Macro languages

Purpose: to improve readability of other code, abstraction, textual manipulation

Examples: Cpp, LaTeX, M4, macros in Lisp

Notable features: usually lexical (character or text) based, with some exceptions (Lisp, Rust)

Macro languages

Feet

- LaTeX:

```
\documentclass[12pt]{article}
\usepackage{latexgun,latexshoot}
\begin{document}
See how easy it is to shoot yourself in the foot? \\
\gun[leftfoot]{shoot} \\
\pain
\end{document}
```

Macro languages

These are used widely, in a huge variety of contexts

Macro languages

These are used widely, in a huge variety of contexts

Often used in conjunction with another language, e.g., the C preprocessor

Macro languages

These are used widely, in a huge variety of contexts

Often used in conjunction with another language, e.g., the C preprocessor

So not often thought about in great detail, but are used to great effect

Macro languages

These are used widely, in a huge variety of contexts

Often used in conjunction with another language, e.g., the C preprocessor

So not often thought about in great detail, but are used to great effect

Particularly *conditional macros* whose expansion depends on other factors

Macro languages

So, in C, we can write code like

```
#ifdef SMALLINT
#define NUMBER short
#else
#define NUMBER int
#endif
```

Macro languages

So, in C, we can write code like

```
#ifdef SMALLINT
#define NUMBER short
#else
#define NUMBER int
#endif
```

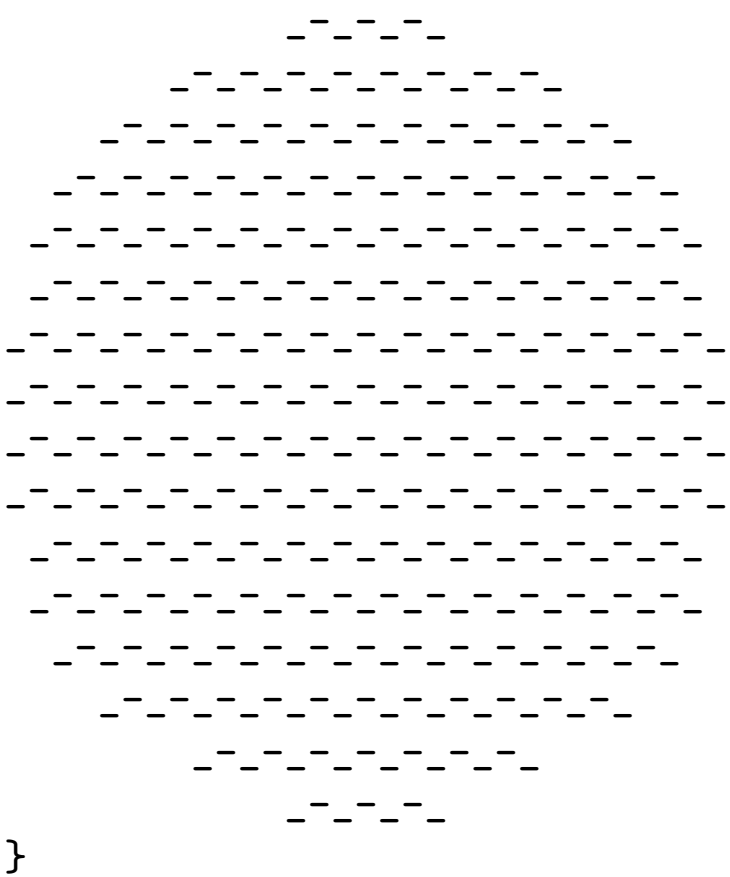
Then if we use `NUMBER` everywhere in our code

```
NUMBER x;
...
```

it takes only a single change to make our code use `short` rather than `int`: very useful for source code portability between architectures

C

```
#define _ F-->00||-F-OO--;
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
```



```
}
```

An enthusiastic use of C macros by Brian Westley

Macro languages

In contrast to most languages, Lisp macros are not text based, but expression based

Macro languages

In contrast to most languages, Lisp macros are not text based, but expression based

And the macro language is Lisp itself, not a separate language: recall that programs are data!

