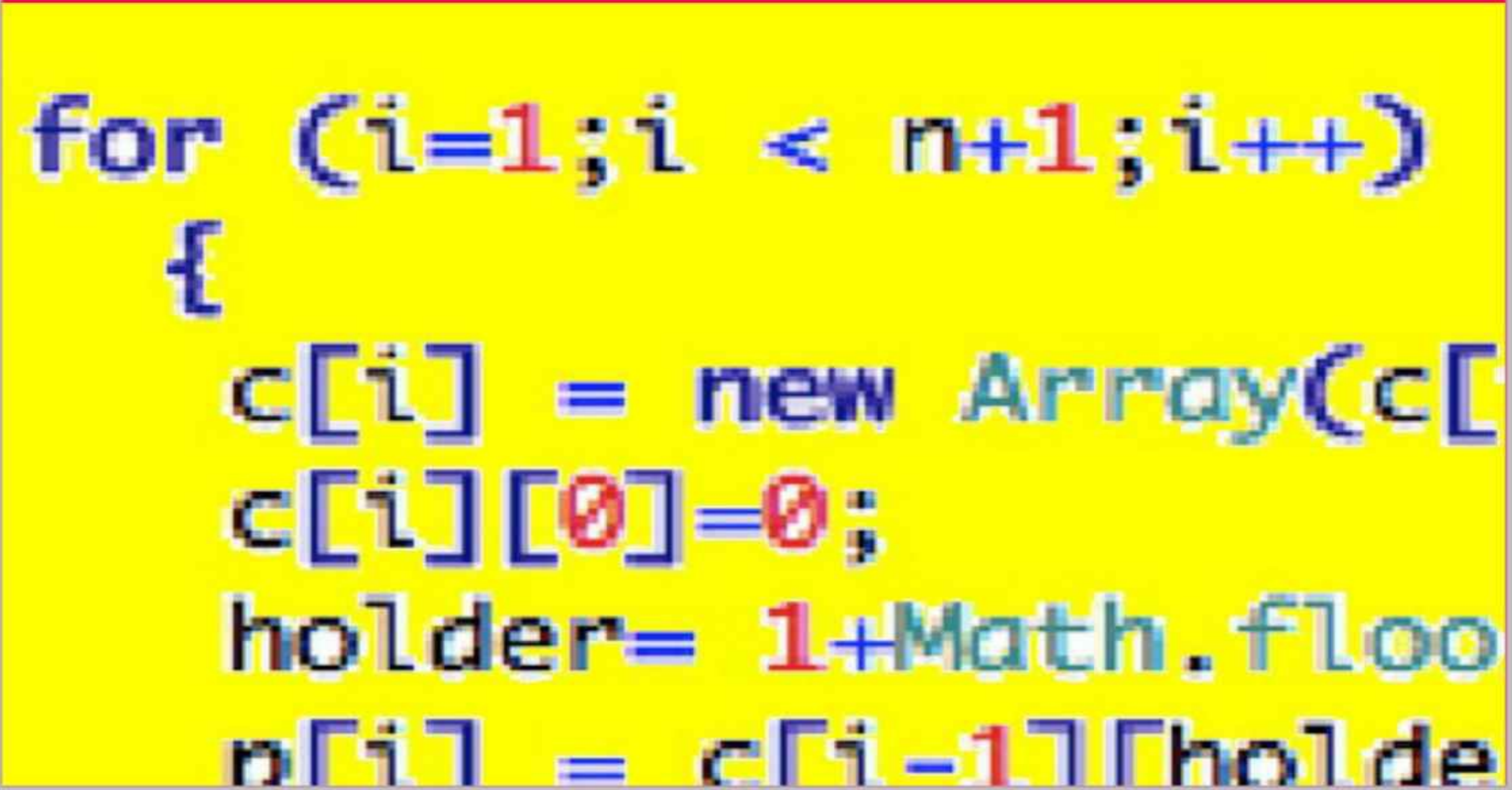
Starting Programming With JavaScript

Fundamentals Of The The JavaScript Computer Language From Someone Who Has Taught It. A Really Good Book For Those With No Previous Knowledge.

RORY BARRETT



```
for (i=1;i < n+1;i++)
   {
    c[i] = new Array(c[
    c[i][0]=0;
    holder= 1+Math.floo
    p[i] = c[i-1][holde
```

Printed in New Zealand by NCEAX
www.nceax.co.nz

About The Author

In addition to this book, Rory Barrett is the author of many books published in New Zealand by ESA Publications. He has authored a number of others and has his own website www.nceax.co.nz.

He was the Head of Mathematics at three high schools in New Zealand: Rutherford High School, Auckland Grammar School, and Macleans College. He was also the Deputy Headmaster at Saint Kentigern College.

In 2014 and 2015 He was in charge of teaching computing at Motueka High School, near Nelson, New Zealand. This book is based on his experiences in teaching JavaScript to the pupils of this school as well as developing his and other websites.

Table of Contents

Introduction

A few words about the book, *"JavaScript"*.

It is necessary before you start your study of JavaScript to have a basic knowledge of HTML, the markup language by which web pages are made. If you lack this then you should learn some basic HTML before you start.

This book contains essential information as you begin a study of JavaScript.

Before you start this book ask yourself these questions:

What do you know about the JavaScript computing language?

Are you keen to use it?

This book will tell you about starting JavaScript in a simple yet factual way that does not blind the reader with science.

It will tell you about basic commands, comments, operations, strings, arrays and many other parts of this computer language.

There are 6 chapters, which cover basic information, sequential programming, iterative programming, conditional programming, dynamic programming, and problems to try and further investigation. Each chapter will provide a stepping-stone to further exploration if so desired.

Before you begin you should have a text editor that automatically produces pages for a variety of computer languages. Microsoft Word is useless for this task. If you are a Windows user then get *Notepad++*. It is an amazing piece of software.
If you, like me prefer Macs, then there are a number such as *Brackets* and *Komodo*.
I knew of Linux but had no experience with it until I got a Raspberry Pi mini computer, which uses a form of Linux called *Raspian*. That operating system comes with a built-in programming editor called *Geany*. I have used Geany and have to admit it is as good as Notepad++.
All editors I have mentioned are free. I actually use *Dreamweaver*, which is very good but unfortunately, it is quite expensive.
Now onto Chapter 1.

Chapter 1: Computers, Associated Concepts, and The First JavaScript Program

Definition of a computer

This is very complicated. All you need to know is that a computer is a device that manipulates data. All useful computers are electronic.

Computer languages

Programming languages range from *low* to *high* level. The lowest level language is called *Machine-Code*. It is written in binary using only 1 or 0. Higher programming languages are designed for humans to interact with computers. Low-level languages are almost indecipherable for most people.

The list below is a sequence of computer languages starting from the lowest, which is *Machine-Code* to *Ruby*. Programs are described as having increasing *abstraction* as their level increases.

Language

1. Machine-code
2. Assembly
3. C
4. C++
5. Java
6. Python
7. Perl
8. JavaScript

9. Ruby

Compilers, Interpreters, and Assemblers

Before use by the computer, languages must be converted to Machine-Code. This task is performed by either by a compiler or an interpreter.

With some languages, the whole program, written in that language, is converted to machine code by compilers. In doing this an executable file is created. Examples of languages which are compiled are C and C++.

An interpreter changes a program line by line to machine code. Perl, JavaScript, and Ruby are interpreted languages.

Is a Server a Computer?

Most people who know about computers regard a *server* as a computer that provides *services* to other computers. While this is true the server is actually a program the server computer is running which provides these services.
If you join computers together to talk with one another then you have a *network* or computer network.

Internet?

The Internet dominates our age. It is an open medium comprised of a vast, global system of networked computers. The process by which computers

exchange data is called *packet switching*. Packet switching is governed by a whole lot of rules called *protocols*. These are very important but also unnecessary to know about if you are just starting in JavaScript.
The Internet is a network of domestic, commercial, government etc. networks with services such as email, the transfer of files, SMS, and all the files of the World Wide Web (WWW).

Definition of a Web Server?

A computer program which serves requested web pages is called a *web server*. A Web *client* is the program of a user which makes the request. A *browser* is a client program requesting files from Web servers. Common browsers include Chrome, Firefox, and Safari.

Web Pages

Files on the Internet with the .html suffix are called web pages. They are opened by browsers. Such files are the basis of the World Wide Web (WWW), probably the most important part of the Internet. The first appearance of HTML was in 1989. It was the invention of the English programmer and scientist Tim Berners-Lee who was employed by the European Laboratory for Particle Physics in Geneva, Switzerland.

HTML has gone through a number of developments since then. At the moment we are in the fifth version called HTML 5. HTML will be used throughout this book as JavaScript was built for use with HTML.

JavaScript

JavaScript is a high-level programming language. Most computer languages have the same program types although some languages are much better for some purposes than for others.

JavaScript is a perfect language of instruction for the teaching of programming. The reasons are that

(i) It is not necessary to install JavaScript.
(ii) It is ideal for courses in programming. Its possibilities vary from the most fundamental to really cutting edge.
(iii)

Is It True That JavaScript and Java Are the Same?

JavaScript and Java are totally different languages even though their names are alike. They are similar when simple programs are made however JavaScript is easier to use and understand. The learning of JavaScript is a great help in learning other more complicated languages such as C++, Java, and Python.

Is HTML The Same As JavaScript?

No, they are not the same. They are quite different. HTML is what is known as a markup language for the creation of static web page content. JavaScript is a computer language which makes possible the creation of dynamic content for web pages. This can be confusing because it is possible to place JavaScript code in an HTML file, although you will learn this is not good practice. Despite that, we will initially put JavaScript code in HTML files as it is easier to learn the better practices later.

