REPETITION WITH PYTHON

José M. Garrido
Department of Computer Science

May 2015

College of Computing and Software Engineering
Kennesaw State University

© 2015, J. M. Garrido
1 Repetition with the While Loop

The *while* loop consists of a conditional expression and block of statements. This construct evaluates the condition before the process block of statements is executed. If the condition is true, the statements in the block are executed. This repeats while the condition evaluates to true; when the condition evaluates to false, the loop terminates.

1.1 While-Loop Flowchart

A flowchart with the *while* loop structure is shown in Figure 1. The process block consists of a sequence of actions.

The actions in the *process* block are performed while the condition is true. After the actions in the process block are performed, the condition is again evaluated, and the actions are again performed if the condition is still true; otherwise, the loop terminates.

![Figure 1: A flowchart with a while-loop.](image)

The condition is tested first, and then the process block is performed. If this condition is initially false, the actions in the block are not performed.

The number of times that the loop is performed is normally a finite number. A well-defined loop will eventually terminate, unless it has been specified as a non-
terminating loop. The condition is also known as the loop condition, and it determines when the loop terminates. A non-terminating loop is defined in special cases and will repeat the actions forever.

1.2 While Structure in Pseudo-Code

The form of the while statement includes the condition, the actions in the process block written as statements, and the keywords while, do, and endwhile. The block of statements is placed after the do keyword and before the endwhile keyword. The following lines of pseudo-code show the general form of the while-loop statement that is shown in the flowchart of Figure 1.

```plaintext
while (condition) do
    (block of statements)
endwhile
```

The following example has a while statement and the block of statements is performed repeatedly while the condition $j \leq \text{MAX\_NUM}$ is true.

```plaintext
while $j \leq \text{MAX\_NUM}$ do
    set $\text{sum} = \text{sum} + 12.5$
    set $y = x \times 2.5$
    add 3 to $j$
endwhile
display "Value of sum: ", $\text{sum}$
display "Value of y: ", $y$
```

1.3 While Loop in the Python Language

The following lines of code show the general form of the while-loop statement in Python; it is similar to the pseudo-code statement and follows the loop definition shown in the flowchart of Figure 1.

```plaintext
while (condition):
    (block of statements)
```

The previous example has a while statement with a condition that checks the value of variable $j$. The block of statements that are repeated are always indented (four columns to the right); these statements are repeated while the condition $j \leq \text{MAX\_NUM}$ is true. The following lines of code show the Python implementation, which is stored in file test2.py. The while statement appears in line 6, the block of statements that are repeated are in lines 7 – 9. The print statement in line 10 is at the end and is outside the loop.
Repetition with Python

The following Linux shell command starts the Python interpreter with the script `test2.py`.

$ python test2.py
Value of sum: 75.0

1.4 Loop Counter

As mentioned previously, in the while-loop construct the condition is tested first and then the statements in the loop block are performed. If this condition is initially false, the statements are not performed.

The number of times that the loop is performed is normally a finite integer value. For this, the condition will eventually be evaluated to false, that is, the loop will terminate. This condition is often known as the loop condition, and it determines when the loop terminates. Only in some very special cases, the programmer can decide to write an infinite loop; this will repeat the statements in the repeat loop forever.

A counter variable stores the number of times (also known as iterations) that the loop executes. The counter variable is incremented every time the statements in the loop are performed. The variable must be initialized to a given value, typically to 0 or 1.

In the following pseudo-code listing, there is a counter variable with name `loop_counter` in the `while` statement. This counter variable is used to control the number of times the block statement is performed. The counter variable is initially set to 1, and is incremented every time through the loop.

```plaintext
Max_Num = 25  // maximum number of times to execute
set loop_counter = 1  // initial value of counter
while loop_counter < Max_Num do
  display "Value of counter: ", loop_counter
```
Repetition with Python

increment loop_counter
endwhile

The first time the statements in the block are performed, the loop counter variable \texttt{loop\_counter} starts with a value equal to 1. The second time through the loop, variable \texttt{loop\_counter} has a value equal to 2. The third time through the loop, it has a value of 3, and so on. Eventually, the counter variable will have a value equal to the value of \texttt{Max\_Num} and the loop terminates. The following listing is the Python code, which is stored in file \texttt{test3.py}.

\begin{verbatim}
# Script for testing a loop counter in a while loop
Max_Num = 15  # maximum number of times to execute
loop_counter = 1  # initial value of counter
while loop_counter < Max_Num :
    print "Value of counter: ", loop_counter
    loop_counter = loop_counter + 1
\end{verbatim}

The following Linux shell command starts the Python interpreter with the script \texttt{test3.py}.

