LINUX VOICE TUTORIAL

BASIC: THE LANGUAGE THAT STARTED A REVOLUTION

JULIET KEMP

Explore the language that powered the rise of the microcomputer – including the BBC Micro, the Sinclair ZX80, the Commodore 64 ***et al***.

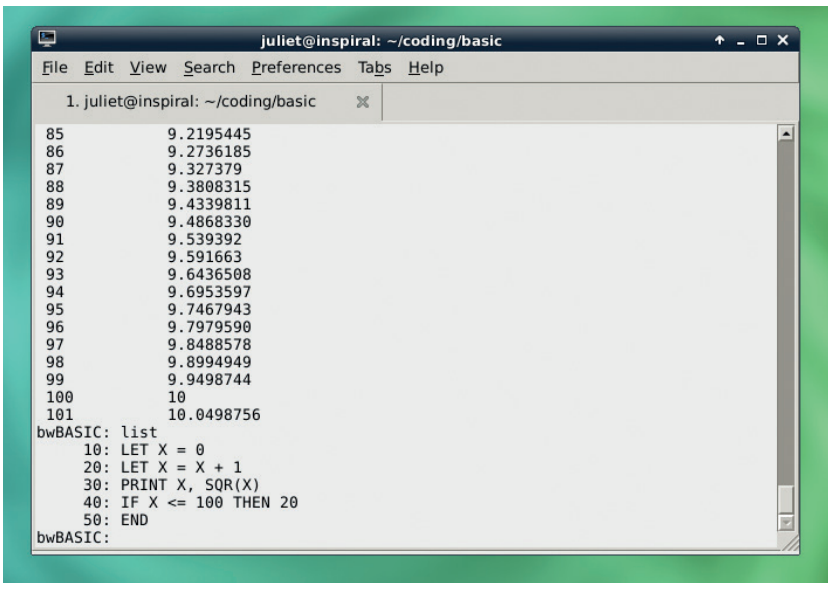
WHY DO THIS?

- Learn the Python of its day
- Gain common ground with children of the 80s
- Realise how easy we've got it nowadays

Like many of my generation, BASIC was the first computer language I ever wrote. In my case, it was on a Sharp MZ-700 (integral tape drive, very snazzy) hooked up to my grandma's old black and white telly. For other people it was on a BBC Micro, or a Spectrum, or a Commodore. BASIC, explicitly designed to make computers more accessible to general users, has been around since 1964, but it was the microcomputer boom of the late 1970s and early 1980s that made it so hugely popular. And in various dialects and BASIC-influenced languages (such as Visual Basic), it's still around and active today.

The very first version of BASIC (which stands for Beginner's All-purpose Symbolic Instruction Code), Dartmouth BASIC, was designed and implemented at Dartmouth College in 1964. It was written by a team of students working (often all night during the initial sessions) under the direction of the designers, John Kemeny and Thomas Kurtz.

In 1964, "computer" still meant a huge mainframe machine, with very limited access. To run a program, you needed to get it onto punch cards, submit your punch cards to be run, then get more punch cards back with the output of your program. It was a slow and opaque process, and initially only a very few people had any kind of access at all. However, in the early 1960s, less mathematically oriented students and researchers were just beginning to use computers for their research.



Here's bwBASIC running the square root program, then using the LIST keyword interactively to show the code listing.

John Kemeny, who spent time working on the Manhattan Project during WWII, and was inspired by John von Neumann (as seen in Linux Voice 004), was chair of the Dartmouth Mathematics Department from 1955 to 1967 (he was later president of the college). One of his chief interests was in pioneering computer use for 'ordinary people' – not just mathematicians and physicists. He argued that all liberal arts students should have access to computing facilities, allowing them to understand at least a little about how a computer operated and what it would do; not computer specialists, but generalists with computer experience. This was fairly far-sighted for the time – Kemeny correctly argued that computers would be a major part of Dartmouth students' future lives even if they weren't themselves 'programmers'.

Dartmouth BASIC

His colleague, Thomas E Kurtz, another Dartmouth mathematics professor, was also enthusiastic about this idea. Their aim was to make computers freely available to all students, in the same way as library books (Dartmouth was famous for its large open access library). Later, Kurtz became director of the Computation Centre, and later the Office of Academic Computing, and the CIS program, at Dartmouth. He and Kemeny also developed True BASIC in the early 1980s, which Kurtz still works on.

