**Back to BASIC in 2014**Jon Silvera  
Founder FUZE Technologies Ltd

There’s a lot of talk about programming at the moment. The Government have gone all-out to get it back on the curriculum to make sure schools are teaching kids how to program from an early start. In fact from this September schools must teach the fundamentals of computing in early primary and text based programming from key-stage 3 onwards (age 12).

Whether they are going about it in the right or wrong way is not really relevant. What is important is that something ***is*** done. As it was in the good old days…

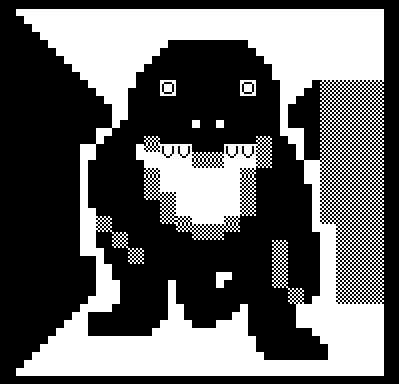
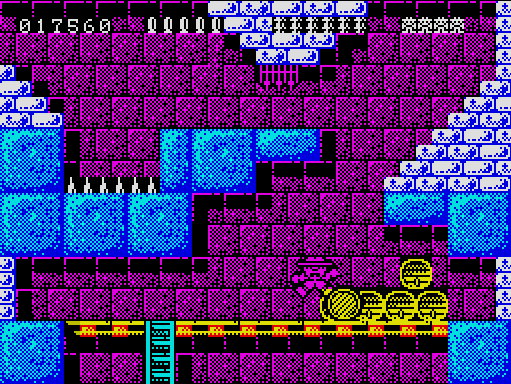
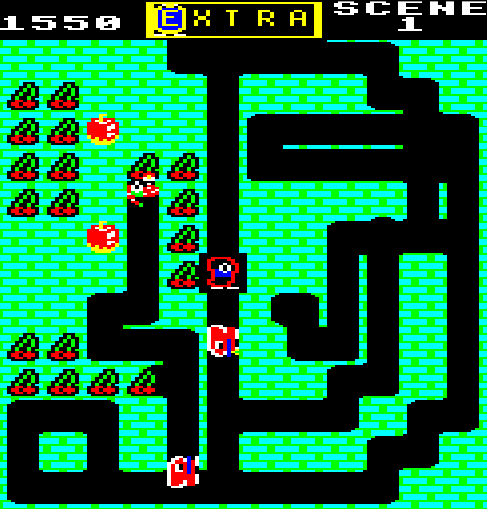
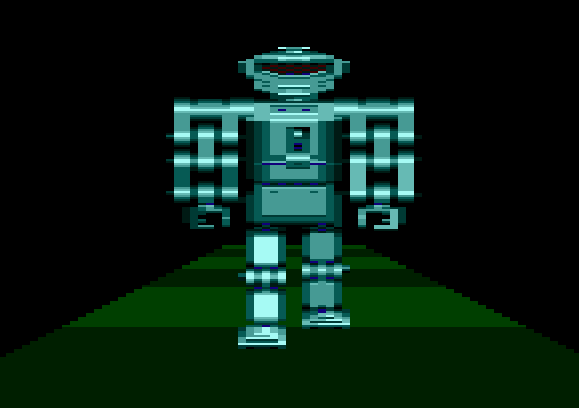
Roughly two million years ago computers were big bulky things with clunky keyboards, terrible graphics, and sound that would make your ears bleed.  
 Actually some of them didn’t have clunky keyboards, they had little rubber pads for keys instead.

These machines were limited in so many ways. For example the Sinclair ZX Spectrum had just one sound channel and the only noise it could make was a   
  
beep. In fact the command it had for this was BEEP! BEEP 1, 0 would make a “C” tone (0) for a second (1). It gets worse… while it was ‘beeping’ it couldn’t do anything else, so to make a game with music playing was impossible.

The Commodore 64 had huge great borders around the screen that were impossible to display anything on, so games were displayed on the tiny letterbox in the middle.

