Enticing Today's Teenagers To Write Computer Code

Shôn Ellerton, February 3, 2017

Last year, my wife and I hosted two 16-year-old students from China as part of an international schools' program here in Australia. One of the students approached me and asked which computer language should he learn. I pondered over the question he asked and then asked him what he wanted to write code for? After shrugging his shoulders, he just said it would be cool to learn how to do all that fancy stuff but didn't know where to start. I didn't mean to put him on the spot, but his response was valid enough. It really isn't very clear-cut how you start to learn how to program, especially with the vast choice of languages and applications we now have available to use. I pre-empted his answer by alluding that he probably wanted to learn how to program to create and design games knowing that he probably would have tried to come up with an answer more scholastically-inclined! After nodding his head in affirmation, I conducted an Internet search and found a variety of courses specialising in creating gameware using classic game engines like Epic Games's *Unreal* which is based on C++, a notoriously difficult language to master. The good news, I discovered, was that you can create your own games by using high-level software development kits provided by popular games engines thus circumventing the need to understand the underlying low-level language. This is similarly the case with creating

websites using web-building platforms like Wordpress and Joomla without needing to understand the details of HTML and CSS. Granted, to build a good game or website still requires plenty of skill and creativity, but it is far easier now then it was a couple of decades ago. So my question is: how do we get kids interested in coding if all these high-level tools exist? Do we need them to be interested at all and, if not, will it matter in the future?

My Own Early Experience

I remember my father introducing me to two classic coin-operated video games during the early 80's: *Space Invaders* and *Galaxians*. Along with the other multitude of video games littering the video arcade, I shudder to think how many quarters I popped into the slots of these machines. During this time, crude handheld game machines were available but they were exceptionally basic and unsatisfying, at least to me. I wanted to create one of these games for myself, so I then convinced my parents that a computer would be useful to help with my homework (a blatant lie). They believed me and then I was given a brand-new *Texas Instruments 99/4A* for my birthday. It's main competitor at the time was the iconic *Commodore 64*. Although the *Commodore 64* had 64KB of RAM vs the 16KB in the TI 99/4A, the TI sported a more advanced 16-bit processor rather than the 8-bit processor the Commodore used. It's probably worth remembering that these computers probably have less computing power than your modern alarm radio clock; however, they served their purpose well to educate those using them to learn how to program in the early 1980's.

After unboxing the TI and connecting everything up to a monitor and a tape deck (floppy disc drives were an expensive extra), I was ready to 'rock-and-roll'. Itching to start programming a new game and with no desk space left, I proceeded to read through what must have been the driest most incomprehensible manual I ever read in my childhood. The TI came with its own dialect of BASIC (TI BASIC) and there was a step-by-step example of how to print the words: "Welcome to the world of computing!" on the screen. Although this was exhilarating, I was dismayed that it would probably take a very long time to program a game as good as *Space Invaders*! Thankfully, there were a variety of monthly magazines published which contained written code in TI BASIC to re-create the games (and other useful applications) the programmers had shared in the magazine. In a world with no Internet, every release of a new magazine was a joyous occasion and I selected those articles containing interesting code to type into the TI. It was certainly tedious typing code into the computer, but damned if

there isn't a better way to know how to code! I then found in the local library, more books containing programs written in BASIC covering not only games but other applications such as money managers, word processors, and inventory management. Only then did I realise that not all BASIC programs are written in the same dialect so I started to learn how to translate these programs into TI BASIC. Not all programs were successful either due to the level of difficulty or the limitations of the computer.

I soon found out that, once you understand the logic of programming, you can quickly learn other high-level languages. This was the case when I was doing a school tour of the *Digital Equipment Corporation* hard disc plant back in Colorado Springs and then invited to do a little programming for them using another language called Pascal. I was freely-provided with a line modem to access their mainframes and bulletin board systems (BBS) which opened out a new vista of untapped resources which would be near-impossible to obtain at the local library. To this day, I still use Usenet services which kind of resembles the old BBS.

My old TI99/4A back in '83

How Low Can You Go?

