Introduction to Programming

Andrew Butterfield

Department of Computer Science
School of Computer Science and Statistics
Trinity College, Dublin

November 1st, 2007
1 Introduction

2 Variables and Values
   • Functions
   • Boolean values
   • Executive Summary

3 Flow Control
   • Loops
As well as operators we can use *functions* in expressions. A function has a name (like a variable name), and a list of *parameters* (sometimes called *arguments*). The general form of a function is:

```
function name(argument1, argument2)
```

There may be zero or more arguments (but the parentheses always appear. You should recognise this form – *alert* is actually a kind of function!

Functions generally *return* (that is, evaluate to) some value.

Shortly you will be adding your own functions to the collection that comes built in to Javascript. For today we will just look at some of the pre-supplied ones.
Some numeric functions

`min` and `max` each take two numbers and return the largest:

```javascript
var bigOne = max(10, 20);
```

`abs` will give the magnitude of a number. This example will write 20 into the document:

```javascript
document.write( abs(-20) );
```

The parameters are values, so we can use expressions and variables instead of literals:

```javascript
alert("The smaller number is: "+min( 2*3, 100/2 ) );
```
Some numeric functions

min and max each take two numbers and return the largest:

var bigOne = max(10,20);

abs will give the magnitude of a number. This example will write 20 into the document:

document.write( abs(-20) );

The parameters are values, so we can use expressions and variables instead of literals:

alert("The smaller number is: "+min( 2*3, 100/2 ) );
Some numeric functions

min and max each take two numbers and return the largest:

var bigOne = max(10,20);

abs will give the magnitude of a number. This example will write 20 into the document:

document.write( abs(-20) );

The parameters are values, so we can use expressions and variables instead of literals:

alert("The smaller number is: "+min( 2*3, 100/2 )");
There are many string functions available - most of them use a slightly different syntax:

```javascript
string.function(argument1,argument2)
```

In other words the string itself actually comes before the function name.

For example, the `toUpperCase` converts all the letters in the string to uppercase:

```javascript
var x = "Some uppercase, Some LoWerCase";
alert( x.toUpperCase() );
```

(there is a `toLowerCase` function as well. Perhaps you can guess what it does?)
The `substring` function takes a slice out of a string: 
"contains three words".substring(9,14)" is the string "three". As with many things in computing we count from 0, not 1 (think of it as an offset from the start).

If you want just one one specific letter (‘t’, say) you can say:

"three".charAt(0)

For every string there is also a variable you can look up to find out how many characters there are in a string: "name".length is 4, for example.
The substring function takes a slice out of a string: "contains three words".substring(9,14)" is the string "three". As with many things in computing we count from 0, not 1 (think of it as an offset from the start).

If you want just one specific letter (‘t’, say) you can say:

"three".charAt(0)

For every string there is also a variable you can look up to find out how many characters there are in a string: "name".length is 4, for example.
The `substring` function takes a slice out of a string: "contains three words".substring(9,14)" is the string "three". As with many things in computing we count from 0, not 1 (think of it as an offset from the start).

If you want just one one specific letter (‘t’, say) you can say:

"three".charAt(0)

For every string there is also a variable you can look up to find out how many characters there are in a string: "name".length is 4, for example.
Booleans

Javascript actually has three types of basic variables. As well as String and Number there are the two Boolean values: true and false:

```javascript
var raining = true;
```

Boolean values represent logical truth. Rather than displaying them directly to the user they are most commonly used to control decision points in the program.

A number of operators can be used to calculate Boolean values, for example:

- The equality test: `==` (for example: `var x = ("yah"=="boo")`)
- Negated equality is written `!=` (pronounce “not equal to”).
- Inequalities: `<`, `>`, `<=`, `>=` (example: `x > 12`).

Hint: a common error to watch out for is confusing the assignment statement with the equality operator: `==`. 
Booleans

Javascript actually has three types of basic variables. As well as String and Number there are the two Boolean values: true and false:

var raining = true;

Boolean values represent logical truth. Rather than displaying them directly to the user they are most commonly used to control decision points in the program.