The BBC Micro had no hardware to display character graphics (sprites) so it was impossible to program games in anything other than text.

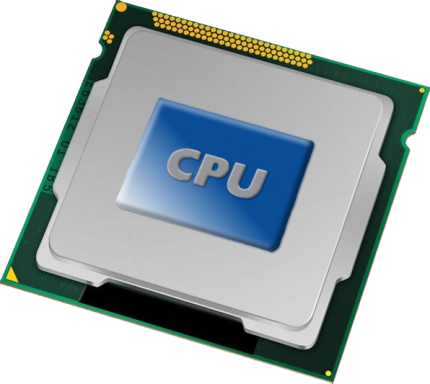
Then the wizards came. They made the Spectrum sing and dance, the BBC into an arcade machine with almost perfect renditions of Pac Man, Mr Do!, Defender and Asteroids; and the Commodore 64 a graphical and audial tour de force.



It really was the strangest time and of course it was not two million years ago, but just thirty odd. It all started back in the mid to late seventies with companies like Atari and Apple. Atari decided to turn its console technology to something more sophisticated.  
  
The Atari 400 and 800 were ahead of their time in 1979. High resolution graphics, 8 Kilobytes of RAM, 256 colours and four channel sound. Apple launched the Apple I in 1976 and then soon after the Apple II. These were powerful machines indeed, but they were also very expensive.

  
Compare this to the Sinclair ZX80 which was released a year after the Atari 400, and in the UK, by the then somewhat lesser known Clive, soon to be Sir Clive, Sinclair and it’s obvious we Brits were getting the thin end of the wedge (computer pun intended). The Sinclair ZX80 had no sound, no graphics, was black and white and had just 1 kilobyte of RAM.

It was however cheap, under a hundred pounds cheap and all things considered it can be held partly responsible for kick-starting the industry that followed.

Actually it would be very unfair to credit any one company for the advent of the Home Computing revolution. It would be more considerate to attribute this to those responsible for designing and manufacturing the CPUs and custom silicon that were the bedrock of all the computers of the day.

While there were many home computers released during 1979 and 1984 there were only three commonly used CPUs. The CPU is the brain of any computer, the **C**entral **P**rocessing **U**nit. Everything else within the computer either is controlled by or feeds information to the CPU.

The CPUs of the time were the Zilog Z80, the MOS 6502 and the less common Motorola 6800. Their designers and engineers are the people responsible for the digital world we live in today.

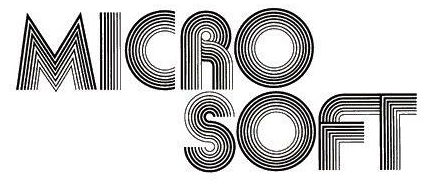
While all these machines were designed by different companies with different specifications and configurations they all had one major thing in common. They were all, every one of them, supplied with the programming language BASIC. Ok *not every single one* came with BASIC, the Jupiter Ace was of course supplied with Forth!

The earliest computers were generally programed in bare metal mode, known as machine code or the ever so slightly more friendly Assembly Language. Machine code is often slated as being the most complicated programming language ever devised. Actually it is one of the simplest and it certainly has the smallest number of instructions to learn. The complexity comes from having to know the hardware inside-out and beyond, how the memory is structured, the input output system and the video capabilities to name but a few. We won’t even go into interrupts, storage or sound.

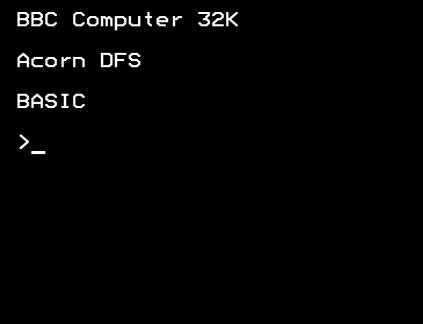
Next came a few languages designed to perform specific tasks. COBOL for example was suited to business applications whereas Fortran to mathematics and science. Actually around this time, the late fifties to the early seventies, there were loads of programming languages being thrown around. A number of them have survived or at least variations of them have. ‘C’ is possibly the best example as this is still the most commonly used language today. At least, it would be correct to add ‘C’ and its many siblings to substantiate this statement.