\$ python test3.py
Value of counter: 1
Value of counter: 2
Value of counter: 3
Value of counter: 4
Value of counter: 5
Value of counter: 6
Value of counter: 7
Value of counter: 8
Value of counter: 9
Value of counter: 10
Value of counter: 11
Value of counter: 12
Value of counter: 13
Value of counter: 14
Value of counter: 15

1.5 Accumulator Variables

An \textit{accumulator} variable stores partial results of repeated calculations. The initial value of an accumulator variable is normally set to zero.

© 2016 J. M. Garrido
For example, the following algorithm in pseudo-code calculates the summation of numbers from input, includes an accumulator variable. The statement accumulates the values of \textit{cval} in variable \textit{total} and it is included in the while loop:

\begin{verbatim}
  total = 0.0
  while j < MAX_NUM
    set cval = j * 1.25
    add cval to total
    increment j
  endwhile
  display "Total accumulated: ", total
\end{verbatim}

After the \textit{endwhile} statement, the value of the accumulator variable \textit{total} is displayed. The following code is the corresponding Python code, which is stored in file \textit{test4.py}.

\begin{verbatim}
# Script for testing an accumulator variable in a while loop

total = 0.0
j = 1
MAX_NUM = 15
while j < Max_num :
    cval = j * 1.25
    total = cval + total
    j = j + 1
print "Total accumulated: ", total
\end{verbatim}

The following Linux shell command starts the Python interpreter with the script \textit{test4.py}.

```
$ python test4.py
Total accumulated:  131.25
```

In programming, each counter and accumulator variable serves a specific purpose and these variables should be well documented.

1.6 Summation of Input Numbers

The following simple problem applies the concepts and implementation of while loop and accumulator variable. The problem computes the summation of numeric values
Repetition with Python

inputed from the main input device. Computing the summation should proceed while the input values are greater than 1.

The pseudo-code that describes the algorithm uses an input variable, an accumulator variable, a loop counter variable, and a conditional expression that evaluates whether the input value is greater than zero.

```
set innumber = 1.5 // number with dummy initial value
set loop_counter = 0
set sum = 0.0 // initialize accumulator variable
display "Enter a number: "
read innumer // read first value
while innumber > 1.0 do
    add innumber to sum
    increment loop_counter
    display "Value of counter: ", loop_counter
display "Enter a number: 
read innumer
endwhile
display "Value of sum: ", sum
```

Listing 1 shows the Python program that implements the summation problem. The program is stored in file `summa.py`.

```
1 # Script for summation of input values a while loop
2 # Script: summa.py
3
4 loop_counter = 0
5 sum = 0.0 # initial value of accumulator variable
6 innumber = input( "Type number: ") # read first value
7 while innumber > 1.0 :
8    sum = sum + innumber
9    loop_counter = loop_counter + 1
10   print "Value of counter: ", loop_counter
11   innumber = input( "Enter a number: ")
```

The following output listing shows the shell commands that start the Python interpreter with file `summa.py`.

© 2016 J. M. Garrido
Repetition with Python

$ python summa1.py
Type number: 1.5
Value of counter: 1
Type number: 2.55
Value of counter: 2
Type number: 1.055
Value of counter: 3
Type number: 4.12
Value of counter: 4
Type number: 1.25
Value of counter: 5
Type number: 0.0
Value of sum: 10.475

2 Repeat-Until Loop

The repeat-until loop is a control flow structure that allows actions to be executed repeatedly based on a given condition. The actions within the process block are executed first, and then the condition is evaluated. If the condition is not true the actions within the process block are executed again. This repeats until the condition becomes true.

Repeat-until structures check the condition after the block is executed; this is an important difference from the while loop, which tests the condition before the actions within the block are executed. Figure 2 shows the flowchart for the repeat-until structure.

The pseudo-code statement of the repeat-until structure corresponds directly with the flowchart in Figure 2 and uses the keywords repeat, until, and endrepeat. The following lines of code shows the general form of the repeat-until statement.

\[
\begin{align*}
\text{repeat} \\
\quad \langle \text{statements in block} \rangle \\
\text{until} \langle \text{condition} \rangle \\
\text{endrepeat}
\end{align*}
\]

The following listing shows the pseudo-code of a repeat-until statement for the problem discussed in the previously section.

set innumber = 1.0 // dummy initial value
set l_counter = 0

© 2016 J. M. Garrido
Repetition with Python

Figure 2: A flowchart with a repeat-until structure.

```plaintext
set sum = 0.0 // accumulator variable
repeat
  add innumber to sum
  increment l_counter
  display "Value of counter: ", l_counter
  display "Type number: 
  read innumer
until innumber <= 0.0
endrepeat
display "Value of sum: ", sum
```

In Python, the repeat-until loop is not directly supported by a syntactic construct. However, it can be implemented with a while statement. Listing 2 shows the Python program that implements a problem with a loop counter. Note that a boolean variable (loop_cond) is used to reference the value of the loop condition. The while statement in line 8 always checks for the reversed condition using the not operator that precedes the boolean variable. The body of the loop includes the statements in lines 9 – 11 and the condition is evaluated at the end. The program is stored in file test5.py.