Is It Possible To Put JavaScript on the Server?

Yes, it is and is frequently done. If you do this then you get server-side JavaScript. This extends JavaScript so that it can access files and databases. Client-side JavaScript does not have this capability. This book will consider only the client-side version.

First task

Type the following up using notepad or some other equivalent text editor such as notepad++ or a Mac editor such as TextEdit. If you are fortunate enough to have Adobe Dreamweaver then use that however it is quite expensive and is not necessary for a basic course in JavaScript.

……………………………………………………………………………………

```
<!DOCTYPE HTML>

<HEAD>

<TITLE> XYZ </TITLE>

<script>
alert("Hello World.");
</script>

</HEAD>

<BODY>
The dog sat on a log.
```

```
<hr>
<p>
<button type="button">Button</button>
<hr>
<p>
<form>
Initial first name:<br>
<input type="text" name="initial">
<br>
Family name:<br>
<input type="text" name="familyname">
</form>

</BODY>

</HTML>
```
....

Save the file as *alert.html*
Click on it or drop the file into the address bar of a browser.
It is a good idea to write down what you see.
If you have done this correctly there should first be a little window saying, "Hello World".

Before we go onto the next chapter it is worth mentioning that the only JavaScript in this first program you wrote was

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

On seeing <*script*>, the script tag, the computer knew that some JavaScript would follow.

The JavaScript command was *alert("Hello World.");*

On receiving this command the computer carried out the procedure leading to the little window with " Hello World".

The <*/script*> tag informed the computer JavaScript was finished so it resumed doing HTML leading to a window showing the sentence, '*The dog sat on a log.'* a button and a form containing some text fields into which information about names could be typed.

It is worth noting that the alert box is a very powerful tool for locating errors in JavaScript code.

One final point about *alert()* you can use ' ' instead of " ". Just be consistent. You can't have ' " or " '.

Finally, it is so easy to use a capital *A* in *alert*() instead of a lower case *a*. If you do this your program won't work. JavaScript is sensitive to lower and upper case. You will probably make this mistake again and again. Most programmers do.

JavaScript is not the only case sensitive language. Take care!

Chapter 2: Sequential Programs

In the last chapter, we introduced you to your first JavaScript program, which was nothing more than the alert command in JavaScript. In this chapter, we will continue our investigation by looking at sequential programming and in the process introduce *document.write*, the standard math operations in JavaScript, strings, variables, arrays and the printing of pictures.

In what follows we will frequently refer to a standard HTML file called Txt.HTML.

By this, we mean an HTML file created with something like Notepad ++ on Windows or TextEdit on a Mac. I use a Mac and use Dreamweaver, however, the latter program is quite expensive.

Type this

```
<!DOCTYPE HTML>
<HEAD>
<TITLE> XYZ </TITLE>

</HEAD>
<BODY>



</BODY>
</HTML>
```

Save it as *Txt.HTML* or some other name but always have the *.HTML* suffix (or *.html*). You will use this program again and again as you work through this book.

What is sequential programming?

Sequential programming is where one command follows another, each doing a particular action. There is NO repetition and NO branching. It is the most straightforward sort of programming.

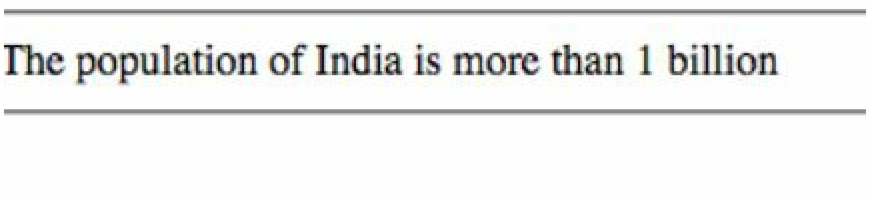
document.write

In between the tags *<body>* and *</body>* write the lines :

```
<hr>

<script>
document.write( "The population of India is more than 1 billion");
</script>

<hr>
```

Save what you wrote then open *Txt.HTML* with your browser.

You should see the following output.



```
The population of India is more than 1 billion
```

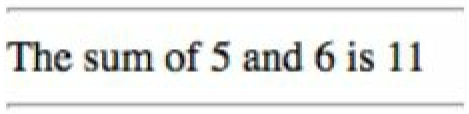
Two comments about this:
(i*) <hr>* is an HTML tag instructing the computer to draw a horizontal line.
This has nothing to do with JavaScript.
(ii) *document.write(......)* is a JavaScript command which tells the computer to write what is between the brackets.

Standard Math Operations

Open *Txt.HTML* in your editor
and amend the program so that it now reads

```
<hr>
<script>
document.write("The sum of 5 and 6 is ");
 document.write(5+6);
</script>
<hr>
```

Save the program then run it. If you do you will get this output.



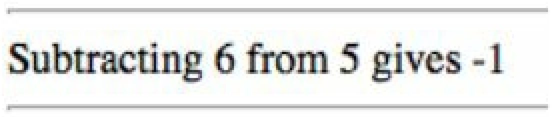
```
The sum of 5 and 6 is 11
```

In similar fashion, we could subtract 6 from 5.

We would have to amend the program to

```
<hr>
<script>
document.write("Subtracting 6 from 5 gives ");
  document.write(5 - 6);
</script>
<hr>
```

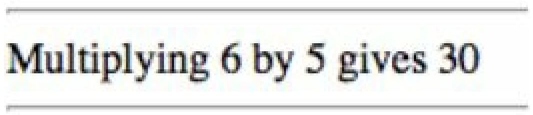
The output is



```
Subtracting 6 from 5 gives -1
```

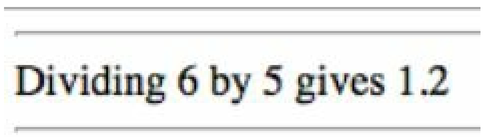
Similar changes would allow you to multiply and divide.
The correct statements would be
`document.write(5*6);` for multiply
and
`document.write(6/5);` for divide.

Outputs should be similar to



```
Multiplying 6 by 5 gives 30
```

and



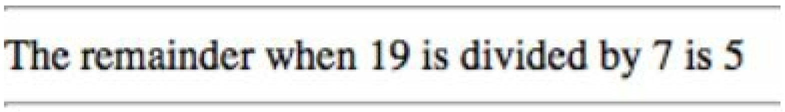
```
Dividing 6 by 5 gives 1.2
```

A very interesting mathematical operation is **mod**; 19 mod 7 = 5. It is the remainder when we divide 19 by 7. We use % for this in JavaScript.

```
<script>
```

```
document.write("The remainder when 19 is divided by 7 is ");
 document.write(19%7);
</script>
```

results in



```
The remainder when 19 is divided by 7 is 5
```

Two things to notice:
(i) You don't put ' and ' or "and " around 13 + 25 if you want a calculation.
(ii) The semicolon (;). JavaScript statements normally work if you haven't got this, but not always. As a result, it is a good idea to always put it at the end of a statement.

Strings

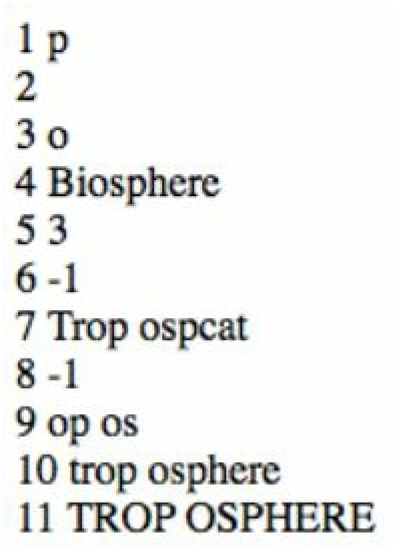
A string is a collection of characters enclosed by " " or ' '. Strings are incredibly important in JavaScript, in fact in most computer languages. You will get to use strings in many practical situations as we proceed but before we go on we need to have a look at some string operations. For anyone using databases, string operations are really important no matter what language you are using.

Open JavaScript.HTML in your text editor and between the *<script>* and *</script>* tags write the following

```
<script>
document.write(1+"  "+"Trop osphere" .charAt(3)+"<br>");
document.write(2+"  "+"Trop osphere".charAt(4)+"<br>");
document.write(3+"  "+"Trop osphere".charAt(5)+"<br>");
document.write(4+"  "+"Bi"+"os"+"phere"+"<br>");
document.write(5+"  "+"Trop osphere".indexOf("p ")+"<br>");
document.write(6+"  "+"Trop osphere".indexOf("ant")+"<br>");
document.write(7+"  "+"Trop osphere".replace("here","cat")+"<br>");
document.write(8+"  "+"Trop osphere".search("e ph")+"<br>");
document.write(9+"  "+"Trop osphere".substring(2,7)+"<br>");
document.write(10+"  "+"Trop osphere".toLowerCase()+"<br>");
document.write(11+"  "+"Trop osphere".toUpperCase()+"<br>");
</script>
```

then run.

The output should be



```
1 p
2
3 o
4 Biosphere
5 3
6 -1
7 Trop ospcat
8 -1
9 op os
10 trop osphere
11 TROP OSPHERE
```

Running through each line.

1. The letters in a string start from position 0 so the statement "+"Trop osphere" .charAt(3) writes down the character of position 3 which is p.

2. The letters in a string start from position 0 so the statement "+"Trop osphere" .charAt(4) writes down the character of position 4 which is a blank.

