

Loops

At the end of this chapter, you will be able to:

- Use the For Loop
- Use the While Loop

Instruction 1: Use the For Loop

1. JavaScript reads and executes codes top down, one line at a time. To tell JavaScript to repeatedly execute a group of codes continuously until a condition is met, you apply the concept of looping.
2. The For Loop is one way to do it:

```
for (Statement1; Statement2; Statement3) {
    code to execute
}
```

- a. Statement1 is executed before the loop begins
 - b. Statement2 states the condition where the loop will continuously execute.
 - c. Statement3 is executed each time the “code to execute” section runs.
3. We will create a simple Cordova program to illustrate the use of a For Loop.
 4. Create a new project, C9.
 5. Follow Instruction1 in Chapter 4 to include jQuery and jQuery Mobile libraries into your project.
 6. Enter the following codes in index.js:

```
1  (function () {
2
3      $(document).ready(function () {
4
5      });
6
7      function looptest() {
8          var endNumber;
9          endNumber = 5;
10
11         for (i = 0; i <= endNumber; i++) {
12             alert(i);
13         }
14     }
15
16     looptest();
17
18 })();
```

- a. In line 8, we define endNumber as a variable.
- b. We assign the value 5 to endNumber in line 9.

- c. In line 11, JavaScript repeatedly execute line 12 by counting *i* from 0 to *endNumber* and incrementing *i* by 1 one each round in the loop.
- d. The statement on line 12 pops *i* in an alert on each round in the loop.

7. Here is the result of running this program: Alerts pop with the values 0 to 5.



8. We will make a minor change to the value of *i* in the for loop:

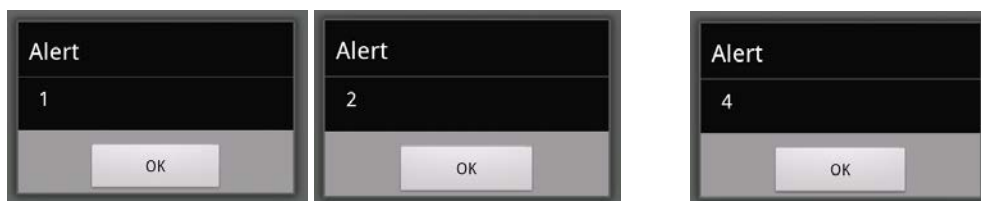
```

1      function looptest(){
2          var endNumber;
3          endNumber = 5;
4
5          for (i=1;i<endNumber;i++){
6              alert(i);
7          }
8      }
9
10     looptest();

```

- a. The value of *i* has been changed from 0 in the previous example to 1.
- b. The loop has been changed to execute as long as *i* is smaller than *endNumber* (5).

9. Here is the result of running this program: Alert pop with the values 1 to 4.



10. We will develop that will count numbers based on what the user enters. Design the following user interface:

 A user interface form is shown. It has a label 'From:' followed by a text input field containing the number '2'. Below that is a label 'To:' followed by a text input field containing the number '3'. At the bottom of the form is a button labeled 'Add'.

11. Based on the values 2 and 5 entered, this program will pop the values 2 to 5 in alert boxes.

12. Edit index.html:

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <!--
5       Customize the content security policy in the meta tag below as
6       needed. Add 'unsafe-inline' to default-src to enable inline JavaScript.
7       For details, see http://go.microsoft.com/fwlink/?LinkID=617521
8     -->
9     <meta http-equiv="Content-Security-Policy" content="default-src
10      'self' data: gap: https://ssl.gstatic.com 'unsafe-eval'; style-src
11      'self' 'unsafe-inline'; media-src *">
12
13     <meta http-equiv="content-type" content="text/html; charset=UTF-
14 8" />
15
16     <meta name="format-detection" content="telephone=no">
17     <meta name="msapplication-tap-highlight" content="no">
18     <meta name="viewport" content="user-scalable=no, initial-
19 scale=1, maximum-scale=1, minimum-scale=1, width=device-width">
20
21     <link rel="stylesheet" type="text/css" href="css/index.css">
22     <link rel="stylesheet" href="css/jquery.mobile-1.4.5.css">
23
24     <script src="lib/jquery-1.11.2.min.js"></script>
25     <script src="lib/jquery.mobile-1.4.5.min.js"></script>
26     <script src="scripts/common.js"></script>
27
28     <title>C9</title>
29   </head>
30   <body>
31
32     <div data-role="main" class="ui-content">
33       <form name="calculateform" id="calculateform">
34         <div class="ui-field-contain">
35
36           <div data-role="fieldcontainer">
37             <label for="txtFrom">From:</label>
38             <input type="text" name="txtFrom" id="txtFrom"
39 value="2">
40
41           </div>
42
43           <div data-role="fieldcontainer">
44             <label for="txtTo">To:</label>
45             <input type="text" name="txtTo" id="txtTo"
46 value="3">
47
48           </div>
49
50           <input type="button" value="Add" id="btnAdd">
51
52         </div>
53       </form>
54     </div>
55
56     <script type="text/javascript" src="cordova.js"></script>
57     <script type="text/javascript"

```