Macro languages

In contrast to most languages, Lisp macros are not text based, but expression based

And the macro language is Lisp itself, not a separate language: recall that programs are data!

```
(defmacro head (l) (list 'car l))
```

Macro languages

In contrast to most languages, Lisp macros are not text based, but expression based

And the macro language is Lisp itself, not a separate language: recall that programs are data!

```
(defmacro head (l) (list 'car l))
```

`defmacro` defines the code to be executed (in the compiler or interpreter) when expanding the macro named `head`

Macro languages

In contrast to most languages, Lisp macros are not text based, but expression based

And the macro language is Lisp itself, not a separate language: recall that programs are data!

```
(defmacro head (l) (list 'car l))
```

`defmacro` defines the code to be executed (in the compiler or interpreter) when expanding the macro named `head`

When compiling `(head x)`, the compiler first expands the macro `head` by executing `(list 'car l)`, where `l` has the value `x`, to get `(car x)`

Macro languages

In contrast to most languages, Lisp macros are not text based, but expression based

And the macro language is Lisp itself, not a separate language: recall that programs are data!

```
(defmacro head (l) (list 'car l))
```

`defmacro` defines the code to be executed (in the compiler or interpreter) when expanding the macro named `head`

When compiling `(head x)`, the compiler first expands the macro `head` by executing `(list 'car l)`, where `l` has the value `x`, to get `(car x)`

The compiler then compiles `(car x)`

Macro languages

The full power of Lisp applies to macroexpansion

Macro languages

The full power of Lisp applies to macroexpansion

```
(defmacro baz (l m)
  (if (> m 0)
      (list 'car l)
      (list 'cdr l)))
```

Will expand `baz` at compile time to a `car` or a `cdr`

Macro languages

LaTeX

Sometimes macros are used in their own right, e.g., LaTeX, the typesetting language is macro based

Macro languages

LaTeX

Sometimes macros are used in their own right, e.g., LaTeX, the typesetting language is macro based

These slides are written in LaTeX

Macro languages

LaTeX

Sometimes macros are used in their own right, e.g., LaTeX, the typesetting language is macro based

These slides are written in LaTeX

```
\begin{frame}
\frametitle{Macro languages}
\framesubtitle{\LaTeX}

Sometimes macros are used in their own right, e.g.,
\LaTeX, the typesetting language is macro based

\paruncover2{These slides are written in \LaTeX}
\end{frame}
```

Macro languages

LaTeX

The basic datatype is text and you define macros as convenient shorthands for things you like to do to that text

Macro languages

LaTeX

The basic datatype is text and you define macros as convenient shorthands for things you like to do to that text

For example, a new chapter needs a new page, a big heading, lots of space before the next text

Macro languages

LaTeX

The basic datatype is text and you define macros as convenient shorthands for things you like to do to that text

For example, a new chapter needs a new page, a big heading, lots of space before the next text

So you define a macro, say `\chapter`, that does all this

Macro languages

LaTeX

The basic datatype is text and you define macros as convenient shorthands for things you like to do to that text

For example, a new chapter needs a new page, a big heading, lots of space before the next text

So you define a macro, say `\chapter`, that does all this

Usually, someone else has created an entire macro package so you don't have to

Scripting Languages

Purpose: control of other elements of a system, e.g., programs, “glue” to join elements together

Examples: DOS batch, sh, Python, Sed, Perl, Ruby, JavaScript . . .

Notable features: not particularly good at classical number crunching; generally lots of string processing and process manipulation

Scripting Languages

Purpose: control of other elements of a system, e.g., programs, “glue” to join elements together

Examples: DOS batch, sh, Python, Sed, Perl, Ruby, JavaScript . . .

Notable features: not particularly good at classical number crunching; generally lots of string processing and process manipulation

They are called “scripts” as they are (were originally) lists of things to be done

Scripting Languages

Purpose: control of other elements of a system, e.g., programs, “glue” to join elements together

Examples: DOS batch, sh, Python, Sed, Perl, Ruby, JavaScript . . .