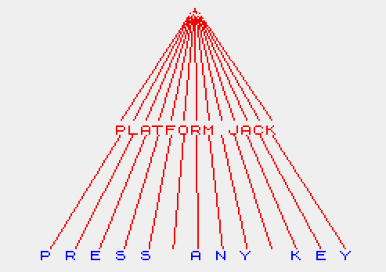
So what about BASIC then? BASIC (**B**eginner’s **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) was written to simply programming into a more understandable, less mathematically reliant language that could be used to develop a wide variety of applications.

First launched in 1964 as Dartmouth BASIC by John Kemeny and Thomas Kurtz its accessibility meant it went on to spawn many other versions over the next 20 or so years.

The first software product sold by Microsoft, was a version of BASIC co-written by Bill Gates. It went on to ship with every IBM PC and many others for years to come. In fact a number of Microsoft BASIC products are still incredibly popular. Visual BASIC, VB.NET and .NET are all in common use today.

**Back to the eighties.**

Ok so in their heyday the popular computers of the time, Apple, Tandy, Commodore, Sinclair, Atari and the Acorn BBC Micro were all supplied with BASIC built in. Switch on and you’d soon see the “Ready > \_” prompt or something similar. Type something in, press the ENTER key and see what happened. At first you’d invariably get a ‘Syntax Error’ but you soon picked up the user guide and started entering in small tutorials. The first thing to learn was how to load a game from a cassette or if you were very lucky a disk drive!

Then came the magazines with game and program listings you could type in. Hours, countless hours, were spent copying in these listings, often with very poor print quality only to be bombarded with error after error when you came to run them.

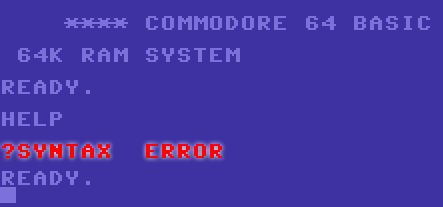
The one on the left was in ZX Computing in 1985 and was my claim to fame, “Minehead’s Jumping John Silvera” – whatever next!

The learning curve was high but very quick and before you knew it you were experimenting with your own programs, showing off to friends and if you had it in you, you’d be applying for a programming job in a fledgling gaming publisher or trying to set up you own.

That an industry was born is not surprising as over 50 million 8 BIT home computers were sold globally. The exposure to programming grabbed the interest of so many.

What happened next however was a bit odd. At the time, programming was seen as the next big thing and as such it was taught in schools, done at home or at a friend’s place and so on. However it reached a critical mass fairly quickly. Programmers went on to develop more powerful programming languages, gaming systems took over, small publishing companies grew into huge corporates and programming became something of an elite occupation. As the industry settled down, programming all but dropped off the curriculum. All of a sudden, some thirty odd years later there’s a huge wakeup call by the Government as it realises we don’t have enough programmers available in the UK to supply demand. As such a great deal of economic damage is underway due to the loss of revenues being generated by farming out programming requirements overseas.

And so thirty odd years on, here we are again. However this time around there are no accessible Home Computers, no plug in and program systems to learn from. The entry point is a complex programming environment developed by programmers for programmers. So where do we start?

At primary school children are being introduced to visual programming environments like Scratch that simplify the programming experience and do a great job of introducing computational concepts. They do not however offer the exposure to real programming that inspired so many the first time round.

There is nothing quite like typing in and RUN’ing your own program.

In secondary school it looks like Python, another language derived from the early days, is being positioned as the language to best prepare for the real world. However, and from a personal perspective, Python still has quite a high entry level. Certainly to jump right in is beyond the scope of many who’ve not had any previous programming experience and as such there is a concern that the starting point is too high and will lose many potential candidates before they get started.

BASIC however, has also been developed over the years, not to make it more complicated but more to keep it up-to-date and comparable to many other modern languages. Commands that were once considered bad practice (GOTO & GOSUB) along with line numbers are gone and structured programming is the name of the game.

