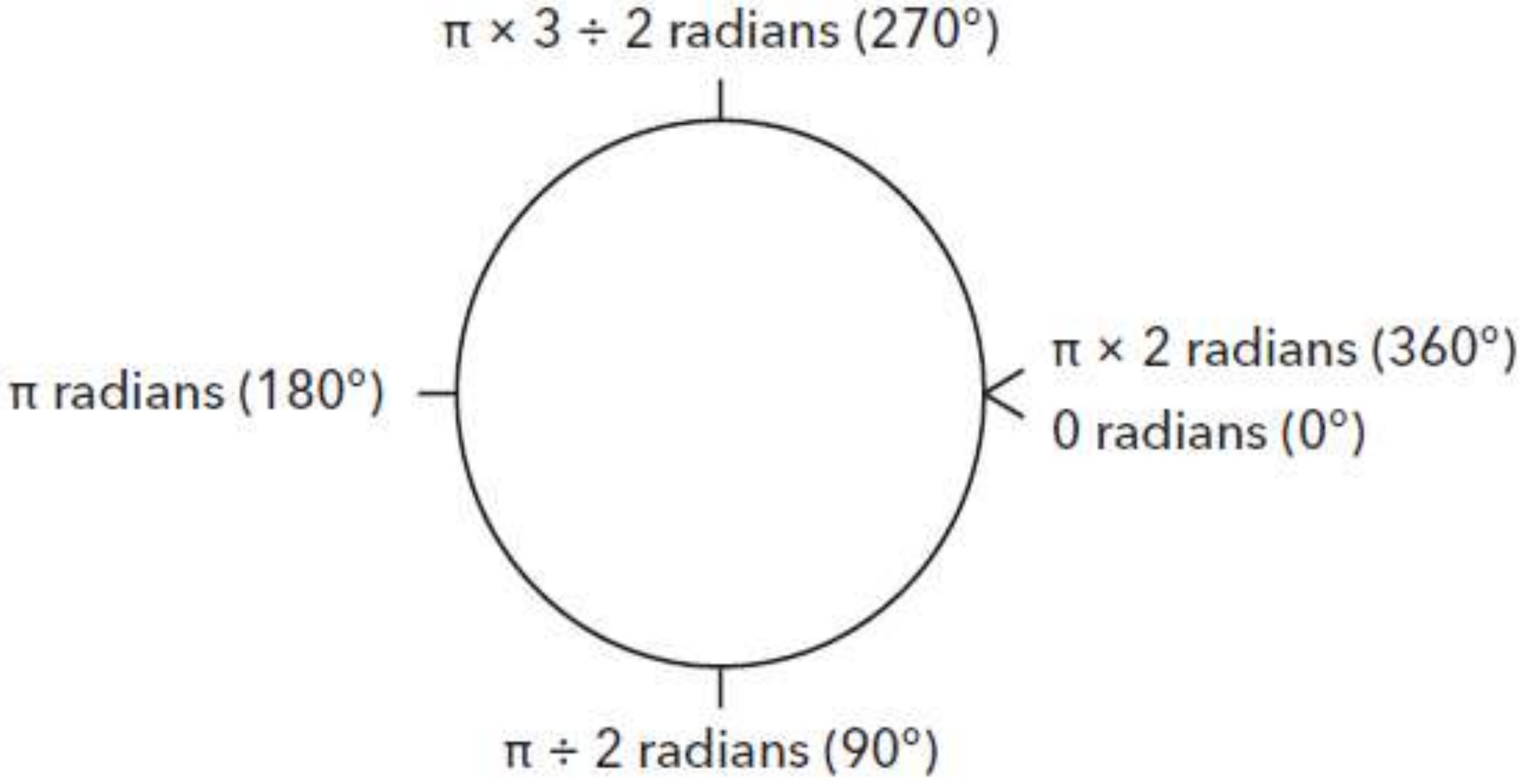


Figure 13-9: Degrees and radians, starting from the right side of the circle and moving clockwise

code will create a quarter circle, a half circle, and a full circle

- ctx.lineWidth = 2;
- ctx.strokeStyle = "Green";
- ctx.beginPath();
- u ctx.arc(50, 50, 20, 0, Math.PI / 2, false);
- ctx.stroke();
- 210 Chapter 13
- ctx.beginPath();
- v ctx.arc(100, 50, 20, 0, Math.PI, false);
- ctx.stroke();
- ctx.beginPath();
- w ctx.arc(150, 50, 20, 0, Math.PI * 2, false);
- ctx.stroke();

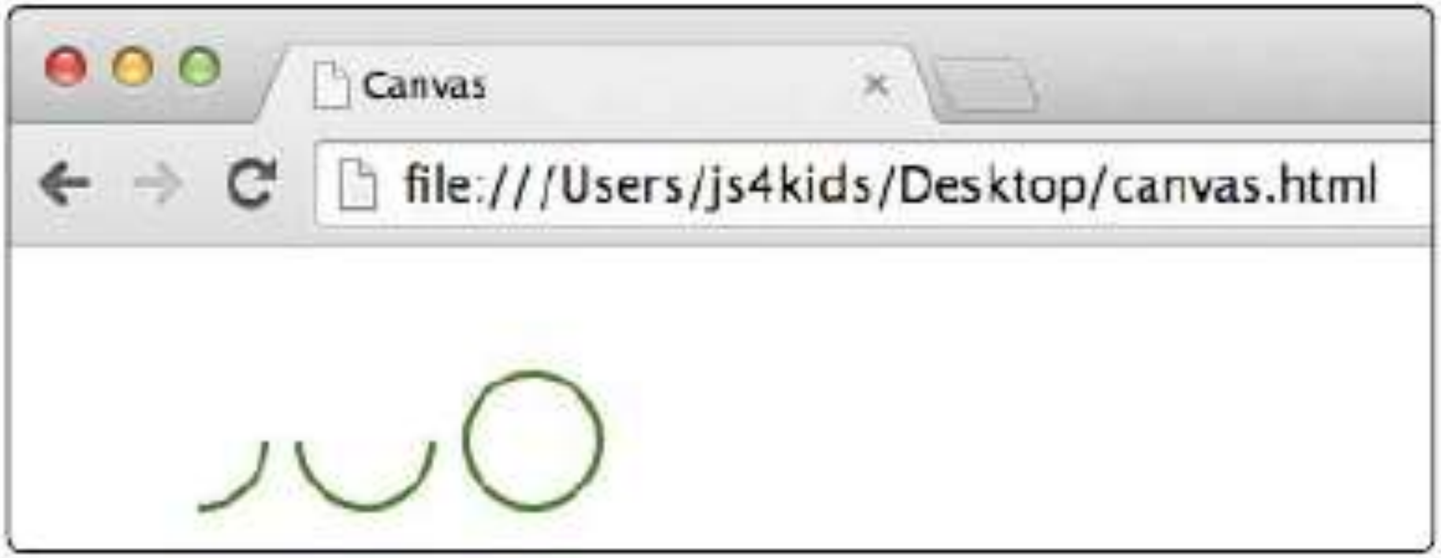


Figure 13-10: Drawing a quarter circle, a half circle, and a full circle

Drawing a Quarter Circle or an Arc

- The first block of code draws a quarter circle.
- At u, after calling beginPath, we call the arc method. We set the center of the circle at the point (50, 50) and the radius to 20 pixels. The starting angle is 0 (which draws the arc starting from the right of the circle), and the ending angle is Math.PI / 2. Math.PI is how JavaScript refers to the number π (pi). Because a full circle is π × 2 radians, π radians means a half circle, and π ÷ 2 radians (which we're using for this first arc) gives us a quarter circle. Figure 13-11 shows the start and end angles.

