smolBASIC

first steps in text-based coding

GILES BOOTH

Published by Giles Booth, London, UK

Edition 1.1

suppertime.co.uk/blogmywiki
Mastodon: @blogmywiki@mastodon.social

smolBASIC

first steps in text-based coding

GILES BOOTH

Contents

1 Introduction

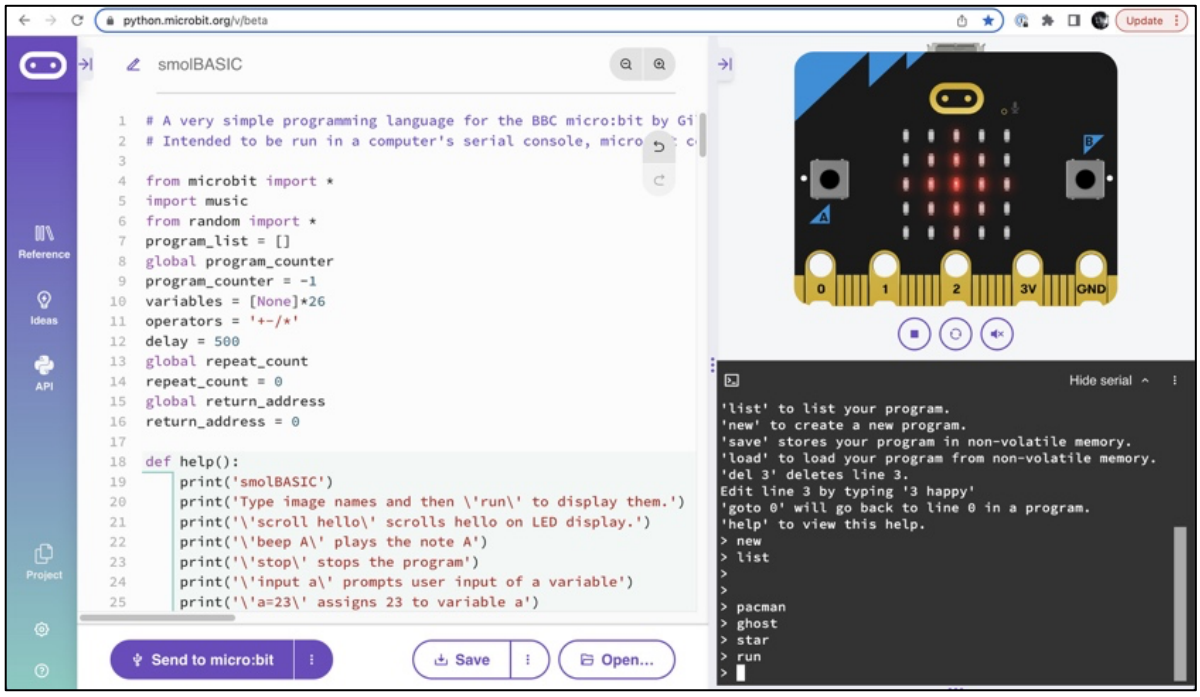
What is smolBASIC?

smolBASIC is very simple text-based programming language for the BBC micro:bit, written in MicroPython. You can use it in an online simulator, or in conjunction with a micro:bit V2 board.

smolBASIC could be a child's first text-based programming language, or even their first ever programming language. It's also simple enough that older students can modify it themselves, for example adding instructions to show more images or access more sensors on the micro:bit or play more sounds.

It aims to be as simple as possible, so it avoids capital letters and punctuation such as quotation marks, commas, or brackets that students find hard to type.

Despite being so simple, it also lets you save smolBASIC programs to the micro:bit's non-volatile memory, so you don't lose your work (if you remember to save it!)



SmolBASIC Python source code running in the simulator in the official BBC micro:bit Python Editor

Why use smolBASIC?

Its simplicity and immediate physical results make smolBASIC an engaging first text-based language.

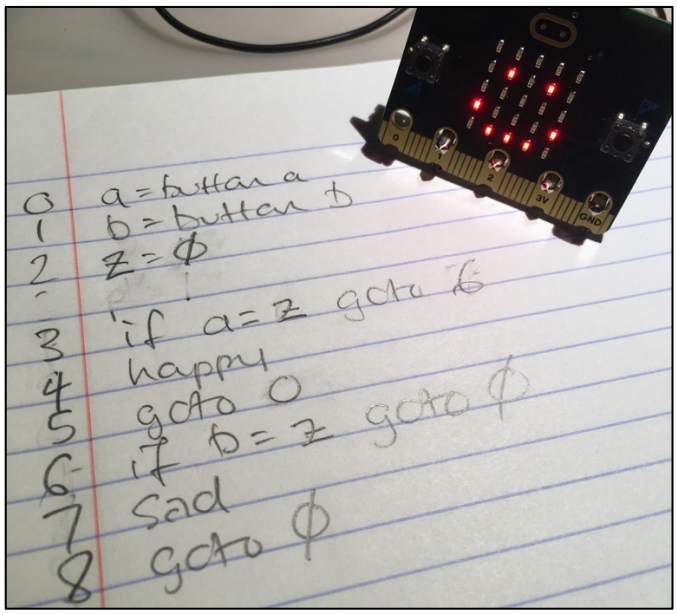
A student can simply type

```
heart
pacman
happy
run
```

and those icons appear in sequence on the BBC micro:bit's LED display. They have written and executed a text-based program and sequence without worrying about line numbers, syntax, or punctuation.

Its limitations, which bring simplicity, also allow students to access some of the clarity of thought that comes with learning assembly language or machine code, but without having to learn hexadecimal, binary or memorise op codes. It encourages thinking, using computational thinking skills to plan programs on paper, cross things out, revise before even typing any code.

By default, every instruction takes half a second to execute. Even the pace at which a program runs allows students to think about how their programs work and what they do.



Plan for an emotion badge program – you can see where I deleted a line that wasn't needed and changed line numbers before I typed the program in

Why smolBASIC not Python?

smolBASIC, as the name suggests, is based on BASIC, although it has some key differences.