Widening computer access meant dealing with two problems. One was the non-intuitive nature of ALGOL and FORTRAN, the most popular languages at the time. Kemeny and Kurtz felt that the more instruction was needed to begin to write programs in a language, the fewer students would end up using it. BASIC was written to be intuitive, using keywords like GOODBYE to log off. And although this very first version of BASIC was compiled, it was still "compile and go" – meaning that from the programmer's point of view, compiling and executing the program was a single step, and feedback was immediate. (Later versions were interpreted, meaning that programs ran without an intermediate step in which the whole program was compiled into machine code.) This all made it easier for non-specialists to start programming.

The second problem was that computers were still large, expensive machines taking up a whole room. Actually providing each student and faculty member with a computer was not remotely feasible. However, a new idea had just arisen which would make

computer access much easier. This was time-sharing, in which multiple teletypes were connected to a single central computer. The computer would then allocate a certain amount of time to each simultaneous user. So the user could type in a BASIC program, and see it run, from their teletype in another room. A time-sharing scheme had just been implemented at MIT by John McCarthy, who recommended the system to Kemeny and Kurtz. But the Dartmouth Time-Sharing System, which went live, along with BASIC, on 1 May 1964, was the first successfully implemented large-scale such system.

Later, a few local secondary schools were also added to the network, and eventually the Dartmouth Educational Network was formed, allowing over 40 colleges, 20 secondary schools, and a variety of other institutions to access computing facilities remotely. Eighty percent of Dartmouth students were able to learn to program using BASIC and the DTSS.

The first BASIC program run from a terminal ran on 1 May, 1964 (exactly 50 years ago as I write this), and consisted, depending on who you ask, either of an implementation of the Sieve of Eratosthenes (which finds prime numbers), or of this line:

```
PRINT 2 + 2
```

For historical resonance, try that in the emulators discussed below before you get started with the rest of the programs.

ALGOL

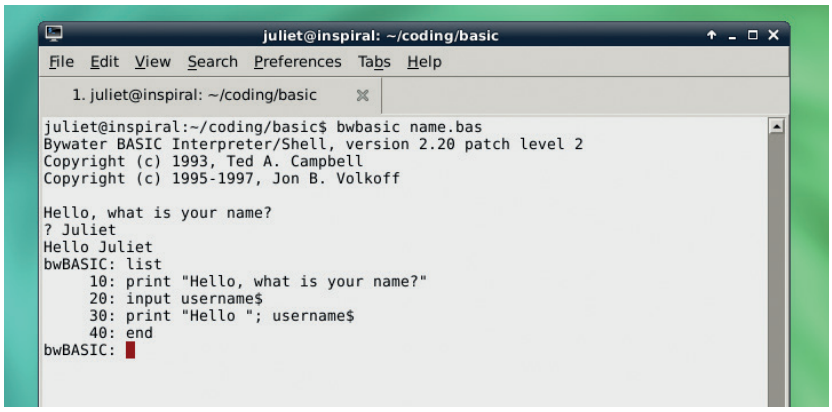
BASIC was loosely based on FORTRAN II and a little bit of ALGOL 60. Kemeny and Kurtz initially tried to produce a cut-down version of one of these languages; when this didn't work, they moved on to creating their own.

ALGOL, which exists in several variants, is imperative and procedural. ALGOL 58 was intended to avoid the problems seen in FORTRAN, and eventually gave rise to a huge number of languages including C and Pascal. ALGOL 60 improved on ALGOL 58, introducing nested functions and lexical scope, among other things. While very popular among research scientists, it was never commercially popular due to its lacking a standard input/output library. It has, though, had a huge effect on computer language development, largely due to the fact that it was used as a standard algorithm description for years.