3 The letters in a string start from position 0 so the statement "+"Trop osphere" .charAt(5) writes down the character of position 5 which is o.

4 The + signs when dealing with strings join them together. The strings, in this case, are "Bi ", "os" and "phere". When they are joined together you get Biosphere.
5. This statement instructs the computer to reveal the position of "p ",(p followed by a space), in "Trop osphere". "p " is at position 3 in "Trop osphere" so the computer returns 3.
6. This statement instructs the computer to reveal the position of the string "ant" in "Trop osphere". The string "ant" is not in "Trop osphere" so the computer gives -1.
7. This statement instructs the computer to replace "here" in "Trop osphere" by "cat'. This is done and gives the string "Trop ospcat".
8. This statement instructs the computer to search for "e ph" in "Trop osphere". This is done and is not found so -1 is returned.
9. This instruction asks for a substring of the string "Trop osphere". This means a set of letters from "Trop osphere". The substring asked for starts from the character at position 2 which is 0 then stops before the character at position 7. This is the substring "op os".
10. This statement instructs the computer to make all characters lowercase.
11. This statement instructs the computer to make all characters uppercase.

You may be mystified by <br>. It is an HTML tag and without this tag, all output would be horizontal instead of vertical. <br> is a very useful HTML tag for causing lines to have vertical alignment.

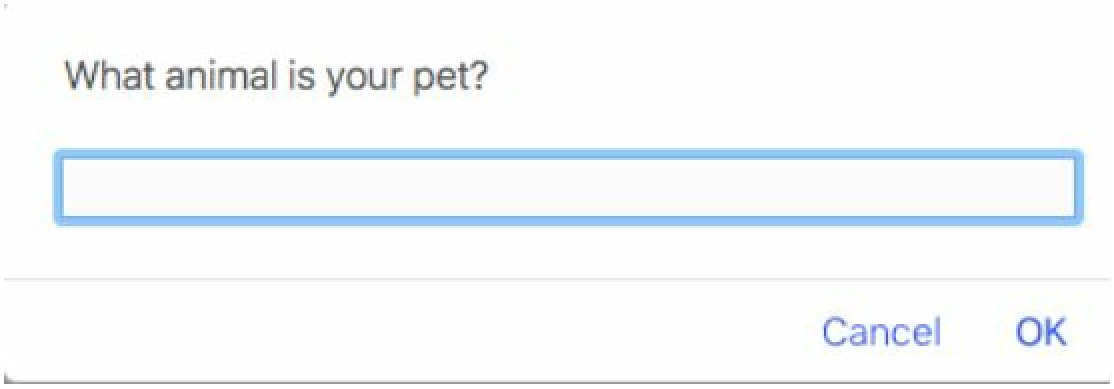
The Extremely Important Concept Of Variables

Open *Txt.HTML* in your text editor and alter *<script>* ..... *</script>* to

```
<script>
 var pet = prompt('What animal is your pet?');
document.write(pet);
</script>
```

After running

This is what you will see


What animal is your pet?

Followed by a screen showing the animal you entered. Say you entered Dog then you'd get

Dog

On receiving the line *var pet = prompt(What animal is your pet?');*
the computer places what you entered into a slice of memory labeled **pet**. Each time the program meets the word pet then the pet you entered will be brought up. You have just created a *string* variable called pet. Note that pet is not

enclosed by " " or ' '. The computer understands that it is to write down what is at memory **pet** and NOT the string 'pet'.

In the previous example

```
<script>
document.write(1+"  "+"Trop osphere" .charAt(3)+"<br>");
...
...
...
document.write(11+"  "+"Trop osphere".toUpperCase()+"<br>");
</script>
```

we could have created a variable called **str**. The code would be

```
<script>

 var str = "Trop osphere";

document.write(1+"  "+str .charAt(3)+"<br>");
...
...
...
document.write(11+"  "+str.toUpperCase()+"<br>");
</script>
```

Doing this would have saved a lot of time.

There are number variables which can be created just like string variables.

var numm = 10; creates an integer variable **numm** with the value 10.
var numbar = 6.5; creates a real number variable with the value 6.5.

Another number type is called *boolean*. It only takes the values True and False. These are very important in something we study later called *conditional computing*.

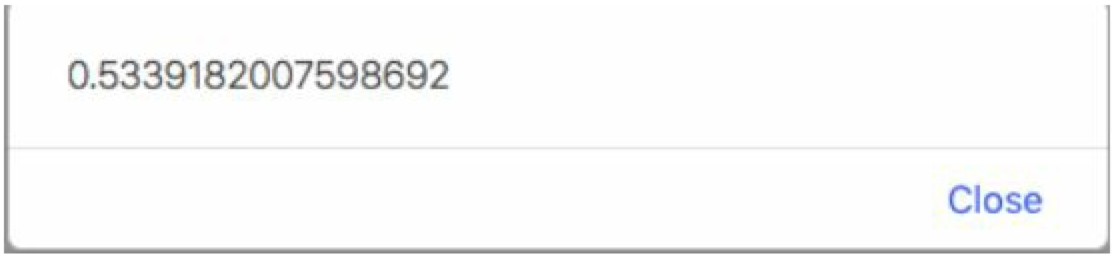
The Math Object

JavaScript has a means, which is built in, to get many mathematical functions called the Math object.

If you need random numbers from 0 to 1 then you use *Math.random()*. Here is how this is done.

```
<script>
var numm = Math.random() ;
alert(numm);
</script>
```

When running this you would get the output of a little window like this.



```
0.5339182007598692
```

The number you get is probably different but you will get a decimal between 0 and 1.

The Math object makes a random number between 0 and 1 that is assigned to the variable **numm**.

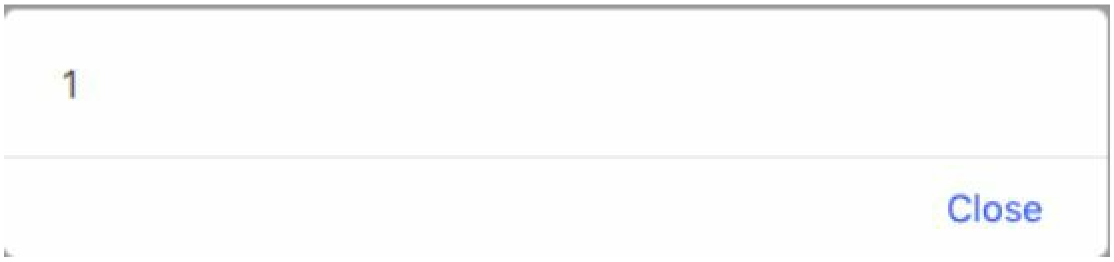
numm is shown by alert.

What about a random integer between 0 and 10?

Here is one way of doing this.

```
<script>
var numm = Math.round(10*Math.random()) ;
alert(numm);
</script>
```

The output for me was



```
1
```

Pressing close then refreshing the page gives different integers between 0 and 10.

The Math object makes a random number between 0 and 1 which is multiplied by 10 then this number is rounded by Math.round() and assigned to the variable **numm**.

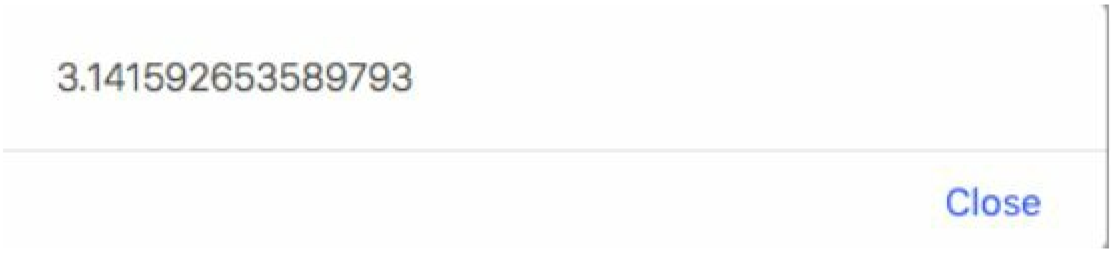
The alert procedure then shows **numm.**

There are many other useful functions available through the Math object.

Here's another example of this.

```
<script>
var num = Math.PI;
alert(num);
</script>
```

The output is



```
3.141592653589793
```

The Math object creates the number Pi or π which goes to the variable **num**.

The alert procedure then displays **num.**

This is of great use if you need to find the circumference ($2\pi r$) or area (πr^2) of a circle then use them in a program.

Arrays

This sub-topic is of great importance, every computer language uses arrays.

Basically, an array is a piece of computer memory that holds a set of values.

Here is a simple example (Don't worry about statements after //. These are comments and don't affect the actual program.)

```
<script>
var numberHold = new Array();//creates new array numberHold

numberHold[0] = 3;
numberHold[1] = 6;
numberHold[2]=-2;//assigns values to numberHold[0] etc

var numRandom = Math.round(2*Math.random());//creates a number 0,1 or
2 randomly

alert(numberHold[numRandom]);//shows the value of
numHold[numRandom]
</script>
```

If you run this program you will get an output of alert showing one of 3, 6 or -2.

The program first creates a new array then puts 3, 6 and -2 into the variables *numberHold[0]*, *numberHold[1]* and *numberHold[2]*.

*var numRandom = Math.round(2*Math.random())* is one of 0,1 or 2.

Alert demonstrates whichever of *numHold[0]*, *numHold[1]* and *numHold[2]* is selected.

You need to know that an array *Ar* always begins at 0, the first in the list contained by *Ar* is *Ar[0]*.
You will have noticed the //. Any statement following // is a comment. These do not affect the program. We will have more to say about this in the next chapter.