BASIC was developed in the 1960s at Dartmouth College in New Hampshire to make computing more accessible to a wide range of students studying humanities as well as scientists. BASIC's commands and syntax are easy to learn, as they closely match natural language. It was widely adopted in home computers of the 1970s and 1980s, but it has not been chosen as a basis for smolBASIC for reasons of nostalgia. BASIC was picked because it is easy to learn, arguably even more so than Python.

smolBASIC has an extremely limited set of instructions and syntax structures, making it even smaller than Tiny BASIC, for example.

Some key differences between traditional versions of BASIC and smolBASIC:

- Commands are not executed when they are entered. They are stored in a program for later execution. So, programs have no line numbers when entered.
- Program lines **do** have numbers when listed, so we can use 'goto' statements. This also encourages some planning of programs, for example on paper, before you start typing code.
- Line numbers start at 0 and must be consecutive in steps of 1.
- You can change a line by typing its line number and a new command.
- You can delete individual lines. All subsequent lines move up and are renumbered accordingly.
- You can only have one command per line.
- Arithmetic operations can only be carried out on variables.
- Conditional statements can only consist of comparing two variables and then jumping to another numbered line with a 'goto' statement.
- There are many single word instructions to show different images on the micro:bit's LED display, such as 'heart', 'happy', 'sad.'
- Quotation marks are not used to mark the start and end of strings.

smolBASIC and the curriculum

You can use smolBASIC, by stealth if you like, to teach some fundamental concepts of programming:

- sequence
- variables
- selection / conditionals ('if…then')
- iteration / loops

The planning needed for even simple programs is an opportunity to practice skills of computational thinking:

- algorithms – planning a sequence of instructions to solve a problem.
- abstraction – keeping only what is needed.
- decomposition – breaking a problem down into smaller chunks.
- pattern recognition – taking advantage of similarities to make a solution simpler.

It's also a good way to practice logical thinking. In the emotion badge example above, the code made shorter than it could be by applying logic. Button variables can only have the value of 1 (if it was pressed) or 0 (not pressed). So, if the value of a is not zero it must be 1, so we don't need to test for that, we can just show a happy face on the display.

smolBASIC could be used to satisfy any requirements to teach a text-based language, for example in KS3 in England, and as it's simpler than many other text-based languages, it could be introduced for younger children, as soon as they are learning to read and spell some simple words.

Encourage older students to extend smolBASIC

You might think that the Python code for the smolBASIC interpreter is pretty badly written. You'd probably be right. But the idea is that the interpreter itself should be easy for students to read, modify and extend. So clever, but opaque, language constructs are out.

Students could also translate the smolBASIC commands into other **human** languages, to give younger students the opportunity to do some text-based coding in their mother tongue.

The two types of smolBASIC

There are two versions of smolBASIC.

The simplest one just runs in the online micro:bit simulator or a real micro:bit V2. You just need a computer, or preferably a computer and a micro:bit to run this.

There's a more complex graphical version, smolBASIC-GFX, that uses an old Raspberry Pi as a dumb terminal, allowing you to build an inexpensive, self-contained computer with colour and graphics that you program in smolBASIC. That's covered in chapter 6.

I recommend you begin with the simpler version.

2 Quick start

You don't even need a BBC micro:bit to try smolBASIC.

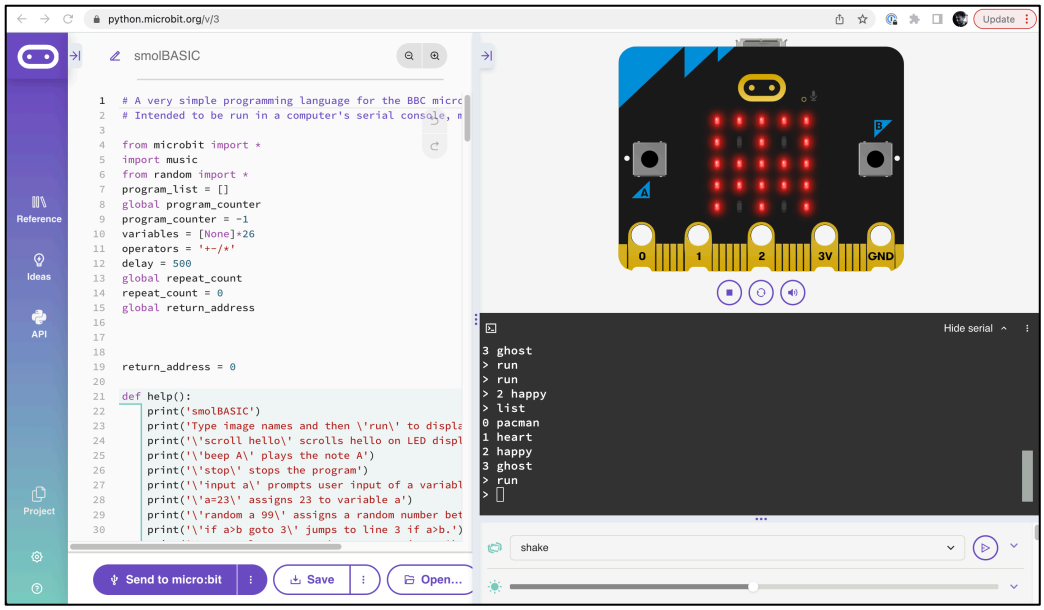
The quickest, simplest way is to load the simple, non-graphical version in the micro:bit Python Editor and use the simulator.

Go to https://github.com/blogmywiki/smolBASIC and download the smolBASIC.hex or smolBASIC.py file.

Load or drag and drop either file into the online micro:bit Python Editor at https://python.microbit.org/

Using the simulator

Click the play button on the simulator on the right. Expand the simulator serial console by clicking on 'Show serial.' You may want to collapse the Reference panel and arrange the screen so the serial console is as large as possible:



That's it! You can type instructions into the simulator's serial console (bottom right) and see the results straight away.