The main attraction to BASIC remains its instant accessibility – enter your program and RUN, get an error and fix it and try again.

In the old days those original computers ran BASIC slowly, it was its Achilles heel. You could just about do anything with it apart from do it quickly, and that was why once you had mastered the basic principles you quickly advanced on to more complicated, compiled languages or dived into the machine and became one with the metal.

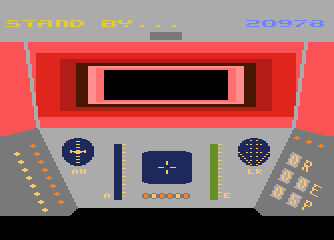
Today, BASIC runs on machines measured in Gigahertz not Megahertz and with Gigabytes or RAM not Kilobytes. It is fast enough to program sophisticated games, especially when compared to the kind of stuff being published on the Pads, Phones and Tablets of the day.

So if you’re interested in learning to program but were wondering where to start you’ll do yourself no harm in following our tutorials over the coming months.

First off though, I need to introduce you to FUZE and FUZE BASIC.



As you’ll have no doubt gathered from my observations above, I was there. From the Sinclair ZX80, 81 and Spectrum and even the QL, the Commodore 64 and the Amiga, the BBC Model B, the Atari 800, 130XE and many ST’s, and then on to the PC. We won’t mention the consoles, oh the consoles!

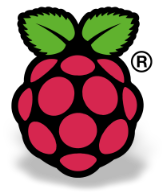
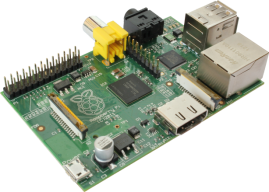
My favourites were of course the BBC due to Elite, the Spectrum as I started delving into machine code but most of all was the Atari 8 BIT series. When Lucasfilm released Ball Blazer and Rescue on Fractulas I was totally blown away. I spent a few months leaning the basics of 6502 machine code on my new Atari 130XE. It actually changed my entire outlook on computing and at the same time changed me as a person.

Up until then I had failed miserably at school, was totally under the impression I was pretty thick and had no idea what to do with myself. The discovery that not only could I program, but that I actually understood the inner workings of, what was at the time, something so very technical changed my perspective. A year later and I was teaching all age groups to program, employed as a professional computer tutor, at just eighteen.

My confidence boosted, I left Minehead in Somerset and moved to London to make my start in life. The rest as they say is … well actually it’s not history, it is very much the here and now as everything seems to have led to this point.

A few years ago I brought an old BBC Micro down from the loft to show my kids how all this computing lark began. To my surprise my two girls Molly, Gracie and my son David were all intrigued by the BASIC welcome and the Syntax Error response returned by about every input.

They wanted to know more so we spent a few days learning a few programming commands and playing a few classic games (Lemonade Shop I think). It got me thinking. Wouldn’t it be great to bring back a computer in the same vein? Something that brought access to programming right to the forefront just like it was back in the eighties.

When I started the journey I looked for a system that booted straight to BASIC. The Maximite from the Australian tech guru Geoff Graham does this incredibly well. A simple single chip board with display, keyboard, sound and power inputs it can be setup in minutes and boots to a very BBC like BASIC in about a second.

The Maximite was however superseded by the then new Raspberry Pi as this represented a much more powerful and flexible system. The only downside is that it did not have an accessible version of BASIC directly suited to our needs.

Enter Gordon Henderson, the author of the WiringPi libraries and a cool version of BASIC called RTB (Return To BASIC). RTB is designed specifically to support the workings of the Raspberry Pi. A deal was struck between FUZE and Gordon to produce FUZE BASIC which includes a vast array of enhancements to tailor FUZE BASIC more in line with the requirements of the newly revised IT curriculum so that we could best target the education market with our FUZE Platform.

At FUZE we focus on the language FUZE BASIC to deliver a learning experience far more accessible to broader age and ability group than more complex languages.

Quite simply, BASIC is easier to pick up and learn than just about any other language ever devised. You do not need to be adept at maths, you do not need to understand the operating system to any great extent and you certainly don’t need to have programmed before.

