

# Code Samples

Code samples are possibly *the* most important tool in your author's arsenal. However, the art of explaining code in books is a tricky one. Firstly, a few golden rules.

- Aim for a general max of about 20 lines.
- If you absolutely have to, go to 30 lines.
- If you write a page or more of code, your editor will ask for revisions.
- Break it down – the rule of thumb is to separate where you would normally comment.
- Never use in-code comments – they're a sign you need to write something outside the code sample about it!
- When you intend to describe a specific section in the sample *highlight* it.

## The sample

Let's look at a number of ways we could describe this fairly simple Python code sample. It's a very simple programme that scans the input and detects the largest number and smallest number in that input.

```
numArray = [1, 5, 3, 12, 14, 75, 37, 83]
largestNumber = None
smallestNumber = None

print ("\nOur numbers are: ", numArray)

for num in numArray:
    if smallestNumber is None and largestNumber is None:
        smallestNumber = num
        largestNumber = num
    elif num > largestNumber:
        largestNumber = num
    elif num < smallestNumber:
        smallestNumber = num
    print ("Smallest Number:", smallestNumber, "Largest Number:",
largestNumber, "Current Number:", num)

print ("The largest number is:", largestNumber)
print ("The smallest number is:", smallestNumber)
```

## Breaking it down

Just pasting this one in there, and explaining it in a paragraph underneath isn't going to work. A more advanced audience will probably get it, but the more beginner your audience, and the earlier in the book you are, the more you should break code samples down. You can always refer the reader to the sample available for download from Packt, or more preferably these days on a github repo. The example's a little contrived, as it's designed to explain the maximum amount of information about the lines, to a basic reader.

We're going to try breaking this one down significantly now. Everything from this point is a mock example of explaining code.

To start with, open up `sample.py`.

```
numArray = [1, 5, 3, 12, 14, 75, 37, 83]
largestNumber = None
smallestNumber = None
```

First, we declare our variables. `numArray` is our input. It houses each of the comma-separated **integers** (whole numbers) which our program will search for. Both `largestNumber` and `smallestNumber` hold the Python special type `None`. This special type allows you to set a null value for the variable. You'll see why we use this soon.

Let's start by displaying to the user what numbers Python will be processing.

```
print ("\nOur numbers are: ", numArray)
```

`Print()` sends the item in brackets to the user display inside Python. It is a standard Python **Built-In Function**. Items inside quotation marks are **strings**.

The `\n` is an **escape sequence**. You'll notice that `\n` is not printed when executing the programme. What you instead notice is that it in fact creates a line break in the output. See *Chapter X, Python Escape Sequences* for more information.

Now, we begin the main task of the program: processing the input. For this we will be using a **for loop**. A **for** loop repeatedly executes either until the condition at the beginning of the statement is complete.

```

for num in numArray:
    if smallestNumber is None and largestNumber is None:
        smallestNumber = num
        largestNumber = num
    elif num > largestNumber:
        largestNumber = num
    elif num < smallestNumber:
        smallestNumber = num
    print ("Smallest Number:", smallestNumber, "Largest Number:", \
largestNumber, "Current Number:", num)

```

The first line states `for num in numArray`. This quite simply means that Python will run through each of the integers in the array called `numArray`, and for each of them will perform the commands after the colon.

We then indent to perform the first step. This is an `if` statement. This asks Python to check whether or not the two variables `smallestNumber` and `largestNumber` contain the value `None`. If they do, then Python executes the steps indented following the colon. In this case, the variables are set to the value of the value of `num`.

This only runs the first time. After that, we use `elif` which allows us to check if the current integer in this iteration is larger or smaller than the one held in `largestNumber` or `smallestNumber` respectively. If this is the case, Python then tells the program to change the values of `largestNumber` or `smallestNumber` to the new largest or smallest integers.

The final section of our for loop tells Python to tell the user what the current iteration has found. Having completed this iteration, the `for` loop resets, and checks again until the original conditions have been met.

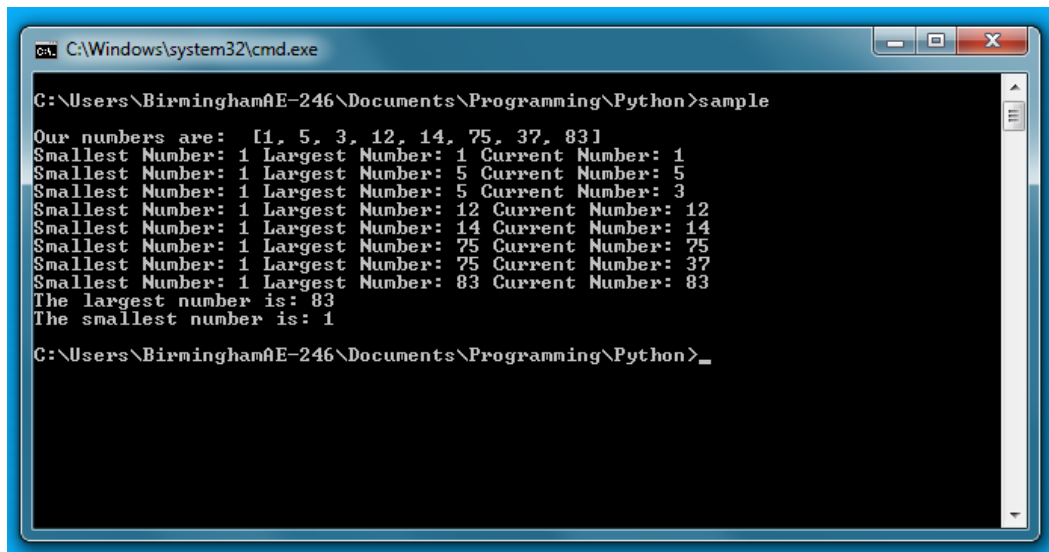
With the logic complete, and the answer found, Python displays the final output:

```

print ("The largest number is:", largestNumber)
print ("The smallest number is:", smallestNumber)

```

The complete user output is:



```
C:\Windows\system32\cmd.exe

G:\Users\BirminghamAE-246\Documents\Programming\Python>sample

Our numbers are: [1, 5, 3, 12, 14, 75, 37, 83]
Smallest Number: 1 Largest Number: 1 Current Number: 1
Smallest Number: 1 Largest Number: 5 Current Number: 5
Smallest Number: 1 Largest Number: 5 Current Number: 3
Smallest Number: 1 Largest Number: 12 Current Number: 12
Smallest Number: 1 Largest Number: 14 Current Number: 14
Smallest Number: 1 Largest Number: 75 Current Number: 75
Smallest Number: 1 Largest Number: 75 Current Number: 37
Smallest Number: 1 Largest Number: 83 Current Number: 83
The largest number is: 83
The smallest number is: 1

G:\Users\BirminghamAE-246\Documents\Programming\Python>_
```

0000OS\_01\_01.png

And that's the completed explanation.

## Explaining this sample

You'll notice this example uses a number of devices. Firstly, it heavily breaks down the code sample into its main logic blocks. It anticipates that the reader cannot look at the whole code sample and instantly understand it.

Key to explaining the code is linking up sections highlighted in the code which are of particular interest (using Code Highlighted [PACKT]) with those explained in the body text. Using Code in Text [PACKT] helps further link the samples and body text.

The most important thing is that you constantly evaluate each sample, and try to work out if you've explained the core concepts of that code sample before – or whether or not the target audience will understand it.