Coding Projects & Ideas

This section features some fantastic coding projects and ideas that you can use in your own code. Want to add something a little extra to your code? How about creating a loading screen, or some amazing text animations using ASCII characters?

There's code here to track the ISS in real-time, and we even look back at some retro coding, using BBC Micro BASIC, to give you a better understanding of how other forms of code come together and work.

There are always more ideas to code, so keep learning and keep coding!

Passing Variables to Python

Here's an interesting coding tutorial: how to pass a system variable to Python. By this we mean, if you create a variable within a Windows batch file or Linux Bash script, then you're able to use that same variable from inside Python.

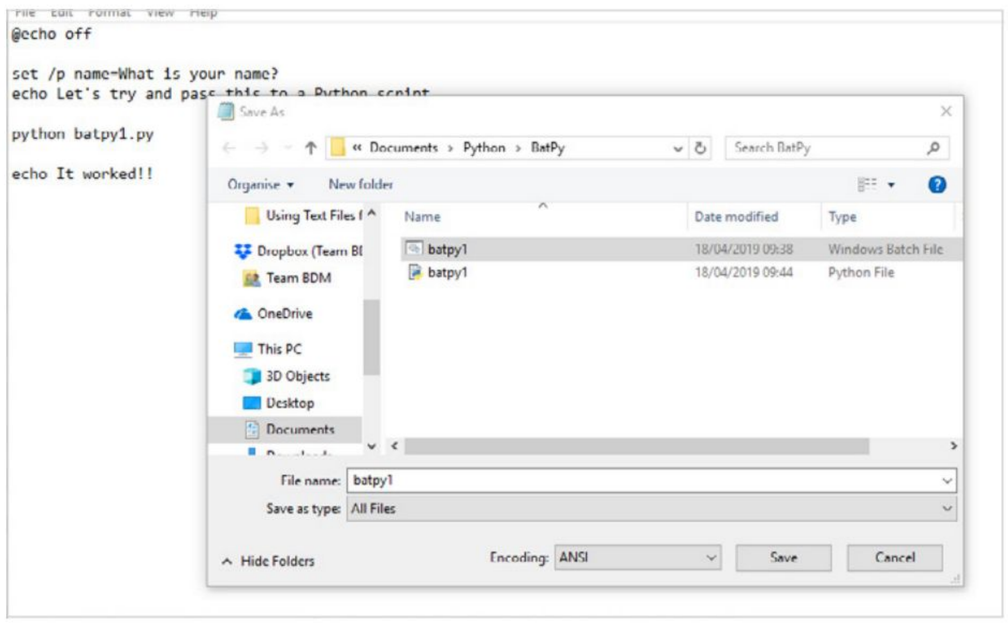
THE WINDOWS WAY

The two systems use slightly different ways to accomplish the same task; Windows is slightly easier (by a single command) so we'll start with that.

STEP 1 Let's begin by creating a sample folder within your Python directory; call it **batpy**, for example. Within the batpy folder, create a new text file and enter the following:

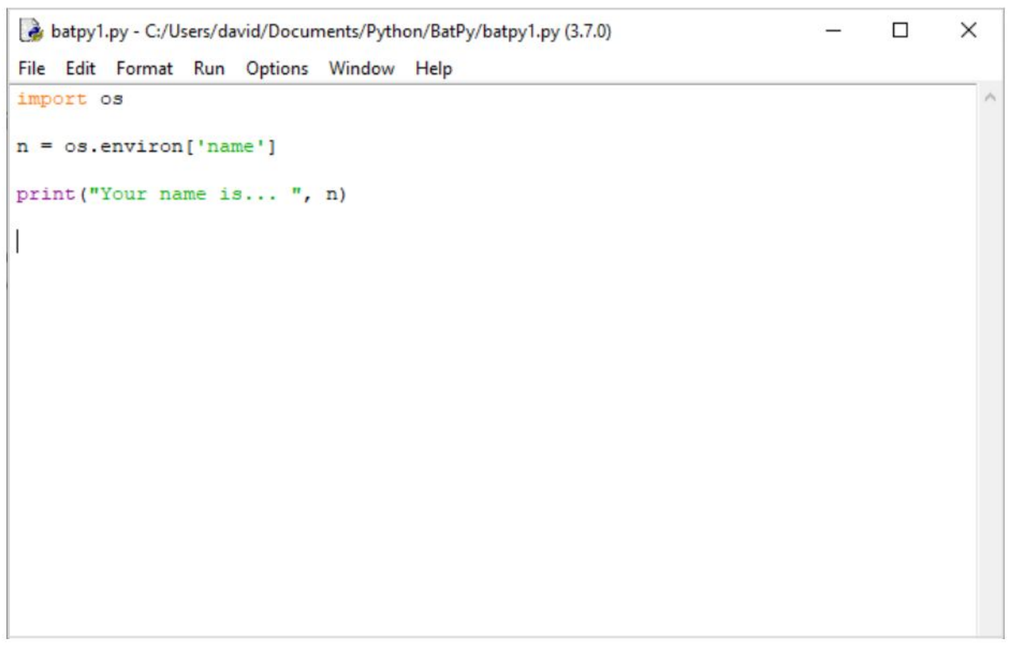
```
@echo off

set /p name=What is your name?
echo Let's try and pass this to a Python script...

python batpy1.py

echo It worked!!
```

STEP 2 We won't go into the intricacies of Windows batch files here. Suffice to say this code will ask for a user's name, store the name as a variable called **name**, display a message, then open the Python code we're about to create. We need to save the file as a batch file, click **File > Save As**, call it **batpy1.bat** and set **Save As Type** as **All Files**.

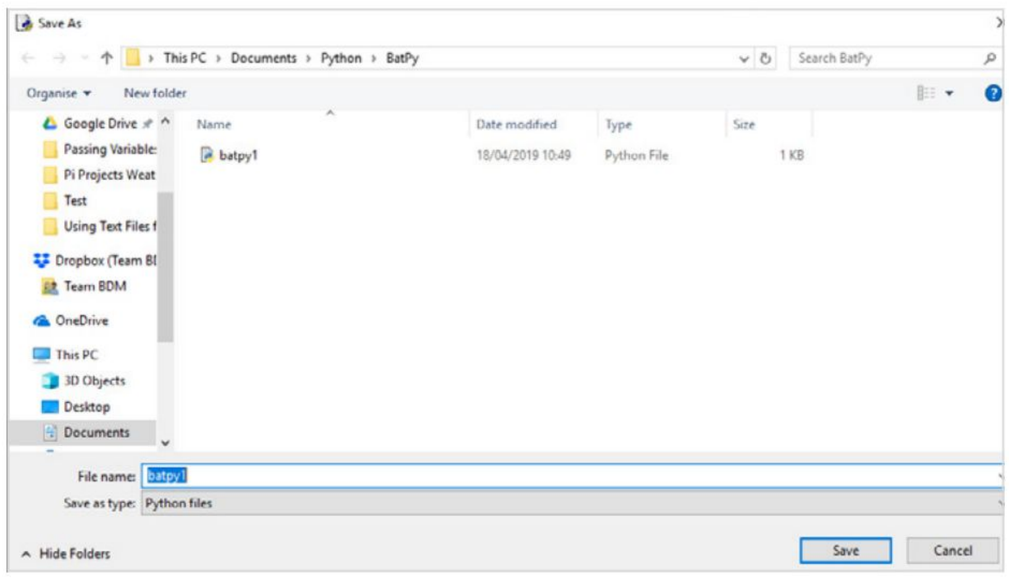


STEP 3 Now open Python and create a New File. Enter the following code:

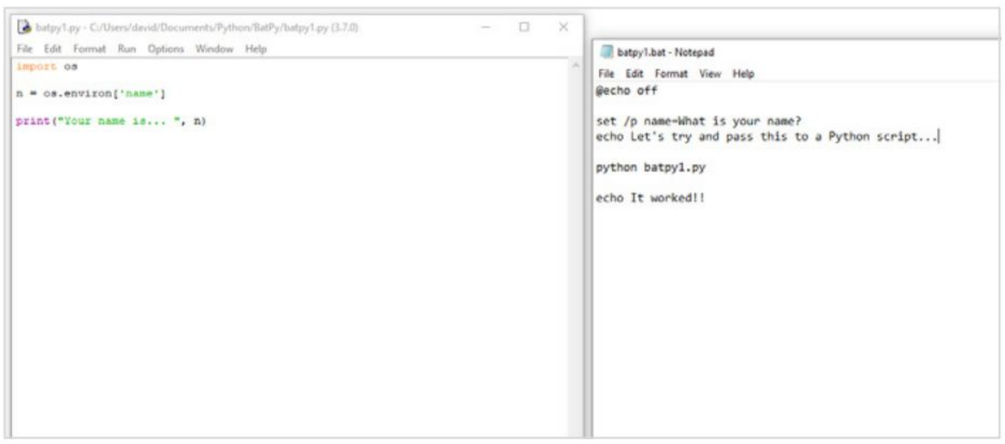
```
import os

n = os.environ['name']

print("Your name is... ", n)
```



STEP 4 Here we're using the OS module that will utilise the system variables. When we create a variable in a batch file, (or Bash script) we're storing the contents of the variable as a system variable. In Python, we're passing **n** as the variable content of the command **os.environ**. Os.environ is calling the system variable '**name**', which we created in the batch file. Save the code as **batpy1.py**.



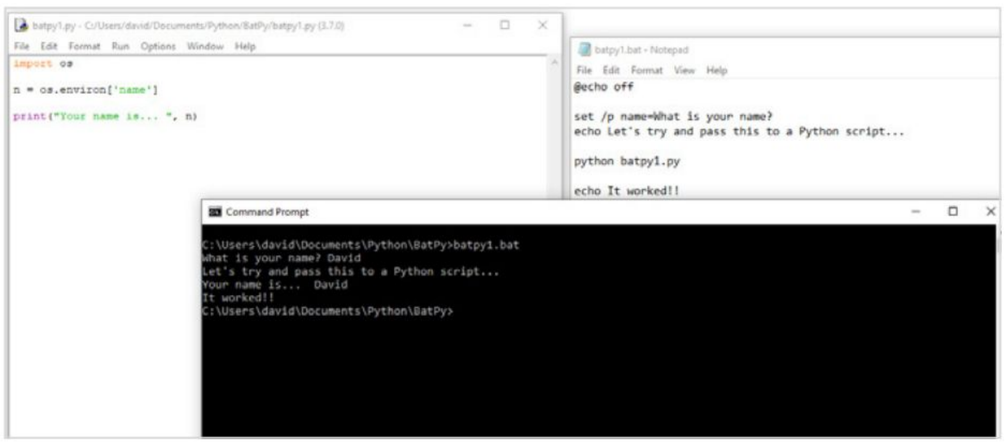
STEP 5 To recap: we have a batch file called **batpy1.bat**, which asks the user their name and stores the info as a system variable called **name**. It'll then print to the screen a message, then run the Python code, and finally print "It worked!!". The Python code, called **batpy1.py**, uses the OS module to call **os.environ['name']**, stored as **n**. It'll then print the value of n after a message.



STEP 6 Drop into a Windows command prompt and navigate to the folder Batpy. To run a Windows batch file, simply enter the command:

```
batpy1.bat
```

Enter your name as instructed and the reply will be passed to Python, then back to the batch file to end.