Notable features: not particularly good at classical number crunching; generally lots of string processing and process manipulation

They are called “scripts” as they are (were originally) lists of things to be done

Often, but not exclusively, interpreted rather than compiled

Scripting Languages

Feet

- DOS batch: You aim the gun at your foot and pull the trigger, but only a weak gust of warm air hits your foot

Scripting Languages

Feet

- DOS batch: You aim the gun at your foot and pull the trigger, but only a weak gust of warm air hits your foot
- Sh, csh, bash: You can't remember the syntax for anything so you spend five hours reading man pages then your foot falls asleep. You then shoot the computer and switch to Perl

Scripting Languages

Feet

- DOS batch: You aim the gun at your foot and pull the trigger, but only a weak gust of warm air hits your foot
- Sh, csh, bash: You can't remember the syntax for anything so you spend five hours reading man pages then your foot falls asleep. You then shoot the computer and switch to Perl
- Perl: You shoot yourself in the foot, but nobody can understand how you did it. Six months later, neither can you

Scripting Languages

Feet

- DOS batch: You aim the gun at your foot and pull the trigger, but only a weak gust of warm air hits your foot
- Sh, csh, bash: You can't remember the syntax for anything so you spend five hours reading man pages then your foot falls asleep. You then shoot the computer and switch to Perl
- Perl: You shoot yourself in the foot, but nobody can understand how you did it. Six months later, neither can you
- Ruby: Your foot is ready to be shot in roughly five minutes, but you just can't find anywhere to shoot it

Scripting Languages

Feet

- DOS batch: You aim the gun at your foot and pull the trigger, but only a weak gust of warm air hits your foot
- Sh, csh, bash: You can't remember the syntax for anything so you spend five hours reading man pages then your foot falls asleep. You then shoot the computer and switch to Perl
- Perl: You shoot yourself in the foot, but nobody can understand how you did it. Six months later, neither can you
- Ruby: Your foot is ready to be shot in roughly five minutes, but you just can't find anywhere to shoot it
- JavaScript: You've perfected a robust, rich user experience for shooting yourself in the foot. You then find that bullets are disabled on your gun

Scripting Languages

Also widely used

Scripting Languages

Also widely used

The original scripts were the job control languages for the early mainframes

Scripting Languages

Also widely used

The original scripts were the job control languages for the early mainframes

For example, IBM's JCL was used to describe a list of programs to be loaded and run: recall batch processing

Scripting Languages

Feet

- JCL: You send your foot down to MIS with a 4000-page document explaining how you want it to be shot. Three years later, your foot comes back deep-fried

Scripting Languages

Sh

The Unix command line language, originally sh (the Bourne Shell), lately bash (the Bourne Again shell)

Scripting Languages

Sh

The Unix command line language, originally sh (the Bourne Shell), lately bash (the Bourne Again shell)

Simple textual lists of things (programs) to be done

Scripting Languages

Sh

The Unix command line language, originally sh (the Bourne Shell), lately bash (the Bourne Again shell)

Simple textual lists of things (programs) to be done

```
#!/bin/sh
setxkbmap -option "compose:menu" -option "ctrl:nocaps"
dispwin -L
[ "$XAUTHORITY" ] && cp -f "$XAUTHORITY" ~/.Xauthority
```

Scripting Languages

Sh

Shell scripts are widely used to automate repetitive and complex tasks

Scripting Languages

Sh

Shell scripts are widely used to automate repetitive and complex tasks

Originally reasonably simple but they have grown more complex over time

Scripting Languages

Sh

Shell scripts are widely used to automate repetitive and complex tasks

Originally reasonably simple but they have grown more complex over time

Crucially, they do not require a GUI so can be deployed automatically over large numbers of machines

Scripting Languages

Sh

Shell scripts are widely used to automate repetitive and complex tasks

Originally reasonably simple but they have grown more complex over time

Crucially, they do not require a GUI so can be deployed automatically over large numbers of machines

Somewhat low-level, so not so good for more complex tasks, or less complex programmers

Scripting Languages

Sed

Example: Sed. A very simple scripting language, capable of just one task

Scripting Languages

Sed

Example: Sed. A very simple scripting language, capable of just one task

String manipulation: sed is a *stream editor*

Scripting Languages

Sed

Example: Sed. A very simple scripting language, capable of just one task

String manipulation: sed is a *stream editor*

Reads files one line at a time (as a *stream*) and edits them, one line at a time, according to given rules

Scripting Languages

Sed

Example: Sed. A very simple scripting language, capable of just one task

String manipulation: sed is a *stream editor*

Reads files one line at a time (as a *stream*) and edits them, one line at a time, according to given rules