It is important to add here that the Python, C+ (et al), Java and PHP programmers might scoff at us BASIC students, but I assure you BASIC has something for everyone. Even the most adept coders will find BASIC a great platform to test out ideas and experiment.

First thing then is to get FUZE BASIC.

Visit [www.fuze.co.uk](http://www.fuze.co.uk) and then go to the resources page and download the latest FUZE boot image.

You will need to unzip this file – be warned, unzipped it is 8gb and so requires a 8Gb SD Card to install it on. You will need software to copy the image as it is not as simple as just copying it over. There is a perfect utility for this called Win32 Disk Imager at <http://sourceforge.net/projects/win32diskimager/>

Download and run this, select the unzipped image and the destination SD card and then select write.   
  
This is also perfect for making backups of Boot SD Cards once you have set them up exactly as you want to.

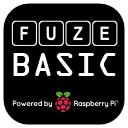
Next boot your Raspberry Pi with your new FUZE image. There isn’t anything particularly different with the boot image over the standard Raspberry Pi one, just a new background and a FUZE BASIC icon.

First off, navigate to the Fuze folder, double click it (enter it) it and then double click the Programs folder. Then create a new folder called MagPi.

Next off we need to download a few graphics for our game tutorial.

Please go to [www.fuze.co.uk](http://www.fuze.co.uk) then to the Resources page. You will see a Tab called Tutorials and within this a section called The MagPi Tutorial.

I’m hoping you are up to speed with downloading and saving files. You need to download the six Sprites. These are the Player Ship, the enemy rocks and the ever important bullet.

Download and copy these six files into the MagPi folder we created earlier.

Back to the FUZE desktop, double click the FUZE BASIC icon to start FUZE BASIC.

The Welcome Screen will appear and you’ll be presented with the Ready >\_ prompt.

First off we should get used to the environment. The Ready> prompt means you are in Direct mode. Type in “Hello” and press Enter (not with the quotes). You’ll get and “Equals expected”. That’s exactly what should happen. The computer has no idea what Hello means. Try this;

Type;

**Number1=10 (press enter)**

**Number2=10 (press enter)**

**Answer = Number1+Number2**

I suspect there are many of you already more than comfortable with what’s going on here but we need to explain to the newcomers.

The words “Number1”, “Number2” and “Answer” are just names. They are called variables. Variables are tags we store values in. We could have used any words but generally it is best to use names that make sense. If we write a program using variables like N1 and N2 and A then when we come back to them later we’ll have no idea what everything means. Whereas “ShipX” and “ShipY” are obvious. Try and make this a habit. You will appreciate it later.

So, we stored the number 10 in the variable name “Number1” and 10 in the variable “Number2”. We then said that the variable “Answer” = “Number1” + “Number2”.

At this point you should know what the value of “Answer” is. Do you? I hope so or we’re in big trouble!

Type;

**Print Answer (press enter)**

You should see

**20**

If anything else whatsoever happens then something has not gone to plan and you should go back and check where you went wrong.

On the basis you did see “**20**” returned we can proceed.

We are currently in “Direct” mode. This is where we can enter commands and expect an immediate response. We can’t do loops or a sequence of commands but we can check variables and enter simple instructions.

It’s not programming though is it? Press **F2** to enter the FUZE BASIC Editor.

You will see a blank screen with a green flashing cursor and a dotted line across the bottom. This is the Editor environment. Here we can enter a list of program instructions that will be saved and executed (**RUN**) whenever we want.

Press **F2** again. This will take you back to Direct Mode. Actually it will ask you for a file name. In this first case don’t bother, just press **F2** again and it will put you in Direct mode again. One last time, press **F2** again and you will be back in the Editor. You get the idea; **F2** takes you between the Editor and Direct mode.