start and end angles

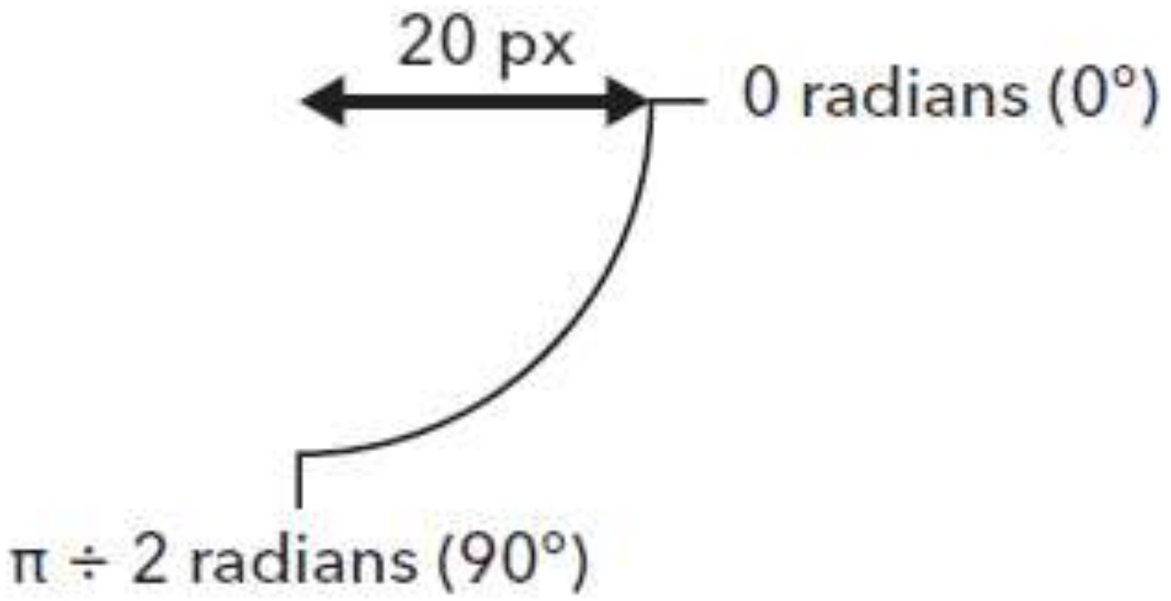


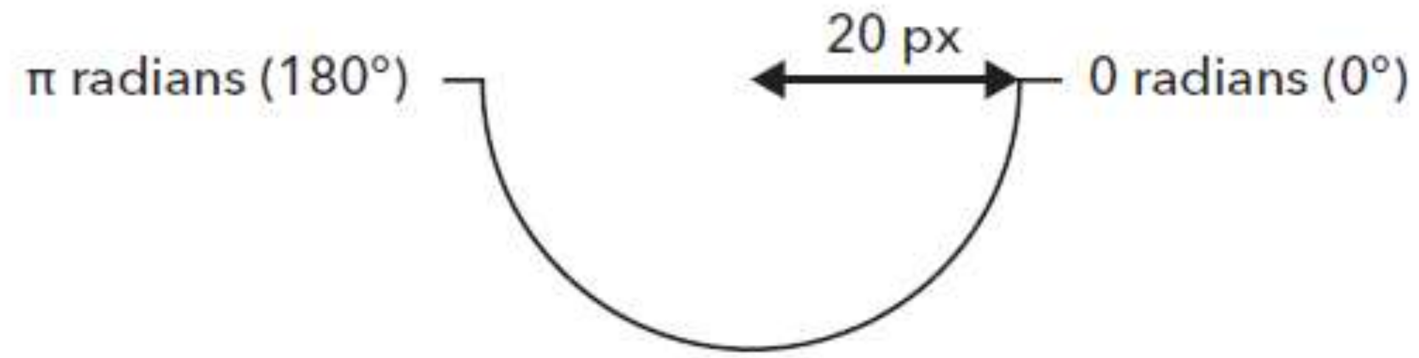
Figure 13-11: The start angle (0 radians, or 0°) and end angle (π ÷ 2 radians, or 90°) of the quarter-circle

start and end angles

- We pass false for the final argument,which tells arc to draw in a clockwise direction.
- If you want to draw in a counter clockwise

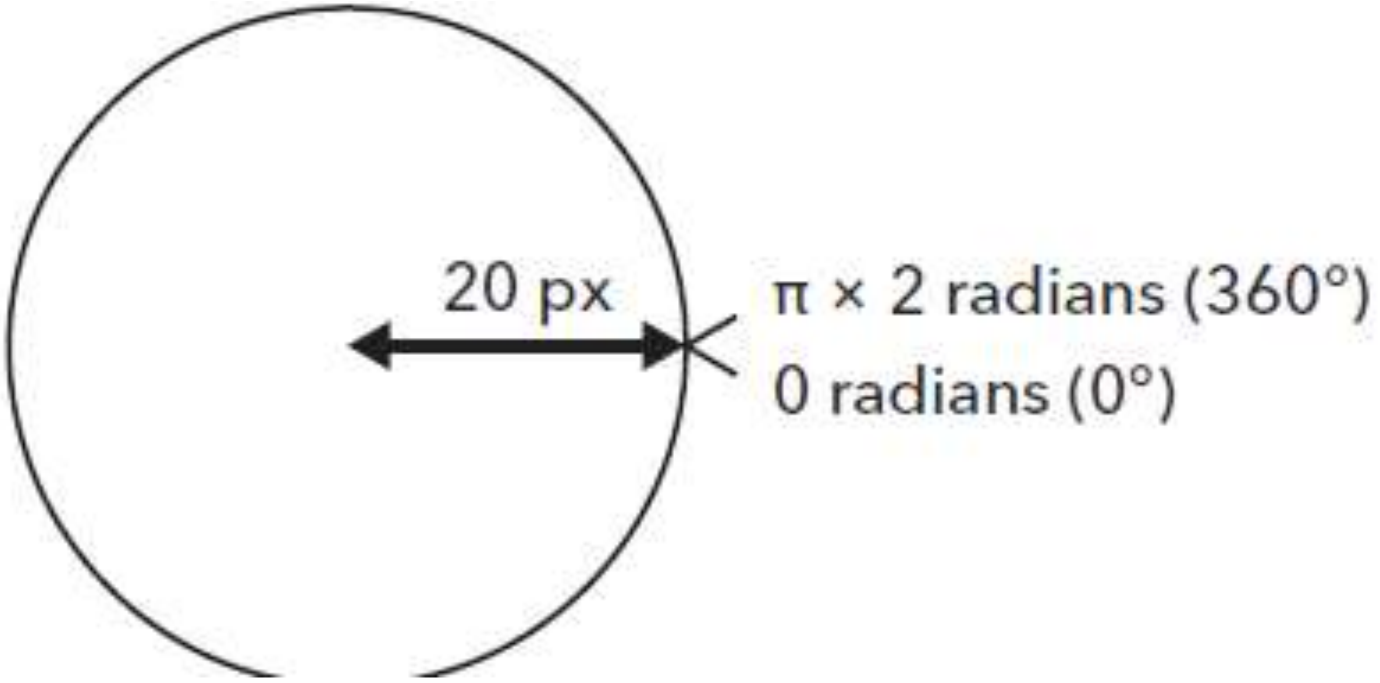
direction, pass true for this final argument.

Drawing a Half Circle



The start angle (0 radians, or 0°) and end angle (π radians, or 180°) of the half circle

Drawing a Full Circle



- *The start angle (0 radians, or 0°) and end angle (π . 2 radians, or 360°) of the full circle*

Drawing Lots of Circles with a Function

```
var circle = function (x, y, radius) {
ctx.beginPath();
ctx.arc(x, y, radius, 0, Math.PI * 2, false);
ctx.stroke();
};
```

draw some colorful concentric circles

```
ctx.lineWidth = 4;
ctx.strokeStyle = "Red";
circle(100, 100, 10);
ctx.strokeStyle = "Orange";
circle(100, 100, 20);
ctx.strokeStyle = "Yellow";
circle(100, 100, 30);
ctx.strokeStyle = "Green";
circle(100, 100, 40);
```

```
ctx.strokeStyle = "Blue";
circle(100, 100, 50);

ctx.strokeStyle = "Purple";
circle(100, 100, 60);
```

colorful concentric circles

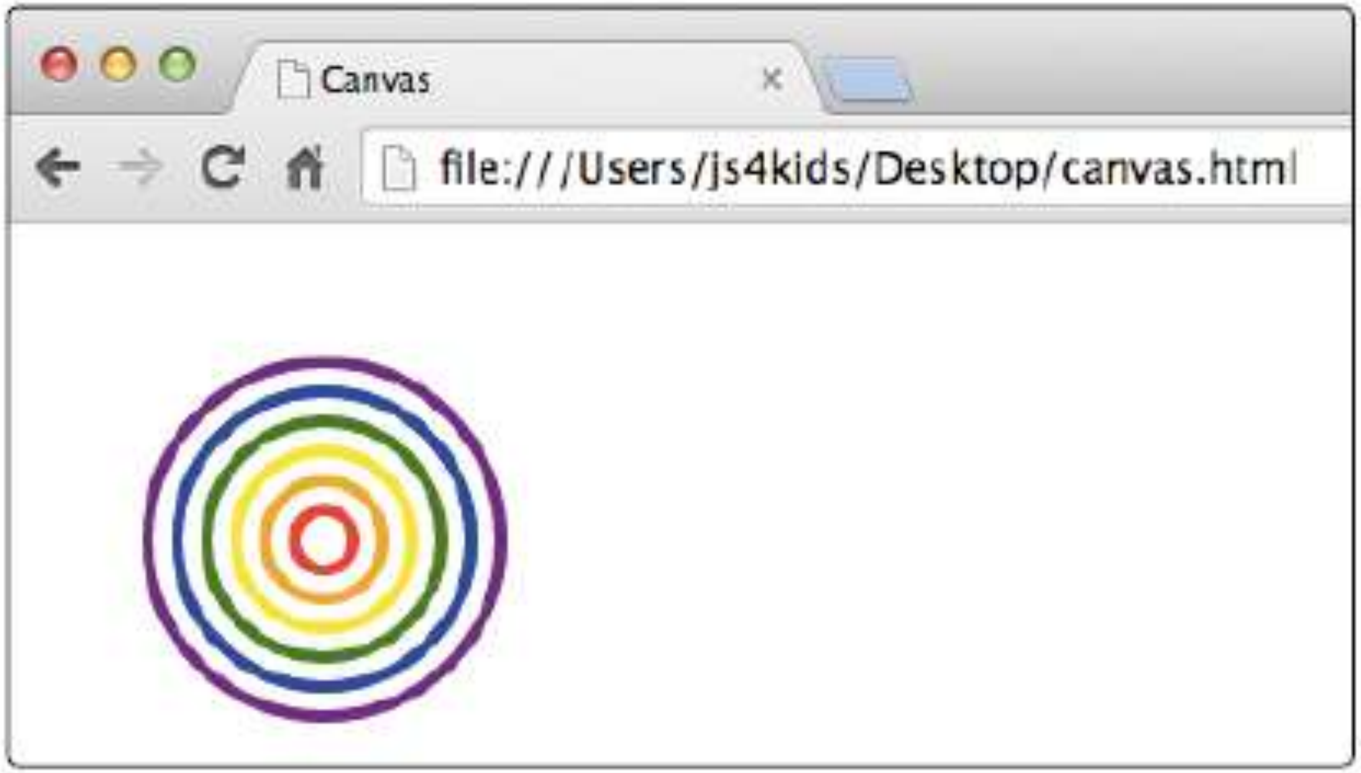
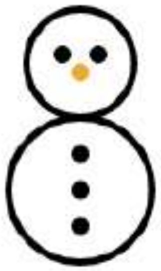