```

src="scripts/platformOverrides.js"></script>
46     <script type="text/javascript" src="scripts/index.js"></script>
47     </body>
48 </html>

```

13. Edit index.js:

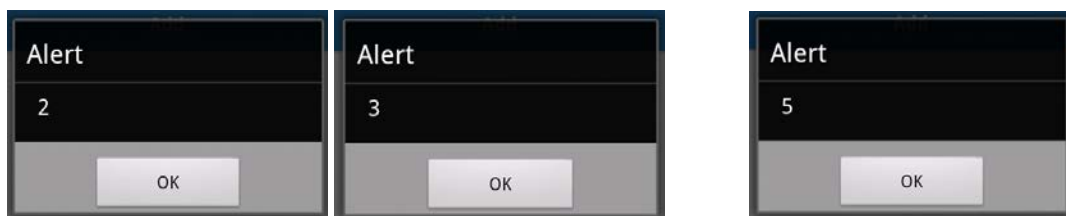
```

1  (function () {
2
3      $(document).ready(function () {
4          $("#btnAdd").bind("click", function () {
5              adder();
6          });
7      });
8
9      function adder() {
10         var from, to;
11         from = parseInt($("#txtFrom").val());
12         to = parseInt($("#txtTo").val());
13
14         for (i = from; i <= to; i++) {
15             alert(i);
16         }
17     }
18 })();

```

- a. Line 10 declares 2 variables, from and to.
- b. Line 11 copies the value entered in the text input declared in index.html's line 31 to the variable "from" after converting it into an integer.
- c. Line 12 copies the value entered the text input declared in line 36 to the variable "to" after converting it into an integer.
- d. Line 14 assigns the value "from" to i. In this example, the value 2 is assigned to i. Line 15 will pop the value of i in an alert box repeatedly while i is less than or equal to the value "to" which in this case is 5. Line 14 will increment i by 1 in each round of the loop.

14. Here is the result of running this program: Alerts pop with the values 2 to 5.



15. Instead of popping values in alert boxes, we will like to add all the numbers between the "from" text field to the "to" text field. If the user enters "2" for "from" and "5" for "to", we will calculate $2+3+4+5$ and pop the number 14 in an alert.

16. To do this, change the adder() function in the example to the following codes:

```

1 function adder() {
2     var from, to, total;
3
4     total = 0;
5     from = parseInt($("#txtFrom").val());
6     to = parseInt($("#txtTo").val());
7
8     for (i = from; i <= to; i++) {
9         total = total + i;
10    }
11
12    alert(total);
13 }

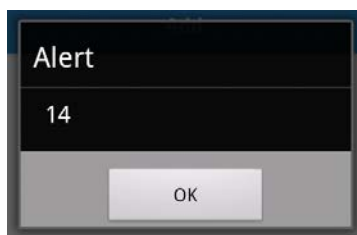
```

- a. We have added a new variable “total” in line 2.
- b. In line 4, we initialize the value of total to 0.
- c. In line 9, we add the value of i to total in each round of the loop.

17. The table below traces the values of the variables as the loop executes:

Iteration	from	to	i	total
1	2	5	2	2
2	2	5	3	5
3	2	5	4	9
4	2	5	5	14

18. The following is the result of running this program. An alert pops with the value 14.



19. We will change our program again to show the results within a set of `<div></div>` tags in the user interface instead of popping an alert. Change index.html:

```

1 <div data-role="main" class="ui-content">
2     <form name="calculateform" id="calculateform">
3         <div class="ui-field-contain">
4
5             <div data-role="fieldcontainer">
6                 <label for="txtFrom">From:</label>

```