I got to creating some pretty good games on the old TI99/4A but there was one irksome issue which I simply could not overcome: performance. Playing an action game programmed with TI BASIC was like trying to swing a tennis racquet through a wall of jelly. It was excruciatingly slow. I then proceeded to expand my horizons by learning *assembly language*, which is about as low as you can go (in terms of programming languages parlance), short of writing in machine language. This was too much hard graft to learn (the manuals were almost incomprehensible) and it was not until years later that I was introduced to a somewhat higher level language called *C*.

```
LINE  MEM   HEX   LABEL  OP     OPERAND      COMMENTS
NUM   ADDR  CODE         CODE
0001                     DEF    TRYIT        (Put the word TRYIT into
                                             the REF/DEF table)
0002                TRYIT                    (The start of the program)
0003  0000  0200         LI     R0,298       (Loads register 0 with a
      0002  012A                              value of decimal 298,
                                              which will be the
                                              position on the screen
                                              where text is displayed)
0004  0004  0201         LI     R1,CD        (Loads register 1 with
      0006  0016                              the memory address where
                                              the text is to be dis-
                                              played is located.)
0005  0008  0202         LI     R2,12        (Loads register 2 with
      000A  000C                              the length of the text
                                              to be displayed.)
0006  000C  0420         BLWP   @>2024       (Loads the subroutine
      000E  2024                              that will display the
                                              text on the screen.)
0007  0010  04E0         CLR    @>837C       (This will clear the
      0012  837C                              status byte so as to
                                              avoid false errors upon
                                              return to ExBasic.)
0008  0014  045B         B      *R11         (This will return you to
                                              ExBasic upon completion
                                              of the assembly program.)
0009  0016  A8A5   CD     DATA   >A8A5,>ACAC,>AF80,>B4A8,>A5B2,>A580
      0018  ACAC
      001A  AF80                             (This is the text to be
      001C  B4A8                              displayed. It is added
      001E  A5B2                              with the DATA directive
      0020  A580                              because the screen bias
                                              of >60 has to be added
                                              to the ASCII code of each
                                              character.)
0010  2004               AORG   >2004        (This updates the LFAL (Last
0011  2004  3FF8         DATA   >3FF8        Free Address in Low Memory),
                                              and puts the address of the
                                              REF/DEF table here).
0012  3FF8               AORG   >3FF8        (Start of the REF/DEF Table
                                              where we have the name and
                                              starting address of the
                                              program.)
0013  3FF8    53         TEXT   'TRYIT'      (The name of the program.)
0014  3FFE  3F86         DATA   >3F86        (The starting address.)
0015                     END                  (Closing statement.)
```

Some TI99/4A Assembly Code

Today's Computing Languages? Which to Learn?

So back to today, the student I hosted asked the question which language should he learn. Today, we have a baffling array of languages to choose from. You only need to do an Internet search of today's list of computing languages to find that one out. Back in the 80's, it mattered as to what computer you had and you were limited as to what languages you can use, but your average PC can emulate just about any type of language you wish to learn.

Just to set the record straight, I am not including SQL (Structured Query Language) here. Agreed, it is highly desirable to learn it if you are dealing with manipulating data, put it is not considered a programming language. Moreover, it is unlikely that your teenager will be remotely interested in dealing with data sets, unless you were as geeky as I was with creating inventories of my hiking pursuits around the Colorado mountains, another hobby of mine I had.

C#.NET

My own personal experience in the field I work in (namely, project management systems) is that C#.NET has always been the dominating language and I have not seen much in the way of its decline either. VB.NET (Visual Basic) does pretty much the same thing as all the .NET languages gets compiled into the same low-level language, but C#.NET is far more common and is generally preferred by the programmer community due to its less wordy and more terse structure. A little more cryptic than VB but worth the extra time to learn. You can do pretty much everything with C#.NET and you can start learning how to use it and create some pretty interesting applications. Although this is quite a bit more complicated than the BASIC I grew up with, you can start simple and eventually learn to create complex applications complete with an array of re-usable classes and customised global functions. Unless web design is of particular interest, learning to build a Windows client application may be a better place to start.

