LIVERPOOL JOHN MOORES UNIVERSITY

School of Computer Science and Mathematics

Further Programming with MicroPython and the BBC micro:bit

Congratulations on completing the ‘Getting Started’ section!

Firstly, the tasks set at the end of the last section can be answered:

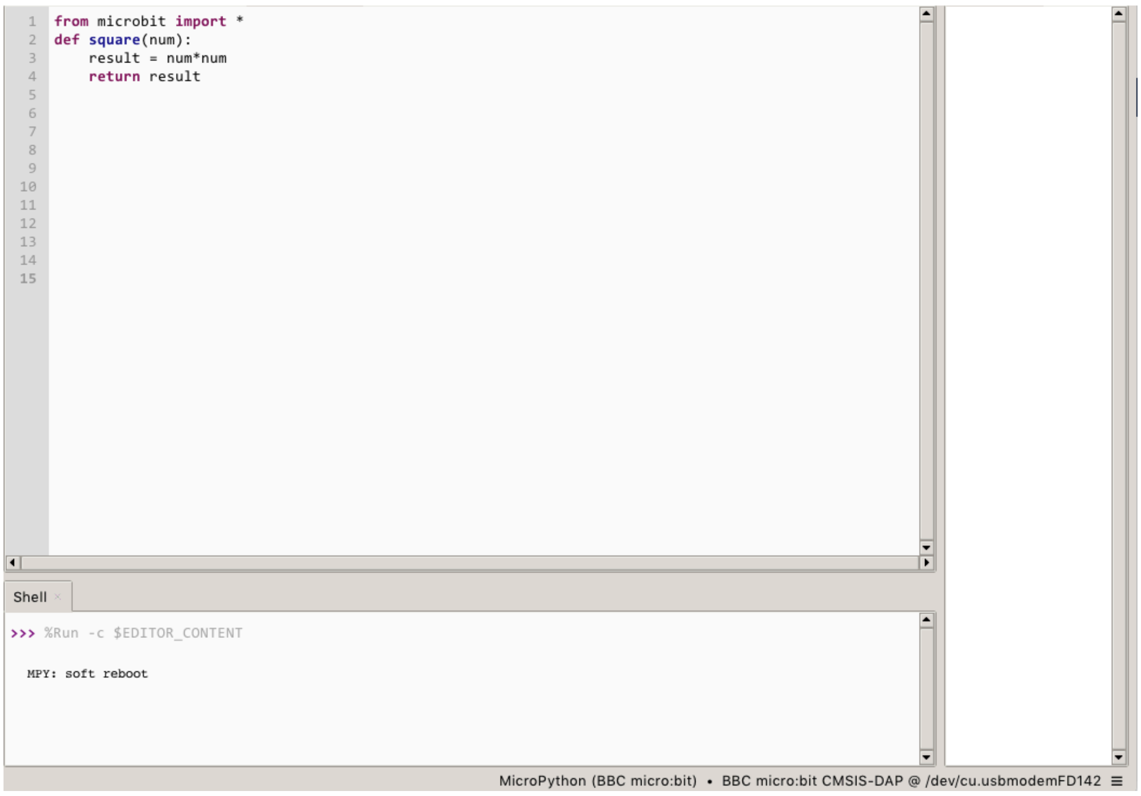
To have a message scroll every time you press button a:

```
from microbit import *
while True:
    if button_a.is_pressed():
        display.scroll("Hello  Welcome to LJMU Computer Science and Mathematics")

```

To store the scrolling message in a variable:

```
from microbit import*
m="Hello  Welcome to LJMU Computer Science and Mathematics"
display.scroll(m)
```

There now follows a look at how functions are used in (Micro)Python. Functions are the building blocks of reusable code in Python. They allow you to break down complex tasks into smaller, manageable chunks, making your code more organized and efficient. A function starts with a ‘def’ followed by the function name. The function body then uses the variable passed to it in the function to produce an output using any of the python features, ‘if’, ‘while’, etc. The code in Figure 1 shows the square function, where the input into the function is squared.