Listing 2 Python program with a loop counter.

```python
1 # Script: test5.py
2 # This script tests a loop counter in a repeat-until loop
```
Repetition with Python

# implemented with a while statement

Max_Num = 15  # maximum number of times to execute
loop_counter = 1  # initial value of counter
loop_cond = False
while not loop_cond :
    print "Value of counter: ", loop_counter
    loop_counter = loop_counter + 1
    loop_cond = loop_counter >= Max_Num # until true

The following output listing shows the shell commands that start the Python interpreter with file test5.py.

$ python test5.py
Value of counter:  0
Value of counter:  1
Value of counter:  2
Value of counter:  3
Value of counter:  4
Value of counter:  5
Value of counter:  6
Value of counter:  7
Value of counter:  8
Value of counter:  9
Value of counter: 10
Value of counter: 11
Value of counter: 12
Value of counter: 13
Value of counter: 14

Listing 3 shows the Python program that implements the summation problem. The program is stored in file summrep.py.

Listing 3  Python program that computes a summation.

# Script: summrep.py
# This script computes a summation using a repeat-until loop implemented with a while statement

sum = 0.0
loop_counter = 0
innumber = input( "Enter a number: ") # first number
lcond = innumber <= 0.0 # first number
while not lcond:
Repetition with Python

10    \texttt{sum} = \texttt{sum} + \texttt{innumber}
11    \texttt{loop\_counter} = \texttt{loop\_counter} + 1
12    \texttt{print} "Value of counter: ", \texttt{loop\_counter}
13    \texttt{innumber} = \texttt{input}( "Enter a number: ")
14    \texttt{lcond} = \texttt{innumber} <= 0.0
15
16    \texttt{print} "Value of sum: ", \texttt{sum}

The following output listing shows the shell commands that start the Python interpreter with file \texttt{summrep.py}.

\begin{verbatim}
$ python summrep.py
Enter a number: 4.7
Value of counter: 1
Enter a number: 7.88
Value of counter: 2
Enter a number: 0.8
Value of counter: 3
Enter a number: 2.145
Value of counter: 4
Enter a number: 0.0
Value of sum: 15.525
\end{verbatim}

3 For Loop Structure

The \texttt{for} loop structure explicitly uses a loop counter; the initial value and the final value of the loop counter are specified. The \texttt{for} loop is most useful when the number of times that the loop is carried out is known in advance. In pseudo-code, the \texttt{for} statement has the following general form:

\begin{verbatim}
for \langle \texttt{counter} \rangle = \langle \texttt{initial\_val} \rangle \text{ to } \langle \texttt{final\_val} \rangle 
do
\text{Block of statements}
\end{verbatim}

On every iteration, the loop counter is automatically incremented. The last time through the loop, the loop counter reaches its final value and the loop terminates. The \texttt{for} loop is similar to the \texttt{while} loop in that the condition is evaluated before carrying out the operations in the repeat loop.

The following listing in pseudo-code uses a \texttt{for} loop for the repetition part of the summation problem. Variable \texttt{j} is the counter variable, which is automatically
incremented and is used to control the number of times the statements in a block is to be performed.

```
for j = 1 to num do
    set sum = sum + 12.5
    set y = x * 2.5
endfor
display "Value of sum: ", sum
display "Value of y: ", y
```

In Python, the simple use of the `for` statement uses function `range`. The first, last, and the increment values of the loop counter are specified. The last value specified is not really included as one of the values of the loop counter. The increment is optional; if not included, its value is 1.

Listing 4 shows the Python program that includes a simple for-loop. The program is stored in file `test6.py`. The loop statement is in line 6. Note that values of $j$ are: 1, ..., 9 and the value of this loop counter is displayed in line 9 in every iteration of the loop.

