FOR Loops - Reading

We often want to repeat a loop a certain number of times. WHILE loops can do this – however, **FOR loops** are made for this task. FOR loops differ quite a lot in the syntax in different languages, but they all perform in a similar way to that shown below.

```
FOR i ← 1 TO 10
    OUTPUT i
ENDFOR i
```

The output from this loop will be: 1 2 3 4 5 6 7 8 9 10.

Normally we use descriptive names for variables. In FOR loops where we are just using a variable name for an **iterator**, we use 'i' and then 'j', 'k' if there are nested loops. For example:

```
FOR i ← 1 TO 12
    FOR j ← 1 TO 12
        OUTPUT i * j
    ENDFOR
ENDFOR
```

This loop will output every times table from 1×1 up to 12×12.

We can change what happens to the iterator variable (i) at the end of each **iteration** by using STEP.

```
FOR i ← 1 TO 10 STEP 2
    OUTPUT str(i)
ENDFOR
```

The output from this loop will be: 1 3 5 7 9
This is because i is now being increased by 2 each iteration. The next iteration would be 11, but we are only looking for values 1 TO 10 so this is not included.

The value of the STEP can also be negative. This allows the program to count down.

```
FOR i ← 10 TO 1 STEP -1
    OUTPUT i
ENDFOR
```

The output from this loop will be: 10 9 8 7 6 5 4 3 2 1

Wherever possible you should try to use variables rather than typing the numbers directly into a FOR loop. This will make your code more readable. Rewriting the loops for times tables would look like this:

```
maxTimesTable ← 12
numberOfRows ← 12
FOR i ← 1 TO maxTimesTable
    FOR j ← 1 TO numberOfRows
        OUTPUT i * j
    ENDFOR
ENDFOR
```