A number of operators can be used to calculate Boolean values, for example:

- The equality test: \( \text{==} \) (for example: \( \text{var x = ("yah"=="boo")} \))
- negated equality is written \( \text{!=} \) (pronounce “not equal to”).
- Inequalities: \( <, >, <=, >= \) (example: \( x > 12 \)).

Hint: a common error to watch out for is confusing the assignment \( = \) statement with the equality operator: \( == \).
Javascript actually has three types of basic variables. As well as String and Number there are the two Boolean values: true and false:

```javascript
var raining = true;
```

Boolean values represent logical truth. Rather than displaying them directly to the user they are most commonly used to control decision points in the program.

A number of operators can be used to calculate Boolean values, for example:

- The equality test: `==` (for example: `var x = ("yah"=="boo")`)
- Negated equality is written `!=` (pronounce “not equal to”).
- Inequalities: `<`, `>`, `<=`, `>=` (example: `x > 12`).

Hint: a common error to watch out for is confusing the assignment `=` statement with the equality operator: `==`. 
The Conditional Statement (I)

We can choose between executing different statements using the conditional. Here’s an example (it assumes the variable `age` has been assigned a value somehow):

```javascript
if( age < 18 ) {
    alert("Leave the bar area immediately!");
} else {
    alert("Welcome in");
}
```

The statement chooses between two blocks of statements based on the truth of the Boolean value between the first parentheses (the condition).

The else clause in the statement is optional – sometimes we don’t need it, and in those cases we can leave it out.
We can choose between executing different statements using the conditional. Here’s an example (it assumes the variable age has been assigned a value somehow):

```javascript
if( age < 18 ) {
    alert("Leave the bar area immediately!");
}
```

The statement chooses between two blocks of statements based on the truth of the Boolean value between the first parentheses (the condition).

The `else` clause in the statement is optional – sometimes we don’t need it, and in those cases we can leave it out.
You can write several statements within each block, including more if statements:

```javascript
if( location == "Ireland" ) {
  if( month != "August" ) {
    document.write("Probably raining, then.");
    raining = true;
  }
}
```

We can often avoid more complex nestings like this by using boolean operators in the condition instead:

```javascript
if( (location=="Ireland") && (month != "August") ){
  document.write("Probably raining, then.");
  raining = true;
}
```
Logical conjunction: $a \land b$ is true if both $a$ and $b$ are true, and false if either is false (pronounce $\land$ “and”).

Logical disjunction: $a \lor b$ is true if either of $a$ or $b$ are true (pronounce it “or”).

Logical negation: $\neg a$ has the opposite truth value to $a$ (pronounce it “not”).

For example, to test a number $x$ to see that it is larger than 5 and smaller than 10 we can write:

$$(x > 5) \land (x < 10)$$
Boolean values

Boolean operators

- Logical conjunction: \texttt{a && b} is true if both \( a \) and \( b \) are true, and false if either is false (pronounce \texttt{&&} “and”).
- Logical disjunction: \texttt{a || b} is true if either of \( a \) or \( b \) are true (pronounce it “or”).
- Logical negation: \texttt{!a} has the opposite truth value to \( a \) (pronounce it “not”).

For example, to test a number \( x \) to see that it is larger than 5 and smaller than 10 we can write:

\[(x > 5) \land (x < 10)\]
Boolean values

Logical conjunction: \( a \land b \) is true if both \( a \) and \( b \) are true, and false if either is false (pronounce \( \land \) “and”).

Logical disjunction: \( a \lor b \) is true if either of \( a \) or \( b \) are true (pronounce it “or”).

Logical negation: \( \neg a \) has the opposite truth value to \( a \) (pronounce it “not”).

For example, to test a number \( x \) to see that it is larger than 5 and smaller than 10 we can write:

\[(x > 5) \land (x < 10)\]
Variables are named storage locations; introduce them with `var`.

The value stored in a variable can be changed with the assignment statement (for instance, `x = 10;`).