Figure 13-14: Colorful concentric circles, drawn using our circle function

Try It Out!



- How would you modify our circle function to make it fill the circle instead of outline it? Add a fourth argument, a Boolean, that says whether the circle should be filled or outlined. Passing true indicates that you want the circle to be filled. You can call the argument fillCircle.
- Using your modified function, draw this snowman, using a mix of outlined and filled circles.

Programming Challenges

- #1: A Snowman-Drawing Function
- Building on your code for drawing a snowman
- write a function that draws a snowman at a specified location, so that calling this . . .
- drawSnowman(50, 50);

would draw a snowman at the point (50, 50).

#2: Drawing an Array of Points

Write a function that will take an array of points like this:

```
var points = [[50, 50], [50, 100], [100, 100], [100, 50], ↩
[50, 50]];
drawPoints(points);
```

and draw a line connecting the points. In this example, the function would draw a line from (50, 50) to (50, 100) to (100, 100) to (100, 50) and back to (50, 50).

Now use this function to draw the following points:

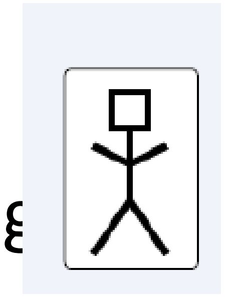
```
var mysteryPoints = [[50, 50], [50, 100], [25, 120], ↩
[100, 50], [70, 90], [100, 90], [70, 120]];
drawPoints(mysteryPoints);
```

Hint: You can use points[0][0] to get the first *x-coordinate* and points[0][1] to get the first *y-coordinate.*

#3: Painting with Your Mouse

Using jQuery and the mousemove event, draw a filled circle with a radius of 3 pixels at the mouse position whenever you move your mouse over the canvas. Because this event is triggered by every tiny movement of the mouse, these circles will join into a line as you move the mouse over the canvas. Hint: Refer to Chapter 10 for a reminder of how to respond to mousemove events.

#4: Drawing the Man in Hangman



- we created our own version of the g Hangman.
- Now you can make it closer to the real game by drawing part of a stick man every time the player gets a letter wrong.
- Hint: Keep track of the number of times the player has guessed incorrectly.
- Write a function that takes this number as an argument and draws a different part of the body depending on the number passed in.

Unit 4

MAKING THINGS MOVE ON THE CANVAS

Moving Across the Page

- Creating canvas animations in JavaScript

```
<!DOCTYPE html>
<html>
<head>
<title>Canvas Animation</title>
</head>
<body>
<canvas id="canvas" width="200" height="200"></canvas>
<script>

// We'll fill this in next
</script>
</body>
</html>
```

script element

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var position = 0;
setInterval(function () {
ctx.clearRect(0, 0, 200, 200);
ctx.fillRect(position, 0, 20, 20);
position++;
if (position > 200) {
position = 0;
}
}, 30);
```

Clearing the Canvas

```
ctx.clearRect(0, 0, 200, 200)
```

Drawing the Rectangle

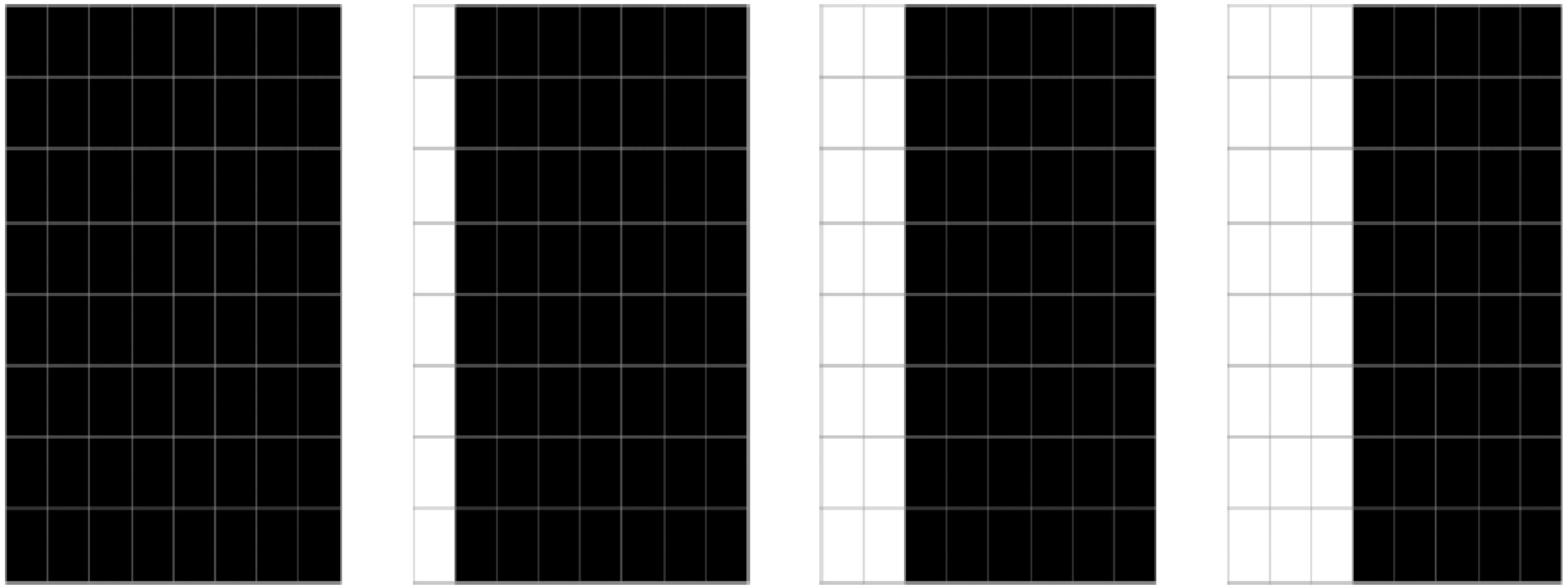
```
ctx.fillRect(position, 0, 20, 20)
```

Changing the Position

```
if (position > 200)
```

Viewing the Animation in the Browser

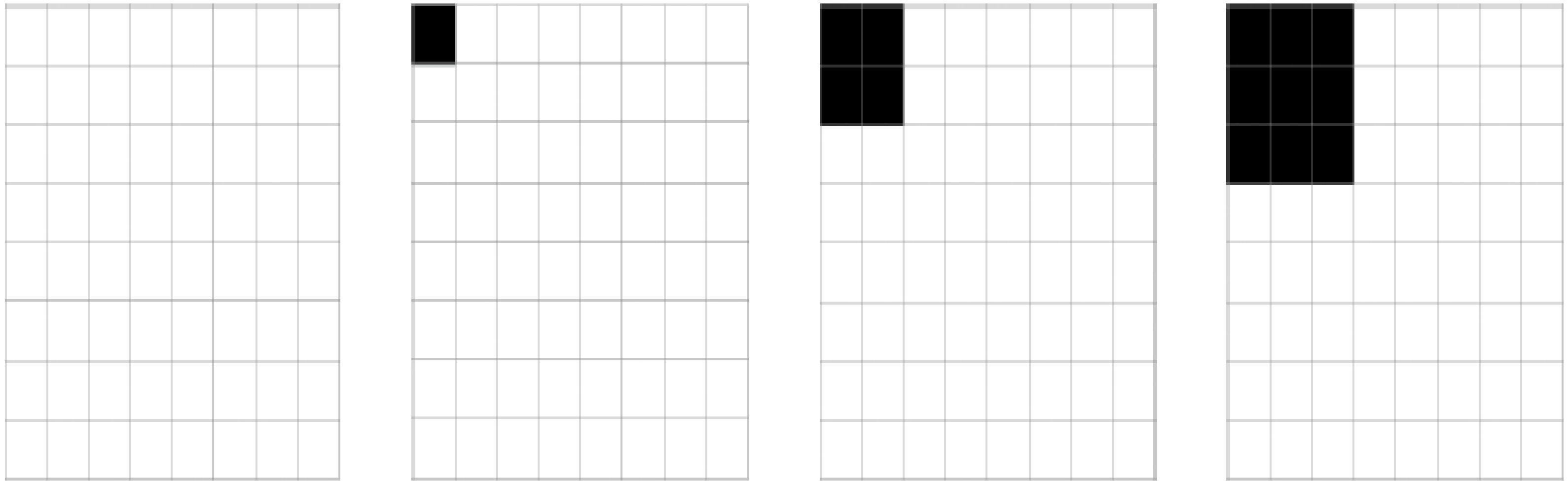
A close-up of the top-left corner of the canvas for the first four steps of the animation. At each step, position is incremented by 1 and the square moves 1 pixel to the right



Animating the Size of a Square

