# Using Javascript For Teaching Introductory Programming

**Paul Roper**

Nelson Marlborough Institute of Technology
Nelson, New Zealand

proper@nmit.ac.nz

Proceedings of the 15th Annual NACCQ, Hamilton New Zealand July, 2002 www.naccq.ac.nz

## ABSTRACT

The choice of programming language for the delivery of the CBC introductory programming modules PD100, PP100 and the optional PP110 is an issue often debated. Since 2001 NMIT has used Javascript to replace Pascal. Javascript has all the selection and iteration constructs needed, arrays, does not require a compiler, the syntax leads easily into Java, and is perceived as trendy and immediately useful to students interested in the web. The GUI design features of VB or Delphi etc do not distract students. As Javascript is "object-based" students get a gentle introduction to some O-O concepts. The downsides are Javascript is loosely typed, it is necessary to incorporate teaching some HTML and the unfriendly or nonexistent error reporting at run time from the popular browsers.

To address this last issue, the author has written a Javascript Authoring Tool, ScrypTik. It allows editing multiple files with syntax highlighting, unlimited undo/redo and thorough cross checking of all user-defined identifiers both within Javascript and surrounding HTML named elements. A Javascript parser checks the code syntax. A click on the 'view' button will run a HTML file in a browser. Although ScrypTik development lagged behind the students needs during the early part of the first year, the overall level of enthusiasm and standard of assessments has justified making this risky language transition.

With students using ScrypTik and adopting a broad delivery plan of design basics, HTML basics, Javascript programming basics, followed by integrating HTML and Javascript, supported by our own design booklet and Intranet site, Javascript is a viable teaching language for beginners. Students who leave with a CBC have useful skills and those returning have had exposure to 'event-driven' software that should help them with future language learning.

## 1. INTRODUCTION

This paper describes the reasoning, consequences, coping strategies and outcomes from changing the introductory programming language from 'educationally sound' Pascal to innovative Javascript. I have no empirical evidence for student satisfaction or knowledge level before or after this change. Any claims I make are based purely on my experience with the students. I propose that by delivering programming in a medium empathetic with students, we increase our chances of maintaining interest and enthusiasm.

## 2. LANGUAGE CHOICE

Since CBC was first delivered at NMIT in 1988, the introductory programming language taught was COBOL then Pascal (apart from two years when the computer scientists decided upon C) The choice of language is usually based on some or all of (not exhaustive):

- Staff bias
- Does it have all the basics (selection, iteration, arrays, sub-tasks)?
- Future pathways, does it lead anywhere?
- Implementation and support of compiler/editor
- Is there a student take home version of compiler/ editor?
- Is it from Microsoft?
- Is it marketable (will it attract or repel students)?
- Is it capable of implementing the design basics from PD100?

Our choice was based on some of the above but strongly influenced by a desire to position ourselves as e-commerce focused and recognising that today software development is often web-application development. Many students have concocted static web pages and creating dynamic pages on the client side is appealing. We also did not want to use a RAD (Rapid Application Development) tool such as Visual Basic or Delphi where students are tempted to rush into placing fancy components on forms (GUI) instead of attending to the basics. DBC students are taught Visual Basic in year two. Java was not considered as we believe immediate immersion in objects is too difficult for beginners.

We chose Javascript because staff were comfortable with it, it has all the basic structured programming constructs, its syntax is in the C, Java and C# family, it requires only a browser not a compiler to run, it is marketable and it will handle the design basics for PD100.

## 3. THE SUBJECT OF PROGRAMMING

Many CBC students find programming unnatural and difficult. They lose enthusiasm and com,plaints include Pascal is of no use to them because they don't see job vacancies for Pascal programmers. Although we as tutors can convince ourselves that Pascal is excellent for enforcing the good habits essential for novices, it is all but impossible to convince disgruntled students that 'the language doesn't matter, you are learning principles that you can apply to any programming language'. This is part of the educational issue of 'academic' versus 'vocational'. We should try to encompass both, however I believe it preferable to engage students by using a contemporary technology and working around any shortcomings.

## 4. JAVASCRIPT IS NOT PERFECT FOR TEACHING BEGINNERS

There are several issues that need to be faced when using Javascript as an introductory programming language. Because it is a scripting language, which to some degree can control another environment, usually HTML in a browser, it is necessary to teach that other environment. HTML is a slight distraction but immensely useful for students regardless of their future with programming. Fortunately many students are already familiar with HTML.

Javascript is case sensitive yet the environment in which it lives (HTML) is not. Beginners often exhibit 'case blindness', they are presented with an error and seem incapable of processing the fact that 'if' and 'If' are not the same. Ideally, a first language should not be case sensitive.

Javascript is loosely typed. Combined with Javascript making any variable declared inside a function without the keyword "var" into a global, this makes enforcing good programming habits difficult. We explain that Javascript is unusual in these aspects and the student algorithm designs must still include data typing.

As Javascript code is interpreted and runs inside the browser after the page is loaded, any errors are reported in, for a beginner, an unfriendly and frequently misleading manner. In order to correct errors, students must go back to their editor, change something and return to the browser and 'refresh'. This editing, saving and swapping applications for often trivial reasons is tedious and distracting.