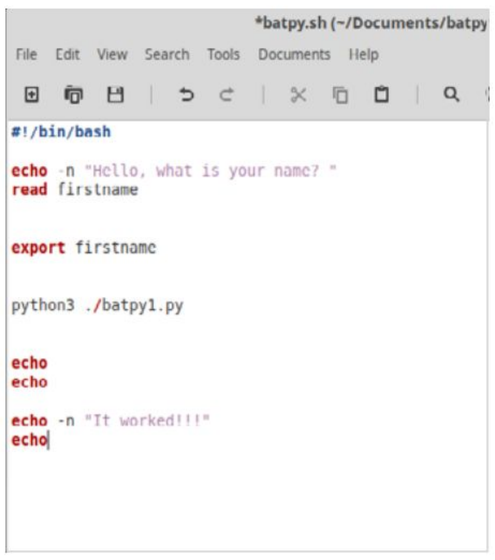


THE LINUX WAY

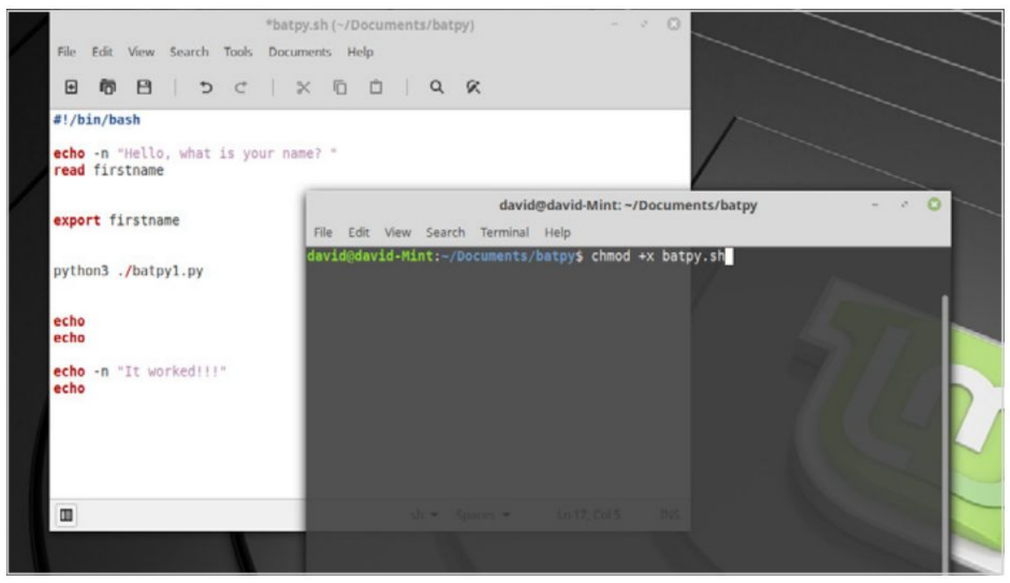
Linux's version of passing variables to Python is slightly different, just because it's Linux! It's easy, though, and here's how it's done.

STEP 1 Start by opening your favourite Linux text editor, and entering the following:

```
#!/bin/bash

echo -n "Hello, what is your name? "
read firstname

export firstname

python3 ./batpy1.py

echo
echo

echo -n "It worked!!!"
echo
```



STEP 2 Drop into a Terminal session and make a new directory called batpy (**mkdir batpy**). Save the Bash script as **batpy.sh**, and from the Terminal enable the script as executable with: `chmod +x batpy.sh`.

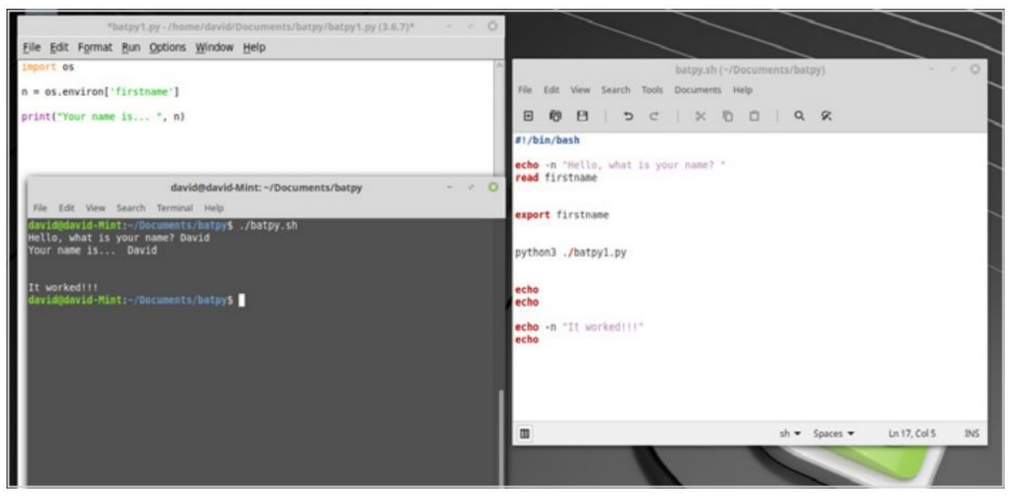


STEP 3 Now create a new Python file, and enter the following:

```
import os

n = os.environ['firstname']

print("Your name is... ", n)
```

Save the Python code as **batpy1.py**.