```

7         <input type="text" name="txtFrom" id="txtFrom"
value="2">
8     </div>
9
10    <div data-role="fieldcontainer">
11        <label for="txtTo">To:</label>
12        <input type="text" name="txtTo" id="txtTo" value="3">
13    </div>
14
15    <input type="button" value="Add" id="btnAdd">
16 </div>
17 </form>
18
19 <div id="result">
20
21 </div>
22 </div>

```

- a. We added a new division “result” in line 19 and 21. The “result” division will be used in displaying the calculation results.

20. Edit index.js:

```

1 (function () {
2
3     $(document).ready(function () {
4         $("#btnAdd").bind("click", function () {
5             adder();
6         });
7     });
8
9     function adder() {
10         var from, to, total;
11
12         total = 0;
13         from = parseInt($("#txtFrom").val());
14         to = parseInt($("#txtTo").val());
15
16         for (i = from; i <= to; i++) {
17             total = total + i;
18         }
19
20         $("#result").html("Total from " + from + " to " + to + " = " +
total);
21     }
22 })();

```

- a. Line 20 adds the statement “Total from 2 to 5 = 14” within the result division. The syntax for adding HTML text to a division is as follows:

```
jQueryID.html("text to add")
```

Thus, to get the result `<div></div>` to display the text, we say:

```
$("#result").html("Total from " + from + " to " + to + " = " + total);
```

- b. The “+” operators on line 20 concatenate the values of the variables “from”, “to” and “total” to the sentence “Total from...to...= total”.

21. Here is the result of running this program:

22. Our next example is a multiplication table. It accepts 1 input from the user, “From”. If the user enters “2”, the program will write the multiplication table from 2x1...2x10.

23. Edit index.html:

```
1 <div data-role="main" class="ui-content">
2   <form name="calculateform" id="calculateform">
3     <div class="ui-field-contain">
4
5       <div data-role="fieldcontainer">
```



```

6         <label for="txtFrom">From:</label>
7         <input type="text" name="txtFrom" id="txtFrom"
value="2">
8     </div>
9
10        <input type="button" value="Multiplier" id="btnMultiply">
11    </div>
12 </form>
13
14 <div id="result">
15
16 </div>
17
18 </div>

```

24. Edit index.js:

```

1 (function () {
2
3     $(document).ready(function () {
4         $("#btnMultiply").bind("click", function () {
5             multiply();
6         });
7     });
8
9     function multiply() {
10        var from, htmlString, result;
11        from = parseInt($("#txtFrom").val());
12
13        htmlString = "";
14        for (i = 1; i <= 10; i++) {
15            result = from * i;
16            htmlString += from.toString() + " x " + i.toString() + " = "
+ result.toString() + "<br>";
17        }
18        $("#result").html(htmlString);
19    }
20
21 })();

```

- a. Variables from, htmlString and result are declared in line 10.
- b. Line 11 assigns the value entered by the user in the “txtFrom” text field to the JavaScript variable “from”.
- c. Line 13 initializes htmlString to an empty string.
- d. Line 14 runs the loop 10 times from i=1 to i=10.
- e. Line 15 multiplies “from” with “i” and assigns the value to result on each round in the loop.

- f. Line 16 forms the content of `htmlString` using the values contained in “from”, “i” and “result”.
- g. Line 18 displays the value of `htmlString` in the user interface.

25. The table below traces the values of the variables as the loop executes:

Iteration	from	i	result	htmlString
1	2	1	2	2 x 1 = 2
2	2	2	4	2 x 1 = 2 2 x 2 = 4
3	2	3	6	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6
4	2	4	8	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8
5	2	5	10	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10
6	2	6	12	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10 2 x 6 = 12
7	2	7	14	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10 2 x 6 = 12 2 x 7 = 14
8	2	8	16	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10 2 x 6 = 12 2 x 7 = 14

				2 x 8 = 16
9	2	9	18	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10 2 x 6 = 12 2 x 7 = 14 2 x 8 = 16 2 x 9 = 18
10	2	10	20	2 x 1 = 2 2 x 2 = 4 2 x 3 = 6 2 x 4 = 8 2 x 5 = 10 2 x 6 = 12 2 x 7 = 14 2 x 8 = 16 2 x 9 = 18 2 x 10 = 20

26. Run the program with various values in the “From” field and note the results.
27. We will change our codes to present a multiplication table based on 2 values that the user enters into the text field, “From” and “To”:

28. Edit index.html:

```

1 <div data-role="main" class="ui-content">
2   <form name="calculateform" id="calculateform">
3     <div class="ui-field-contain">
4
5       <div data-role="fieldcontainer">
6         <label for="txtFrom">From:</label>
7         <input type="text" name="txtFrom" id="txtFrom"
value="2">

```

```

8         </div>
9
10        <div data-role="fieldcontainer">
11            <label for="txtFrom">To:</label>
12            <input type="text" name="txtTo" id="txtTo" value="5">
13        </div>
14
15        <input type="button" value="Multiplier" id="btnMultiply">
16    </div>
17 </form>
18
19 <div id="result">
20
21 </div>
22
23 </div>

```

- a. Lines 6 and 7 declare a “Multiply from” text field.
- b. Lines 11 and 12 declare a “Multiply to” text field.

29. Edit index.js:

```

1 function multiply() {
2     var from, to, htmlString, result;
3     from = parseInt($("#txtFrom").val());
4     to = parseInt($("#txtTo").val());
5
6     htmlString = "";
7     for (j = from; j <= to; j++) {
8         for (i = 1; i <= 10; i++) {
9             result = j * i;
10            htmlString += j.toString() + " x " + i.toString() + " = " +
11            result.toString() + "<br>";
12        }
13    }
14    $("#result").html(htmlString);
15 }

```

- a. Variables “from”, “to”, “htmlString” and “result” are declared in line 12.
- b. Line 3 assigns the value entered by the user in the “from” text field to the JavaScript variable “from”.
- c. Line 4 assigns the value entered by the user in the “to” text field to the JavaScript variable “to”.
- d. Line 6 initializes htmlString to an empty string.
- e. Line 7 runs the loop based on the user’s input in the “from” and “to” text fields. In this example, it runs the loop based on j = 2 to 5.
- f. Line 8 runs a nested loop 10 times from i=1 to 10.

- g. Line 9 multiplies `j` with `i` and assigns the value to `result` on each round in the loop.
- h. Line 10 forms the content of `htmlString` using the value contained in the variables “`j`”, “`i`” and “`result`”.
- i. Line 13 displays the value of `htmlString` in the user interface.

30. The table below traces the values of the variables as the loop executes. For simplicity, only the execution of the outer loop on line 8 will be shown in detail:

Iteration	from	to	j	i	result	htmlString
1	2	5	2	1..10	2,4,6...20	Multiplication table of 2
2	2	5	3	1..10	3,6,9...30	Multiplication table of 2 Multiplication table of 3
3	2	5	4	1..10	4,8,12..40	Multiplication table of 2 Multiplication table of 3 Multiplication table of 4
4	2	5	5	1..10	5,10,15..50	Multiplication table of 2 Multiplication table of 3 Multiplication table of 4 Multiplication table of 5

31. Run the program to obtain the following results (screen capture is incomplete):

```

3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
5 x 1 = 5
5 x 2 = 10

```

Instruction 2: Use the While Loop

1. The while loop executes a block of code continuously until a specific condition is met:

```

while (condition) {
    code block to be executed
}

```

2. You may specify various conditions in a while loop using one or more of the condition operators as follows:

Meaning	Operators	Example
Equal	===	x === y
Not Equal	!==	x !== y
Less than	<	x < y
Less than or equal to	<=	x <= y
Greater than	>	x > y
Greater than or equal to	>=	x >= y
Test for more than 1 condition	&&	(x === 2) && (y === 4)
Test if value is either or something		(x === 2) (y === 4)
Test if value is not something	!	!(x === y)

3. We will create a Cordova application to illustrate the use of a while loop.
4. Edit index.html:

```

1 <div data-role="main" class="ui-content">
2
3   <div id="result">

```

```

4
5     </div>
6
7 </div>

```

5. Edit index.js:

```

1 (function () {
2
3     function looptest() {
4         var endNumber, htmlString, i;
5         endNumber = 5;
6         i = 0;
7         htmlString = "";
8         while (i <= endNumber) {
9             htmlString = htmlString + i.toString() + "<br>";
10            i++;
11        }
12        $("#result").html(htmlString);
13    }
14
15    looptest();
16
17 })();

```

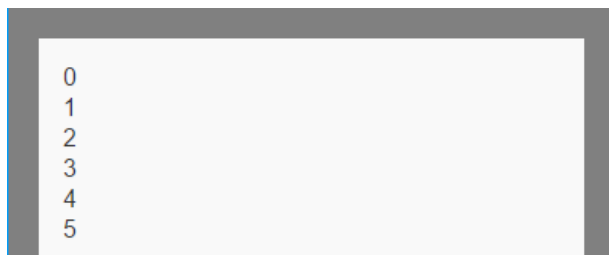
- a. Line 4 declares 3 variables, endNumber, htmlString and i.
- b. Line 5 sets endNumber to 5.
- c. Line 6 sets i to 0.
- d. Line 7 sets htmlString to be an empty string.
- e. Line 8 states that the statements in line 9 and 10 will be executed continuously as long as i is less than or equal to endNumber.
- f. Line 9 concatenates i with htmlString and adds a
 tag to break to the next line.
- g. Line 10 adds 1 to i. It is like writing $i = i + 1$.
- h. Line 12 prints the content of htmlString to the result division on index.html's line 3 to 5.

6. The table below traces the values of the variables as the loop executes:

Iteration	i	endNumber	htmlString
1	0	5	0
2	1	5	0 1

3	2	5	0
			1
			2
4	3	5	0
			1
			2
			3
5	4	5	0
			1
			2
			3
			4
6	5	5	0
			1
			2
			3
			4
			5

7. Run the program to observe its output.



8. The program may be amended to increment *i* by 2 instead of 1. The codes underlined below will increment *i* by 2. Amend your code in the previous example to do this.

```

1 function looptest() {
2     var endNumber, htmlString, i;
3     endNumber = 5;
4     i = 0;
5     htmlString = "";
6     while (i <= endNumber) {
7         htmlString = htmlString + i.toString() + "<br>";
8         i = i + 2;
9     }
10    $("#result").html(htmlString);
11 }
12
13 looptest();

```

9. The table below traces the values of the variables as the loop executes:

Iteration	i	endNumber	htmlString
1	0	5	0
2	2	5	0 2
3	4	5	0 2 4

10. Run the program to observe its output.



11. The program may be amended to decrement instead of increment *i* by 1. The codes underlined below are the changes to be made to do this. Amend your codes in the previous example to do this.

```

1  function looptest(){
2      var endNumber, htmlString, i;
3      endNumber = 0;
4      i=5;
5      htmlString = "";
6      while (i>=endNumber){
7          htmlString = htmlString + i.toString() + "<br>";
8          i--;
9      }
10     $("#result").html(htmlString);
11 }

```

- Line 3 sets endNumber to 0.
- Line 4 sets i to 5.
- Line 6 sets the condition to continuously execute the statements in line 7 and 8 while i is greater than or equal to endNumber which is set to 0.
- Line 8 deducts 1 from i. It is similar to writing $i = i - 1$.

12. This table traces the values of the variables as the loop executes:

Iteration	i	endNumber	htmlString
1	5	0	5
2	4	0	5 4
3	3	0	5 4 3
4	2	0	5 4 3 2
5	1	0	5 4 3 2 1
6	0	0	5 4 3 2 1 0

13. Run the program to observe its output.



```

5
4
3
2
1
0

```

14. The program may be amended to decrement *i* by 2 instead of 1. The code underlined below will decrement *i* by 2. Amend your code in the previous example to do this.

```

1  function looptest(){
2      var endNumber, htmlString, i;
3      endNumber = 0;
4      i=5;
5      htmlString = "";
6      while (i>=endNumber){
7          htmlString = htmlString + i.toString() + "<br>";
8          i = i - 2;

```

```

9      }
10     $("#result").html(htmlString);
11  }

```

15. The table below traces the values of the variables as the loop executes:

Iteration	i	endNumber	htmlString
1	5	0	5
2	3	0	5 3
3	1	0	5 3 1

16. Run the program to observe its output.



17. The example below illustrates a nested while loop. Amend your codes in the previous example to the codes below:

```

1  function looptest(){
2      var startNumber, endNumber, htmlString, i, j;
3      startNumber = 0;
4      endNumber = 3;
5      i = 0;
6      j = 0;
7      htmlString = "";
8      while (i<=endNumber){
9          while (j<=3){
10             htmlString = htmlString + "(" + i.toString() + "," +
11             j.toString() + ")" + "<br>";
12             j++;
13             j = 0;
14             i++;
15         }
16         $("#result").html(htmlString);
17     }

```

- a. Line 2 declares startNumber, endNumber, htmlString, i and j as variables.

- b. Line 3 sets startNumber to 0.
- c. Line 4 sets endNumber to 3.
- d. Line 5 sets i to 0.
- e. Line 6 sets j to 0.
- f. Line 7 sets htmlString to an empty string.
- g. Line 8 sets the condition where lines 9 to 14 will continuously execute while i is less than or equal to endNumber, currently set to 3.
- h. Line 9 sets the condition where lines 10 to 11 will continuously execute while j is less than or equal to 3.
- i. Line 10 forms the content of htmlString on each round of the loop, managed by the condition in line 9.
- j. Line 11 increments j by 1.
- k. Line 13 sets j to 0, in preparation for another round in the loop, managed by the condition in line 8.
- l. Line 14 increments i by 1.

18. The table below traces the values of the variables as the loop executes:

Iteration	i	startNumber	endNumber	Nested Loop		
1	0	0	3	Iteration	j	htmlString
				1	0	(0,0)
				2	1	(0,0) (0,1)
				3	2	(0,0)(0,1)(0,2)
				4	3	(0,0)(0,1)(0,2)(0,3)
2	1	0	3	Iteration	j	htmlString
				1	0	(1,0)
				2	1	(1,0) (1,1)
				3	2	(1,0)(1,1)(1,2)
				4	3	(1,0)(1,1)(1,2)(1,3)
3	2	0	3	Iteration	j	htmlString

				1	0	(2,0)
				2	1	(2,0) (2,1)
				3	2	(2,0)(2,1)(2,2)
				4	3	(2,0)(2,1)(2,2)(2,3)
4	3	0	3	Iteration	j	htmlString
				1	0	(3,0)
				2	1	(3,0) (3,1)
				3	2	(3,0)(3,1)(3,2)
				4	3	(3,0)(3,1)(3,2)(3,3)

19. Run the program to observe its output.

```
(0,0)
(0,1)
(0,2)
(0,3)
(1,0)
(1,1)
(1,2)
(1,3)
(2,0)
(2,1)
(2,2)
(2,3)
(3,0)
(3,1)
(3,2)
(3,3)
```

20. The next example illustrates the use of more than one condition to control the while loop. Design the following user interface.

Start finding from:

Find until:

Divisible by:

Find

This program accepts a range of numbers from the user and finds the first number within the range that is divisible by the user's input.

The following screen captures show the results of running the program based on the user's input:

Start finding from:

Find until:

Divisible by:

Find

First number divisible by 2 found: 2

Start finding from:

Find until:

Divisible by:

Find

First number divisible by 7 found: 7

21. Edit index.html:

```

1 <div data-role="main" class="ui-content">
2   <form name="calculateform" id="calculateform">
3
4     <div data-role="main" class="ui-content">
5       <div class="ui-field-contain">
6         <div data-role="fieldcontainer">
7           <label for="txtFrom">Start finding from:</label>
8           <input type="text" name="txtFrom" id="txtFrom"
value="2">
9         </div>
10
11        <div data-role="fieldcontainer">
12          <label for="txtTo">Find until:</label>
13          <input type="text" name="txtTo" id="txtTo"
value="5">
14        </div>

```

```

15
16         <div data-role="fieldcontainer">
17             <label for="txtDivisible">Divisible by:</label>
18             <input type="text" name="txtDivisible"
id="txtDivisible" value="8">
19         </div>
20         <input type="button" value="Find" id="btnFind">
21     </div>
22 </div>
23 </form>
24
25 <div id="result">
26
27 </div>
28
29 </div>

```

22. Edit index.js:

```

1 (function () {
2
3     $(document).ready(function () {
4         $("#btnFind").bind("click", function () {
5             div();
6         });
7     });
8
9     function div() {
10         var from, to, divisible, htmlString, found, i;
11         from = parseInt($("#txtFrom").val());
12         to = parseInt($("#txtTo").val());
13         divisible = parseInt($("#txtDivisible").val());
14         htmlString = "";
15         i = from;
16         found = false;
17         while ((!found) && (i <= to)) {
18             if (i % divisible == 0) {
19                 found = true;
20             }
21             else {
22                 i++;
23             }
24         }
25
26         if (found) {
27             $("#result").html("First number divisible by " + divisible +
" found:" + i);
28         }
29         else {
30             $("#result").html("Not Found");
31         }
32     }
33
34 })();

```

- a. Line 10 declare variables from, to, divisible, htmlString, found and i
- b. Line 11 assigns the user's input in the “txtFrom” text field to the JavaScript variable “from”
- c. Line 12 assigns the user's input in the “txtTo” text field to the JavaScript variable “to”
- d. Line 13 assigns the user's input in the “txtDivisible” text field to the JavaScript variable “divisible”
- e. Line 14 initializes “htmlString” to an empty string.
- f. Line 15 assigns the value of “from” to “i”.
- g. Line 16 initializes “found” to false.
- h. Line 17 controls the execution of the loop for the codes between line 18 and 24. The loop will continue running as long as the following 2 conditions are not met
 - “found” is equal to false
 - “i” is less than or equal to “to”
- i. Line 18 checks if “i” is divisible by “divisible” using the modular (%) function. A value is divisible if the % function returns the value 0 (e.g. 2%2 will be equal to 0. 5%3 will not be equal to 0 – it will be equal to 2. Once a divisible instance is found, the variable “found” is set to true. On the next round in the loop, the condition in line 22 will be fulfilled since found is now true. The loop will then exit.
- j. Line 22 adds 1 to i.
- k. Line 26 checks if the loop has exited with “found” set to true. If it has, it displays the text "First number divisible by"
- l. If the divisible is not found, line 30 runs to display "Not Found."

23. The table below traces the values of the variables as the loop executes to check for the first number divisible by 8 within the numbers 2 to 5.

Iteration	from	to	Divisible	found	i
1	2	5	8	FALSE	2
2	2	5	8	FALSE	3
3	2	5	8	FALSE	4
4	2	5	8	FALSE	5

24. The table below traces the values of the variables as the loop executes to check for the first number divisible by 2 within the numbers 2 to 5.

Iteration	from	to	divisible	found	i
1	2	5	2	TRUE	2

25. The table below traces the values of the variables as the loop executes to check for the first number divisible by 7 within the numbers 2 to 10.

Iteration	from	to	divisible	found	i
1	2	10	7	FALSE	2
2	2	10	7	FALSE	3
3	2	10	7	FALSE	4
4	2	10	7	FALSE	5
5	2	10	7	FALSE	6
6	2	10	7	TRUE	7

Exercise


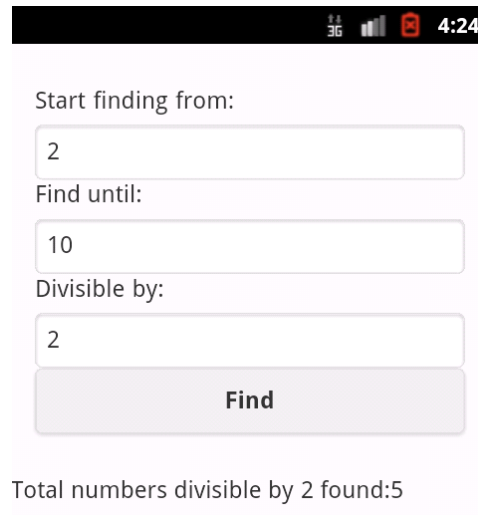
1. Rewrite the Multiplication Table program in Instruction 1, point 29 using a while loop.
2. Amend the program in Instruction 2 point 15 to count backwards as follows:

```
(3,3)
(3,2)
(3,1)
(3,0)
(2,3)
(2,2)
(2,1)
(2,0)
(1,3)
(1,2)
(1,1)
(1,0)
(0,3)
(0,2)
(0,1)
(0,0)
```

3. Write a program to count the total count of numbers between “Start finding from” and “Find until” that are divisible by “Divisible by”.

For example, the numbers between 2 and 10 that are divisible by 3 (2,3,4,5,6,7,8,9,10) are 3, 6 and 9. Thus, 3 numbers between 2 and 10 are divisible by 3.

The numbers between 2 and 10 that are divisible by 2 (2,3,4,5,6,7,8,9,10) are 2, 4, 6, 8 and 10. Thus 5 numbers between 2 and 10 are divisible by 2.

 <p>Start finding from: <input type="text" value="2"/> Find until: <input type="text" value="10"/> Divisible by: <input type="text" value="3"/> Find</p> <p>Total numbers divisible by 3 found:3</p>	 <p>Start finding from: <input type="text" value="2"/> Find until: <input type="text" value="10"/> Divisible by: <input type="text" value="2"/> Find</p> <p>Total numbers divisible by 2 found:5</p>
---	--

Codes

Codes for this chapter can be found by searching VC9-Loops on Github.com.