```
var size = 0;
ctx.fillRect(0, 0, size, size);
   size++;
   if (size > 200) {
   size = 0;
```

In each step of this animation, size is incremented by 1 and the width and height of the square grow by 1 pixel



A Random Bee

- A New circle Function

```
var circle = function (x, y, radius, fillCircle) {
ctx.beginPath();
u ctx.arc(x, y, radius, 0, Math.PI * 2, false);
v if (fillCircle) {
w ctx.fill();
} else {
x ctx.stroke();
}
};
```

Drawing the Bee

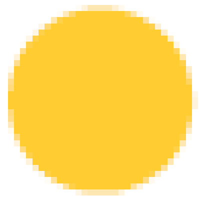
```
var drawBee = function (x, y) {
ctx.lineWidth = 2;
ctx.strokeStyle = "Black";
ctx.fillStyle = "Gold";
circle(x, y, 8, true);
circle(x, y, 8, false);
circle(x - 5, y - 11, 5, false);
circle(x + 5, y - 11, 5, false);
circle(x - 2, y - 1, 2, false);
circle(x + 2, y - 1, 2, false);
};
```

- set the lineWidth,strokeStyle, and fillStyle properties for our drawing
- set the lineWidth to 2 pixels and the strokeStyle to Black. This means that our outlined circles.
- The fillStyle is set to Gold, which will fill the circle for bee body with a nice yellow color.

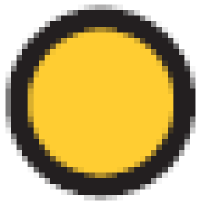
- The first circle draws the bee's body using a filled circle with a center at the point (x, y) and a radius of 8 pixels.

```
circle(x, y, 8, true);
```

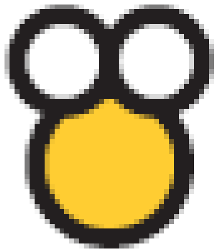
- set the fillStyle to Gold, this circle will be filled in with yellow like so:



- This second circle draws a black outline around the bee's body that's the same size and in the same place as the first circle
- circle(x, y, 8, false);
- Added to the first circle, it looks like this



- circle(x - 5, y - 11, 5, false);
- circle(x + 5, y - 11, 5, false);



Updating the Bee’s Location

```
var update = function (coordinate) {
var offset = Math.random() * 4 - 2;
coordinate += offset;
 if (coordinate > 200) {
coordinate = 200;
}
 if (coordinate < 0) {
coordinate = 0;
}
 return coordinate;
};
```

Updating the Bee’s Location

- Changing the Coordinate with an offset Value
- Checking If the Bee Reaches the Edge
- Returning the Updated Coordinate

```
x = update(x);
y = update(y);
```

Animating Our Buzzing Bee

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var x = 100;
var y = 100;
setInterval(function () {
ctx.clearRect(0, 0, 200, 200);
drawBee(x, y);
x = update(x);
y = update(y);
ctx.strokeRect(0, 0, 200, 200);
}, 30);
```

- To clear the canvas .
- Draw the bee at the point (x, y). The first time the function is called, the bee is drawn at the point (100, 100)
- The update function takes a number, adds a random number between –2 and 2 to it, and returns that updated number. So the code x = update(x) basically means “change x by a small, random amount.”

- call strokeRect at x to draw a line around the edge of the canvas.
- This makes it easier for us to see when the bee is getting close to it. Without the border, the edge of the canvas is invisible.

Bouncing a Ball!

- make a ball that bounces around the canvas

<u>ball.html</u>

```
<!DOCTYPE html>
<html><head><title>A Bouncing Ball</title></head>
<body>
<canvas id="canvas" width="200"
    height="200"></canvas>
<script>
// We'll fill this in next
</script></body></html>
```

The Ball Constructor

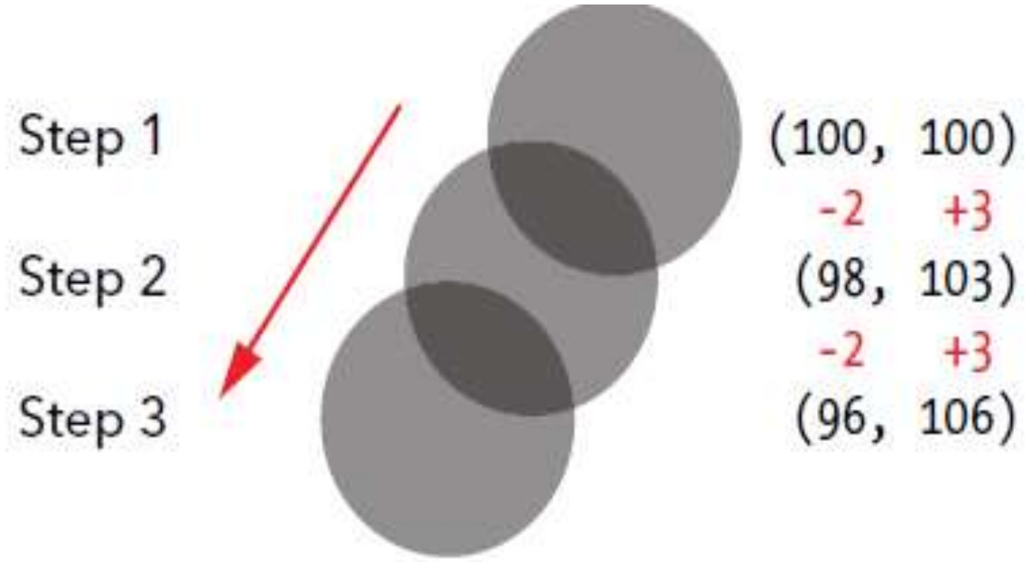
```
var Ball = function () {
this.x = 100;
this.y = 100;
this.xSpeed = -2;
this.ySpeed = 3;
};
```

Drawing the Ball

```
var circle = function (x, y, radius, fillCircle) {
ctx.beginPath();
ctx.arc(x, y, radius, 0, Math.PI * 2, false);
if (fillCircle) {
ctx.fill();
} else {
ctx.stroke();
}
};
Ball.prototype.draw = function () {
circle(this.x, this.y, 3, true);
};
```

Moving the Ball

```
Ball.prototype.move = function () {
this.x += this.xSpeed;
this.y += this.ySpeed;
};
```



The first three steps of the animation, showing how the x and y properties change

