### Visit:<https://www.microbit.co.uk/app/> and then click on `Create Code` and `MicroPython`

### Hello, World!

The traditional way to start programming in a new language is to get your computer to say, “Hello, World!”. This is easy with MicroPython:

from microbit import \*

display.scroll("Hello, World!")

Each line does something special. The first line:

from microbit import \*

...tells MicroPython to get all the stuff it needs to work with the BBC micro:bit. All this stuff is in a module called microbit (a module is a library of pre-existing code). When you import something you’re telling MicroPython that you want to use it, and \* is Python’s way to say everything. So, from microbit import \* means, in English, “I want to be able to use everything from the microbit code library”.

The second line:

display.scroll("Hello, World!")

...tells MicroPython to use the display to scroll the string of characters “Hello, World!”.

### Image

MicroPython comes with lots of built-in pictures to show on the display. For example, to make the device appear happy you type:

from microbit import \*

display.show(Image.HAPPY)

I suspect you can remember what the first line does. The second line uses the display object to show a built-in image.

Here’s a list of the built-in images:

• Image.HEART

• Image.SMILE

• Image.SAD

• Image.RABBIT

• Image.COW

• Image.PACMAN

• Image.DUCK

• Image.TORTOISE

• Image.GHOST

• Image.SWORD

• Image.GIRAFFE

• Image.SNAKE

### DIY Images

Of course, you want to make your own image to display on the micro:bit, right?

That’s easy. Each LED pixel on the physical display can be set to one of ten values. If a pixel is set to 0 (zero) then it’s off. It literally has zero brightness. However, if it is set to 9 then it is at its brightest level. The values 1 to 8 represent the brightness levels between off (0) and full on (9).

Armed with this information, it’s possible to create a new image like this:

from microbit import \*

boat = Image("05050:"

"05050:"

"05050:"

"99999:"

"09990")

display.show(boat)

### Buttons

So far we have created code that makes the device do something. This is called output. However, we also need the device to react to things. Such things are called inputs. It’s easy to remember: output is what the device puts out to the world whereas input is what goes into the device for it to process.

The most obvious means of input on the micro:bit are its two buttons, labelled A and B. Somehow, we need MicroPython to react to button presses. This is remarkably simple:

from microbit import \*

sleep(10000)

display.scroll(str(button\_a.get\_presses()))

All this script does is sleep for ten thousand milliseconds (i.e. 10 seconds) and then scrolls the number of times you pressed button A. That’s it! While it’s a pretty useless script, it introduces a couple of interesting new ideas:

1. The sleep function will make the micro:bit sleep for a certain number of milliseconds.

2. There is an object called button\_a and it allows you to get the number of times it has been pressed with the `get\_presses` method.

Since get\_presses gives a numeric value and display.scroll only displays characters, we need to convert the numeric value into a string of characters. We do this with the str function (short for “string” ~ it converts things into strings of characters).

The third line is a bit like an onion. If the parenthesis are the onion skins then you’ll notice that display.scroll contains str that itself contains button\_a.get\_presses. Python attempts to work out the inner-most answer first before starting on the next layer out. This is called nesting.

Let’s pretend you’ve pressed the button 10 times. Here’s how Python works out what’s happening on the third line:

Python sees the complete line and gets the value of get\_presses:

display.scroll(str(button\_a.get\_presses()))

Now that Python knows how many button presses there have been, it converts the numeric value into a string of

characters:

display.scroll(str(10))

Finally, Python knows what to scroll across the display:

display.scroll("10")

While this might seem like a lot of work, MicroPython makes this happen extraordinarily fast.

### Handling an Event

If we want MicroPython to react to button press events we should put it into an infinite loop and check if the button `is\_pressed`. An infinite loop is easy:

while True:

# Do stuff

While checks if something is True to work out if it should run its block of code. Since True is

obviously True for all time, you get an infinite loop!.