Expressions can be used to calculate values (which may be used directly, or stored in variables for later use).

The type of a variable is significant. Javascript offers numbers, strings and booleans.

Boolean values may be used to make decisions in a program, via the `if` statement.
Variables are named storage locations; introduce them with `var`.

The value stored in a variable can be changed with the assignment statement (for instance, `x = 10;`).

Expressions can be used to calculate values (which may be used directly, or stored in variables for later use).

The type of a variable is significant. Javascript offers numbers, strings and booleans.

Boolean values may be used to make decisions in a program, via the `if` statement.
Variables are named storage locations; introduce them with `var`.

The value stored in a variable can be changed with the assignment statement (for instance, `x = 10;`).

Expressions can be used to calculate values (which may be used directly, or stored in variables for later use).

- The type of a variable is significant. Javascript offers numbers, strings and booleans.
- Boolean values may be used to make decisions in a program, via the `if` statement.
Variables are named storage locations; introduce them with `var`.

The value stored in a variable can be changed with the assignment statement (for instance, `x = 10;`).

Expressions can be used to calculate values (which may be used directly, or stored in variables for later use).

The type of a variable is significant. Javascript offers numbers, strings and booleans.

Boolean values may be used to make decisions in a program, via the `if` statement.
Variables are named storage locations; introduce them with `var`.

The value stored in a variable can be changed with the assignment statement (for instance, `x = 10;`).

Expressions can be used to calculate values (which may be used directly, or stored in variables for later use).

The type of a variable is significant. Javascript offers numbers, strings and booleans.

Boolean values may be used to make decisions in a program, via the `if` statement.
The if statement from the previous section controls the flow of the program execution. It allows us to choose whether certain sections of the program will be executed or not.

Another control structure that uses boolean values to control the execution is a loop.

A loop is a way of managing repetition in a program.

- As long as there are matches in the box, light a match.
- For each book, count the pages

Loops are generally controlled by conditions, much like if statements. Instead of selecting between blocks to be performed a loop condition controls whether a block is repeated again (and again, and again, and again...)
One of the simplest, but most powerful, loop constructs in javascript is the **while loop**:

```javascript
while( condition ){
    statement;
}
```

The block containing the statements will be executed zero or more times, according to the following simple rules:

- Test the condition. If it is true, execute the statement and *do this step again*
- If the condition is false then proceed to the next statement after the while.
Since the condition is tested *before* we perform the block ("loop body") it may be that we never perform it! This is OK, it just means that conditions in the program were such that we didn’t need to loop.

**Important:** Something in the loop body must eventually cause the condition to become false. Without this *the statement will never end!* This is called an “infinite loop”.
A common pattern in loops is to repeat something a specific number of times. We can use a numeric variable to control this; here is a loop that prints ten rows of a multiplication table:

```javascript
var counter = 1;
while( counter <= 12 ){
    document.write("<br />7 times "+counter+" = "+(counter*7) );
    counter = counter + 1;
}
```

And the output:

7 times 1 = 7  
7 times 2 = 14  
7 times 3 = 21  
7 times 4 = 28  

(and so on...)  experiment: remove the line in the body that changes counter.  
What happens? try this in mozilla/firefox
A common pattern in loops is to repeat something a specific number of times. We can use a numeric variable to control this; here is a loop that prints ten rows of a multiplication table:

```javascript
var counter = 1;
while( counter <= 12 ){
    document.write("<br />7 times "+counter+" = "+(counter*7) );
    counter = counter + 1;
}
```

And the output:

7 times 1 = 7  
7 times 2 = 14  
7 times 3 = 21  
7 times 4 = 28  

(and so on...)  

experiment: remove the line in the body that changes counter.  
What happens? try this in mozilla/firefox
A common pattern in loops is to repeat something a specific number of times. We can use a numeric variable to control this; here is a loop that prints ten rows of a multiplication table:

```javascript
var counter = 1;
while( counter <= 12 ){
    document.write("<br />7 times "+counter+" = "+(counter*7) ");
    counter = counter + 1;
}

And the output:

7 times 1 = 7
7 times 2 = 14
7 times 3 = 21
7 times 4 = 28

(and so on...) experiment: remove the line in the body that changes counter.
What happens? try this in mozilla/firefox
Another common looping pattern is to repeat the code an arbitrary number of times, until some condition is satisfied. In the lab last week we had to accept whatever input the user entered. Here is how we could have checked the input to see that it was in the acceptable range for guesses in the game:

```javascript
guess = prompt("Enter a number from 1--10");
while( (guess<0) || (guess>10) ){
    guess = prompt("No, between 1 and 10, please.\nTry again")
}
```

The condition reads like this:

*If the number is less than zero or the number is greater than ten then perform the loop body, and try again.*

We don’t know how often the loop will repeat, but eventually the user should enter some value that is in the desired range and we can continue with the rest of the program.
Another common looping pattern is to repeat the code an arbitrary number of times, until some condition is satisfied. In the lab last week we had to accept whatever input the user entered. Here is how we could have checked the input to see that it was in the acceptable range for guesses in the game:

```javascript
guess = prompt("Enter a number from 1--10");
while( (guess<0) || (guess>10) ){
    guess = prompt("No, between 1 and 10, please.\nTry again")
}
```

The condition reads like this:

*If the number is less than zero or the number is greater than ten then perform the loop body, and try again.*

We don’t know how often the loop will repeat, but eventually the user should enter some value that is in the desired range and we can continue with the rest of the program.
When code like this appears in a program it is useful to add a note to a programmer reading it later. You can insert a comment in a javascript program like this:

```javascript
guess = prompt("Enter a number from 1--10");

// Validate the input
while( (guess<0) || (guess>10) ){
    guess = prompt("No, between 1 and 10, please.\nTry again")
}
```

Everything from the // characters to the end of the line is ignored by javascript – it’s assumed to be some english commentary on the code. Use these liberally! Comments allow you to annotate your program so that you can read it later on.
To summarize our flow control statements so far:

- The *conditional* statement *if* allows us to choose between two blocks of code, based on a boolean condition.
- The *while* statement allows us to repeat a block of code based on the value of some boolean condition.

With these statements we will be able to write some very sophisticated programs. In fact, from here on *everything* is a matter of finding more convenient and appropriate ways to express our ideas.
Summary

To summarize our flow control statements so far:

- The *conditional* statement *if* allows us to choose between two blocks of code, based on a boolean condition.
- The *while* statement allows us to repeat a block of code based on the value of some boolean condition.

With these statements we will be able to write some very sophisticated programs. In fact, from here on *everything* is a matter of finding more convenient and appropriate ways to express our ideas.
To summarize our flow control statements so far:

- The *conditional* statement `if` allows us to choose between two blocks of code, based on a boolean condition.

- The `while` statement allows us to repeat a block of code based on the value of some boolean condition.

With these statements we will be able to write some very sophisticated programs. In fact, from here on *everything* is a matter of finding more convenient and appropriate ways to express our ideas.
To summarize our flow control statements so far:

- The *conditional* statement *if* allows us to choose between two blocks of code, based on a boolean condition.
- The *while* statement allows us to repeat a block of code based on the value of some boolean condition.

With these statements we will be able to write some very sophisticated programs. In fact, from here on *everything* is a matter of finding more convenient and appropriate ways to express our ideas.
To summarize our flow control statements so far:

- The *conditional* statement `if` allows us to choose between two blocks of code, based on a boolean condition.
- The `while` statement allows us to repeat a block of code based on the value of some boolean condition.

With these statements we will be able to write some very sophisticated programs. In fact, from here on *everything* is a matter of finding more convenient and appropriate ways to express our ideas.
To summarize our flow control statements so far:

- The *conditional* statement *if* allows us to choose between two blocks of code, based on a boolean condition.

- The *while* statement allows us to repeat a block of code based on the value of some boolean condition.

With these statements we will be able to write some very sophisticated programs. In fact, from here on *everything* is a matter of finding more convenient and appropriate ways to express our ideas.