Using a real micro:bit

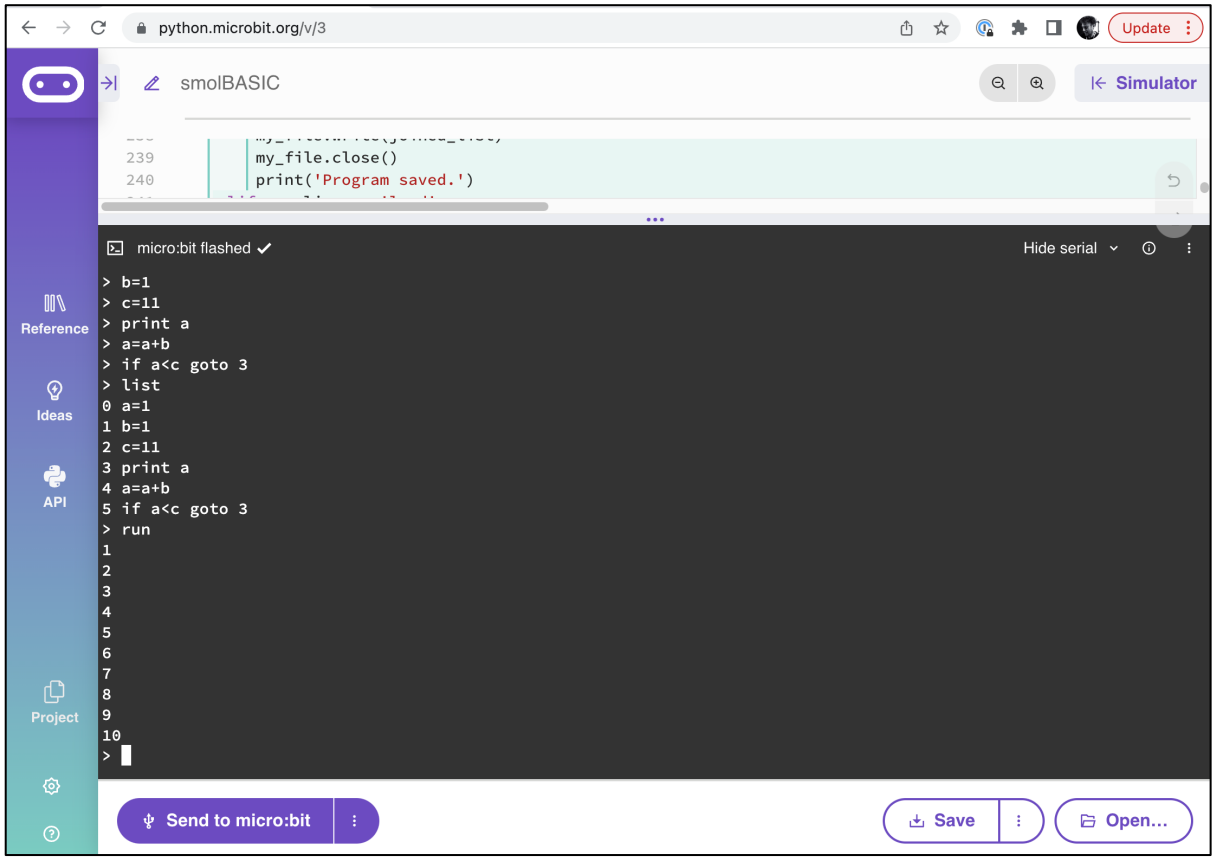
If you want to use a real micro:bit, make sure you're using a recent version of the Chrome or Edge web browsers. Connect your micro:bit V2 to your computer's USB port, click on the 'send to micro:bit button' and follow the instructions on screen.

Once you have sent the smolBASIC.py program to your micro:bit, click on 'show serial' underneath the code. You may need to press the reset button on the back of the micro:bit a couple of times until you see the prompt:

```
> smolBASIC
Type 'help' for a list of commands.
```

Now you can type your instructions into the serial console and see the results on the micro:bit and responses and program listings in the serial console too. Remember, the code is running on the micro:bit, not in the browser.

You may want to close the Reference and Simulator panels and make the serial console larger:



Instead of using the micro:bit Python Editor, you can use any USB serial console, for example https://googlechromelabs.github.io/serial-terminal/

3 Enter and run your first program

If you read the Introduction, you've already seen your first program:

```
heart
pacman
happy
run
```

Type those words in carefully, being careful not to add any spaces and pressing enter after each word.

Remember, unlike a Python shell or BASIC computer, instructions you type are stored not carried out. They are only carried out when you **run** the program.

You should see three images appear in turn on the micro:bit's LED display (either in the simulator, or on a real micro:bit if you're using one).

Type

```
list
```

and your program is listed for you:

```
0 heart
1 pacman
2 happy
```

If you saw a ? appear on the micro:bit's LED display at any point, it means you made a mistake typing an instruction. You can type over a line by entering its line number and instruction again:

```
1 pacman
```

Then type `run` to run the program again.

Save your program

It's good idea to get into the habit of saving your program. Type

```
save
```

If the program saved successfully, you'll see this message:

```
Program saved.
```

You can only save one program at a time, but saved programs stay in the micro:bit's flash memory even when you disconnect them from a power source like your computer or if you press the reset button on the back of your micro:bit.

Load your program

You can reload the saved program by typing

```
load
```

Fast and slow

Each instruction, by default, takes half a second to run. You can speed up execution of programs by typing

```
fast
```

Run the program again, and you'll probably just see the ghost icon. The heart and pacman images were shown, but over-written immediately by the next image. To slow program running down again, type

```
slow
```

If you make a mistake or want to change your program, you have a few options.

Start a new program

You can type

```
new
```

to delete the whole program and start again.

Deleting lines

You can delete single lines with

```
delete 1
```

Every subsequent line will shuffle up when you do that, so what was line 2 becomes the new line 1.