STEP 4 Back in the Terminal, enter: `./batpy.sh` to run the Bash script. As you can see, the results are the same as in Windows. The major differences are making the Bash script executable, and adding the **export firstname** command to the Bash file. In Linux, you need to export system variables before they can be accessed by the subsystem, in this case the Python code.



Retro Coding

There's a school of thought on the Internet, that to master the foundations of good coding skills you need to have some experience of how code was written in the past. In the past is a bit of a loose term, but mostly, it means coding from the 80s.

THE GOLDEN ERA OF CODE

```
A 002 TRITEXT
A 003 CALC
A 002 RUNAVG
A 002 UNITCONV
A 002 SQUARE
A 002 HWORLD

]LOAD UNITCONV
]LIST

 5  HOME
 10  PRINT "---UNIT CONVERTER---"

 20  INPUT "[F]AH-CEL OR [C]EL-FA
     H: ";C$
 30  INPUT "ENTER UNIT: ";UN
 40 X = (UNIT - 32) * 5 / 9
 50 Y = (UNIT * 9 / 5) + 32
 60  IF C$ = "F" THEN  PRINT "CEL
     SIUS: ";X
 70  IF C$ = "C" THEN  PRINT "FAH
     RENHEIT: ";Y

]
```

While it may seem a little counterproductive to learn how to code in a language that's virtually obsolete, there are some surprising benefits to getting your hands dirty with a bit of retro coding. Firstly, learning old code will help you build the structure of code as, regardless of whether it is a language that was developed yesterday, or forty years ago, code still demands strict discipline to work correctly. Secondly, everyday coding elements such as loops, sub routines and so on are a great visual aid to learn in older code, especially BASIC. Lastly, it's simply good fun.

GOING BASIC

The easiest retro language to play around with is, without doubt, BASIC. Developed back in the mid-sixties, BASIC (Beginner's All-Purpose Symbolic Instruction Code) is a high-level programming language whose design was geared toward ease of use. In a time when computers were beginning to become more accessible, designers John Kemeny and Thomas Kurtz needed a language that students could get to grips with, quickly and easily. Think of BASIC as a distant relation to Python.

```
1050 REM FOR I=DLSTART TO DLEND
1060 REM PRINT I,PEEK(I)
1070 REM NEXT I
1080 REM
1090 POKE 512,0
1100 POKE 513,6
1110 REM
1120 FOR I=1536 TO 1550
1130 READ A
1140 POKE I,A
1150 NEXT I
1160 REM
1170 FOR I=DLSTART+6 TO DLSTART+28
1180 POKE I,130
1190 NEXT I
1240 POKE 54286,192
2000 REM
2010 DATA 72
```

THE BEEB

The problem with BASIC is that there were so many different versions available, across multiple 8-bit platforms, with each having its own unique elements on top of the core BASIC functions. The BASIC that was packaged with the Commodore 64 was different to that on the ZX Spectrum, or the Atari home computers, due to the differing hardware of each system. However, it's widely recognised that one of the 'best', and possibly most utilised, form of BASIC from the 80s was that of BBC BASIC.

BBC BASIC was used on the Acorn BBC Micro range of computers, utilising the MOS 6502-based processor technologies. It was one of the quickest examples of BASIC and, thanks to an inline assembler, it was also capable of allowing the developers of the time to write code for different processor types, such as the Zilog Z80 – a CPU present in the ZX Spectrum, as well as many arcade machines.