Another example

Try this

```
<script>
var girls =
["Jane","Fiona","Mary","April","May","June","Julie","Averill","Shirley","
var numm = Math.round(11*Math.random());
alert(girls[numm]);
</script>
```

This program if run repeatedly gives a random sequence of windows with one girl's name from the list shown. This is a way of creating arrays when you know the values needed. It is not necessary to use new Array(). You just make a variable and set it equal to square brackets, which contain the contents of the array.

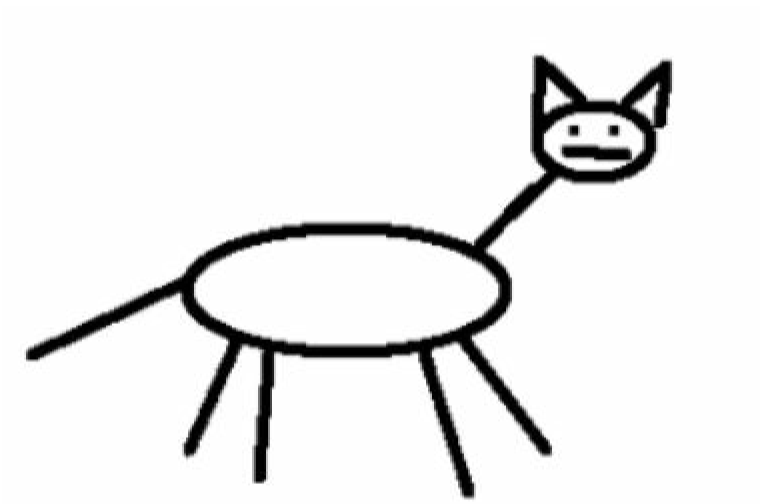
The next chapter will show you more examples using arrays.

Printing a picture

Make sure you have a picture file of a cat with the suffix *.png* in the same folder as *Txt.HTML* then change the code to

```
<script>
document.write("<img src='  cat.png' alt='cat '  width='20%'><br>");
</script>
```

When I do this the output is



The reason for the *20%* is to adjust the size of the picture to 20% of the page width. This is an *HTML* command, not a JavaScript command.

Chapter 3: Iterative Programs

In the last chapter, we gave you examples of sequential programming in JavaScript. In the process, we introduced the *document.write* command, the standard math operations in JavaScript, strings, variables, arrays and the printing of pictures.

We will continue to refer to a standard HTML file called *Txt.HTML*.

In this chapter, we are going to introduce you to writing comments as you program, a very important part of programming called *iteration*, more commonly known as *loops*, a concept called *functions* and JavaScript files with the suffix *.js*.

We start with something very simple.

Suppose you wanted to print the word 'Frog' 6 times going down the page.

```
<script>
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
</script>
```

will do this.

Comments

Although what this piece of code does is almost self-evident this is not always true. Sometimes programs are really complicated. It is always good practice to write comments in the program, which do not affect the running of the program but help someone who has a look at the program later to understand what is going on. The way in which this is done in JavaScript is very similar to what is done in the C, C++, Java and PHP languages.

Here is how the program above could have comments.

```
<script>
/* This program was written in 2017. The writer was
Paul Revere*/

document.write( "Frog"+"<br>"); //This is the first line of the program.
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>");
document.write( "Frog"+"<br>"); //This is the last line of the program.

/*This program prints the word Frog
six times down a page*/
</script>
```

// for single lines
/*..........*/ if more than one line. Often it is more meaningful to write /*..........*/ before important lines of code. Examples of this will be given later.

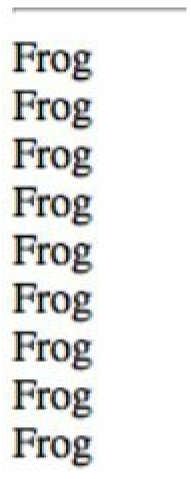
Loops

The above program could be written much more compactly. If the task was to write Frog down the page 50 times there are fewer inefficient ways of doing this than to write out *document.write("Frog"+"<br>")* fifty times. Luckily, there are methods by which this is done called *loops*. The first method uses a **for** loop.

Type the following piece of code into *Txt.HTML*.

```
<script>
/*This will print FROG 50 times going down the page */
for (i = 1;i<51;i++)
{document.write( "Frog"+"<br>"); }
</script>
```

Part of the output is shown below


```
Frog
Frog
Frog
Frog
Frog
Frog
Frog
Frog
Frog
```

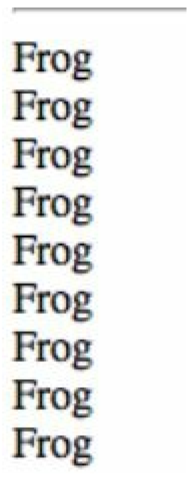
Note that there is now a comment above the code. */*This will print FROG 50 times going down the page */*

Are there other loops? Yes! One of the best is the **while** loop. Here is how a **while** loop does what **for** did previously.

Put this code in *Txt.HTML*.

```
<script>
/* 50 printings of Frog down page using the while loop*/
var count = 1; // This variable counts the number of printings

while (count< 51)
{
document.write( "Frog"+"<br>");
count++; //this increases count by 1 each time a loop is completed
}
</script>
```

The output is once again.



```
Frog
Frog
Frog
Frog
Frog
Frog
Frog
Frog
Frog
```

These loops are sufficient for most purposes. There are others. Check them with Google if you're interested.

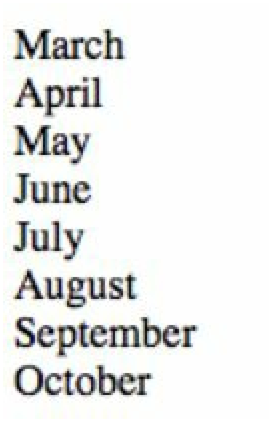
Which is best? **for** or **while**? Neither, a good programmer could use either. The **for** loop is best if you have a definite beginning and end. The **while** loop is best when you are uncertain where to stop.
Let us take a situation where **for** would be better.

If we had an array of the 12 months and wanted to print the 3rd month to the 10th then do this.

```
<script>
/* This program prints all months from the 3rd to the 10th*/

var months =
["January","February","March","April","May","June","July","August","Sep
months

/*This will print months 3-10 down the page*/
for (i = 2;i<10;i++)
```

```
{document.write( months[i]+"<br>"); }
</script>
```

The output is



```
March
April
May
June
July
August
September
October
```

A couple of points need to be raised.
(i) Remember that an array always starts from 0 so the 3rd will be number 2 and the 10th will be number 9.
(ii) I have called the array containing the months *months*. I could have called it just about anything but in selecting variable names give the variables names that are similar to what they stand for. You must take care in selecting names for variables, as a name may be a *reserved word*. Reserved words are used by JavaScript and have special meaning. If you named *months* with the word *name* instead your results would be strange. The reason is that name is a reserved word.

Let's use **while** in a situation where it would be better.
Suppose you wanted to print all names in the array of months until you encounter one with less than 6 letters.

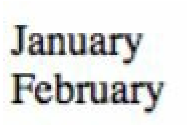
str.length is the number of characters in the string *str*.

Here is a program which will do this:

```
<script>
/* This program prints all months until there is one with less than 6 letters*/


var count =0;//This counts the number of loops.

var months =
["January","February","March","April","May","June","July","August","Sep
months

var howManyLetters = months[count].length ;/*This variable will hold the
number of letters in a month as we go through the loop. It starts with the
length of the first member of months*/


while (howManyLetters>6)//This will print required months down the page
        {
        document.write( months[count]+"<br>");

        count++; //Each time we do a loop count increases by 1

        howManyLetters = months[count].length;//each time we increase
        count we make howManyLetters the length of the next member of
        months
        }
```

```
</script>
```

The output is shown below



```
January
February
```

This is what we expect. March only has 5 letters.

Indentation

Something that was done in the previous little program needs stressing. Notice how the program is set out beneath the *while*.

```
while (howManyLetters>6)//This will print required months down the page
          {
          document.write( months[count]+"<br>");

          count++; //Each time we do a loop count increases by 1

          howManyLetters = months[count].length;//each time we increase
          count we make howManyLetters the length of the next member of
          months
           }
```

with the brackets and their contents moved to the right. This is called *indentation*.

The contents of the {} are a subprogram of the main program. It is good practice and manners to do this. If you are the only person who will ever work on your programs then how you set them out is up to you but if you will have others working on the program then this sort of indentation is of great help in assisting to see what you have done.

Although indenting is not essential in JavaScript, in some languages, such as *Python,* you have to indent otherwise the program won't work.

Functions

A function is a program that gets used a lot and is made available in a way that other programs can use.
In the head or some other part of the file *Txt.HTML*. type

```
<script >
function initials()
{
/* This function is a program that gets you to input your first and family names, gets the first initials of each then alerts them as a word */
var firstName =prompt("What is your first name?");//This gets you to input your first name
var familyName =prompt("What is your family name?");//This gets you to input your family name
var namStr = firstName.charAt(0)+  familyName.charAt(0)//This combines first initials of the first and family names
alert(namStr);
}//end of function initials()

</script>
```

Now somewhere in the body type *<script> initials(); </script>*.
If *Txt.HTML* has opened then the function *initials()* operates and we get the expected alert.

This may seem a fairly simple demonstration so we will make it a bit more interesting by giving what is called a *parameter*.