C and C++

C and *C++* are rather difficult-to-learn low-level languages which, if mastered, can open the doors to creating other languages and applications requiring high efficiency and performance, like games and electronic devices. My experience using C was, somewhat, short-lived when, after installed 40+ *Borland C* floppy discs into my under-powered PC during the mid-90s, it crashed repeatedly and I

succumbed to frustration and impatience; both common symptoms of working with this language. If you want to program devices or robots, you will probably find this language a compulsory requirement. In any case, those who master *C* will be the envy of those who don't know it. At least, I was envious.

Students programming robots using C

Java

My experience with Java is even less and I have, personally, not run across many applications in the field using it (except for those using Android phones). I always like the idea that this is a portable language that can run across different operating environments. Java is a well-known language, however, and I predict it will be around for some time to come. Learning Java is not terribly easy but, from those I have spoken to who have programmed in it, found it to be a very rewarding language to learn. Great if you want to create games and other apps for your Android phone!

Python and Perl

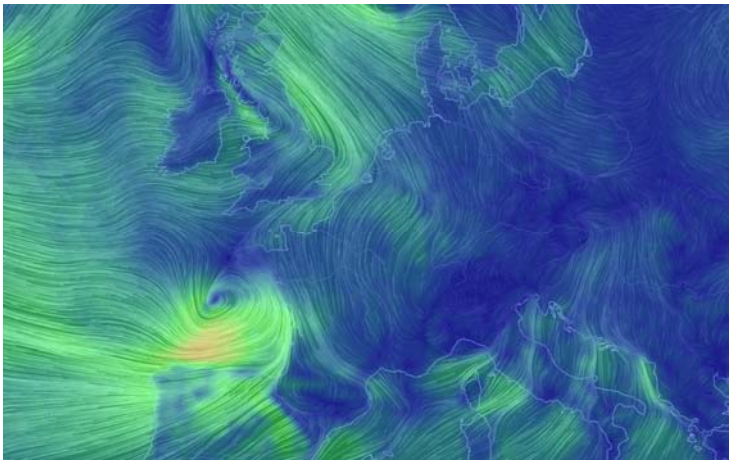
These are both scripting languages with Python being a relatively new language and seeming to be all the rave basking in the sunshine of relative simplicity. Many of my friends and relatives in the scientific and business community are learning Python due to its intuitive nature unlike Perl. I'm not fond of the way Python forces programmers to indent and would rather stick with Perl's bracket-based structure, but this is just my personal preference. Whether Python or Perl offers a rich enough environment for someone to learn at an early age is open to conjecture and then there is the issue of the longevity of the language, although Perl has been around quite some time now.

COBOL and FORTRAN, Really??

Don't fall off the edge of your seat, these two 1950s-originated languages are still with us today. Along with *C*, I consider these to be the trio of the granddaddies of computer languages.

Let's start with FORTRAN (Formula Translation Language).

Should your teenager have a fascination with supercomputers having the ability to number-crunch complex fluid dynamics problems, weather patterns, or other physics-related problems, then this language might be of interest. I learnt FORTRAN 77 during my engineering university days and found it very easy to learn not being wholly unlike the BASIC language I first started to learn. Unfortunately, our FORTRAN lecturer was as dry as a bone that was dug up from an Egyptian tomb so I ended up dozing off most of the time during these lectures. C++ is also popular in this field but, as mentioned above, it is considerably more difficult to learn.



Beautiful output from Earth School Null showing weather patterns calculated by supercomputers

Now let's move on to COBOL, or Common Business-Oriented Language.

Interestingly, much news has been publicised about the many failed attempts of Java to kill off the world's many existing COBOL-based systems. I personally find COBOL fascinating.

Fascinating that it:

- has only had less than 10 major revisions since 1959
- has survived since 1959
- is still used today by the majority of major financial and critical-infrastructure industries such as airline ticketing and ATMs
- it was created by an equally interesting person, Grace Hopper
- it is a language which fewer and fewer people know how to use (although there has been a recent resurgence of some universities teaching it)
- it is downright stable
- it is secure by the fact that it is becoming more obscure

A bit of a weird language to learn (and very wordy!), but COBOL compilers can be used in today's PC's. Last year, I tried out Microfocus's COBOL compiler for MS Visual Studio.