Other emulators

Lots of other emulators are also available for various early microcomputers and for BASIC. Here are a few options:

- A list of Spectrum emulators **www.worldofspectrum.org/emulators.html**.
- Two ZX81 emulators are available for Linux: SZ81 (**http://sz81.sourceforge.net**), and Z81 (**www.svgalib.org/rus/z81.html**).
- Dartmouth BASIC (RFO BASIC) is available for Android **http://laughton.com/basic**.
- And if you're looking for type-in programs to try out, the book BASIC Computer Games is available as an online scan. (NB: this worked when I first looked at it, then didn't a week later. I include it here in the hope that the problem is temporary.) **www.atariarchives.orgbasicgames**.



Our Name program running on bwbasic, again with the LIST keyword shown.

Running Dartmouth BASIC

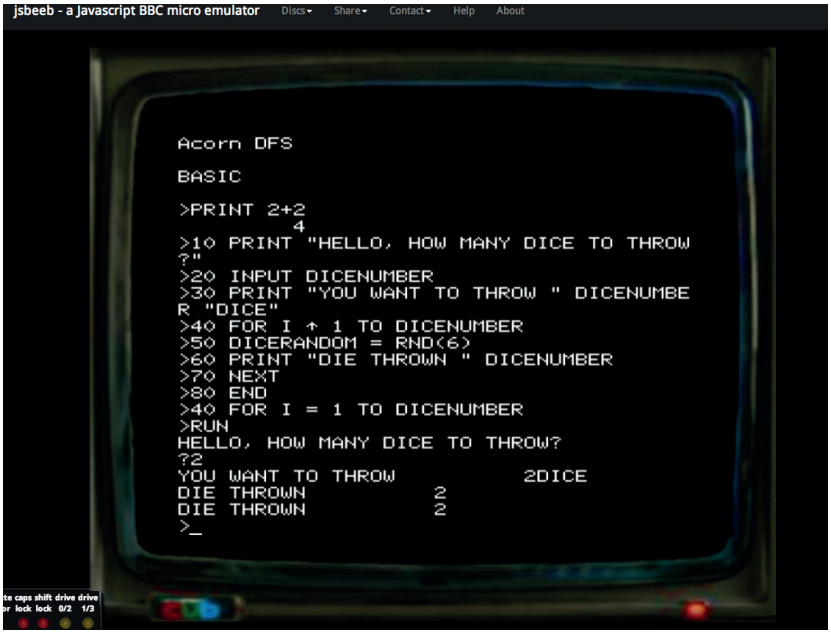
An emulator is still theoretically available online, but the online version no longer works at time of writing, and the download version only exists for Mac and Windows. (It's also seven years old so may not work on either anyway; I was unable to test it.)

However, at least some Dartmouth BASIC programs ought to run with a modern BASIC interpreter. The Dartmouth BASIC manual from October 1964 is available online from **Bitsavers.org** (a fantastic resource). The second program listing in the manual will run with the bwbasic interpreter (available as a package for Debian/Ubuntu) and ought to run on any other BASIC interpreter, as it is pretty straightforward:

```
10 LET X = 0
20 LET X = X + 1
30 PRINT X, SQR(X)
40 IF X <= 100 THEN 20
50 END
```

As is fairly obvious (BASIC was after all designed to be easy to read), this is just a loop that prints out x and its square root for the values 1 to 101. A couple of notes: firstly, BASIC is case-sensitive in general, but in bwbasic, commands and functions are not case-sensitive. **LET** and **let** will do the same thing. (This is not true of all BASICs – many insist on caps.)

Line numbers, as in the loop here, are used as labels. They are also used by the compiler to automatically order the lines. You could write the lines of code backwards in your file (from 50 down to 10), and the compiler would rearrange them for you and run them in the correct order. It is a good idea to number your lines in 10s rather than 1s, to make it easier to insert new lines in between. Unfortunately, bwbasic doesn't include the **RENUMBER** command, which is in the ANSI BASIC standard, though it does include **DO NUM** and **DO UNNUM** (which number and un-number the program lines, but do not change any **GOSUB** or **GOTO** statements). Dartmouth BASIC didn't have **RENUMBER** either, though.



Our Dice program typed into BBC BASIC simulator. Note the error in line 40 (later corrected by reentering the line).

This doesn't use **GOTO**, as the **IF**/**THEN** statement only needs a single line. Run it with **bwbasic test.bas** to try it out. You can also use bwbasic interactively.