`sed -e 's/hello/goodbye/'`
Substitutes `goodbye` for occurrences of `hello` in its input

Scripting Languages

Sed

Uses *regular expressions* for patterns

Scripting Languages

Sed

Uses *regular expressions* for patterns

Can edit files too large to fit into memory all at once

Scripting Languages

Sed

Uses *regular expressions* for patterns

Can edit files too large to fit into memory all at once

Usually used as a component in shell scripts

Scripting Languages

Sed

Uses *regular expressions* for patterns

Can edit files too large to fit into memory all at once

Usually used as a component in shell scripts

Limited to line-by-line editing

Scripting Languages

Sed

Uses *regular expressions* for patterns

Can edit files too large to fit into memory all at once

Usually used as a component in shell scripts

Limited to line-by-line editing

More general is *awk*, but better is *Perl*

Scripting Languages

Perl

Perl is/was used a lot in Web page creation and manipulation, amongst many other things

Scripting Languages

Perl

Perl is/was used a lot in Web page creation and manipulation, amongst many other things

The basic datatype is the string

Scripting Languages

Perl

Perl is/was used a lot in Web page creation and manipulation, amongst many other things

The basic datatype is the string

The basic operation is pattern matching

Scripting Languages

Perl

Perl is/was used a lot in Web page creation and manipulation, amongst many other things

The basic datatype is the string

The basic operation is pattern matching

It is also procedural, has first class functions, has objects and so on

Scripting Languages

Perl

Perl is/was used a lot in Web page creation and manipulation, amongst many other things

The basic datatype is the string

The basic operation is pattern matching

It is also procedural, has first class functions, has objects and so on

The syntax is the usual C/Java/whatever, but with a few features

Scripting Languages

Perl

```
open IN, '<', 'infile';
open OUT, '>outfile';
$count = 0;
while (<IN>) {
  s/world/everybody/ if (/hello/);
  print OUT;
  $count++;
}
close IN;
close OUT;
print "$count lines\n";
```

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables
- `$count`: scalar (single value) variable names are prefixed by `$`

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables
- `$count`: scalar (single value) variable names are prefixed by `$`
- array variable names are prefixed by `@`

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables
- `$count`: scalar (single value) variable names are prefixed by `$`
- array variable names are prefixed by `@`
- function variable names are prefixed by `&`

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables
- `$count`: scalar (single value) variable names are prefixed by `$`
- array variable names are prefixed by `@`
- function variable names are prefixed by `&`
- `$f` is separate from `@f` and `&f`

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables
- `$count`: scalar (single value) variable names are prefixed by `$`
- array variable names are prefixed by `@`
- function variable names are prefixed by `&`
- `$f` is separate from `@f` and `&f`
- `<>`: operator returns a single line of the file each time it is called

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables
- `$count`: scalar (single value) variable names are prefixed by `$`
- array variable names are prefixed by `@`
- function variable names are prefixed by `&`
- `$f` is separate from `@f` and `&f`
- `<>`: operator returns a single line of the file each time it is called
- and false at end of file

Scripting Languages

Perl

- `open`: takes strings `'<'` to indicate input and `'>'` to indicate output; then a file name
- or joined onto the filename
- `IN`: filestream variables are syntactically different from normal variables
- `$count`: scalar (single value) variable names are prefixed by `$`
- array variable names are prefixed by `@`
- function variable names are prefixed by `&`
- `$f` is separate from `@f` and `&f`
- `<>`: operator returns a single line of the file each time it is called
- and false at end of file
- it assigns to the variable `$_`

Scripting Languages

Perl

- `$_` is a *default* argument and can be left out of many places, e.g., `print;` is equivalent to `print $_;`

Scripting Languages

Perl

- `$_` is a *default* argument and can be left out of many places, e.g., `print;` is equivalent to `print $_;`
- `statement if (test);` as well as the usual `if (test) { statements; }` (and with `else`)

Scripting Languages

Perl

- `$_` is a *default* argument and can be left out of many places, e.g., `print;` is equivalent to `print $_;`
- `statement if (test);` as well as the usual `if (test) { statements; }` (and with `else`)
- `//` for pattern matching; in this case it looks to see if the default `$_` contains the string `hello`