Editing lines

You can edit lines by typing the line number and a new instruction.

```
1 happy
```

4 Core language features

Output images on the LED display

smolBASIC has access to many of the micro:bit's built-in images, plus a star image that uses LEDs of different brightness. It would be a good challenge for students to add more by modifying the source Python program.

Images you can display with one-word instructions include:

```
heart
happy
sad
confused
meh
angry
asleep
confused
yes
no
duck
small heart
pacman
ghost
skull
rabbit
diamond
small diamond
star
```

Clear the LED display

```
clear
```

Scroll text on the LED display

```
scroll hello world
```

Output text to the serial console

```
print hello world
```

Sound and music

You can play very simple tunes using the **beep** command. You can't currently change the duration or tempo of the notes, but in fast mode there will be no gaps between the notes.

```
0 beep c
1 beep d
2 beep e
3 beep c
```

Sleep / pause / wait

You can pause your program for 2000 milliseconds (2 seconds) using **wait**:

```
wait 2000
```

Loops

You can repeat sections of code using **repeat** and **again**. This program plays the opening notes of Frère Jacques twice:

```
0 repeat 2
1 beep c
2 beep d
3 beep e
4 beep c
5 again
```

Goto

You can create an infinite loop using **goto** statements. Be careful though: you can only break out of a smolBASIC program like this by pressing the reset button on the back of the micro:bit or disconnecting its power source. So always **save** your code before running it!

```
0 beep c
1 beep d
2 beep e
3 beep c
4 goto 0
```

Variables

There are 26 possible variables. They have single letter names a - z. Variables can be numerical or strings of characters with no quotation marks are used.

```
a=23
b=hello
```

You can print variables but not scroll them on the LED display.

```
print a
print b
```

Input

You can assign user input typed in the console into a variable using

```
input c
```

Random numbers

Assign a random number between 0 and 99 to a variable with

```
random a 99
```

Mathematical operations

Maths operations can only be performed on variables and must be in this format

```
a=b*c
```

Only +, -, / and * operators are supported.

Selection / branching / conditionals

smolBASIC can branch conditionally with **if** and **goto** instructions. You must compare two variables.

```
if a>b goto 6
```

It supports <, > and = as operators.

Stopping a program

Stop a program executing with

```
stop
```

Sensor readings

You can assign micro:bit sound, temperature, and light level sensor readings to variables:

```
a=sound
b=temp
c=light
```

Sound and light readings are in the range 0-255.

Temperature readings are in degrees C.

Button inputs

You can read button A and button B presses using variables.

```
a=button a
```

puts 1 in variable **a** if micro:bit button A was pressed since the last test, otherwise 0.

```
b=button b
```

puts 1 in variable **b** if button B was pressed since the last test, otherwise 0.

You can use any variable a-z, but a and b seem sensible.

5 Sample programs

These programs are presented with their line numbers to make them easier to read, but remember you don't need to type the line numbers in when you type them in. Line numbers are only needed if you're correcting a line.

Voting age

```
0 print Enter your age
1 input a
2 b=17
3 if a>b goto 6
4 print You are too young to vote
5 stop
6 print You can vote
```

Guess the number game

```
0 random a 10
1 print Guess my number 1-10
2 input b
3 if a=b goto 6
4 if a>b goto 8
5 if a<b goto 10
6 print Correct!
7 stop
8 print Too low!
9 goto 1
10 print Too high!
11 goto 1
```

Looping just with ‘goto’

This will count to 10.

```
0 a=1
1 b=1
2 c=11
3 print a
4 a=a+b
5 if a<c goto 3
```

Test if button A was pressed

```
0 a=button a
1 clear
2 c=1
3 if a=c goto 6
4 print you did not press button A
5 goto 8
6 print you pressed button A
7 happy
8 beep A
```

Emotion badge

```
0 a=button a
1 b=button b
2 z=0
3 if a=z goto 6
4 happy
5 goto 0
6 if b=z goto 0
7 sad
8 goto 0
```

6 Graphical version

If you have a spare old Raspberry Pi and a USB keyboard, you can turn your micro:bit into a self-contained smolBASIC computer with a colour display.

Instead of using USB to communicate with a laptop or desktop computer that acts as your keyboard and screen, it uses a Raspberry Pi running PiGFX as a dumb terminal. Plug a USB keyboard and display into the Pi, and attach the Pi's serial, 3.3v power and GND pins to the micro:bit.

smolBASIC still runs in Python on your micro:bit, but all communication with it is done via the Pi's keyboard and screen.

Building it

You could use an old PiZero or, as I did, an old Raspberry Pi Model B as your terminal. PiGFX boots very quickly because it's not using Linux, it runs on 'bare metal'. Note that you should use a 1GB SD card for PiGFX - I could not get it to work with larger cards with 1GB FAT partitions, possibly because I was formatting them on a Mac.

Follow the installation instructions at https://github.com/fbergama/pigfx.

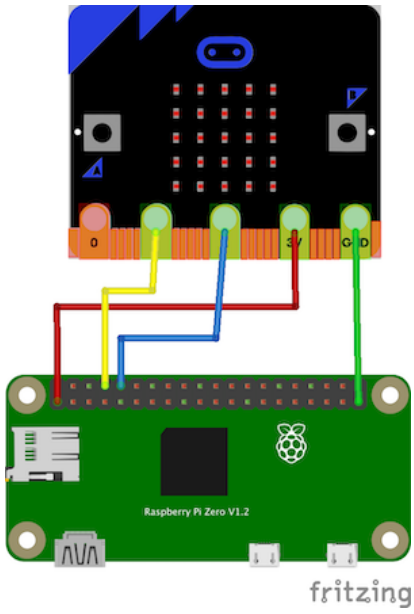
Configure PiGFX to run at 9600 baud. Connect a USB keyboard to the Pi and the Pi to a monitor via HDMI or composite video out (NTSC).

Connect micro:bit RX pin 1 to the Pi UART TX pin on the Pi (Pin 8 - GPIO 14).

Connect micro:bit TX pin 2 to the Pi UART RX pin on the Pi (Pin 10 - GPIO 15).