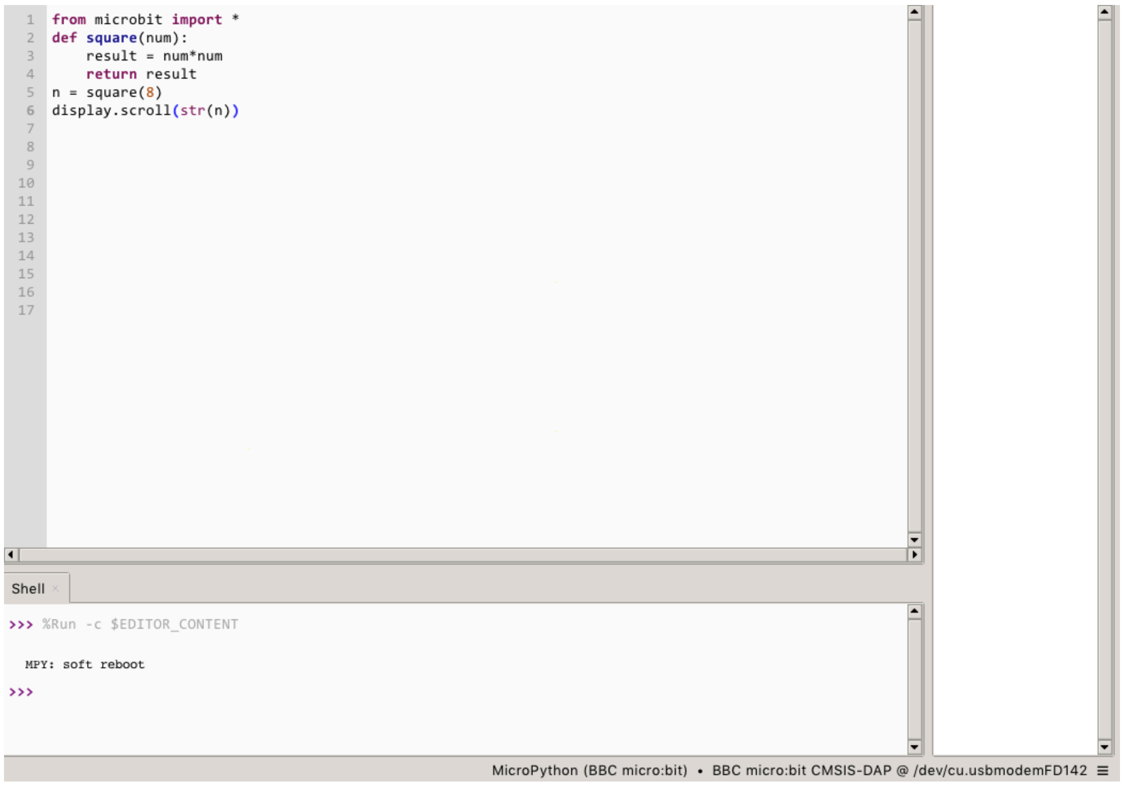
```
from microbit import *
def square(num):
    result = num*num
    return result
```

```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
```

MicroPython (BBC micro:bit) • BBC micro:bit CMSIS-DAP @ /dev/cu.usbmodemFD142

Figure 1: A Function Definition in Python

We can now call the function in the program. Then if you flash your micro:bit with this program (press green button with arrow at top of Thonny interface), you will see '64' (8 squared) scroll across the micro;bit.



```
from microbit import *
def square(num):
    result = num*num
    return result
n = square(8)
display.scroll(str(n))
```

```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
>>>
```

Shell
MicroPython (BBC micro:bit) • BBC micro:bit CMSIS-DAP @ /dev/cu.usbmodemFD142