Replace the declaration of initials() with *initials(n)* in the following way.

```
function initials(n)
{
/* This program gets you to input your first and family names, gets the nth
character of each then alerts them as a word */
this.n = n;//tells program that the n used by the program is the n given
between the brackets
var firstName =prompt("What is your first name?");//This gets you to input
your first name
var familyName =prompt("What is your family name?");//This gets you to
input your family name
var namStr = firstName.charAt(n)+  familyName.charAt(n)//This combines
nth characters to a single string
alert(namStr);
}//end of function initials(n)
```

The section which used to be *<script > initials(); </script>* is replaced by

```
<script >
var n = prompt("What is the position of the character in the names?");// we
enter n
initials(n);
</script>
```

We are being hypocritical in using *n* as a variable name. It would be better to call it charAtPos or something more descriptive.

The .js file

We now demonstrate a powerful tool called a JavaScript file. The suffix of a JavaScript file is *.js*. Often you will repeat the same code over many, even, scores of web pages. The *.js* file lets you do this efficiently without having to write the same code repeatedly.

Using a text editor create your functions in a totally distinct file from those with the *.HTML* suffix. You can name that file how you wish but don't use reserved words.

To show how this works the *initials(n)* function is placed in a file called *hold.js*. The file *hold.js* must be in the same folder as the *HTML* file. If you want your program to fail then place the .HTML and .js files in folders, which are different.

The HTML is now the following

```
<!DOCTYPE HTML>

<HEAD>

<TITLE> ABC </TITLE>

<script src="hold.js"></script>

</HEAD>
```

```
<BODY>
<hr>
<script>
var n = prompt("What is the position of the character in the names?");// we
enter n
initials(n);

</script>

<hr>

</BODY>

</HTML>
```

<script src="hold.js"></script> is the vital line

This line informs JavaScript that it should inspect the source file (src) *hold.js* for functions, not the *HTML* file.

Chapter 4: Conditional Programs

In the last chapter, we introduced you to iterative programming in JavaScript. In the process, we introduced you to writing comments as you program, a very important part of programming called *iteration*, more commonly known as *loops*, functions, and JavaScript files with the suffix *.js*.

In this chapter, we will explore the situation when you have different branches in a program. The branch that is taken depends on conditions. Such programming is called *conditional programming*. Associated with conditional programming are the logic operators AND(&&), OR(||) and NOT(!). It is very important to see how JavaScript programs handle equality and inequality.

Continue with the HTML file called *Txt.HTML*. however, all functions will be placed in *hold.js*.

You should have *Txt.HTML* as

```
<!DOCTYPE HTML>

<HEAD>

<TITLE> TTT </TITLE>

<script src="hold.js"></script>

</HEAD>
```

```
<BODY>

<script>
//programs with functions in hold.js
</script>

</BODY>

</HTML>
```

The line *<TITLE> TTT </TITLE>* is optional. If you like you can totally delete this

We start with something very simple.

Suppose you randomly generated 0 or 1. If 0 was generated then an alert with 'Meow' would come up. If 1 was generated then the output would be an alert with 'BowWow'.

Create a new function *randomCatDogNoise(n)* in hold.js.

The initial step is to write what is below.

```
function randomCatDogNoise(n)
{
}
```

This is the randomCatDogNoise function. Because there is no code between the curly brackets this function does nothing. If some JavaScript instructions are between the curly braces then *randomCatDog(n)* may do something. Notice the parameter *n*.

The conditional operators, in JavaScript, are **if** and **if/else**. The next example uses **if/else**.

Type the following.

```
function randomCatDogNoise(n)
{
this.n = n;// n means the parameter

if (n ==0)
        {
        alert("Meow");


        }
else
        {
        alert("BowWow");
        }

}//end of randCatDogNoise(n)
```

Note == we use ' equals ' for a comparison. This is critical.
Never write (n = 0); if you do this JavaScript assumes you are giving *n* the value of 0.

Never forget : == for comparison and = for assignment.

In the body of *Txt.HTML* write

```
<script>
var n = Math.round(Math.random());//Randomly create 1 or 0
randomCatDogNoise(n);//activates the function randomCatDogNoise
</script>
```

Note the use of *randomCatDogNoise* for the function name. Function names like variable names should be related to their purpose.

Instead of **if/else** use **if/else if/else** if the number of choices exceeds 2.

As an example consider the random generation of numbers 0,1, 2, 3, 4 and 5. If 0 or 1 you get 'Chair', 2 or 3 are 'Table' and 4,5 are 'Knife'.

Here is the code. Try it yourself first but don't spend more than half an hour.

```
function randomChairTableKnife(n)
{
this.n = n; ;// n means the parameter
if (n <2)
        {
        alert("Chair");//the alert says "Chair" if 0 or 1 are generated

        }
else if (n<5)
        {
        alert("Table");// the alert says "Table" if 2 or 3 are generated
        }
else
        {
        alert("Knife");// the alert says "Knife" if 4 or 5 are generated
        }

}//end of randomChairTableKnife(n)
```

The code in the body of JavaScriptCarrier.HTML would be

```
<script>
var n = Math.round(5*Math.random());//Randomly creates  0,1,.....5
randomChairTableKnife(n);
</script>
```

Note < being used as a comparison operator. The operator < means *less than*, similarly > is *greater than*, >= is *greater than or equal* and <= is *less than or equal*.

There are three choices in the program above hence only one *else if*. For four choices you'd need 2 *else ifs*, for 7 choices you'd need 5 *else if*s etc. It is necessary to end with *else*.

You could use a *switch* operator. The switch operator does this as well. It is not necessary. If you're interested in the switch operator do a Google search.

What about a program that only uses **if** and does not need **if/else** or **if/else if/else**.

Suppose you had the array
["elephants","cats","tigers","lions","monkeys","cows", "sheep", "frogs"]

and needed the printing of all words in the array with a minimum of 6 letters.

Here is a function does this. Try it yourself before reading what follows.

In *hold.js* type the following.

```
function typeAnimals(n)
{


}
```

Now fill it as follows:

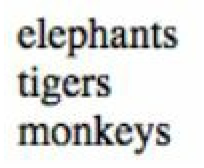
```
function typeAnimals(n)
```

```
{
this.n = n;// n in parameter

var animals= ["elephants","cats","tigers","lions","monkeys","cows",
"sheep", "frogs"];//array of animals

for(i=0;i<animals.length;i++)//animals.length is the number of elements in
animals. In this case it is 8
        {

        if (animals[i].length >= n)//if the number of letters in animals[i] is at
        least n then animals[i] is printed.
                {
                document.write(animals[i]+"<br>");//instruction to print
                animals[i]
                }

        }
}//end of function typeAnimals(n)
```

Now type the following code into the body of *Txt.HTML.*

```
<script>
var n = 6;// We want to print elements in the array which have at least six
letters
typeAnimals(n);
</script>
```

Open *Txt.HTML* and your output should be


```
elephants
tigers
monkeys
```

Conditional Operators

This too is of great importance. Those who are good at maths and logic find this easy but others may find it difficult.

AND

The operator AND is used between two conditions if it is necessary for both to be true. The operator AND is && in JavaScript

Consider this problem. A computer generates 1 or 0 randomly then selects one of C or D.

If 1 AND C are picked an alert goes up saying "Congratulations, you picked 1 and C" else an alert goes up saying "Sorry you need 1 and C".

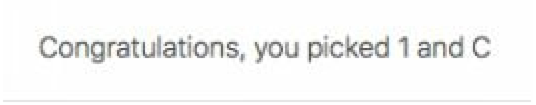
Type this into *hold.js*.

```
function ranNumLetter()
{
var numm = Math.round(Math.random());//selects 0 or 1 randomly
var lettNum = Math.round(Math.random());//selects 0 or 1 randomly

var let = "C";

if (lettNum==1){let = "D";}//the letter is C for letterNum being 0 and D
otherwise.

if ((numm ==1)&&(let=="C")) //both 1 and C have to be generated
        {alert("Congratulations, you picked 1 and C")}
else
        { alert("Sorry, you need 1 and C")}



}//end of function ranNumLetter()
```

Now type the following code into the body of *Txt.HTML*

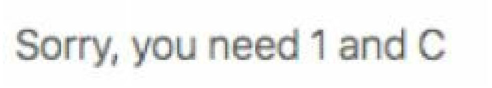
```
<script>
ranNumLetter();
</script>
```

Open Txt.HTML in your favorite browser to see



```
Congratulations, you picked 1 and C
```

or



```
Sorry, you need 1 and C
```

OR

The operator OR goes between two conditions if one or both of them are true. The operator OR is represented by || in JavaScript.

To demonstrate it we will use the function *typeItems(n)* used before and modify it so that it prints items of length *m* OR length *n*.

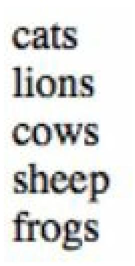
Here is the function that should be typed into *hold.js*.

```
function typeAnimals1(m, n)
{
this.m = m;// uses m in parameter
this.n = n;// uses n in parameter

var animals= ["elephants","cats","tigers","lions","monkeys","cows",
"sheep", "frogs"];//array of animals

for(i=0;i<animals.length;i++)//things.length is the number of elements in
animals. In this case it is 8
        {

        if ((animals[i].length == m)|| (animals[i].length == n))//if the
        number of letters in animals[i] is m or n then in animals[i is printed.
                {
                document.write(animals[i ]+"<br>");//instruction to print
                animals[i]
                }
        }
}//end of function typeAnimals1(m,n)
```

Now type the following code into the body of *Txt.HTML.*

```
<script>
var m = 4;// We want to print elements in the array which have four letters
var n = 5;// We want to print elements in the array which have five letters
typeAnimals1(m,n);
</script>
```

After opening *Txt.HTML* in your favorite browser the output is



```
cats
lions
cows
sheep
frogs
```

All the animals with 4 or 5 letters!

