LBSC 690: Information Technology
Lecture 07
Programming and Javascript

William Webber
CIS, University of Maryland

Spring semester, 2012

Programming languages: the machine

- The language understood by the machine is binary (machine code):

  00000000 11001000 01001000 00100000

- This can be transliterated to and from a human-readable language (assembly language):

  add $4, $6, $8

- But that assembly language is:
  - too low-level
  - not portable (each processor type has its own assembly language)

  and so is rarely written directly today

Higher level programming languages

- Modern programmers work in one or more higher level languages (HLL)
- Different HLL offer different features, abstractions, speeds, portabilities
- But every HLL must be translated into machine language for the machine to run it
- There are two processes by which translation can take place:
  - Compilation
  - Interpretation

Compilation and compiled languages

- In compilation, a program written in one language is converted in its entirety to another language before it is run
- Think of translating a book from English to German
- Compilation may be:
  - To machine code
  - To another (generally lower level) HLL
  - To an intermeidate representation

  The latter two representations then need to be further translated, either through compilation or interpretation (see next) into machine code

Interpretation and interpreted languages

- In intepretation, when a program is being run, it is read by another program, called an interpreter . . .
- and the intepreter executes the instructions line by line using machine code
- Think of interpreting an English speaker to a German listener, a sentence at a time

Compilation versus interpretation

Compiled languages:

- Are faster (can be up to 100 times faster)
- Allow error checking before program is run

Interpreted languages:

- Are more flexible
- Do not need separate compilation step
- Are generally more portable

BUT distinction between the two somewhat blurred (e.g. Java can be intepreted or compiled; many “interpreted” languages are first compiled into an intermediate representation (byte code)).

Javascript

We will be looking at Javascript

- Developed by Netscape programmers in mid 1990s
- Implemented in all modern web browsers
- Mainly used for adding automation to web pages:
  - checking forms for errors before submission
  - animating web banners and other toys
  - implementing desktop-like rich interfaces (e.g. Gmail)
- But can also be used as a general purpose programming language

Working with Javascript

Two environments for experimenting with javascript:

- Online Javascript console: `http://jsconsole.com`. For typing simple examples and seeing their result.
- Embedding your program in an HTML file:

```
<html><body><script language="javascript">
document.write(2 + 2);
</script></body></html>
```

  and loading the file up in your browser.
- For latter, use Firefox's "Tools > Web developer > Error console"

A simple example

```
<html><body><script language="javascript">
var i = 0;
var val = 1;

while (i < 10) {
    i = i + 1;
    val = val * 2;
    document.write("" + i + ": " + val + "<br>\n");
}
</script></body></html>
```

- A program is a series of commands (statements) to the computer (the Javascript interpreter in the browser)
- The interpreter executes these commands one line at a time
- We use document.write() to cause the interpreter to write output to the browser window

Expressions

- Basic unit is an expression, which has a value
- Examples expressions and their values are:

Expression	Value
`2`	`2`
”cat”	`"cat"`
`10 + 2 * 5`	`20`
`3 > 2`	`true`

- *Type these into the Javascript console and see the results*

Types

- Values have types
- There are six basic types in Javascript:

Type	Example	Description
Number	2	Numerical value (signed, fractional)
String	"Hello"	Text (note the quotes)
Boolean	$3 > 2$	True or false
Function	document.write()	Executes a group of code
Object	{"name": "Eve", "age": 6}	Aggregates compound values
Undefined	undefined	Special value for undefined variables (see below)

Operators

- Operators combine values to create new values
- Numeric operators, as you'd expect: $2 + 2$, $5 - 2.1$
- Boolean operators, test for a condition, resolve to true or false: $3 > 5$
- String operators, concatenate two strings: "Hello, " + "world" $\rightarrow$ `"Hello, world"`
- Brackets can be used to specify precedence: $((3 + 3) > 8) \;||\; !((5 * 1) < 6)$
  - $||$ OR's two Boolean expressions, && AND's them, NOT's a single expression

Statements

- A statement contains an expression, and ends with an “;”.
- While an expression has a value, a statement is executed for its side-effect:
  - Print out a value to the screen
  - Save a value to a database
  - Assign a value to a variable (see next)
- A program is made up of a sequence of statements

Variables

```
var name = "John Smith";
document.write(name + "<br">);
name = name + ", Junior";
document.write(name + "<br">);
```

- Variables allow us to capture the value of an expression for later reuse.
- A variable gives a “name” to a value
- This name can be reassigned later.