The BBC Micro was designed and built by Acorn Computers – a company that is historically responsible for the creation of the ARM CPU - the processor that's used in virtually every Android phone and tablet, as well as smart TVs, set top boxes and so on. The BBC Micro was born in a time when the UK government was looking for a countrywide computer platform to be used throughout education. Different companies bid, but it was the BBC's Computer Literacy Project (the BBC Micro) that was chosen, due to its ruggedness, upgradability, and potential for education. As a result, the BBC Micro, or the Beeb as it's affectionately known, became the dominant educational computer throughout the 80s.

BEEBEM

Naturally, you could scour eBay and look for a working BBC Micro to play around on, and it'll be a lot of fun. However, for the sake of playing around with some retro code, we'll use one of the best BBC Micro emulators available: BeebEm.

BeebEm was originally developed for UNIX in 1994 by Dave Gilbert and later ported to Windows. It is now developed by Mike Wyatt and Jon Welch, who maintain the Mac port of the emulator, and is therefore available for Windows 10, Linux and macOS, as well as other platforms.

If you're using Windows 10, simply navigate to **http://www.mkw.me.uk/beebem/index.html**, and download the BeebEM414.exe that's displayed in the main screen.

Once downloaded, launch the executable and follow the on-screens instructions to install it. Linux users can find the installation files, as well as full instructions, at: **http://beebem-unix.bbcmicro.com/download**.html. MacOS users can get everything they need from: **http://www.g7jjf.com/**.

BBC BASIC

Once installed and powered up, BeebEm will display the default BBC system start-up, along with a couple of beeps. Those of you old enough to have been in a UK school in the 80s will certainly recall this setup.

In BASIC, we use line numbers to determine which lines of code run in sequence. For example, to print something to screen we'd enter:

```
10 print “hello”
```

Once you've typed the above in, press Enter and then type:

```
run
```

We can of course expand the code to include variables, multi-line print statements and so on:

```
1 CLS
10 Input “Hello, what’s your name? “ n$
20 print
30 print “Hi, “ n$ “ I hope you’re well today.”
```

Type RUN to execute the code, you can also type LIST to view the code you've entered.

```
HELLO, WHAT'S YOUR NAME? DAVID

HI, DAVID I HOPE YOU'RE WELL TODAY.
>LIST
    1 CLS
   10 INPUT "HELLO, WHAT'S YOUR NAME? "
N$
```

As you can see, variables are handled in much the same way as Python, a print statement on its own displays a blank line, and CLS clears the screen – as used in conjunction with the OS module in Python and when running Windows. We're also able to do some maths work, and play around with variables too:

```
1 CLS
10 input “how old are you? “ a
20 print
30 if a > 40 print “You’re over 40 years old.”
40 if a < 40 print “You’re under 40 years old.”
50 print
```

BBC BASIC also has some interesting built-in features, such as the value of PI:

```
1 REM Area of a circle
2 CLS
20 Input “Enter the radius: “ r
30 let area = PI*r*r
40 print “The area of your circle is: “; area
50 print ‘’’
```

As you'll notice, variables with a dollar ($) represent strings, nothing after the variable, or a hash (#) represent floating point decimals; a whole integer has a % character, and a byte has an ampersand (&). The single quotes after the Print on line 50 indicate a blank line, one for each tick, while REM on line 1 is a comment, and thus ignored by the BASIC compiler.

Needless to say, there's a lot you can learn, as well as having fun, with BBC BASIC. It's a rainy day project and something that's interesting to show the kids – this is how we rolled back in the 80s, kids!

There are a number of sites you can visit to learn BBC BASIC, such as **http://archive.retro-kit.co.uk/bbc.nvg.org/docs.php3.html**. See what you can come up with using BBC BASIC, or other BASIC types for different systems, and let us know what you've created.

OTHER SYSTEMS

Naturally, you don't have to look to the BBC Micro to play around with some retro code. If you grew up with a Commodore 64, then you can always try VICE, the C64 emulator. Likewise, the ZX Spectrum has a slew of great emulators available for every modern system to play around on. In fact, you can probably find an emulator for virtually every 8-bit or 16-bit machine that was produced over the years. Each has their own unique perspective and coding nuances, so find a few and see what you can create.