NOT

The operator NOT is used in front of a condition so that the procedure happens when the condition is not true. The operator NOT is represented by ! in JavaScript.

As an example, we will use the function *typeAnimals(n*) and alter it so that it prints animals which do NOT have *n* or more letters.

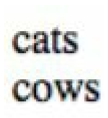
Type this function in *hold.js*

```
function typeAnimals2(n)
{
this.n = n;// n in parameter

var animals= ["elephants","cats","tigers","lions","monkeys","cows",
"sheep", "frogs"];//array of animals


for(i=0;i<animals.length;i++)//animals.length is the number of elements in
animals. In this case it is 8
        {

        if  (!(animals[i].length >= n))//if the number of letters in animals[i]
        is NOT at least n then animals[i]  is printed.
                {
                document.write(animals[i]+"<br>");//instruction to print
                things[i]
                }

        }
}//end of function typeAnimals2(n)
```

Now type the following code into the body of *Txt.HTML*

```
<script>
var n = 5;// We want to print elements in the array which do NOT have at
least five letters
typeAnimals2(n);
</script>
```

After opening Txt.HTML in your favorite browser the output is



```
cats
cows
```

As you can see the words that do not have at least 5 letters have been printed. In this case, there are only 2.

Chapter 5: Creating Dynamic Content

In JavaScript, the ability to access, create, destroy or change HTML elements such as text boxes, drop-down lists, tables etc. is one of its most useful features. In order to do this, it uses a technique called the Dom model. This is based on something called *object oriented programming* which is a method of programming which is quite complicated for beginners. For the purpose of creating dynamic content for web pages knowledge of this type of computing is not absolutely necessary.

First, we will have a look at accessing and altering existing HTML elements using JavaScript.

Task one. Alter *Txt.HTML* so that when the button is pressed an alert showing the word Cat appears.

This is a really easy thing to do and does not actually use the Dom model. However, it is a good first step in learning about this powerful technique.

Here is the *Txt.HTML* code you need.

```
<!DOCTYPE HTML>

<HEAD>
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
</HEAD>
```

```
<BODY>

The dog sat on a log.

<hr>
<p>
<form>
<input type ="button"  id = "butt" onclick=" cat() " value ="button">
</button>
<br>
Initial first name:<br>
<input type="text" name="initial" >

<br>    Family name:<br>
<input type="text" name = "familyName" >
</form>

</BODY>

</HTML>
```

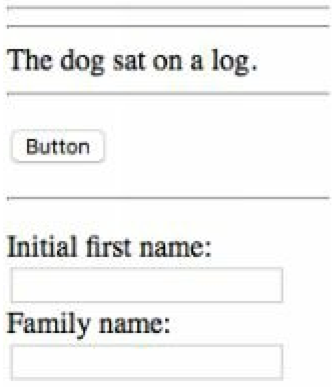
For the purposes of the task the vital line is *<input type ="button" id = "butt" onclick=" cat() " value ="button"></button>*
By putting in the instruction *onClick="cat()"* we are informing the computer that if the button is clicked then the function *cat()* takes place.

The instructions for the function are not found in *Txt.HTML* they are found in *hold.js*.

The code for this function should be obvious but in case it isn't then it is written below.

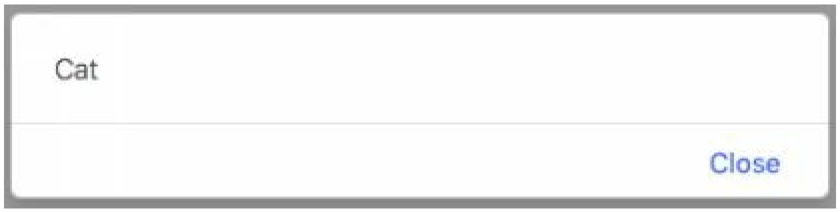
```
function cat()
 { alert('Cat'); }
```

If you type the exact code for Txt.HTML as above and put the function cat() into hold.js then initially you will get the following output.



The dog sat on a log.

Now click on the button and you get the alert window.



Cat

Now we are going to use the Dom model to influence what is on the screen showingTxt.HTML..

Task two. Alter *Txt.HTML* so that when the button is pressed a prompt appears asking your first name initial, then a second prompt appears asking for your family name. After you have filled in the prompts the first initial appears in the corresponding textbox and the family name goes in that text box.

Let the function be called *firstLetterFamilyName()*.

Here is the code you should have in *hold.js*.

```
function firstLetterFamilyName()//This function gets names and puts data from them in text fields
{
var firstNam = prompt("What is your first name?");//obtains first name
var familyNam = prompt("What is your family name?"); //obtains family name

var initLetter = firstNam.charAt(0);//gets first letter of first name

var ob1 = document.getElementById("firstName");//assigns the textfield with the ID "firstName" to the variable ob1
var ob2 = document.getElementById("familyName");//assigns the textfield with the ID "familyName" to the variable ob2

ob1.value = initLetter//puts first initial of first name in correct textfield
ob2.value = familyNam//puts family name in correct textfield


}//end function firstLetterFamilyName()
```

Having put this function in hold.js now you need to make some changes to the file *Txt.HTML*.

The code for *Txt.HTML* is shown below.

```
<!DOCTYPE HTML>

<HEAD>
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
</HEAD>

<BODY>

The dog sat on a log.

<hr>
<p>
<form>
<input type ="button"  id = "butt" onclick=" firstLetterFamilyName() "
value ="button"></button> <br>
Initial first name:<br>
<input type="text" name="initial" id = "firstName">

<br>    Family name:<br>
<input type="text" name="familyname" id = "familyName">
</form>

</BODY>
```

</HTML>

You will note only 3 changes, which are:
(i) Changing the name of the function from *cat()* to *firstLetterFamilyName()*
(ii) Giving the text box for first names the id = "*firstName*"
(iii) Giving the text box for family names the id = "*familyName*"

The *id* is critical. By using it JavaScript is able to get the required elements using the method (built in function) *document.getElementById().*

Once the element has been obtained it has a whole lot of *attributes* which can be changed. Please note the following.

It is very easy to write document.getElementById() as document.getElementByID() with a capital D instead of a lower case. If you do this your program won't work. JavaScript is sensitive to lower and upper case.

****

The next example shows how the text box for family name can be altered in a variety of ways.

function changeColorsFamilyName()
{
var nam = prompt("What is your family name?"); //obtains family name

var ob = document.getElementById("familyName");//assigns the textfield with the ID "familyName" to the variable ob

ob.style.backgroundColor="yellow";//changes the color of the actual textbox
ob.style.width="50%";//changes the width of the textbox to 50% of the containing screen
ob.style.color = "red";//changes the font color in the textbox to red
ob.style.height = "200px"; //changes the height(vertical) of the textbox
ob.style.fontSize="24pt"; //changes the font size in the textbox
ob.style.fontFamily="wingdings";//changes the font used in the textbox to wingdings
ob.value = nam;//enters your family name into the textbox

}//end of changeColorsFamilyName().

In the onclick action on the button change *=" firstLetterFamilyName()* to " *changeColorsFamilyName()".*

There are a huge number of attributes you can adjust, far more than in the example.

The next example shows how the button can be altered in a variety of ways. Make sure the button has an ID. I am calling mine *butt*. Create this function

function changeButton()
{
var nam = prompt("What is your favorite animal?"); //obtains animal name
var ob = document.getElementById("butt");//assigns the button with the ID "butt" to the variable ob
ob.style.backgroundColor="blue";//changes the color of the actual button

```
ob.style.width="75%";//changes the width of the button to 75% of the
containing screen
ob.style.color = "white";//changes the font color in the button to white
ob.style.height = "50px"; //changes the height(vertical) of the button
ob.style.fontSize="30pt"; //changes the font size in the button
ob.style.fontFamily="arial";//changes the font used in the button to arial
ob.value = nam;//enters your favorite animal as text on button

}//end of changeButton()
```

Instead of onclick = " *changeColorsFamilyName()"* change it to onclick=" *changeButton()*"

Having dealt with text boxes and buttons let us now have a look at how you might use a dropdown box, a method frequently used for obtaining choices or multi-choice tests.

Let's add a dropdown box to our Txt.HTML. The code is below.

```
<!DOCTYPE HTML>

<HEAD>
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
</HEAD>

<BODY>
```

```
The dog sat on a log.

<hr>
<p>
<form>
<select id="dropp">
<option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
<option value="mercedes">Mercedes</option>
<option value="audi">Audi</option>
</select>
<br>
<input type ="button"  id = "butt" onclick=" alertSelect()" value
="button"></button> <br>
Initial first name:<br>
<input type="text" name="initial" id = "firstName">

<br>    Family name:<br>
<input type="text" name="familyname" id = "familyName">
</form>

</BODY>

</HTML>
```

The function *alertSelect()* replaces *changeButton()* or *changeColorsFamilyName()*.

Here is the code for *alertSelect()*.

```
function alertSelect()
{
var ob = document.getElementById("dropp");
alert(ob.value);
}
```

Open *Txt.HTML* in your favorite browser then click on the button. You should get an alert with the name of the car but in lower case. This is the value of the choice.

As a final demonstration of the Dom model, we will have a look at tables.

We fill demonstrate three uses of JavaScript in dealing with tables.

(1) Given an existing table fill its cells with the members of an array so that one row has yellow writing and another green writing. The background of the table will be blue.
(2) As for (1) but an extra row will be added. The new row will have red writing.
(3) A completely new table of the same dimensions as that in (1) will be created. This is often described as *being created on the fly*. The table will have a black background with white writing.