Listing 4 Python program that includes a for-loop.

```
1 # Script: test6.py
2 # This script tests a for loop
3 x = 3.45
4 num = 10
5 sum = 0.0
6 for j in range(1, num) :
7    sum = sum + 12.5
8    y = x * 2.5
9    print "Loop counter: ", j
10
11 print "Value of sum: ", sum
12 print "Value of y: ", y
```

The following output listing shows the shell commands that start the Python interpreter with file `test6.py`.

```
$ python test6.py
Loop counter: 1
Loop counter: 2
Loop counter: 3
Loop counter: 4
```

© 2016 J. M. Garrido
Loop counter: 5
Loop counter: 6
Loop counter: 7
Loop counter: 8
Loop counter: 9
Value of sum: 112.5
Value of y: 8.625

3.1 Summation Problem with a For Loop

Using the for-loop construct of the repetition structure, the algorithm for the summation of input data can be defined in a relatively straightforward manner with pseudo-code. The most significant difference from the previous design is that the number of data inputs from the input device is included at the beginning of the algorithm. As in the previous case, the input value is added to variable sum only if the value entered is greater than zero.

```plaintext
set innumber = 1.0  // number with dummy initial value
set sum = 0.0       // initialize accumulator variable
display "Number of input data to read: ", MaxNum
for loop_counter = 1 to MaxNum do
    display "Type number: ", loop_counter
    read innumber
    if innumber > 0.0
        add innumber to sum
        display "Value of counter: ", loop_counter
    endif
endfor
display "Value of sum: ", sum
```

Listing 5 shows the Python source program that implements the summation problem with a for-loop. The program is stored in file summfor.c. Note that in line 5, the upper bound value specified for loop_counter is MaxNum+1.

Listing 5 Python program for computing the summation with for-loop.

```python
1 # Script: summfor.py
2 # This script computes a summation using a for loop
3 sum = 0.0  # initialize value of accumulator variable
4 MaxNum = input ("Enter number of input data to read: ")
5 for loop_counter in range(1, MaxNum+1):

© 2016 J. M. Garrido
Repetition with Python

```python
6       innumber = input ("Enter number: ")
7       if innumber > 0.0 :
8           sum = sum + innumber
9           print "Value of counter: ", loop_counter
10
11       print "Value of sum: ", sum
```

The following output listing shows the shell commands that start the Python interpreter with file `summfor.py`.

```
$ python summfor.py
Enter number of input data to read: 5
Enter number: 12.66
Value of counter: 1
Enter number: 2.432
Value of counter: 2
Enter number: 5.78
Value of counter: 3
Enter number: 23.85
Value of counter: 4
Enter number: 22.12
Value of counter: 5
Value of sum: 66.842
```

3.2 Factorial Problem

The factorial operation, denoted by the symbol $\!$, can be defined in a general and informal manner as follows:

$$ y! = y(y-1)(y-2)(y-3) \ldots 1 $$

For example, the factorial of 5 is:

$$ 5! = 5 \times 4 \times 3 \times 2 \times 1 $$

3.2.1 Mathematical Specification of Factorial

A mathematical specification of the factorial function is as follows, for $y \geq 0$:

$$ y! = \begin{cases} 
1 & \text{when } y = 0 \\
(y-1)! & \text{when } y > 0 
\end{cases} $$

The base case in this definition is the value of 1 for the function if the argument has value zero, that is $0! = 1$. The general (recursive) case is $y! = y(y-1)!$, if the value of the argument is greater than zero. This function is not defined for negative values of the argument.

© 2016 J. M. Garrido
3.2.2 Computing Factorial

In the following Python program, the factorial function \textit{mfact} and has one parameter: the value for which the factorial is to be computed. Listing 6 shows a Python program, \textit{factp.py} that includes function function \textit{mfact}. This function is called in line 22 to compute the factorial of a number and the result value is displayed on the console.

Listing 6 Python source program for computing factorial.

```python
1 #
2 # Program : factp.py
3 # Author : Jose M Garrido, May 28 2014.
4 # Description : Compute the factorial of a number.
5
6 def mfact(num):
7     
8     # This function computes the factorial of num >= 0
9     # it multiplies num * (num-1) * num-2 * ...1
10     
11     res = 1
12     if num > 0:
13         for num in range(num, 1, -1):
14             res = res * num
15         return res
16     elif num == 0:
17         return 1
18     else:
19         return -1
20
21 y = input("Enter a number to compute factorial: ")
22 fy = mfact(y)
23 print "Factorial is: ", fy
```

Note that this implementation returns \(-1\) for negative values of the argument. The following shell commands execute the Python interpreter with program \textit{factp.py} and computes the factorial for several values of the input number.

```
$ python factp.py
Enter a number to compute factorial: 5
Factorial is:  120

$ python factp.py
Enter a number to compute factorial: 0
```

© 2016 J. M. Garrido
Factorial is: 1

$ python factp.py
Enter a number to compute factorial: 1
Factorial is: 1