There is much speculation as to the future value of learning COBOL; however, to be really effective as a COBOL programmer, you will probably need to understand all the other bits and pieces surrounding the COBOL environment (e.g. IBM z/OS, CICS, etc).

If your teenager is interested in the world of programming systems involving financial or other business-related data, this might be an interesting language to think about; however, I would also recommend learning Java as well. Learning how to program databases using one of the well-known vendors such as Oracle, Microsoft, IBM or MySQL along with a sound knowledge of their SQL dialects will be essential.

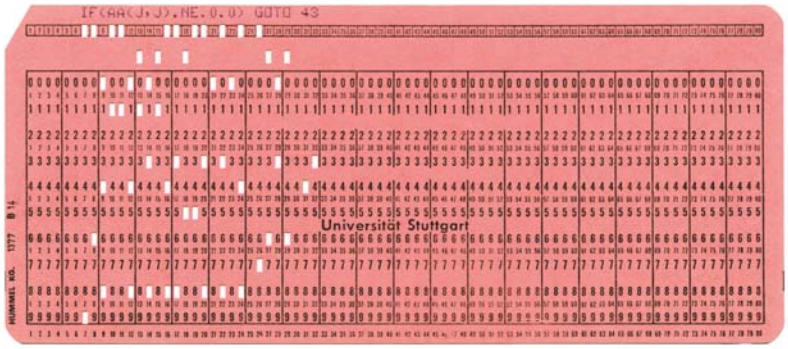
One thing is certain, COBOL programmers are not the easiest to find, many of them being retired or have passed away!

What Would Create That Interest to Learn Computer Programming?

Just to state the obvious, if your teenager does not show any interest in programming, don't force the issue. However, if your teenager is a heavy *user* of

computers and does show an interest in getting an edge over his peers in computing, then it might pay dividends to point out that learning what happens 'behind the scenes' of the machine could be very rewarding indeed, especially when emerging into the technology-related workforce. Before the Internet was mature enough to provide enough rich content in delivering step-by-step tutorials in learning how to program, my generation generally relied on, usually expensive, programming language reference books found at a very good bookstore (itself a dwindling resource). The library never had anything up to date!

In short, to create that spark that would generate an interest in learning a computer language is that there needs to be an end result: creation of a game, an inventory tracker, a dynamic website or whatever. Before the days of university courses offering computer science or business computing, the majority of computer programmers were largely self-taught in their own professions because there was a real *need* to do so. This was the case with my grandfather who learnt FORTRAN 4 (on ancient machines the size of a large truck running on valves complete with punch cards) to calculate statistical results of sugar beet trials in the 1960's.



1960's punch-card containing FORTRAN code

Does It Matter If Nobody Was Interested In Programming?

As technology advances, our distance from the underlying base technology also increases. For example, our definition of *computer literate* typically encompasses someone who can competently put together a reasonable *Microsoft Excel* report together or know how to stitch up a document using *Adobe Acrobat*. Many of today's applications are quite complicated to use without knowing a shred of what happens under the bonnet. For example, heavyweight applications like *AutoCAD* or *Adobe Photoshop* are good examples of this. I have used *AutoCAD* extensively

in the past and still feel somewhat comfortable with it, but as soon as I open up *Adobe Photoshop*, I get that blank look on my face as I struggle to do anything remotely complex with an image. Database products are complex enough to understand in their own right; for example Microsoft's *SQL Server* and *Oracle's* database product. But remember that many of these applications are written using low-level languages like *C* or *C++*, so one can imagine the sheer complexity of the code here. So how do you write *C* itself? Another *C* compiler? Assembly? The list goes on.

We create tools to create more advanced tools, but, ultimately, what would happen if the last person on Earth died who understood the very basics of how machine code or assembly language is put together? Would it make a difference? I don't know the answer to this, but what I do know was that I wanted to re-create that *Space Invaders* game and play for free when I was a young teenager!