Here is the code for Txt.HTML with a table and code for (1).

```
<!DOCTYPE HTML>

<HEAD>
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
</HEAD>

<BODY>

The dog sat on a log.

<hr>
<table id="myTable" border="1" >
<tr>
<td >Animals</td>
<td >Continent</td>
<td >Wild/Tame</td>
</tr>
<tr>
<td > </td>
<td > </td>
<td > </td>
</tr>
<tr>
<td > </td>
<td > </td>
<td > </td>
</tr>
```

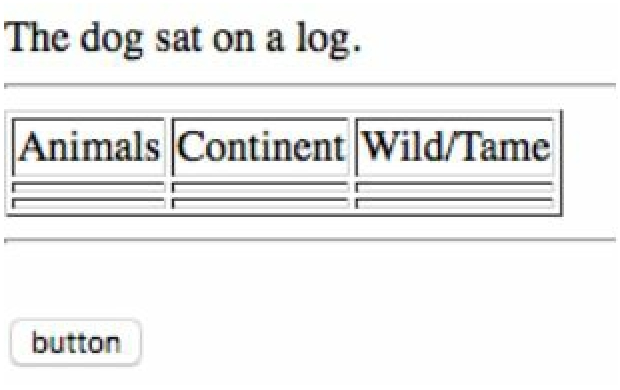
```
</table>
<hr>
<br>
<input type ="button"  id = "butt" onclick=" fillRows1()" value ="button">
</button> <br>

</BODY>
</HTML>
```

Here is the code for *fillRows1()*

```
function fillRows1()
{
document.getElementById('myTable').style.backgroundColor="blue";//chang
background color of table
document.getElementById('myTable').rows[1].style.color="yellow";//change
font color of row 1 to yellow
 document.getElementById('myTable').rows[2].style.color="green";//change
font color of row 2 to green
document.getElementById('myTable').rows[1].cells[0].innerHTML =
"Lion";
document.getElementById('myTable').rows[1].cells[1].innerHTML =
"Africa";
document.getElementById('myTable').rows[1].cells[2].innerHTML = "W";
document.getElementById('myTable').rows[2].cells[0].innerHTML = "Dog";
document.getElementById('myTable').rows[2].cells[1].innerHTML =
"Everywhere";
document.getElementById('myTable').rows[2].cells[2].innerHTML =
```

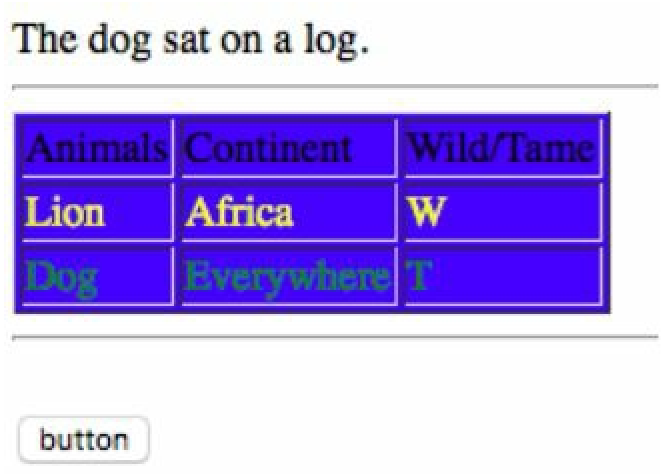
"T";//previous 6 instructions fill cells
}//end fillRows1()

When you open *Txt.HTML* you will get a funny looking little page as below.


The dog sat on a log.

Animals	Continent	Wild/Tame

Clicking on the button activates *fillRows1()* and you get


The dog sat on a log.

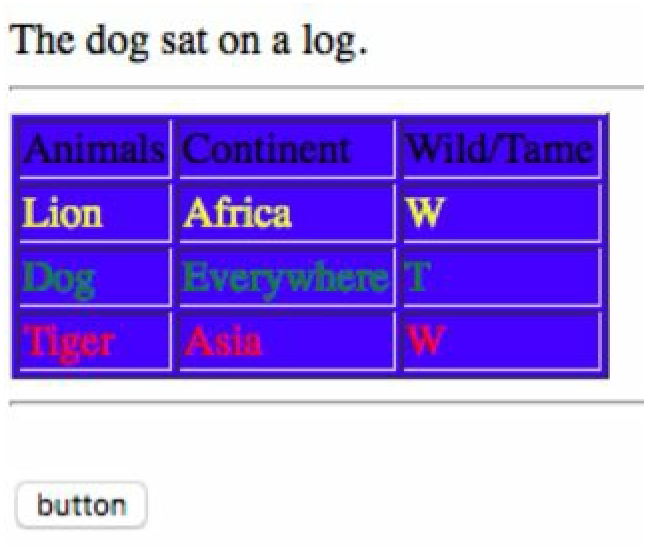
Animals	Continent	Wild/Tame
Lion	Africa	W
Dog	Everywhere	T

What was required has been done for (1). Now to carry out task (2). Take *fillRows1()* and add the following lines of code.

var table = document.getElementById("myTable"); // Find a <table> element with id="myTable"
var row = table.insertRow(3);// Create an empty <tr> element and add it to the 4th row position of the table
row.style.color = "red"; //new row has to have red writing

```
var cell1 = row.insertCell(0);
var cell2 = row.insertCell(1);
var cell3 = row.insertCell(2); // Insert new cells (<td> elements) at the 0,1,2
position of the "new" <tr> element
cell1.innerHTML = "Tiger";
 cell2.innerHTML = "Asia";
cell3.innerHTML = "W"; // Add some text to the new cells
```

Once again open Txt.HTML click the button and now you get the new row with required information in red.



The dog sat on a log.

Animals	Continent	Wild/Tame
Lion	Africa	W
Dog	Everywhere	T
Tiger	Asia	W

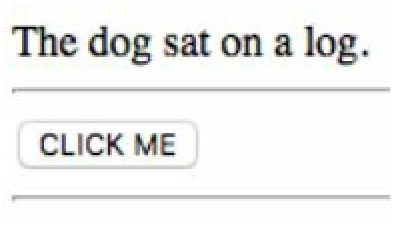
Before proceeding to Task (3) notice that the **value** of textboxes and buttons has been replaced with **innerHTML**. This has caused a lot of grief to many programmers over the years. If you forget just use Google to remind you and so long as you have comments on your previous work you will soon remember.

Task (3) requires the creation of a table. While that is quite straightforward it is a good idea, as preparation, to show something really simple, the creation of a button.

Here is the code for *Txt.HTML*

```
<!DOCTYPE html >
<head>
<title>XYZ</title>
<script src="hold.js"></script>
</head>
<body>
The dog sat on a log.
  <hr>
<script>
/* Create a <button> element*/
var btn = document.createElement("BUTTON");

/*Create a text node*/
 var t = document.createTextNode("CLICK ME");

  /* Append the text to <button>   */
 btn.appendChild(t);

/*Append <button> to <body>*/
 document.body.appendChild(btn);
  </script>
<hr>
</body>
</html>
```

Opening *Txt.HTML* produces the following output.


The dog sat on a log.

Unfortunately clicking on the button produces nothing, as we haven't put in any functions for it to activate. Here is how you could add something to the program so it produced an alert saying cat.

We already have a little function called *cat()* in *hold.js* which does exactly that.

Add these lines of code just below *document.body.appendChild(btn);*

```
/*assigns function cat() to btn onclick event*/
btn.onclick = cat;
```

Now run the program and you will get the required cat alert.

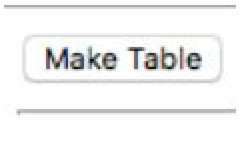
Now to create the table of task (3).

First let's have a new version of *Txt.HTML*

```
<!DOCTYPE HTML>
  <HEAD>
<TITLE> XYZ </TITLE>
  <script src="hold.js"></script>
  </HEAD>
<BODY>
```

```
<input type="button" value="Make Table" onclick="makeTable()" />
<hr />
<div id="dvTable"> </div>
</BODY>
</HTML>
```

Note the use of the <*div*> tag.
As you probably know from HTML a div is basically a holder of content on a web page. We're going to place the table in it after clicking the button. If you open *Txt.HTML* before we have created makeTable() then all you will get is



You can click on the button as much as you like. Nothing happens.

In *hold.js* enter the following function. Carefully read the comments to follow the logic of this program.

```
function makeTable()
 {
/*Array to contain  student  data*/
 var students = new Array();

/*student data entered into array. Note new method called push of entering
into an array. Push is widely used in many computer languages*/
/* When complete we have an array containing student records. This array is
an array of arrays*/
 students.push([" Id", "Name", "Maths"]);    //This is going to be the header
of the table
```

```
 students.push([1, "Mary", 89]);
 students.push([2, "Boris", 65]);
 students.push([3, "Trudy", 53]);
  students.push([4, "Dale", 17]);

/*Now to create table*/
 var table = document.createElement("TABLE");  //Create a HTML Table
element
  table.border = "1";//give the table a border of thickness 1px
table.id = "myTable";// give the table the id ' myTable'

var columnCount = students[0].length;//Get the number of columns in the
table.
var row = table.insertRow(-1);//creates a row to put into the table

for (var i = 0; i < columnCount; i++) {
          var headerCell = document.createElement("TH");
          headerCell.innerHTML = students[0][i];
          row.appendChild(headerCell);
            }   //Add the header row

for (var i = 1; i < students.length; i++)
 {  /*for each student we create a row*/
 row = table.insertRow(-1);
 for (var j = 0; j < columnCount; j++)
           {
            var cell = row.insertCell(-1);
           cell.innerHTML = students[i][j];
```

```
            }//Add the data rows.
}  //Add the students

var dvTable = document.getElementById("dvTable");//get div which has
been created to hold table
dvTable.innerHTML = "";//make the HTML of the div non existent.
Otherwise any HTML would appear before our table.
dvTable.appendChild(table);//put the table in the div

/*change background color of table*/
document.getElementById("myTable").style.backgroundColor="black";

/*changes font color of table*/
document.getElementById("myTable").style.color="white";
}//end of makeTable()
```

At the beginning of this chapter, we mentioned that in addition to the accessing, altering and creation of the elements in HTML we could also destroy them. This is, in actual fact, the easiest thing to do. All you need is the Id of the element to wish to destroy.