Unfortunately, the first program in the Dartmouth manual doesn't run under bwbasic, as it relies on **READ** and **DATA** behaving in certain ways. **READ** is used to read values from the next available **DATA** line. It seems that in 1964 Dartmouth BASIC, when the program ran out of **DATA** lines, it would stop. In bwbasic, it just stops reading in new values, but continues to run (if possible) with any values already present. This demonstrates one problem with translating BASIC programs between different dialects; the detail of the keywords can vary enough to cause problems.

BASIC with microcomputers

In the mid-1970s, advances in technology led to the invention of the microprocessor – a single chip that could act as an entire CPU, rather than the many different components that made up a mainframe CPU. This in turn meant the emergence of microcomputers: small, relatively cheap computers that could be used at home.

The first models were sold in kit form and were very limited (like the Altair 8800, which had only 256 bytes of RAM, and only switches and lights for input/output); but very quickly, home users could get machines that were cheap, fairly easy to set up (they would often plug into a TV as a monitor), and genuinely useful. Classic microcomputers of this era included the Commodore 64 (the single highest-selling computer model of all time); the Sinclair ZX-80, ZX-81 and Spectrum; the BBC Micro; and the Apple II. All of these (and pretty much every other microcomputer of the time) had some variety of BASIC as a built-in primary programming language and operating environment. You didn't just write your programs in BASIC, you used BASIC to run them, and you could type BASIC statements straight in at the prompt once the machine started.

Type-in programs – long listings for the user to type in directly – were very popular in books and in computer magazines. A lack of cheap portable storage media (some machines took tapes, but packaging a tape with a magazine was expensive in the 70s; and few people had modems or bulletin board access), combined with the fact that programs had to be fairly short due to the memory and other limitations of the machines, meant that it was possible to type in even quite complicated programs. However, type-ins could take hours, and the process was error-prone for lots of reasons, including programmer error, typing error, and poor-quality printing. After the arduous process of typing in, the eager reader would then have to track down the bugs they'd introduced. When listings were all written in straight BASIC, this wasn't too hard. But as programs became more complicated, it became more common to have long listings of machine language or assembly code, with only a little snippet of BASIC which handled **POKE**ing this into various locations.

This was nearly impossible to debug. Tactics to resolve this problem included checksum programs to apply to each line of machine code, but it made type-ins ever harder to use. Early on, you could often send a small sum to the programmer in exchange for a tape of the program, and by the mid-1980s it was becoming more common for magazines to include tapes on the cover.

Another issue was that there were lots of different dialects of BASIC (all the manufacturers mentioned above had their own versions). Some programs might be transferable, or universal, since there was a shared core set of keywords, but the detail of keyword implementation varied, and some BASICs had keywords which others did not. (As demonstrated in the two different dialects of BASIC in the next section.) The various dialects meant that some magazines were variant- or machine-specific, and some would add notes for changes to make to the printed listing for different machines. They would also add suggested changes that users could make to alter the printed program, promoting the fundamental idea behind BASIC that programming was something anyone could do.

In 1984, *COMPUTE!* Magazine published a type-in word processor, SpeedScript (later also published as a book), which may have been the high point (in one sense, at least) of type-in programming. In 1988, the magazine discontinued all type-in programs, and type-ins in general faded around that time, though for 8-bit machines they lasted into the 1990s.

BBC BASIC emulator

There are various emulators available for various different manufacturers and brands of machine, but one of the easiest to use (and of a brand which was very popular in the UK at the time) is the JavaScript

implementation of the BBC Micro JSBeeb (at **http://bbc.godbolt.org**). You can load your own disc images, as well as several discs from the **StairwaytoHell.com** archive; but you can also type BASIC files in line-by-line directly to the emulator. (Be warned that some of the keys behave a bit strangely; I had to experiment to work out where it thought keys like **=**, **+**, *****, etc were.)

You can type in the program listings here exactly as given. If you type in a line without a line number, that line will be immediately executed. Lines with line numbers are stored in memory. If you re-enter a given line by number then the previous one is overwritten. You can list the program currently in memory with **LIST**, and delete a range of lines with **DELETE 10-100**.