Bouncing the Ball

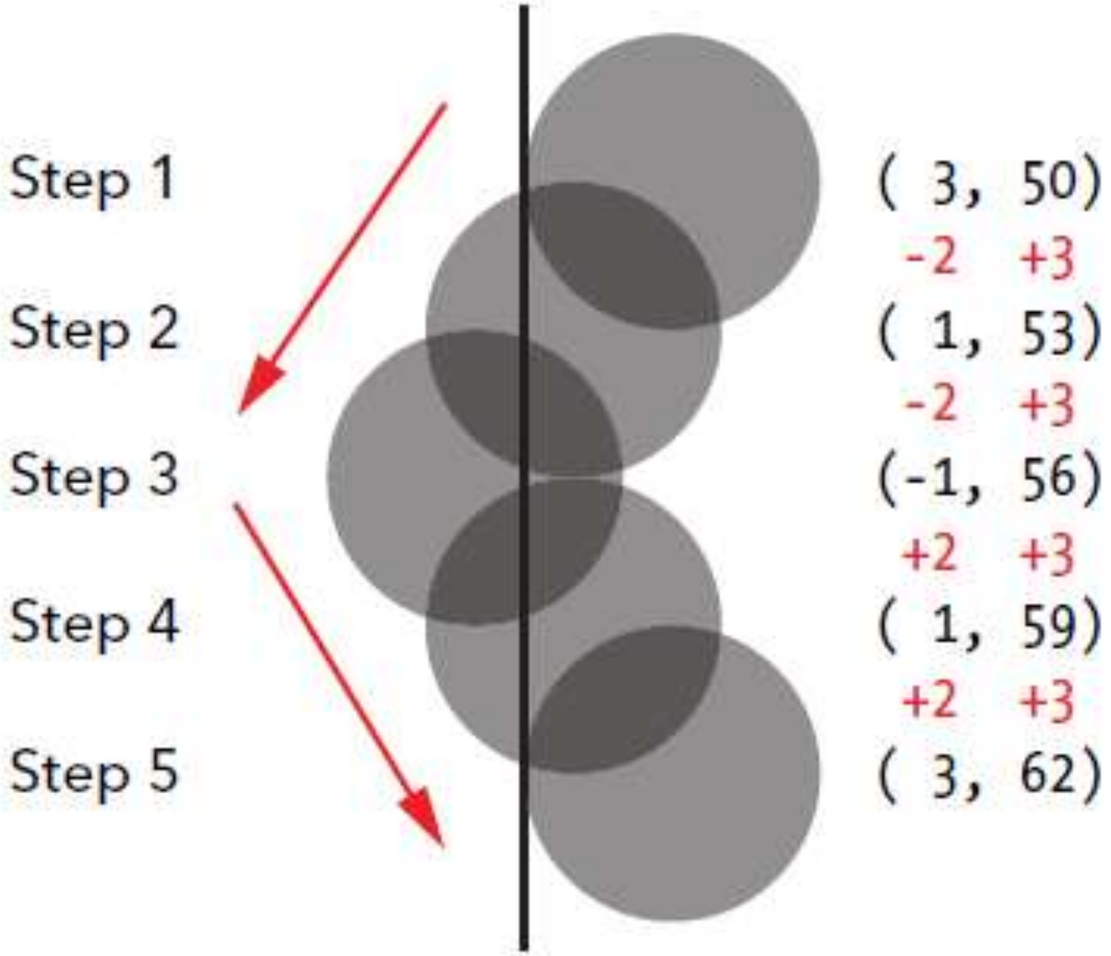
- At every step of the animation, we check to see if the ball has hit one of the walls.
- If it has, we update the xSpeed or ySpeed property by *negating it (multiplying it by –1).*

```
Ball.prototype.checkCollision = function () {
if (this.x < 0 || this.x > 200) {
		this.xSpeed = -this.xSpeed;
}
if (this.y < 0 || this.y > 200) {
		this.ySpeed = -this.ySpeed;
}
};
```

Bouncing the Ball

- determine whether the ball has hit the left wall or the right wall by checking to see if its x property is either less than 0 (meaning it hit the left edge) or greater than 200 (meaning it hit the right edge).
- If either of these is true, the ball has started to move off the edge of the canvas, so we have to reverse its horizontal direction. We do this by setting this.xSpeed equal to -this.xSpeed. For example, if this.xSpeed was -2 and the ball hit the left wall, this.xSpeed would become 2.

How this.xSpeed changes after a collision with the left wall



Animating the Ball

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var ball = new Ball();
setInterval(function () {
ctx.clearRect(0, 0, 200, 200);
ball.draw();
ball.move();
ball.checkCollision();
ctx.strokeRect(0, 0, 200, 200);
}, 30);
```

Programming Challenges

- #1: Bouncing the Ball Around a Larger Canvas

  Our 200-by-200-pixel canvas is a bit small. What if you wanted to increase the canvas size to 400 by 400 pixels or some other arbitrary size? Instead of entering the width and height of the canvas manually throughout your program, you can create width and height variables and set the variables using the canvas object. Use the following code:

- var width = canvas.width;
- var height = canvas.height;

Now if you use these variables throughout your program, you only have to change the properties on the canvas element in the HTML if you want to try out a new size. Try changing the size of the canvas to 500 pixels by 300 pixels. Does your program still work?

#2: Randomizing this.xSpeed and this.ySpeed

- To make the animation more interesting, set this.xSpeed and this.ySpeed to different random numbers (between –5 and 5) in the Ball constructor.

#3: Animating More Balls

- Instead of creating just one ball, create an empty array of balls, and use a for loop to add 10 balls to the array. Now, in the setInterval function, use a for loop to draw, move, and check collisions on each of the balls.

#4: Making the Balls Colorful

- How about making some colored bouncing balls? Set a new property in the Ball constructor called color and use it in the draw method. Use the pickRandomWord function from Chapter 8 to give each ball a random color from this array:
- var colors = ["Red", "Orange", "Yellow", "Green", "Blue", "Purple"];

THANK YOU

UNIT V

GAME DEVELOPMENT

GAME DEVELOPMENT

Making a Snake Game

Building the Block Constructor

Define a Block constructor that will create objects that represent individual blocks on our invisible game grid.

Each block will have the properties col (short for *column) and row,* which will store the location of that particular block on the grid.

The block containing the green apple is at column 10, row 10. The head of the snake (to the left of the apple) is at column 8, row 10.

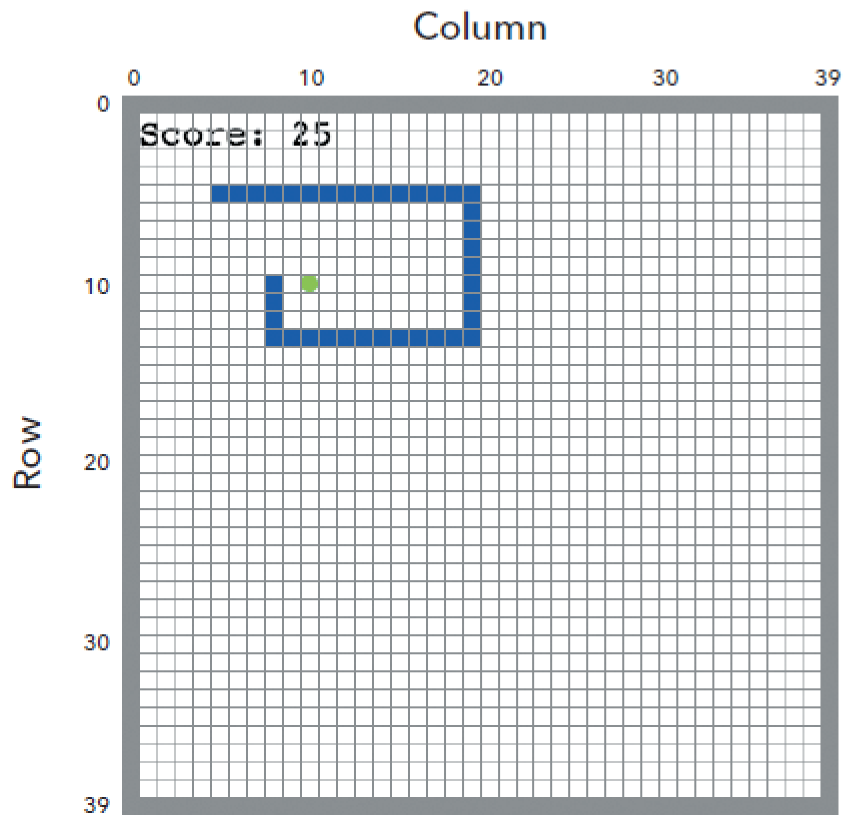


Figure 17-1: The column and row numbers used by the Block constructor

code for the Block constructor

```
var Block = function (col, row) {
this.col = col;
this.row = row;
};
```

```
  var sampleBlock = new Block(5, 5);
```

Adding the drawSquare Method

```
Block.prototype.drawSquare = function (color) {
u var x = this.col * blockSize;
v var y = this.row * blockSize;
ctx.fillStyle = color;
ctx.fillRect(x, y, blockSize, blockSize);
};
```

```
var sampleBlock = new Block(3, 4);
sampleBlock.drawSquare("LightBlue");
```

This square drawn on the canvas and how the measurements for the square are calculated.

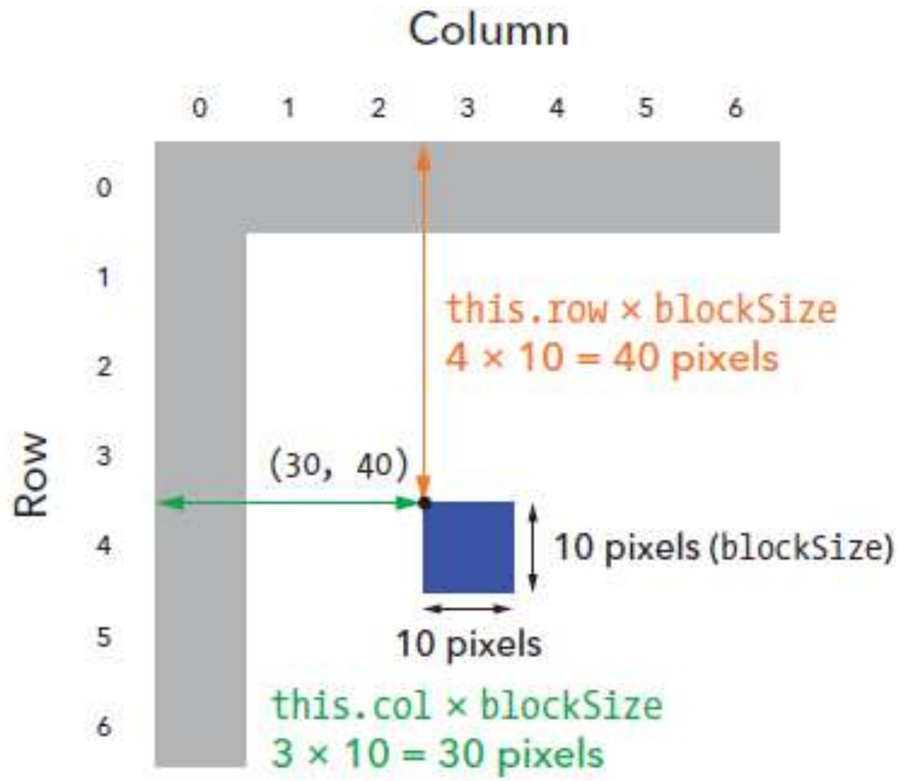


Figure 17-2: Calculating the values for drawing a square

Adding the drawCircle Method