Connect micro:bit GND to any Pi GND pin and micro:bit 3v to the Pi 3v pin.

If you use any other terminal hardware with this, note that the micro:bit (and Raspberry Pi) data pins run at 3v, not 5v, so you may need a level shifter to avoid blowing up your micro:bit.



Colour: ink and paper

The **ink** and **paper** commands to set colours by name: red, green, blue, yellow, cyan, magenta, black and white. **ink** also accepts numbers in the range 0-255 or variable names so you can create random colours.

See https://upload.wikimedia.org/wikipedia/commons/1/15/Xterm_256color_chart.svg for a colour chart.

Clear screen

```
clear screen
```

clears the screen and sets colours back to black and white.

Shapes

```
circle 20 30 40
```

draws a circle at x=20, y=30 with a radius of 40

```
rectangle 100 200 30 40
```

draws a rectangle at x=100, y=200 of width 30 and height 40

Sample program to draw random colour and size circles

There's no break key, so save this before running. Enter fast mode, run it - then press the reset button on the micro:bit to break out of it.

```
0 random x 600
1 random y 400
2 random r 75
3 random c 255
4 ink c
5 circle x y r
6 goto 0
```

Sound level meter demo

This draws a bar that gets longer and changes colour the louder the sound the micro:bit picks up:

```
0 repeat 1000
1 a=sound
2 ink a
3 b=2
4 c=a*b
5 rectangle 0 100 c 50
6 again
```

Change colour depending on light and sound

This draws a circle that changes colour depending on light levels and a rectangle that changes colour depending on sound levels.

```
0 repeat 200
1 a=light
2 ink a
3 circle 200 200 50
4 b=sound
5 ink b
6 rectangle 100 100 50 50
7 again
```

7 Source code

The most up-to-date source code can be found at
https://github.com/blogmywiki/smolBASIC

The source code may be of interest to browse here, however.