Conditionals

```
var a = 3;
if (a < 5) {
  document.write("a is small!<br");
} else if (a < 10) {
  document.write("a is middle-sized!<br");
} else {
  document.write("a is big!<br");
}
```

- `if` statement tests a Boolean condition, executes block of code only if true.
- `else` statement is executed if condition is false.
- `else if` can be used to chain `if` conditionals

Loops

```
var i = 0;
while (i < 10) {
  document.write(i + "<br>");
  i = i + 1;
}
```

```
for (var i = 0; i < 10; i++) {
  document.write(i + "<br>");
}
```

- A loop statement (`while()` or `for()`) executes a block repeatedly as long as a conditional statement is true
- `for()` is a short-hand for a common case of `while()`; the above two code segments are equivalent

Revisiting a simple example

```
<html><body><script language="javascript">
var i = 0;
var val = 1;

while (i < 10) {
    i = i + 1;
    val = val * 2;
    document.write("" + i + ": " + val + "<br>\n");
}
</script></body></html>
```

Read through this program and try to figure out what it does.

Calling functions

```
document.write("Hello, world!");
var a = Math.log(1024, 2);
```

- Functions encapsulate a set of statements to provide reusable functionality.
- They may be called with arguments, and may return values.
- Some functions are called primarily for their side effects (`document.write()`).
- Other functions are called primarily for the value they return (`Math.log()`).

Defining functions

```
function add(a, b) {
    return a + b;
}
```

- Functions are defined using the `function` keyword ...
- followed by the name of the function ...
- and a list of the function's parameters

Function parameters and return

```
function add(a, b) {
    return a + b;
}
document.write(add(2, 5));
```

- Function parameters act as variables, but are visible only inside the function (in computer jargon, they're local variables)
- When a function is called, its parameters are set to the calling arguments
- The returned value (if any) is returned via the `return` statement
- In the calling code, the function evaluates to its returned value

An example of defining functions

```
<html><body><script language="javascript">
function nextSquare(n) {
    if (n < 0) {
        return 0;
    }
    for (i = 1; i * i <= n; i++)
        ;
    return i * i;
}
function displayNextSquare(n) {
    document.write("The␣next␣square␣after␣"
      + n + "␣is␣" + nextSquare(n) + "<br>");
}

displayNextSquare(10);
displayNextSquare(102145);
</script></body></html>
```

Useful input and output functions

```
<html><body><script language="javascript">
var n = Number(prompt("Please enter a number"));
alert("The square of " + n + " is " + n * n);
</script></body></html>
```

- `alert(MESSAGE)` writes a message in a pop-up window.
- `prompt(MESSAGE)` write a message, asks the user to enter some text, and returns the text that was entered.
- Note that `Number()` takes a string and converts it to a number.

Objects

```
var user = { "name" : "Peter", "age" : 24,
      "login" : "pete" };
user.age = 25;
document.write("The user is " + user.name +
      ", age " + user.age + "<br>");
```

- An “Object” is a composite value.
- Object has named properties, each of which has a value.
- Property `foo` of object `bar` is accessed as `foo.bar`, or `foo["bar"]`
- Note that when we say `document.write`, we are accessing the `write` property (a function) of the `document` object.

Arrays

```
var words = [ "one", "fish", "two", "fish" ];
for (var i = 0; i < words.length; i++) {
    document.write(words[i] + "<br>");
}
```

- A special type of object is an array.
- Properties (keys, slots) of array are all sequential numbers.
- First slot is number 0, second 1, and so forth.