```
Block.prototype.drawCircle = function (color) {
var centerX = this.col * blockSize + blockSize / 2;
var centerY = this.row * blockSize + blockSize / 2;
ctx.fillStyle = color;
circle(centerX, centerY, blockSize / 2, true);
};
```

```
var sampleCircle = new Block(4, 3);
sampleCircle.drawCircle("LightGreen");
```

Figure 17-3 shows the circle, with the calculations for the center point and radius.

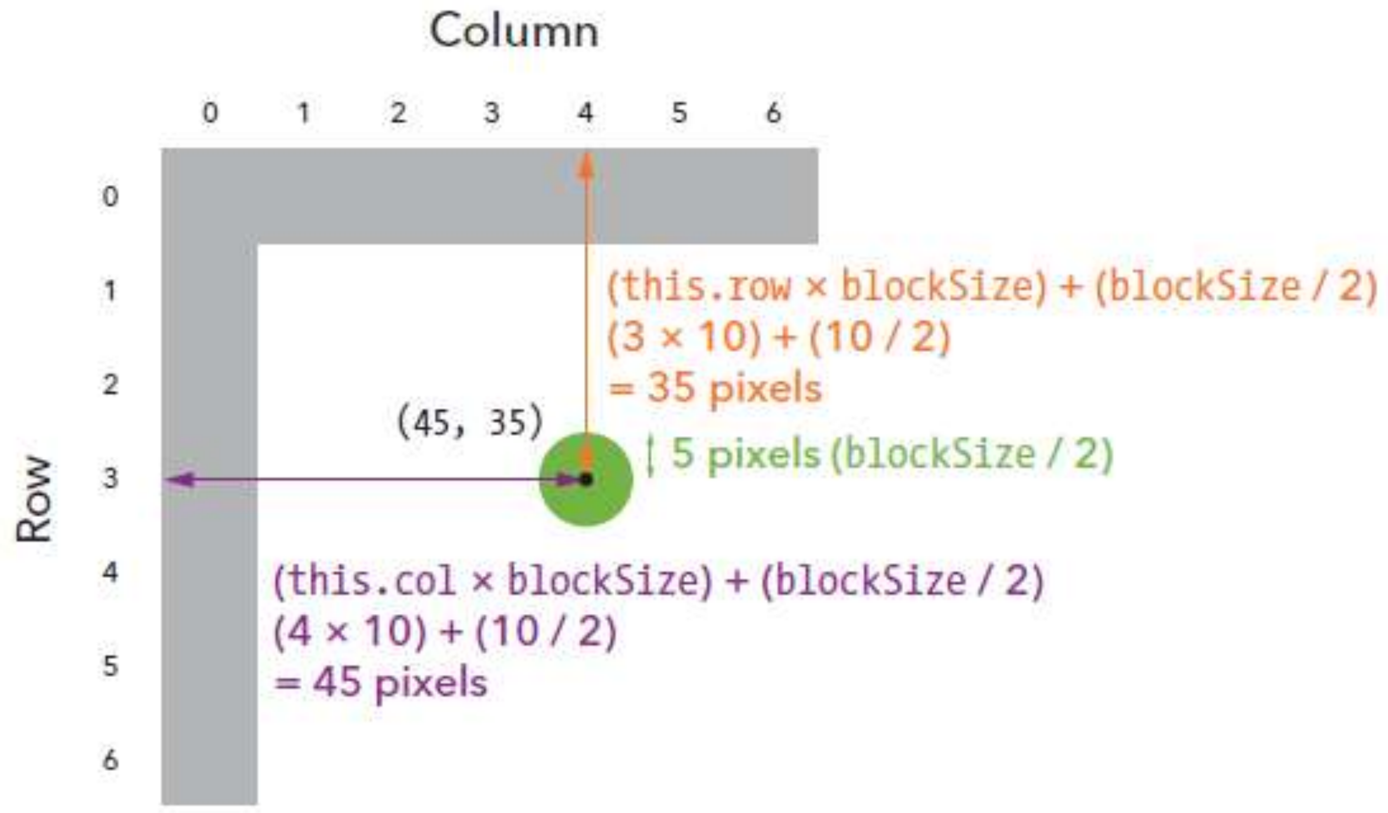


Figure 17-3: Calculating the values for drawing a circle

Adding the equal Method

```
Block.prototype.equal = function (otherBlock) {
return this.col === otherBlock.col && this.row ===
    otherBlock.row;
};
```

create two new blocks called apple and head and see if they’re in the same location

```
var apple = new Block(2, 5);
var head = new Block(3, 5);
head.equal(apple);
False

head = new Block(2, 5);
head.equal(apple);
true
```

Creating the Snake

- Store the snake’s position as an array called segments, which will contain a series of block objects.
- To move the snake, we’ll add a new block to the beginning of the segments array and remove the block at the end of the array.
- The first element of the segments array will represent the head of the snake.

Writing the Snake Constructor

```
var Snake = function () {
this.segments = [
new Block(7, 5),
new Block(6, 5),
new Block(5, 5)
];
 this.direction = "right";
 this.nextDirection = "right";
};
```

Defining the Snake Segments

- The segments property at u is an array of block objects that each represent a segment of the snake’s body. When we start the game, this array will contain three blocks at (7, 5), (6, 5), and (5, 5).

initial three segments of the snake

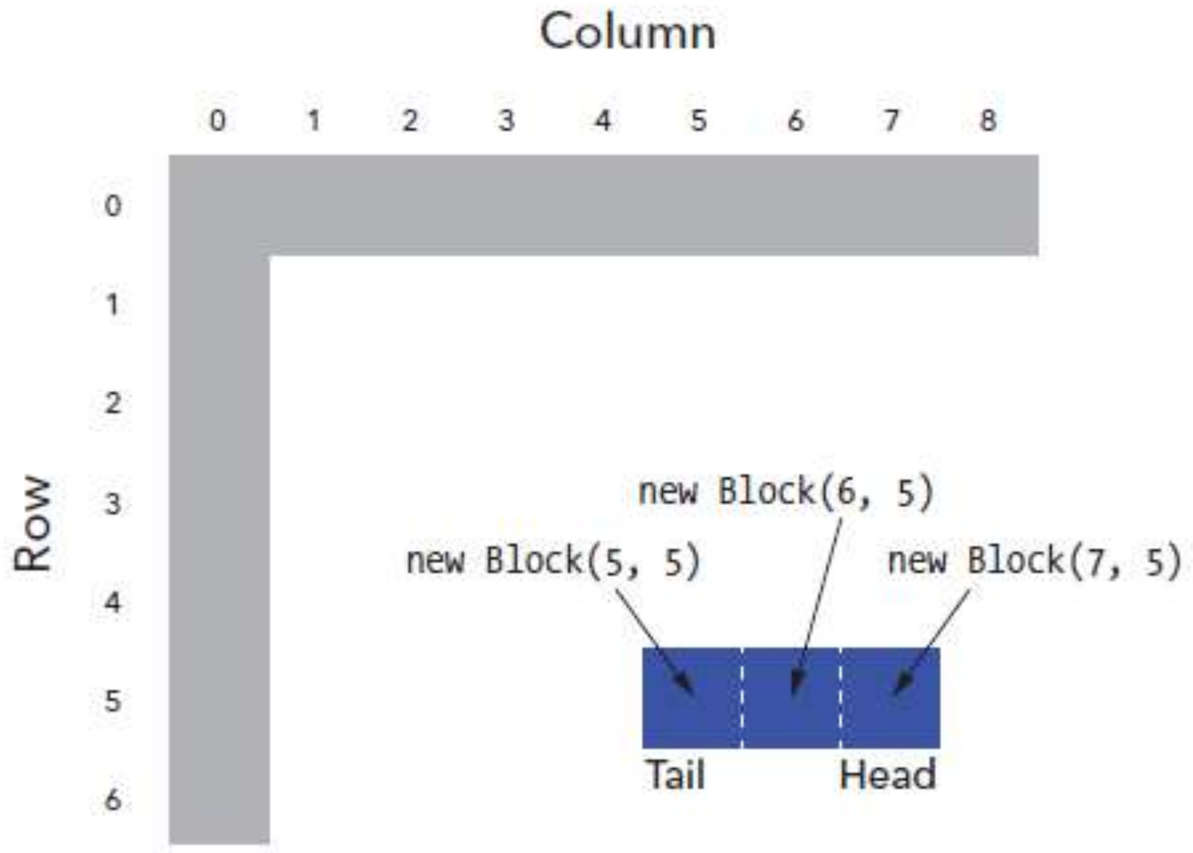


Figure 17-4: The initial blocks that make up the snake

Setting the Direction of Movement

Drawing the Snake

- The direction property at v stores the current direction of the snake. Our constructor also adds the nextDirection property at, which stores the direction in which the snake will move for the next animation step.
- This property will be updated by our keydown event handler when the player presses an arrow key
- For now, the constructor sets both of these properties to "right", so at the beginning of the game our snake will move to the right.

Drawing the Snake