Make sure you’re back in the Editor. Type in the following program

**CYCLE  
 PRINT “Hello MagPi”  
 REPEAT**

You don’t actually need to worry about capitals or lower case – it’s a great habit to type in commands in capitals but it is not essential. However the names we give to variables as we did above with “Number1” and “Answer” etc. are set in stone. If we give a variable the name “NUMBER1” then we must refer to it as such every time in the future. If we expect the number “numBER1” to return the same result, we’re in for a big surprise. The variable “numBER1” has not been defined so will generate an error.

Ok, enough of the dull stuff. You should at this point be in the Editor with the program as listed above. Press **F3** – if at this point the program hasn’t been saved it will ask you to do so. Just enter a name like “test” and press Enter. The program should then run. “Hello MagPi” should display in a never ending list down the screen.

To stop it, hit the **ESC** key.

Press **F2** to go back to the Editor and change the **PRINT** line so that it looks like the following;

**CYCLE  
 PRINT “Hello MagPi ”;  
 REPEAT**

The only difference is that we have added a space in between MagPi and the quotation mark and added a semi colon to the end of the **PRINT** line. The semi colon tells BASIC to display the next item next to the last one and not on a new line, the space just puts a gap in between. Press **F3** to **RUN** the program again. This time instead of a long list of “Helllo MagPi”’s going down the screen, this time it displays “Hello MagPi ” across the screen.

Again, press **ESC** to exit the program and then **F12** to wipe the current program from memory. All going well you should have a blank screen in the Editor. If not then try pressing **F2** and **F12** until you get there. When you are in Direct Mode you can type “**EXIT**” to exit the program.

Right now we need to be in direct mode with no program in memory. If you type “**NEW**” in direct mode it will clear the memory so when you go into the Editor it will be blank.

In direct mode type;

**Dir (press Enter)**

Amongst others, you should see a Directory called MagPi (if you did everything above). Type;

**CD MagPi (press Enter)**

This will put us in the same Directory (or Folder) where we saved the Sprites above. When we create our program we want it to be in the same Folder as the Sprites.

Press **F2** to go to the Editor and enter the following Program;

**PROC Setup  
  
CYCLE  
REPEAT  
  
END  
  
DEF PROC Setup  
  
ENDPROC**

Then press **F3** to **RUN** the program. The first time, it will ask you for a file name. Enter “Game” and press enter. You don’t need to enter the quotation marks or a file extension. The FUZE BASIC Editor will automatically add the file extension “.fuze”. When you press Enter the program will **RUN** but nothing of any interest will happen as we haven’t done anything of interest yet. If you have entered anything incorrectly you may get an error in which case **F2** will take you back to the editor. All going well the screen will just go blank as the program is in an infinite loop (**CYCLE** / **REPEAT**).

Press the **ESC** key and then **F2** to return to the editor.

This is the basic structure of our program. It is important as we progress to try and build some good habits. When naming Variables a popular method is called Camel Text. ThisIsCamelText. The reason we use it is because we’re not allowed to use spaces in variable names. Camel Text (notice the humps) makes things readable at a glance.

Most programming languages are case sensitive and FUZE BASIC is no exception although it is not particularly stubborn about it. Variables name are case sensitive so **CAT**, **cat** and **Cat** would be seen as three different variables. Commands and system variables however are not sensitive. **Print**, **print** and **PRINT** will all work.

As you write larger programs another variable issue will raise its head. Short, non-related variable names WILL cause you grief later. A 200 plus line program will be very difficult to make sense of if you have used things like “**pbx**” for “**PlayerBulletX**” and just “**X**” for the “**PlayerX**”. Those long names sure do take more time when editing, but man they will save hours later when debugging. Also consider at some point your code might be scrutinised by someone else. You do want to impress them don’t you… let alone make your program legible!

Back to our Game

We are going to store our Variables in the **PROCEDURE** Setup along with our Sprites and sound files. This keeps our program nice and tidy. The **CYCLE** and **REPEAT** commands define our main program loop. This is where all the action will happen.

Now we need to load the Sprites so we can start having some fun. Go to the editor with F2 if you’re not already in it.

Edit the code so it becomes;

**PROC Setup  
  
CYCLE  
REPEAT  
  
END  
  
DEF PROC Setup  
  
ENDPROC**