Scripting Languages

Perl

- `$_` is a *default* argument and can be left out of many places, e.g., `print;` is equivalent to `print $_;`
- `statement if (test);` as well as the usual `if (test) { statements; }` (and with `else`)
- `//` for pattern matching; in this case it looks to see if the default `$_` contains the string `hello`
- `s///`: if the string contains `world`, replace it by the string `everybody` (again, using the default `$_`)

Scripting Languages

Perl

- `$_` is a *default* argument and can be left out of many places, e.g., `print;` is equivalent to `print $_;`
- `statement if (test);` as well as the usual `if (test) { statements; }` (and with `else`)
- `//` for pattern matching; in this case it looks to see if the default `$_` contains the string `hello`
- `s///`: if the string contains `world`, replace it by the string `everybody` (again, using the default `$_`)
- `$count++;` lots of C-like features

Scripting Languages

Perl

- `$_` is a *default* argument and can be left out of many places, e.g., `print;` is equivalent to `print $_;`
- `statement if (test);` as well as the usual `if (test) { statements; }` (and with `else`)
- `//` for pattern matching; in this case it looks to see if the default `$_` contains the string `hello`
- `s///`: if the string contains `world`, replace it by the string `everybody` (again, using the default `$_`)
- `$count++;` lots of C-like features
- untyped variables: a variable can hold numbers and strings and other types; so ++ has first to check if the value is a number or a string containing a number, which it then converts to a number

Scripting Languages

Perl

- `$_` is a *default* argument and can be left out of many places, e.g., `print;` is equivalent to `print $_;`
- `statement if (test);` as well as the usual `if (test) { statements; }` (and with `else`)
- `//` for pattern matching; in this case it looks to see if the default `$_` contains the string `hello`
- `s///`: if the string contains `world`, replace it by the string `everybody` (again, using the default `$_`)
- `$count++;` lots of C-like features
- untyped variables: a variable can hold numbers and strings and other types; so ++ has first to check if the value is a number or a string containing a number, which it then converts to a number
- flexibility over `()` around function arguments

Scripting Languages

Perl

- strings are both single and double quoted

Scripting Languages

Perl

- strings are both single and double quoted
- single quote is unevaluated: `'hello\n'` prints as `hello\n`

Scripting Languages

Perl

- strings are both single and double quoted
- single quote is unevaluated: `'hello\n'` prints as `hello\n`
- double quote is interpolated: `"hello\n"` prints as `hello` with a newline

Scripting Languages

Perl

- strings are both single and double quoted
- single quote is unevaluated: `'hello\n'` prints as `hello\n`
- double quote is interpolated: `"hello\n"` prints as `hello` with a newline
- `"The count is $count"` sticks the current value of `$count` into the string at that point

Scripting Languages

Perl

- strings are both single and double quoted
- single quote is unevaluated: `'hello\n'` prints as `hello\n`
- double quote is interpolated: `"hello\n"` prints as `hello` with a newline
- `"The count is $count"` sticks the current value of `$count` into the string at that point

There are several very fat books on Perl

Scripting Languages

Perl

- strings are both single and double quoted
- single quote is unevaluated: `'hello\n'` prints as `hello\n`
- double quote is interpolated: `"hello\n"` prints as `hello` with a newline
- `"The count is $count"` sticks the current value of `$count` into the string at that point

There are several very fat books on Perl

Its flexibility means it is used in a lot of places

Scripting Languages

Perl

- strings are both single and double quoted
- single quote is unevaluated: `'hello\n'` prints as `hello\n`
- double quote is interpolated: `"hello\n"` prints as `hello` with a newline
- `"The count is $count"` sticks the current value of `$count` into the string at that point

There are several very fat books on Perl

Its flexibility means it is used in a lot of places

And it is easy to write unreadable code in Perl

Scripting Languages

JavaScript

JavaScript is an interesting case

Scripting Languages

JavaScript

JavaScript is an interesting case

Originally designed to be a scripting language to manage Web page content, it is now more likely to be thought of as a special-purpose language to enable dynamic Web content

Scripting Languages

JavaScript

JavaScript is an interesting case

Originally designed to be a scripting language to manage Web page content, it is now more likely to be thought of as a special-purpose language to enable dynamic Web content