Simple version – smolBASIC.py

```
# A very simple programming language for the BBC micro:bit by Giles Booth
# Intended to be run in a computer's serial console, micro:bit connected by USB.

from microbit import *
import music
from random import *
program_list = []
global program_counter
program_counter = -1
variables = [None]*26
operators = '+-/*'
delay = 500
global repeat_count
repeat_count = 0
global return_address

return_address = 0

def help():
    print('smolBASIC')
    print('Type image names and then \'run\' to display them.')
    print('\'scroll hello\' scrolls hello on LED display.')
    print('\'beep A\' plays the note A')
    print('\'stop\' stops the program')
    print('\'input a\' prompts user input of a variable')
    print('\'a=23\' assigns 23 to variable a')
    print('\'random a 99\' assigns a random number between 1 and 99 to a')
    print('\'if a>b goto 3\' jumps to line 3 if a>b.')
    print('You can also use < and = as comparisons.')
    print('\'list\' to list your program.')
    print('\'new\' to create a new program.')
    print('\'save\' stores your program in non-volatile memory.')
    print('\'load\' to load your program from non-volatile memory.')
    print('\'del 3\' deletes line 3.')
    print('Edit line 3 by typing \'3 happy\'')
    print('\'goto 0\' will go back to line 0 in a program.')
    print('\'help\' to view this help.')

def parse(instruction):
    global program_counter
    global repeat_count
    global return_address
    try:
        if instruction.startswith('goto '):
            split = instruction.find(' ') + 1
            line_number = int(instruction[split:])
            if line_number > -1 and line_number < len(program_list) + 1:
                program_counter = line_number - 1
            else:
```

```
        print('Goto error: line doesn\'t exist.')
elif instruction.startswith('wait '):
    split = instruction.find(' ') + 1
    wait_time = int(instruction[split:])
    sleep(wait_time)
    print('sleep')
elif instruction.startswith('repeat '):
    split = instruction.find(' ') + 1
    repeat_count = int(instruction[split:])
    return_address = program_counter
elif instruction == 'again':
    repeat_count -= 1
    if repeat_count != 0:
        program_counter = return_address
elif instruction == 'clear':
    display.clear()
elif instruction == 'stop':
    program_counter = len(program_list)
elif instruction.startswith('beep '):
    music.play(instruction[5])
elif instruction.startswith('random '):
    # random a 99 puts a random number between 0 and 99 in variable a
    var_rnd = instruction[7]
    var_index = ord(var_rnd) - 97
    rnd_range = int(instruction[9:])
    variables[var_index] = randint(0,rnd_range)
elif instruction.startswith('if '):
    var1 = ord(instruction[3]) - 97
    operator = instruction[4]
    var2 = ord(instruction[5]) - 97
    goto = int(instruction[12:])
    if goto  < 0 or goto > len(program_list):
        print('Goto error: line doesn\'t exist.')
    else:
        if operator == '=':
            if variables[var1] == variables[var2]:
                program_counter = goto - 1
        elif operator == '>':
            if variables[var1] > variables[var2]:
                program_counter = goto - 1
        elif operator == '<':
            if variables[var1] < variables[var2]:
                program_counter = goto - 1
        else:
            print('If error: comparsion operators must be =, < or >')
elif instruction.startswith('scroll '):
    split = instruction.find(' ') + 1
    display.scroll(instruction[split:])
elif instruction.startswith('input ') and len(instruction) == 7:
    var_input = instruction[6]
    var_index = ord(var_input) - 97
    if var_index >= 0 and var_index <= 26:
        input_text = input()
        try:
            variables[var_index] = int(input_text)
        except ValueError:
            variables[var_index] = input_text
    else:
        print('Variable names must be letter a-z.')
elif instruction.startswith('print '):
    if len(instruction) == 7 and instruction[6].isalpha():
```

```
                var_print = instruction[6]
                var_index = ord(var_print) - 97
                print(variables[var_index])
            else:
                split = instruction.find(' ') + 1
                print(instruction[split:])
        elif len(instruction) == 5 and instruction[3] in operators and instruction[1] == '='
and ord(instruction[0]) > 96 and ord(instruction[0]) < 123 and ord(instruction[2]) > 96 and
ord(instruction[2]) < 123 and ord(instruction[4]) > 96 and ord(instruction[4]) < 123:
            var1 = instruction[0]
            var2 = instruction[2]
            var3 = instruction[4]
            operator = instruction[3]
            if operator == '+':
                variables[ord(var1)-97] = variables[ord(var2)-97] + variables[ord(var3)-97]
            elif operator == '-':
                variables[ord(var1)-97] = variables[ord(var2)-97] - variables[ord(var3)-97]
            if operator == '/':
                variables[ord(var1)-97] = variables[ord(var2)-97] / variables[ord(var3)-97]
            if operator == '*':
                variables[ord(var1)-97] = variables[ord(var2)-97] * variables[ord(var3)-97]
        elif '=' in instruction:
            split = instruction.find('=') + 1
            var_name = instruction[:split-1]
            var_contents = instruction[split:]
            if len(var_name) == 1 and var_name.isalpha():
                if var_contents == 'light':
                    variables[ord(var_name)-97] = display.read_light_level()
                elif var_contents == 'temp':
                    variables[ord(var_name)-97] = temperature()
                elif var_contents == 'sound':
                    variables[ord(var_name)-97] = microphone.sound_level()
                elif var_contents == 'button a':
                    if button_a.was_pressed():
                        variables[ord(var_name)-97] = 1
                    else:
                        variables[ord(var_name)-97] = 0
                elif var_contents == 'button b':
                    if button_b.was_pressed():
                        variables[ord(var_name)-97] = 1
                    else:
                        variables[ord(var_name)-97] = 0
                else:
                    try:
                        variables[ord(var_name)-97] = int(var_contents)
                    except ValueError:
                        variables[ord(var_name)-97] = var_contents
            else:
                print('Variable names must be one letter a-z.')
        elif instruction == 'heart':
            display.show(Image.HEART)
        elif instruction == 'happy':
            display.show(Image.HAPPY)
        elif instruction == 'sad':
            display.show(Image.SAD)
        elif instruction == 'meh':
            display.show(Image.MEH)
        elif instruction == 'confused':
            display.show(Image.CONFUSED)
        elif instruction == 'angry':
            display.show(Image.ANGRY)
```

```
        elif instruction == 'asleep':
            display.show(Image.ASLEEP)
        elif instruction == 'yes':
            display.show(Image.YES)
        elif instruction == 'no':
            display.show(Image.NO)
        elif instruction == 'duck':
            display.show(Image.DUCK)
        elif instruction == 'small heart':
            display.show(Image.HEART_SMALL)
        elif instruction == 'pacman':
            display.show(Image.PACMAN)
        elif instruction == 'ghost':
            display.show(Image.GHOST)
        elif instruction == 'skull':
            display.show(Image.SKULL)
        elif instruction == 'rabbit':
            display.show(Image.RABBIT)
        elif instruction == 'diamond':
            display.show(Image.DIAMOND)
        elif instruction == 'small diamond':
            display.show(Image.DIAMOND_SMALL)
        elif instruction == 'star':
            display.show(Image('00300:'
                               '03630:'
                               '36963:'
                               '03630:'
                               '00300'))
        else:
            display.show('?')
    except:
        print('There\'s a mistake in your program.')
print('smolBASIC')
print('Type \'help\' for a list of commands.')

while True:
    new_line = input('> ')
    if new_line.startswith('del '):
        if ' ' in new_line:
            split = new_line.find(' ') + 1
            line_number = int(new_line[split:])
            if line_number > -1 and line_number < len(program_list):
                del program_list[line_number]
                print('deleted line ' + str(line_number))
            else:
                print('That line doesn\'t exist.')
        else:
            print('You need to tell me which line to delete.')
    elif new_line[:1].isdigit():
        split = new_line.find(' ') + 1
        line_number = int(new_line[:split])
        if line_number >= 0 and line_number < len(program_list):
            program_list[line_number] = new_line[split:]
        else:
            print('Line', line_number, 'doesn\'t exist.')
    elif new_line == 'list':
        for i in range(len(program_list)):
            print(i,program_list[i])
    elif new_line == 'new':
        program_list = []
    elif new_line == 'help':
```

```
        help()
    elif new_line == 'save':
        my_file = open('data', 'w')
        joined_list = ",".join(program_list)
        my_file.write(joined_list)
        my_file.close()
        print('Program saved.')
    elif new_line == 'load':
        try:
            with open('data') as f:
                joined_list = f.read()
                program_list = joined_list.split(',')
            for i in range(len(program_list)):
                print(i,program_list[i])
        except:
            print('No file to load.')
    elif new_line == 'fast':
        delay = 0
    elif new_line == 'slow':
        delay = 500
    elif new_line == 'run':
        program_counter = -1
        while program_counter < len(program_list):
            program_counter += 1
            if program_counter == len(program_list):
                break
            else:
                parse(program_list[program_counter])
                sleep(delay)
#                display.clear() <- used to clear display after showing image, now leaves it
as is
    else:
        if new_line != '':
            program_list.append(new_line)
```