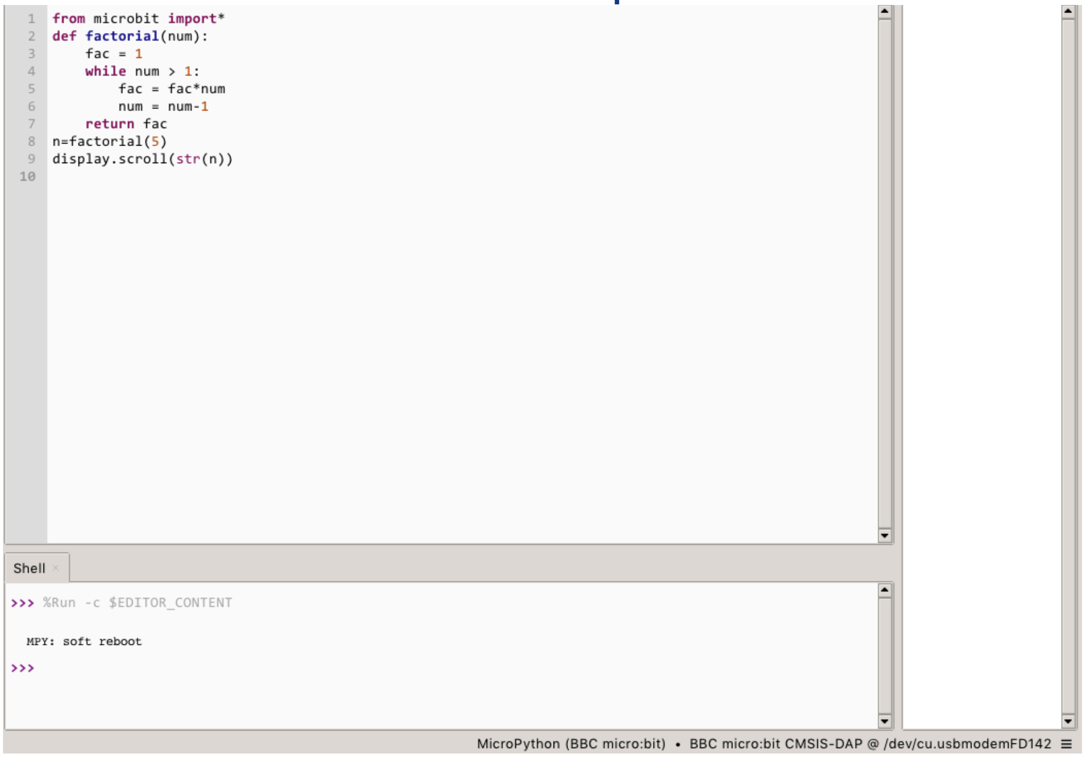
Figure 2: Calling the Function

You can change the number that the function calls and send that program to your micro:bit.

Next a slightly more complex function; factorial, sometimes denoted by the operator '!'.

The factorial of a number is that number multiplied by every whole number less than it down to 1. For example, the factorial of 5 (5!) is 5x4x3x2x1 = 120. The code in Figure 3 show a function for calculating factorials.

The value of the factorial (fac) is originally set to 1 we then multiply this by the number that was input into the function, then decrease this by 1 until that number is 1. In other words, factorial(num) is worked out by:
Setting a variable fac to 1 then multiplying fac by num, giving this value to fac, then decreasing num by 1 and multiplying this new value of fac by the new value of num, carrying on in this way until num is equal to 1. The while loop in the program accomplishes this. We finally write a couple of line to call the function to test it and see the result scroll across the micro:bit. You can of course choose any number to call the factorial function in your program.