Web applications like Google Maps are coded using JavaScript

Scripting Languages

JavaScript

JavaScript is an interesting case

Originally designed to be a scripting language to manage Web page content, it is now more likely to be thought of as a special-purpose language to enable dynamic Web content

Web applications like Google Maps are coded using JavaScript

In Firefox, the browser's appearance, i.e., the buttons, toolbars and so on (the "chrome"), are programmed in JavaScript

Scripting Languages

JavaScript

JavaScript is an interesting case

Originally designed to be a scripting language to manage Web page content, it is now more likely to be thought of as a special-purpose language to enable dynamic Web content

Web applications like Google Maps are coded using JavaScript

In Firefox, the browser's appearance, i.e., the buttons, toolbars and so on (the "chrome"), are programmed in JavaScript

This allows easy modification and extension (using "add-ons")

Scripting Languages

JavaScript

Standard warning:

Java and JavaScript are **different** languages

Scripting Languages

JavaScript

Standard warning:

Java and JavaScript are **different** languages

The name is unfortunate: their syntax is broadly similar but their semantics are wildly different

Scripting Languages

JavaScript

Standard warning:

Java and JavaScript are **different** languages

The name is unfortunate: their syntax is broadly similar but their semantics are wildly different

The name was originally chosen as JavaScript was intended to be “reminiscent” of Java, but now it’s just a source of confusion

Scripting Languages

JavaScript

Standard warning:

Java and JavaScript are **different** languages

The name is unfortunate: their syntax is broadly similar but their semantics are wildly different

The name was originally chosen as JavaScript was intended to be “reminiscent” of Java, but now it’s just a source of confusion

It can be argued that JavaScript is more like Scheme/Lisp than Java in the way it behaves (first-class functions, etc.)

Scripting Languages

JavaScript

Standard warning:

Java and JavaScript are **different** languages

The name is unfortunate: their syntax is broadly similar but their semantics are wildly different

The name was originally chosen as JavaScript was intended to be “reminiscent” of Java, but now it’s just a source of confusion

It can be argued that JavaScript is more like Scheme/Lisp than Java in the way it behaves (first-class functions, etc.)

But you shouldn’t take this analogy too far

Scripting Languages

JavaScript

JavaScript is more properly called *ECMAScript* and there are several close-but-not-perfectly-compatible versions around

Scripting Languages

JavaScript

JavaScript is more properly called *ECMAScript* and there are several close-but-not-perfectly-compatible versions around

- JavaScript: Sun/Mozilla/etc.
- JScript: Microsoft
- ActionScript: Adobe

Scripting Languages

JavaScript

JavaScript is more properly called *ECMAScript* and there are several close-but-not-perfectly-compatible versions around

- JavaScript: Sun/Mozilla/etc.
- JScript: Microsoft
- ActionScript: Adobe

Ecma: formerly *European Computer Manufacturers Association*, now just “Ecma International”, is a standards body

Scripting Languages

JavaScript

```
function getdoc()
{
   var input = document.getElementById("num");
   var num = input.value;

   if (parseInt(num) == num && num > 0 && num < 5000) {
     document.location = "http://www.rfc-editor.org/rfc/rfc"
                         + num + ".txt";
   }
   else {
     alert("Not a valid RFC number!");
     input.value = "";
   }
   return false;
}
```

Scripting Languages

JavaScript

- functions declared by `function`, no type declarations

Scripting Languages

JavaScript

- functions declared by `function`, no type declarations
- variables declared by `var`, no type declarations

Scripting Languages

JavaScript

- functions declared by `function`, no type declarations
- variables declared by `var`, no type declarations
- a variable can hold items of any type

Scripting Languages

JavaScript

- functions declared by `function`, no type declarations
- variables declared by `var`, no type declarations
- a variable can hold items of any type
- syntax reminiscent of Java (and C and so on)

Scripting Languages

JavaScript

JavaScript is easy to learn and use, good for prototyping

Scripting Languages

JavaScript

JavaScript is easy to learn and use, good for prototyping

It is OO, but in a very different way from other languages, particularly Java

Scripting Languages

JavaScript

JavaScript is easy to learn and use, good for prototyping

It is OO, but in a very different way from other languages, particularly Java

We shall be talking more about JavaScript when we examine OO in detail