Graphical version – smolBASIC-GFX.py

```
from microbit import *
import micropython
import music
from random import *
program_list = []
global program_counter
program_counter = -1
variables = [None]*26
operators = '+-/*'
delay = 500
global repeat_count
repeat_count = 0
global return_address
return_address = 0
colours = {'white':'15', 'black':'0', 'red':'9', 'yellow':'11', 'green':'10', 'blue':'12',
'magenta':'13', 'cyan':'14'}

uart.init(baudrate=9600, bits=8, parity=None, stop=1, tx=pin2, rx=pin1)
micropython.kbd_intr(-1) # disable accidental keyboard interrupt

def help():
    print('Type image names and then \'run\' to display them')
    print('\'scroll hello\' scrolls hello on LED display')
    print('\'beep A\' plays the note A')
    print('\'stop\' stops the program')
    print('\'input a\' prompts user input of a variable')
    print('\'a=23\' assigns 23 to variable a')
    print('\'random a 99\' assigns a random number between 1 and 99 to a')
    print('\'if a>b goto 3\' jumps to line 3 if a>b')
    print('You can also use < and = as comparisons')
    print('\'list\' to list your program')
    print('\'new\' to create a new program')
    print('\'save\' stores your program in non-volatile memory')
    print('\'load\' to load your program from non-volatile memory')
    print('\'del 3\' deletes line 3')
    print('Edit line 3 by typing \'3 happy\'')
    print('\'goto 0\' will go back to line 0 in a program')
    print('\'help\' to view this help')

def parse(instruction):
    global program_counter
    global repeat_count
    global return_address
    try:
        if instruction.startswith('goto '):
            split = instruction.find(' ') + 1
            line_number = int(instruction[split:])
            if line_number > -1 and line_number < len(program_list) + 1:
                program_counter = line_number - 1
            else:
                print('Error: goto line doesn\'t exist.')
        elif instruction.startswith('wait '):
            split = instruction.find(' ') + 1
            wait_time = int(instruction[split:])
            sleep(wait_time)
            print('sleep')
        elif instruction.startswith('repeat '):
            split = instruction.find(' ') + 1
```

```
        repeat_count = int(instruction[split:])
        return_address = program_counter
    elif instruction == 'again':
        repeat_count -= 1
        if repeat_count != 0:
            program_counter = return_address
    elif instruction == 'clear':
        display.clear()
    elif instruction == 'clear screen':
        uart.write('\x1B[2J')
    elif instruction == 'stop':
        program_counter = len(program_list)
    elif instruction.startswith('beep '):
        music.play(instruction[5])
    elif instruction.startswith('random '):
        # random a 99 puts a random number between 0 and 99 in variable a
        var_rnd = instruction[7]
        var_index = ord(var_rnd) - 97
        rnd_range = int(instruction[9:])
        variables[var_index] = randint(0,rnd_range)
    elif instruction.startswith('if '):
        var1 = ord(instruction[3]) - 97
        operator = instruction[4]
        var2 = ord(instruction[5]) - 97
        goto = int(instruction[12:])
        if goto  < 0 or goto > len(program_list):
            print('Error: goto line doesn\'t exist.')
        else:
            if operator == '=':
                if variables[var1] == variables[var2]:
                    program_counter = goto - 1
            elif operator == '>':
                if variables[var1] > variables[var2]:
                    program_counter = goto - 1
            elif operator == '<':
                if variables[var1] < variables[var2]:
                    program_counter = goto - 1
            else:
                print('Error: if statement comparsion operators must be =, < or >')
    elif instruction.startswith('scroll '):
        split = instruction.find(' ') + 1
        display.scroll(instruction[split:])
    elif instruction.startswith('input ') and len(instruction) == 7:
        var_input = instruction[6]
        var_index = ord(var_input) - 97
        if var_index >= 0 and var_index <= 26:
            input_text = ''
            new_char_string = ''
            while new_char_string != '\n':
                new_char_byte =  uart.readline()
                if new_char_byte:
                    new_char_string = str(new_char_byte, 'UTF-8')
                    input_text = input_text + new_char_string
                    uart.write(new_char_byte)
            input_text = input_text[:-1]
            try:
                variables[var_index] = int(input_text)
            except ValueError:
                variables[var_index] = input_text
        else:
            print('Variable names must be letter a-z.')
```

```
        elif instruction.startswith('print '):
            if len(instruction) == 7 and instruction[6].isalpha():
                var_print = instruction[6]
                var_index = ord(var_print) - 97
                print(variables[var_index])
            else:
                split = instruction.find(' ') + 1
                print(instruction[split:])
        elif instruction.startswith('ink '):
            split = instruction.find(' ') + 1
            if instruction[split:].isalpha():
                if len(instruction[split:]) > 1:
                    uart.write('\x1B[38;5;' + colours[instruction[split:]] + 'm')
                else:
                    uart.write('\x1B[38;5;' + str(variables[ord(instruction[split:])-97]) +
'm')
            else:
                uart.write('\x1B[38;5;' + instruction[split:] + 'm')
        elif instruction.startswith('paper '):
            split = instruction.find(' ') + 1
            uart.write('\x1B[48;5;' + colours[instruction[split:]] + 'm')
        elif instruction.startswith('circle '):
            split = instruction.find(' ') + 1
            xyr = instruction[split:]
            split = xyr.find(' ') + 1
            x = xyr[:split].strip(' ')
            yr = xyr[split:]
            split = yr.find(' ') + 1
            y = yr[:split].strip(' ')
            r = yr[split:]
            if x.isalpha():
                x = str(variables[ord(x)-97])
            if y.isalpha():
                y = str(variables[ord(y)-97])
            if r.isalpha():
                r = str(variables[ord(r)-97])
            uart.write('\x1B[#'+x+';'+y+';'+r+'c')
        elif instruction.startswith('rectangle '):
            split = instruction.find(' ') + 1
            xywh = instruction[split:]
            split = xywh.find(' ') + 1
            x = xywh[:split].strip(' ')
            ywh = xywh[split:]
            split = ywh.find(' ') + 1
            y = ywh[:split].strip(' ')
            wh = ywh[split:]
            split = wh.find(' ') + 1
            w = wh[:split].strip(' ')