Start with *Txt.HTML* as below.

```
<!DOCTYPE HTML>
  <HEAD>
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
</HEAD>
```

```
  <BODY>
  <input type="button" value="Destroy" onclick="removeButton()" id =
"rmv"/>
<hr >
</BODY>
</HTML>
```

Now to create the code for *removeButton()*.

In *hold.js* type the following.

```
function removeButton()
{/* gets button and puts it in variable butn*/
 var butn = document.getElementById("rmv");

butn.remove()// eliminates butn

}//end of removeButton()
```

Open *Txt.HTML* and you will see the button showing "*Destroy*".
Click it and it disappears.

In this chapter, we have given the barest bones of all the amazing things you can do with JavaScript to create dynamic content. The sky is the limit. There is a wealth of information on Google. Any question you have about this can probably be answered.

Some particularly good sites are

https://www.javascript.com/

https://www.w3schools.com/js/

You can also test your knowledge at my site
http://www.nceax.co.nz/Computing/javascriptMultichoice.html

Chapter 6: Problems to do and where to from here

Here are some problems for you to try. If you can do all of them then you have really understood what was taught in the preceding chapters.

Write JavaScript programs which do the following:
(1) Allow you to input a string *str* and a number *n* (whole) then print *str n* times going down the page

(2)Allow you to input certain strings then print a picture with the name of the string inputted. In this problem, you're going to have to get some pictures and name them

(3)Allow you to input certain strings and allow you to input a whole number less than some upper limit then print a sequence of pictures with the name of the string inputted.

(4)Allow you to input a string *str* then print the characters of the string going down the page. So if you inputted ‘CAT' you would output

```
C
A
T
```

(5)Allow you to input a string *str* and a number *n* (whole) then print new strings with the characters of the string *n* times going down the page from 1 up to *n* characters
Example you might input ‘Technologies’ and 6 so the program outputs

```
T
```

```
Te
Tec
Tech
Techn
Techno
```

(6)Do as in 5 only the output would start from the end so if you inputted ‘Technologies’ and 7 you would get

```
s
es
ies
gies
gies
ogies
logies
```

(7)As in (5) and (6) but now the output is

```
Technologies
Technologie
Technologi
Technolog
```

If you inputted ‘Technologies’ and 4

(8)Allows you to input a string and a number n (whole). If the string starts with a vowel alert ‘Vowel’ otherwise prints the string n times going down the page

(9)Allows you to input three strings then alerts you with the string having the largest number of characters.

(10)Allows you to input a number *N* less than 1000, input another number *n* less than 10 then create a table which has rows with *n* cells and is filled with the *N* numbers. As an example, if your value for *N* was 18 and your value for *n* was 5 then the table you create would be

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18		

If there are any problems you can't do go to the email on www.nceax.co.nz and contact the writer of these problems.

Having mastered the JavaScript of this book how far can you go?

The sky is the limit as regards this. Here are some things you can learn how to do:

(i) Learn about timers and how you can use them to make sliders and other picture displays;

(ii) Learn about Object Oriented Programming using JavaScript;

(iii) Learn how to create persistent databases in a web page;

(iv) Learn how to access and manipulate external databases;

(v) Build apps using several different technologies, including Adobe PhoneGap Build;

You can do almost anything with JavaScript.

If there are any problems you can't do go to the email on www.nceax.co.nz and contact the writer of these problems.

Answers to three problems

(3)<u>HTML</u>

```
<!DOCTYPE HTML>
<HEAD>
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
</HEAD>
<BODY>
<hr>
<br>
<form>
<input type ="button"  id = "butt" onclick ="pix()" value ="button">
</button>
  </form>
</BODY>
</HTML>
```

<u>JavaScript</u>

```
function pix()
 {
var numPix = prompt("How many pictures do you want?");  //input number
of pictures wanted
if(numPix>4)// number of pictures must not exceed 4
          {   alert("Try again. The number of pictures must be less than
          5");                }
else
          {
          var pic = prompt("What picture do you want? Input C for cat, M for
          man, R for rectangle.");             //input picture wanted

           if (!((pic=="C")||(pic=="M")||(pic=="R")))//  pictures must be C,M
          or R
                    {   alert("Try again. You must enter C,M or R");     }
          else
                    {
                    if (pic =="C"){ pic ="<img src='  cat.png' alt='cat '
                    width='20%'><br>"}  else if (pic =="M"){pic = "<img src='
                    man.png' alt='man '  width='20%'><br>"}  else {pic = "<img
                    src=' rectangle.png' alt='rectangle '  width='20%'><br>"}//
                    change pic to code for picture

                     for (i = 1;i<=numPix;i++)  {  document.write(pic+"<br>");
                    }// print pictures
              }
          }
}//end of pix()
```

(7) <u>HTML</u>

```
<!DOCTYPE HTML>
<HEAD>
```

```
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
</HEAD>
<BODY>
<input type="button" value="Display words" onclick="showWords()" id =
"showWords"/>
<hr >
<br>
</BODY>
</HTML>
```

<u>JavaScript</u>

```
function showWords()
{
 var word = prompt("What word would you like to see displayed?");//inputs
word
var numberLettersInWord = word.length;//finds number of characters in
word
var numSubStrings = prompt("What is the number of substrings of
"+word+" that you want displayed?");//inputs number of substrings

 if (numSubStrings>numberLettersInWord)   {   alert("You must ask for a
number of substrings which is no more than the number of letters in your
word!");                 }//ensures a number which is not too large is inputted.
else

{
```

```
//Checks could also be put in for other things but we won't bother. You can
research ways you could warn of other invalid inputs

for(i=0;i<=numSubStrings-1;i++)// prints numSubStrings words (note we're
starting from 0)
{ var str = ""; //initially word is empty string

          for(j=0;j<=numberLettersInWord - 1-i;j++)
           { str = str +word.charAt(      j);//build up substring
          }
          document.write(str+"<br>"); //print string just created.<br> moves
          each substring down a line
      } //end of (i=0;i<=numSubStrings-1;i++)
  }//end of else
}//end of showWords()
```

(10) <u>HTML</u>

```
<!DOCTYPE HTML>
  <HEAD>
<TITLE> XYZ </TITLE>
<script src="hold.js"></script>
  </HEAD>
   <BODY>
  <input type="button" value="Display Table" onclick="makeTable1()" id =
"showTable"/>
<hr > <br>
<div id = dvTable></div>
  </BODY>
  </HTML>
```

JavaScript

```
function makeTable1()
 {
var largestNumberShown = prompt("What whole number less than 1000 do
you want in the table?");//input number that is desired

 if(largestNumberShown>=1000)
          {   alert("You must enter a whole number less than 1000");//warning
          if number in excess of 1000 is entered. There could be other checks
          but we won't bother
          }
else
          {
           var numberColumns = prompt("How many columns do you want in
          your table? Enter a whole number less than 13 but more than 4");

          //could put checks in here to make sure restrictions on columns were
          adhered to

            /*The 2 lines below construct enough rows to accommodate all the
          numbers. Needs some knowledge of mathematics */

          var numberRows = ((largestNumberShown -
          largestNumberShown%numberColumns)/numberColumns + 1);
/* Create a HTML Table element     */
  var table = document.createElement("TABLE");
   table.border = "1";
 table.id = "myTable";

/*Be careful with { and }*/
    for (var i = 0; i < numberRows; i++)
          {
            row = table.insertRow(-1);//insert rows

                      for (var j = 1; j <=numberColumns; j++)
                                {   var cell = row.insertCell(-1);

                                if(numberColumns*i+j<=largestNumberShown)
                                          {cell.innerHTML =
                                          numberColumns*i+j;}//puts correct
                                          numbers in cells. Needs some knowledge of
                                          maths         }//Add the data rows.

                                     }// for (var j = 1; j <=numberColumns; j++)

var dvTable = document.getElementById("dvTable");//get div which
has been created to hold table
dvTable.innerHTML = "";//make the HTML of the div non existent.
Otherwise any HTML would appear before our table.

dvTable.appendChild(table);//put the table in the div
   }
```

```
}//end of makeTable1()
```

In addition to these problems you should also make sure you can do the multi-choice questions at
http://www.nceax.co.nz/Computing/javascriptMultichoice.html
By the end of 2017, the solutions to all problems in the last chapter of this book will be in

http://www.nceax.co.nz/Computing

Be sure to make this site a bookmark!

Conclusion

JavaScript is a computer language.

If you wish to master it you must know its basics inside and out.

This book has thoroughly covered most of those basics.

Before you proceed further with this language you must master the material in the book to the extent that you can do the problems in the last chapter.

Once you have mastered the material in this book you are ready for all the other things that can be done with the JavaScript programming language!

Good Luck.