The four lines below comprise the first program I remember writing in BASIC:

```
10 PRINT "HELLO, WHAT IS YOUR NAME?"
20 INPUT NAME$
30 PRINT "HELLO " NAME$
40 END
```

Once you've typed that in, type **RUN**, which runs the lines in memory, and it should do what you would expect. BASIC listings at this sort of level are pretty self-explanatory! Note that to get a string variable, you need to use a name ending in **$**; without that the default variable type is numeric. Here, if you don't use the **$**, it will try to translate the input into a number (and doubtless output something odd).

You can also define arrays in BASIC with this line:

```
DIM MyVariable(20)
```

which will create a numeric array of length 20. Keywords in BBC BASIC must be in capitals; variable names can be lower case or upper case as you prefer (but are case sensitive). (It was common at the time just to stick caps lock on and use that for everything, to avoid errors with keywords.)

Note that if you would rather run this on bwbasic, you need to change line 30:

```
30 PRINT "HELLO "; USERNAME$
```

which is one illustration of the differences between different versions of BASIC.

Now here's a dice simulation to type into the BBC BASIC simulator:

```
10 PRINT "HELLO, HOW MANY DICE TO THROW?"
20 INPUT DICENUMBER
30 PRINT "YOU WANT TO THROW " DICENUMBER " DICE."
40 FOR I = 1 TO DICENUMBER
50 DICERANDOM = RND(6)
60 PRINT "DIE THROWN " DICENUMBER
70 NEXT
80 END
```

This demonstrates the **FOR...NEXT** loop. As with modern code, you specify start and end, and optionally step up (1 being the default). At line 50, we use the keyword **RND** to generate a random number. With BBC BASIC, **RND** without a parameter generates a random number between 0 and 1 (exclusive of 1); **RND(number)** generates a random integer between 1 and number (inclusive of number). Run this with **RUN** and try throwing some dice.

GOTO Considered Harmful

BASIC contained, from a reasonably early version, the GOTO statement. A couple of years later, Dutch computer scientist Edsger Dijkstra wrote his classic essay *Go To Statement Considered Harmful*, arguing that the GOTO statement encourages messy programming and makes it too easy to lose track of the program process (roughly, what is happening in the course of the program, where, and when).

However, in early versions of BASIC, due to interpreter limitations in handling FOR or WHILE (and single-line IF statements), GOTO was essential. Modern versions of BASIC deprecate it for uses other than returning to the top of a main loop.

The same simulation for bwbasic is a little different in the way it generates the random numbers:

```
35 RANDOMIZE TIMER
40 FOR I = 1 TO DICENUMBER
50 DICERANDOM = RND
60 PRINT "DIE THROWN "; CINT( DICERANDOM * 5 + 1)
70 NEXT
80 END
```

bwbasic only implements **RND** without the parameter, so our random number is somewhere between 0 and 0.9999.... The **CINT** keyword (not available in BBC BASIC, although **INT** does something similar) rounds a number down to the integer below it. So to generate our 1–6 random number, we multiply by 5, add 1, and round down.

An easy improvement of this program would be to enable the user to specify how many sides the dice have, as well as how many dice to throw. Beyond that, play around with it as you like.

BBC BASIC has also been updated and made available for various platforms including Z80-based computers. The manual and downloads for the Z80 and DOS version are available online here (**www.bbcbasic.co.uk/bbcbasic/bbcbasic.html**). These versions are intended to be as compatible as possible with the BBC BASIC that ran on the BBC Micro series computers, so the manuals available here are your best option if you want to experiment more with the emulator. From the same site, you can also download Brandy BASIC for Linux, which you will have to compile to run.

Despite some disparagement over the years, BASIC had a significant impact on a generation of coders and on a particular approach to more intuitive programming. That built-in BASIC prompt during the microcomputer era also meant that a generation of computer users were accustomed to the idea of programming and adapting the computer for your own purposes – in itself a hugely positive idea. Modern computers are far superior in almost all regards to those early microcomputers, and modern programming languages far more powerful and flexible than BBC BASIC and its ilk. But the sheer ease of access does set BASIC apart from the rest. At least, I'm pretty sure that's not just the nostalgia talking... LV

Juliet Kemp is a programming polyglot, and the author of O'Reilly's *Linux System Administration Recipes*.