            h = wh[split:]
            if x.isalpha():
                x = str(variables[ord(x)-97])
            if y.isalpha():
                y = str(variables[ord(y)-97])
            if w.isalpha():
                w = str(variables[ord(w)-97])
            if h.isalpha():
                h = str(variables[ord(h)-97])
            uart.write('\x1B[#'+x+';'+y+';'+w+';'+h+'r')
        elif len(instruction) == 5 and instruction[3] in operators and instruction[1] == '='
and ord(instruction[0]) > 96 and ord(instruction[0]) < 123 and ord(instruction[2]) > 96 and
ord(instruction[2]) < 123 and ord(instruction[4]) > 96 and ord(instruction[4]) < 123:
```

```
        var1 = instruction[0]
        var2 = instruction[2]
        var3 = instruction[4]
        operator = instruction[3]
        if operator == '+':
            variables[ord(var1)-97] = variables[ord(var2)-97] + variables[ord(var3)-97]
        elif operator == '-':
            variables[ord(var1)-97] = variables[ord(var2)-97] - variables[ord(var3)-97]
        if operator == '/':
            variables[ord(var1)-97] = variables[ord(var2)-97] / variables[ord(var3)-97]
        if operator == '*':
            variables[ord(var1)-97] = variables[ord(var2)-97] * variables[ord(var3)-97]
    elif '=' in instruction:
        split = instruction.find('=') + 1
        var_name = instruction[:split-1]
        var_contents = instruction[split:]
        if len(var_name) == 1 and var_name.isalpha():
            if var_contents == 'light':
                variables[ord(var_name)-97] = display.read_light_level()
            elif var_contents == 'temp':
                variables[ord(var_name)-97] = temperature()
            elif var_contents == 'sound':
                variables[ord(var_name)-97] = microphone.sound_level()
            elif var_contents == 'button a':
                if button_a.was_pressed():
                    variables[ord(var_name)-97] = 1
                else:
                    variables[ord(var_name)-97] = 0
            elif var_contents == 'button b':
                if button_b.was_pressed():
                    variables[ord(var_name)-97] = 1
                else:
                    variables[ord(var_name)-97] = 0
            else:
                try:
                    variables[ord(var_name)-97] = int(var_contents)
                except ValueError:
                    variables[ord(var_name)-97] = var_contents
        else:
            print('Variable names must be one letter a-z.')
    elif instruction == 'heart':
        display.show(Image.HEART)
    elif instruction == 'happy':
        display.show(Image.HAPPY)
    elif instruction == 'sad':
        display.show(Image.SAD)
    elif instruction == 'meh':
        display.show(Image.MEH)
    elif instruction == 'confused':
        display.show(Image.CONFUSED)
    elif instruction == 'angry':
        display.show(Image.ANGRY)
    elif instruction == 'asleep':
        display.show(Image.ASLEEP)
    elif instruction == 'yes':
        display.show(Image.YES)
    elif instruction == 'no':
        display.show(Image.NO)
    elif instruction == 'duck':
        display.show(Image.DUCK)
    elif instruction == 'small heart':
```

```
            display.show(Image.HEART_SMALL)
        elif instruction == 'pacman':
            display.show(Image.PACMAN)
        elif instruction == 'ghost':
            display.show(Image.GHOST)
        elif instruction == 'skull':
            display.show(Image.SKULL)
        elif instruction == 'rabbit':
            display.show(Image.RABBIT)
        elif instruction == 'diamond':
            display.show(Image.DIAMOND)
        elif instruction == 'small diamond':
            display.show(Image.DIAMOND_SMALL)
        elif instruction == 'star':
            display.show(Image('00300:'
                               '03630:'
                               '36963:'
                               '03630:'
                               '00300'))
        else:
            display.show('?')
    except Exception as e: print(e)
#    except:
#        uart.write('There\'s a mistake in your program.')

sleep(500)

# use print for text etc as it adds a newline. Use uart.write for sending control codes
music.play['c']
uart.write('\x1B[2J')       # clear screen
# draw micro:bit logo
uart.write('\x1B[#3;11;7;10A0;2;15;7;0;3;15;1;0;7;15;1;0;2;15;1;0;7;15;1;0;1;15;1;0;2;15;1;0;
3;15;1;0;2;15;1;0;1;15;1;0;7;15;1;0;2;15;1;0;7;15;1;0;3;15;7;0;2;')
uart.write('\x1B[#3;240;5d')
uart.write('\x1B[38;5;11m') # set foreground to yellow
uart.write('\x1B[48;5;12m') # set background to blue
print('BBC micro:bit computer system')
uart.write('\x1B[0m')       # set default colours
print('smolBASIC')

while True:
    uart.write('\n>')
    new_line = ''
    new_char_string = ''
    while new_char_string != '\n':
        new_char_byte =  uart.readline()
        if new_char_byte:
            new_char_string = str(new_char_byte, 'UTF-8')
            new_line = new_line + new_char_string
            uart.write(new_char_byte)
    new_line = new_line[:-1]
    new_line = new_line.rstrip()
    if new_line.startswith('del '):
        if ' ' in new_line:
            split = new_line.find(' ') + 1
            line_number = int(new_line[split:])
            if line_number > -1 and line_number < len(program_list):
                del program_list[line_number]
                print('deleted line',line_number)
            else:
                print('That line doesn\'t exist.')
```

```
    else:
        print('You need to tell me which line to delete.')
elif new_line[:1].isdigit():
    split = new_line.find(' ') + 1
    line_number = int(new_line[:split])
    if line_number >= 0 and line_number < len(program_list):
        program_list[line_number] = new_line[split:]
    else:
        print('Line',line_number,'doesn\'t exist.')
elif new_line == 'list':
    for i in range(len(program_list)):
        print(i,program_list[i])
elif new_line == 'new':
    program_list = []
elif new_line == 'help':
    help()
elif new_line == 'save':
    my_file = open('data', 'w')
    joined_list = ",".join(program_list)
    my_file.write(joined_list)
    my_file.close()
    print('Program saved.')
elif new_line == 'load':
    try:
        with open('data') as f:
            joined_list = f.read()
            program_list = joined_list.split(',')
    except:
        uart.write('No file to load.')
elif new_line == 'fast':
    delay = 0
elif new_line == 'slow':
    delay = 500
elif new_line == 'run':
    program_counter = -1
    while program_counter < len(program_list):
        program_counter += 1
        if program_counter == len(program_list):
            break
        else:
            parse(program_list[program_counter])
            sleep(delay)
else:
    if new_line != '':
        program_list.append(new_line)
```