```
Snake.prototype.draw = function () {
for (var i = 0; i < this.segments.length; i++) {
this.segments[i].drawSquare("Blue");
}
};
```

```
var snake = new Snake();
snake.draw();
```

Moving the Snake

- create a move method to move the snake one block in its current direction. To move the snake, we add a new head segment (by adding a new block object to the beginning of the segments array) and then remove the tail segment from the end of the segments array.

- The move method will also call a method, checkCollision, to see whether the new head has collided with the rest of the snake or with the wall, and whether the new head has eaten the apple. If the new head has collided with the body or the wall, we end the game by calling the gameOver function.

Adding the move Method

```
Snake.prototype.move = function () {
 var head = this.segments[0];
 var newHead;
 this.direction = this.nextDirection;
 if (this.direction === "right") {
newHead = new Block(head.col + 1, head.row);
}
```

Adding the move Method contd..

```
else if (this.direction === "down") {
newHead = new Block(head.col, head.row + 1);
} else if (this.direction === "left") {
newHead = new Block(head.col - 1, head.row);
} else if (this.direction === "up") {
newHead = new Block(head.col, head.row - 1);
}
```

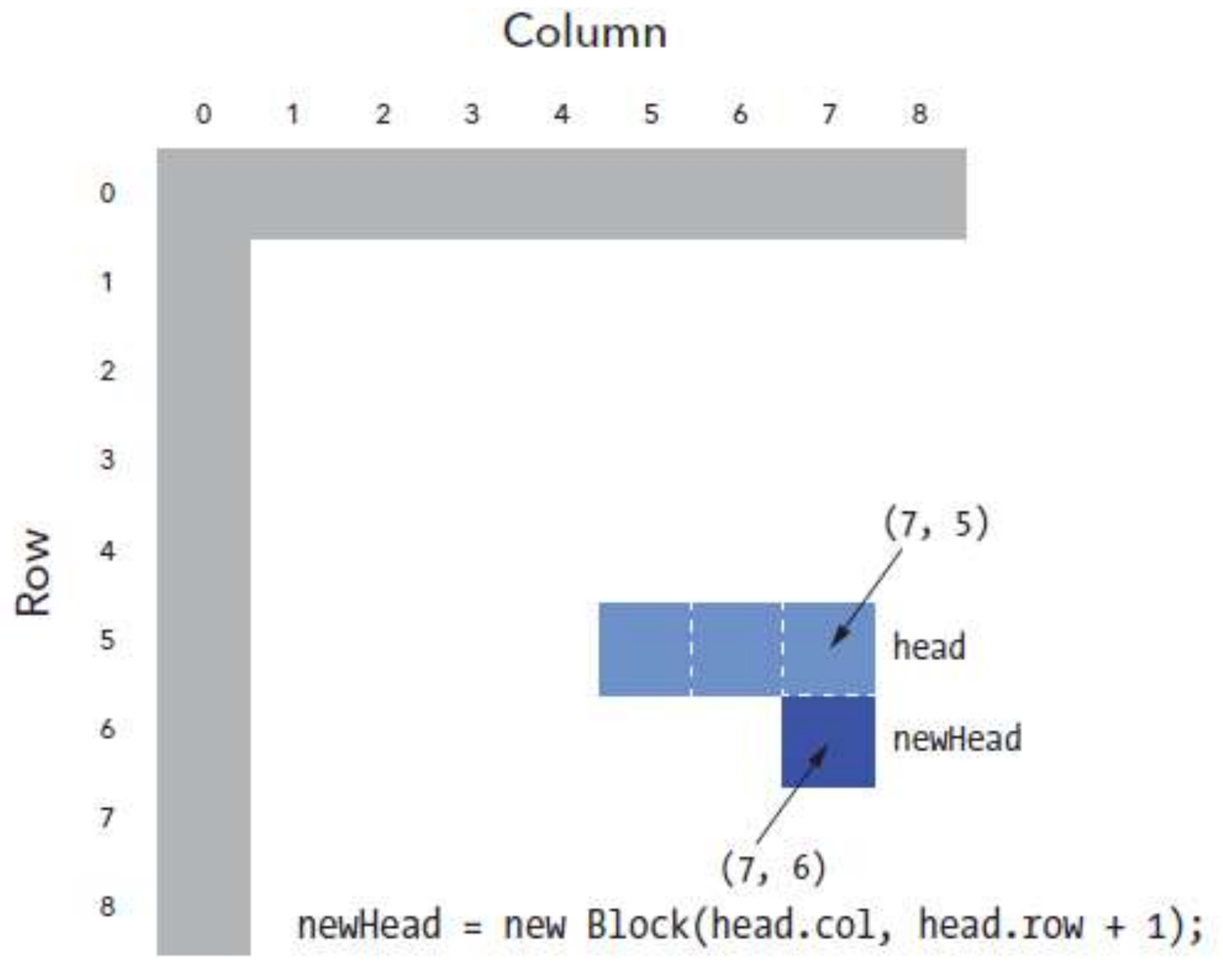
Adding the move Method contd..

```
if (this.checkCollision(newHead)) {
gameOver();
return;
}
z this.segments.unshift(newHead);
{ if (newHead.equal(apple.position)) {
score++;
apple.move();
} else {
this.segments.pop();
}    };
```

Creating a New Head

- save the first element of the this.segments array in the variable head.
- create the variable newHead, which we’ll use to store the block representing the new head of the snake.
- set this.direction equal to this.nextDirection, which updates the direction of the snake’s movement to match the most recently pressed arrow key.

Creating newHead when this.nextDirection is "down"



```
newHead = new Block(head.col, head.row + 1);
```

Figure 17-5: Creating newHead when this.nextDirection is "down"

Checking for Collisions and Adding the Head

- call the checkCollision method to find out whether the snake has collided with a wall or with itself.
- The code for this method in a moment, but as you might guess, this method will return true if the snake has collided with something. If that happens, the body of the if statement calls the gameOver function to end the game and print “Game Over” on the canvas.

- The return keyword that follows the call to gameOver exits the move method early, skipping any code that comes after it. We reach the return keyword only if heckCollision returns true, so if the snake hasn’t collided with anything, we execute the rest of the method.

- As long as the snake hasn’t collided with something, we add the new head to the front of the snake at by using unshift to add newHead to the beginning of the segments array. For more about how the unshift method works on arrays, see “Adding Elements to an Array”

Eating the Apple

- use the equal method to compare newHead and apple.position. If the two blocks are in the same location, the equal method will return true, which means that the snake has eaten the apple.

Eating the Apple CONTD…

- If the snake has eaten the apple, we increase the score and then call move on the apple to move it to a new location. If the snake has not eaten the apple, we call pop on this.segments.
- This removes the snake’s tail while keeping the snake the same size (since move already added a segment to the snake’s head).
- When the snake eats an apple, it grows by one segment because we add a segment to its head without removing the tail.

Adding the checkCollision Method

- this.segments.pop();
- Each time we set a new location for the snake’s head, we have to
- check for collisions. Collision detection, a very common step in
- game mechanics, is often one of the more complex aspects of game
- programming. Fortunately, it’s relatively straightforward in our
- Snake game.

- We care about two types of collisions in our Snake game: collisions
- with the wall and collisions with the snake itself. A wall
- collision happens if the snake hits a wall. The snake can collide
- with itself if you turn the head so that it runs into the body. At the
- start of the game, the snake is too short to collide with itself, but
- after eating a few apples, it can.

checkCollision method