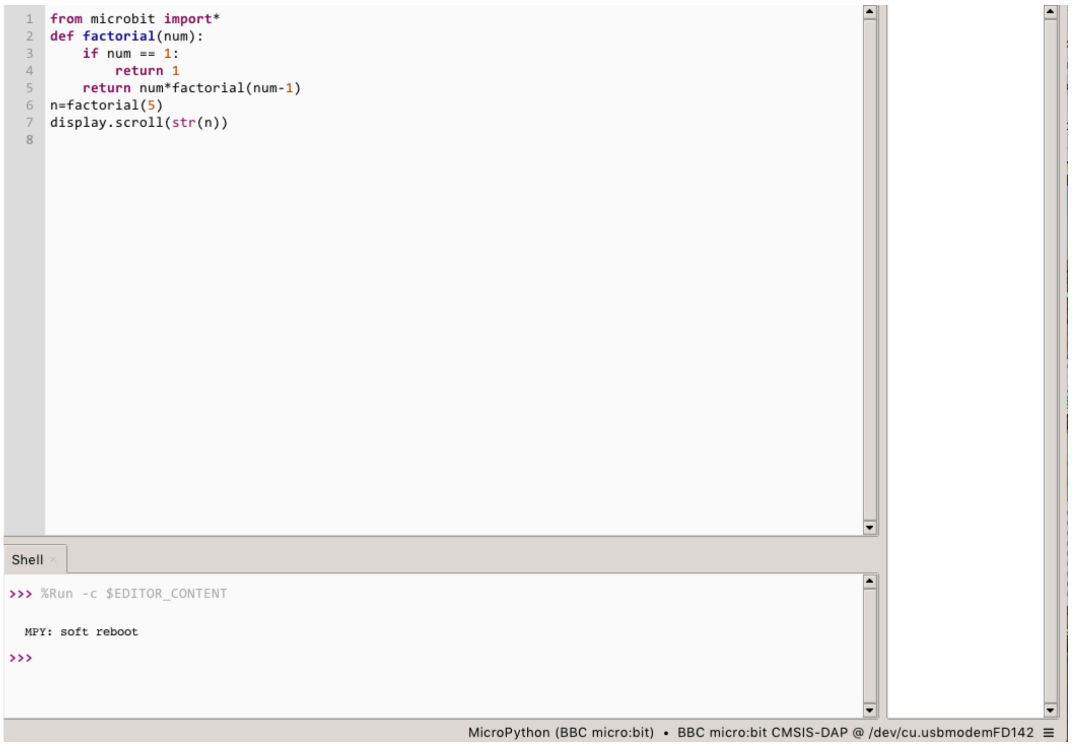
```
from microbit import*
def factorial(num):
    fac = 1
    while num > 1:
        fac = fac*num
        num = num-1
    return fac
n=factorial(5)
display.scroll(str(n))
```

```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
>>>
```

Shell
MicroPython (BBC micro:bit) • BBC micro:bit CMSIS-DAP @ /dev/cu.usbmodemFD142

Figure3: The Factorial Function in Python

A major issue in writing programs and coding solutions to problems is producing the most efficient code. In mathematics and computer science recursion is a powerful algorithmic technique. Basically, recursion is a function calling itself. Notice on our previously defined factorial function the factorial of a number n is equal to the n multiplied by the factorial of n – 1. For example, the factorial of 4 (= 4x3x2x1) is equal to 4 multiplied by the factorial of 3 (=3x2x1). We could use recursion to write a more efficient piece of code.



```
from microbit import*
def factorial(num):
    if num == 1:
        return 1
    return num*factorial(num-1)
n=factorial(5)
display.scroll(str(n))
```