The run-time environment (i.e browsers) is unpredictable, inconsistent and unstable. The major browsers are not 100% compatible. To be browser independent scripts must check which browser they are running on and then execute appropriate code. Further, each new browser release introduces additional incompatible features.

## 5. SCRYPTIK

With a group of 24 students using notepad and Internet Explorer attempting to get their first simple programs running by copying code from the whiteboard, it quickly became apparent that the ratio of errors (and the time taken to correct them) to tutor was not viable. Javascript was either a really dumb idea or I needed a tool that would locate the syntax and quirky errors before trying to run in a browser. NetObjects Scriptbuilder V3 looked good, it parsed the Javascript in a limited way (no checking for variable's existence) but made no attempt to cross check against the surrounding HTML. Trawling the web produced nothing else, so I decided to develop my own.

ScrypTik is a source code editor and syntax checker that is effectively a Javascript IDE (Integrated Development Environment). It has taken a year of spare time development to get the Javascript IDE, ScrypTik to a useable release 1.0. It now reports approximately 150 Javascript syntax and HTML integration errors. In the second year of teaching Javascript when the students reached the same beginning coding stage, ScrypTik found most errors and tutoring was relatively relaxed and stress free.

ScrypTik's main features are:

- It can edit multiple files using a 'tabbed' user interface rather than a classical MDI (blame Borland).
- File types expected (but not restricted to) are .htm, .html, .js or .asp.
- Re-open list of the last ten files edited.
- The left gutter of the editor always displays line numbers.
- Syntax highlighting for HTML or Javascript, hopefully a future version will allow both at once.
- Virtually unlimited undo/redo (32000).
- Many quick key commands appropriate for a programmer's editor, eg Cntl Y deletes a line.
- Built in pop-up ASCII code chart from which decimal or hex may be copied.
- Template text for most common HTML elements (headings, bold, center, tables, lists, images, forms, inputs etc) and Javascript code chunks (if, else, switch, for, while etc).inserted from menu selections.
- A navigation and information tree hierarchy showing global variables, Javascript sections, all functions, all HTML named (id) elements and all events (eg a button with an 'onClick =').
- A pane to show all errors. Selecting an entry in this or the navigation pane highlights the appropriate line in the editor.
- Errors reported in three categories, hint, warning and error, all phrased appropriately for beginners.
- One button click to run the file in a browser. If several browsers are installed the user is offered a choice at view time.

## 6. PD100 & PP100 COURSE DELIVERY

ScrypTik is only one of the resources used. A common question students ask after a few weeks of PD100 is "Is there a book in the library about Structured English and Structure Diagrams?" Since the answer was always the same (no), several years ago Matthias Otto and I put together a 'booklet' on program design. This is available to students as a Word document and most print themselves a copy (approx 50 pages).

We have a school intranet site and PD100 has various sample programs, information and links.

All these resources are used extensively in conjunction with the 'chalk and talk'.

The delivery plan for PD100 and PP100 is:

- Basic algorithm designs
- Basic raw HTML tags
- Implement the basic design exercises in Javascript using prompt and alert dialogs for input and output. If output is more than one data item then use document.writeln().
- Design using sub-tasks.
- Implement sub-tasks using Javascript functions.
- Revisit some earlier exercises and make them look better using HTML forms, <INPUT> components and buttons.
- Array design and implementation.
- Design and coding exercises using everything covered so far.

Note that the boundary between PD100 and PP100 is not clearly defined. PD100 is not completely finished until part way into the time allocation for PP100. Scryptik is used continuously after the first two points above.

ScrypTik supports the delivery of PP110 also:

- Manipulating images for animation (changing and moving), drag, drop and snap.
- Introduction to the DOM (Document Object Model) leading to creating some simple objects.

## 7. CONCLUSION

There are no silver bullets when it comes to the best language for introductory programming. Niklaus Wirth did a great job of designing Pascal as a teaching language. I used Object Pascal (Delphi) to write SrypTik. But things have moved on. In general, today's students spend a large amount of time on the web. They are familiar with web pages to varying degrees. By delivering programming closely aligned to something they relate to, we are in a better position to capture and maintain their attention. In the process they learn the basics of programming and also cope with 'event-driven' software and the concept of an 'object'. Javascript syntax is that of C and Java. These factors should be useful preparation for whatever they try next. The use of ScrypTik minimises the frustration experienced by novice programmers having to solve problems that are not central to the required task.

My perception is that overall student enthusiasm for the compulsory programming modules of CBC is vastly improved from when we used Pascal.

## 8. FUTURE DEVELOPMENTS

In writing ASP server-side scripts in VBScript, I found myself using ScrypTik solely for the line numbering feature of the editor. After wrestling with the quoting and line continuation challenges of VBScript and embedded SQL, I would like to add a VBScript parser and a SQL parser to ease debugging of server-side scripting.

NMIT is happy to supply ScrypTik to other public tertiary providers and their students free until December 2003. A message stating where ScrypTik was developed will be displayed in the 'splash' and 'about' screens and Help file. Feedback and experiences from educators would be nice, we could have a user group mailing list if sufficient demand.