```
Snake.prototype.checkCollision = function (head) {
u var leftCollision = (head.col === 0);
var topCollision = (head.row === 0);
var rightCollision = (head.col === widthInBlocks - 1);
var bottomCollision = (head.row === heightInBlocks -
    1);
v var wallCollision = leftCollision || topCollision || ⍰
rightCollision || bottomCollision;
var selfCollision = false;
for (var i = 0; i < this.segments.length; i++) {
if (head.equal(this.segments[i])) {
y selfCollision = true;
}
}
z return wallCollision || selfCollision;
};
```

Checking for Wall Collisions

- we create the variable leftCollision and set it to the value of
- head.col === 0. This variable will be true if the snake collides with
- the left wall; that is, when it is in column 0. Similarly, the variable
- topCollision in the next line checks the row of the snake’s head to
- see if it has run into the top wall.

- After that, we check for a collision with the right wall by
- checking whether the column value of the head is equal to
- widthInBlocks - 1. Since widthInBlocks is set to 40, this checks
- whether the head is in column 39, which corresponds to the right
- wall, as you can see back in Figure 17-1. Then we do the same
- thing for bottomCollision, checking whether the head's row property
- is equal to heightInBlocks - 1.

- At v, we determine whether the snake has collided with a wall
- by checking to see if leftCollision *or topCollision or rightCollision*
- *or bottomCollision is true, using the || (or) operator. We save the*
- Boolean result in the variable wallCollision.

Checking for Self-Collisions

- To determine whether the snake has collided with itself, we create
- a variable at w called selfCollision and initially set it to false. Then
- at x we use a for loop to loop through all the segments of the snake
- to determine whether the new head is in the same place as any
- segment, using head.equal(this.segments[i]).

- The head and all of the other segments are blocks, so we can use the equal method that we defined for block objects to see whether they are in the same place.
- If we find that any of the snake's segments are in the same place as the new head, we know that the snake has collided with itself, and we set selfCollision to true (at y). Finally, at z, we return wallCollision || selfCollision, which will be true if the snake has collided with either the wall or itself.

Setting the Snake’s Direction with the Keyboard

- write the code that lets the player set the snake’s direction using the keyboard. We’ll add a keydown event handler to detect when an arrow key has been pressed, and we’ll set the snake’s direction to match that key.

Adding the keydown Event Handler

```
var directions = {
37: "left",
38: "up",
39: "right",
40: "down"
};
```

keyboard events

```
$("body").keydown(function (event) {
var newDirection = directions[event.keyCode];
 if (newDirection !== undefined) {
snake.setDirection(newDirection);
}
});
```

- At u we create an object to convert the arrow keycodes into
- strings indicating the direction they represent (this object is
- quite similar to the keyActions object we used in “Reacting to the
- Keyboard” on page 244). At v we attach an event handler to the
- keydown event on the body element. This handler will be called when
- the userpresses a key (as long as they’ve clicked inside the web
- page first).

- This handler first converts the event’s keycode into a direction
- string, and then it saves the string in the variable newDirection. If
- the keycode is not 37, 38, 39, or 40 (the keycodes for the arrow keys
- we care about), directions[event.keyCode] will be undefined.

- At w we check to see if newDirection is not equal to undefined.
- If it’s not undefined, we call the setDirection method on the snake,
- passing the newDirection string. (Because there is no else case in
- this if statement, if newDirection is undefined, then we just ignore
- the keypress.)

Adding the setDirection Method

- The setDirection method takes the new direction from the keyboard
- handler we just looked at and uses it to update the snake’s
- direction. This method also prevents the player from making
- turns that would have the snake immediately run into itself.

- For
- example, if the snake is moving right, and then it suddenly turns
- left without moving up or down to get out of its own way, it will
- collide with itself. We’ll call these *illegal turns because we do not*
- want to allow the player to make them. For example, Figure 17-6
- shows the valid directions and the one illegal direction when the
- snake is moving right.

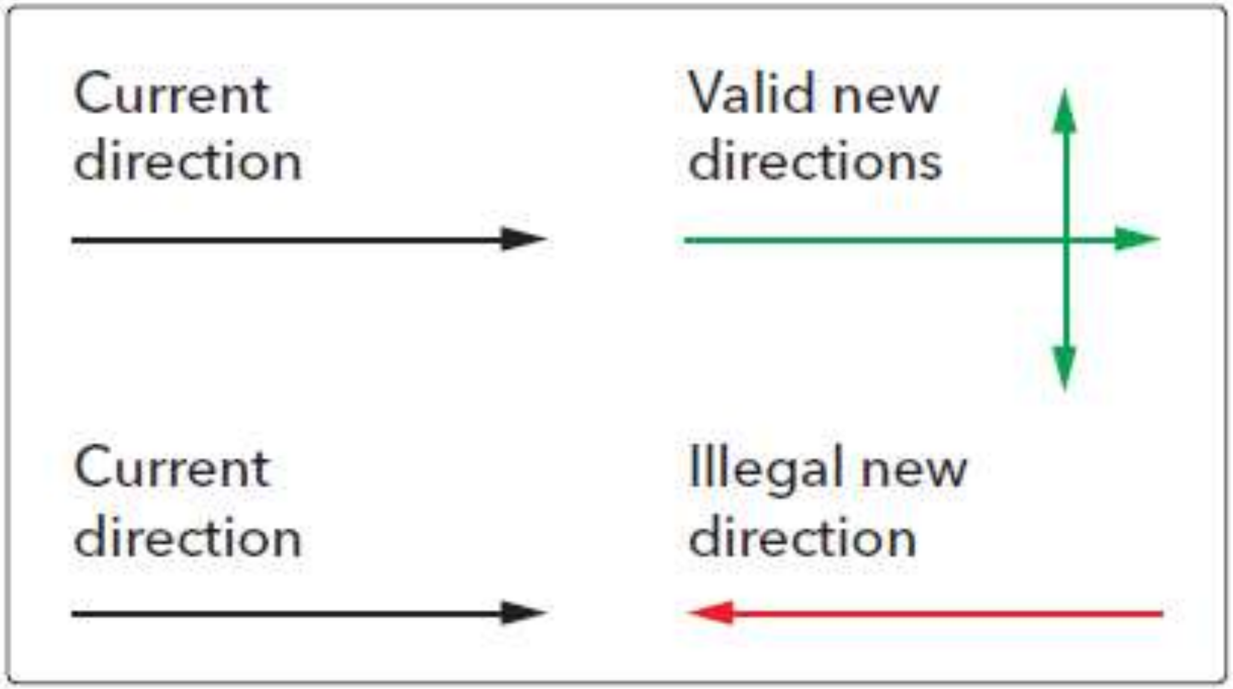


Figure 17-6: Valid new directions based on the current direction

- The setDirection method checks whether the player is trying
- to make an illegal turn. If they are, the method uses return to end
- early; otherwise, it updates the nextDirection property on the snake
- object.

code for the setDirection method

```
Snake.prototype.setDirection = function (newDirection) {
u if (this.direction === "up" && newDirection === "down") {
return;
} else if (this.direction === "right" && newDirection === "left") {
return;
} else if (this.direction === "down" && newDirection === "up") {
return;
}
else if (this.direction === "left" && newDirection
    === "right") {
return;
}
this.nextDirection = newDirection;
};
```

- The if...else statement at u has four parts to deal with the
- four illegal turns we want to prevent. The first part says that
- if the snake is moving up (this.direction is "up") and the player
- presses the down arrow (newDirection is "down"), we should exit the
- method early with return.

- The other parts of the statement deal
- with the other illegal turns in the same way.
- The setDirection method will reach the final line only if
- newDirection is a valid new direction; otherwise, one of the return
- statements will stop the method.
- If newDirection *is allowed, we set it as the snake's nextDirection*
- property,

Creating the Apple

- the apple as an object with three components:
- a position property, which holds the apple’s position as a
- block object; a draw method, which we’ll use to draw the apple; and
- a move method, which we’ll use to give the apple a new position once
- it’s been eaten by the snake.

Writing the Apple Constructor

- The constructor simply sets the apple’s position property to a new block object.

```
var Apple = function () {
this.position = new Block(10, 10);
};
```

- This creates a new block object in column 10, row 10, and
- assigns it to the apple’s position property. We’ll use this constructor
- to create an apple object at the beginning of the game.

Drawing the Apple

- Apple.prototype.draw = function () {
- this.position.drawCircle("LimeGreen");
- };

- The apple’s draw method is very simple, as all the hard work is
- done by the drawCircle method (created in “Adding the drawCircle
- Method” on page 270). To draw the apple, we simply call the
- drawCircle method on the apple’s position property, passing the
- color "LimeGreen" to tell it to draw a green circle in the given block.

- var apple = new Apple();
- apple.draw();

Moving the Apple

- The move method moves the apple to a random new position within
- the game area (that is, any block on the canvas other than the
- border). We’ll call this method whenever the snake eats the apple
- so that the apple reappears in a new location.

```
Apple.prototype.move = function () {
 var randomCol = Math.floor(Math.random() *
   (widthInBlocks - 2)) + 1;
var randomRow = Math.floor(Math.random() *
   (heightInBlocks - 2)) + 1;
this.position = new Block(randomCol,
   randomRow);
};
```

- we create the variables randomCol and randomRow. These
- variables will be set to a random column and row value within the
- playable area. As you saw in Figure 17-1, the columns and rows for
- the playable area range from 1 to 38, so we need to pick two random
- numbers in that range.

- To generate these random numbers, we can call Math.floor
- (Math.random() * 38), which gives us a random number from 0 to 37,
- and then add 1 to the result to get a number between 1 and 38
- (for more about how Math.floor and Math.random work,

- This is exactly what we do at u to create our random column
- value, but instead of writing 38, we write (widthInBlocks - 2). This
- means that if we later change the size of the game, we won’t also
- have to change this code. We do the same thing to get a random
- row value, using Math.floor(Math.random() * (heightInBlocks - 2)) + 1.

- Finally, at v we create a new block object with our random
- column and row values and save this block in this.position. This
- means that the position of the apple will be updated to a new random
- location somewhere within the playing area.

test out the move method

- var apple = new Apple();
- apple.move();
- apple.draw();