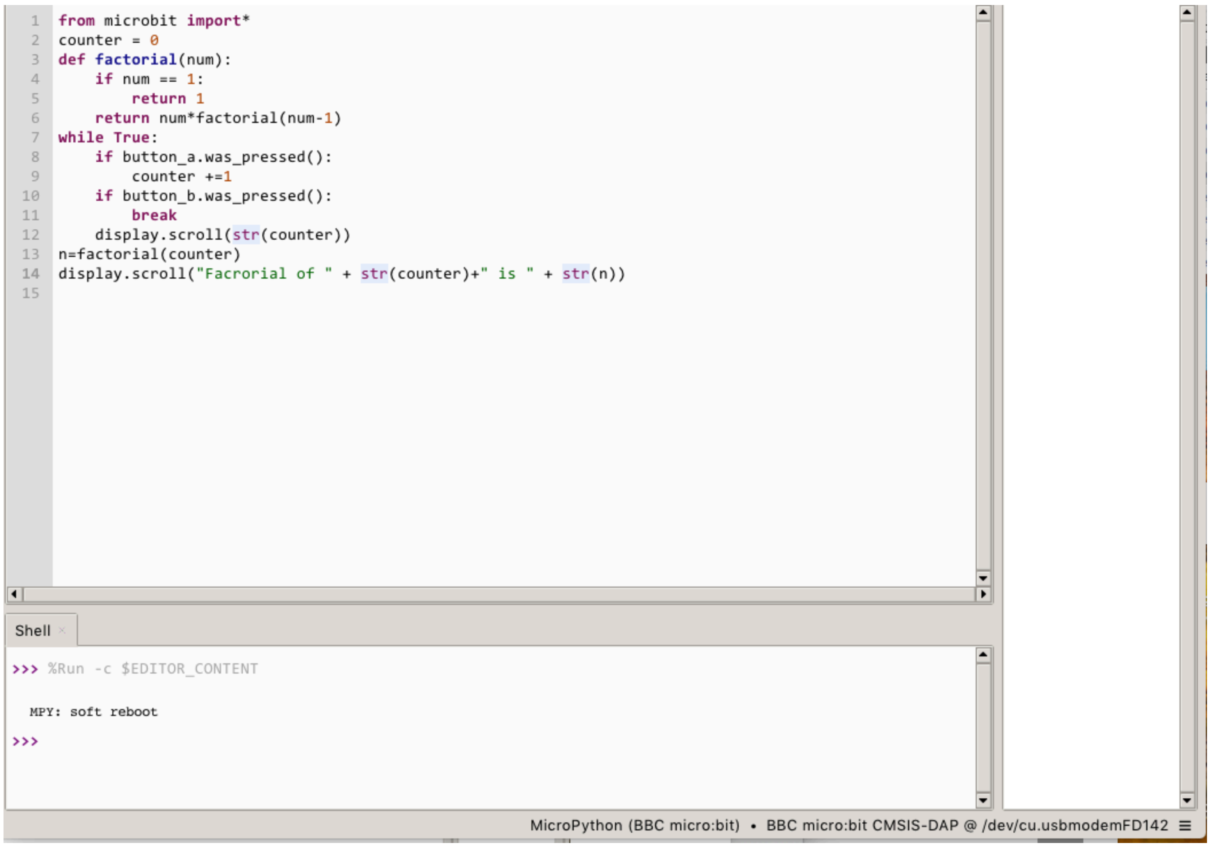
```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
>>>
```

Shell
MicroPython (BBC micro:bit) • BBC micro:bit CMSIS-DAP @ /dev/cu.usbmodemFD142

Figure 4: The Factorial Function in Python using Recursion

Notice for the code in Figure4, to check if two things are equal the '==' symbol is used to distinguish from '=' used as a variable assignment.

Finally in this part, the number passed to a function could be input using the micro:bit's buttons to make a more interactive experience.



```
from microbit import*
counter = 0
def factorial(num):
    if num == 1:
        return 1
    return num*factorial(num-1)
while True:
    if button_a.was_pressed():
        counter +=1
    if button_b.was_pressed():
        break
    display.scroll(str(counter))
n=factorial(counter)
display.scroll("Facrorial of " + str(counter)+" is " + str(n))
```

```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
>>>
```

Shell
MicroPython (BBC micro:bit) • BBC micro:bit CMSIS-DAP @ /dev/cu.usbmodemFD142

Figure 5: An Interactive Factorial Function in Python on a micro:bit