# Python Part III - Repeating Actions with Loops <br> Jean-Yves Sgro 

February 23, 2017

## Contents

1 Software Carpentry: Repeating Actions with Loops ..... 1
1.1 Overview: ..... 1
1.2 Key points summary ..... 1
2 Rationale ..... 2
3 Example 1: writing each letter of a word ..... 2
3.1 Explicit print() request ..... 2
3.2 for loop ..... 3
4 Function len ..... 5
5 Exercises ..... 5
5.1 From 1 to N with range() ..... 5
5.2 Computing Powers With Loops ..... 6
5.3 Reverse a String ..... 7
References and/or Footnotes ..... 8

```
## Warning: package 'knitr' was built under R version 3.5.2
```


## 1 Software Carpentry: Repeating Actions with Loops

This lesson "Repeating Actions with Loops" is lesson 02 from Software Carpentry ("Programming with Python" 2016).

### 1.1 Overview:

Questions

- How can I do the same operations on many different values?

Objectives

- Explain what a for loop does.
- Correctly write for loops to repeat simple calculations.
- Trace changes to a loop variable as the loop runs.
- Trace changes to other variables as they are updated by a for loop.


### 1.2 Key points summary

- Use for variable in collection to process the elements of a collection one at a time.
- The body of a for loop must be indented.
- Use len(thing) to determine the length of something that contains other values.


## 2 Rationale

In the last lesson, we wrote some code that plots some values of interest from our first inflammation dataset, and reveals some suspicious features in it, such as from inflammation-01.csv


Figure 1.
Analysis of inflammation-01.csv.
We have a dozen data sets right now in inflammation-*. csv files, and more on the way. We want to create plots for all of our data sets with a single statement.

To do that, we'll have to teach the computer how to repeat things i.e. use the same method on multiple files without specifying the commands for each file.

## 3 Example 1: writing each letter of a word

An example task that we might want to repeat is printing each character in a word on a line of its own.

```
word = 'lead'
```


### 3.1 Explicit print() request

We can access a character in a string using its index. For example, we can get the first character of the word 'lead', by using word[0]. One way to print each character is to use four print() statements:

```
print(word[0])
print(word[1])
print(word[2])
print(word[3])
```

1
e
a
d
This is a bad approach for two reasons:

- It doesn't scale: if we want to print the characters in a string that's hundreds of letters long, we'd be better off just typing them in.
- It's fragile: if we give it a longer string, it only prints part of the data, and if we give it a shorter one, it produces an error because we're asking for characters that don't exist.

```
word = 'tin'
print(word[0])
print(word[1])
print(word[2])
print(word[3])
```

IndexError Traceback (most recent call last)
<ipython-input-1-a51226538da7> in <module>()
3 print(word[1])
4 print (word[2])
----> 5 print(word[3])
IndexError: string index out of range

## 3.2 for loop

Here's a better approach: for every character in the defined word: print this character until all characters have been printed, which translates in Python language as such:

```
word = 'lead'
for char in word:
    print(char)
```

1
e
a
d

This is shorter-certainly shorter than something that prints every character in a hundred-letter string-and more robust as well:

```
word = 'oxygen'
for char in word:
    print(char)
```

- 

x
y
g
e
n

The improved version uses a for loop to repeat an operation-in this case, printing-once for each thing in a collection. The general form of a loop is:

```
for variable in collection:
    do things with variable
```

Using the oxygen example above, the loop might look like this:


Figure 2.
Oxygen word loop.
where each character (char) in the variable word is looped through and printed one character after another. The numbers in the diagram denote which loop cycle the character was printed in (1 being the first loop, and 6 being the final loop).

We can call the loop variable anything we like, but there must be a colon at the end of the line starting the loop, and we must indent anything we want to run inside the loop.
Unlike many other languages, there is no command to signify the end of the loop body (e.g. end for); what is indented after the for statement belongs to the loop.

Here's another loop that repeatedly updates a variable called length:

```
length = 0
for vowel in 'aeiou':
    length = length + 1
print('There are', length, 'vowels')
('There are', 5, 'vowels')
```

It's worth tracing the execution of this little program step by step:

- Since there are five characters in 'aeiou', the statement on line 3 will be executed five times.
- The first time around, length is zero (the value assigned to it on line 1) and vowel is 'a'.
- The statement adds 1 to the old value of length, producing 1, and updates length to refer to that new value.
- The next time around, vowel is 'e' and length is 1 , so length is updated to be 2 .
- After three more updates, length is 5 ;
- since there is nothing left in 'aeiou' for Python to process, the loop finishes and the print statement on line 4 tells us our final answer.

Note that a loop variable is just a variable that's being used to record progress in a loop. It still exists after the loop is over, and we can re-use variables previously defined as loop variables as well:

```
letter = 'z'
for letter in 'abc':
    print(letter)
print('after the loop, letter is', letter)
```

a
b
c
('after the loop, letter is', 'c')

Note also that finding the length of a string is such a common operation that Python actually has a built-in function to do it called len:

```
print(len('aeiou'))
```

5

## 4 Function len

len() returns the length (the number of items) of an object.
len() is much faster than any function we could write ourselves, and much easier to read than a two-line loop; it will also give us the length of many other things that we haven't met yet, so we should always use it when we can.

## 5 Exercises

### 5.1 From 1 to N with range()

Python has a built-in function called range that creates a sequence of numbers.
Range can accept 1-3 parameters:

- If one parameter is input, range creates an array of that length, starting at zero and incrementing by 1.
- If 2 parameters are input, range starts at the first and ends just before the second, incrementing by one.
- If range is passed 3 parameters, it starts at the first one, ends just before the second one, and increments by the third one.
- For example, range (3) produces the numbers 0, 1, 2, while range (2, 5) produces 2, 3, 4, and range (3, 10, 3) produces 3, 6, 9.

Using range(): write a loop that uses range() to print the first 3 natural numbers:

## 1

2
3
Note: There is a difference between Python 2 and Pyton 3 in the output printed by range() with Python 2 providing an output containing all elements:

## In Python 2:

```
range(3)
```

$[0,1,2]$

## In Python 3:

```
range(3)
range(0, 3)
```

However, just typing range() will not output a line by line print-out as requested in the exercise. For this a loop is needed. Remember here that the the range will start with the first parameter and end with the last parameter minus 1. Therefore the range has to go until 4 even though we want to go till 3:

```
for i in range(1, 4):
    print(i)
```

1
2
3

### 5.2 Computing Powers With Loops

Power is built in Python, for example: $5^{3}$ can be calculated with the exponentiation operator $* *$ :

```
print(5 ** 3)
```

125

As an exercise, write a loop that will calculate the same result as $5 * * 3$ but only using the multiplication operator $*$ without using exponentiation.

The solution provided may need clarification.

- First, a variable result is initiated with a given value that will change within the loop.
- Then a for loop is initiated in combination with a range of values that will specify how many iterations the loop will need to go through: we need 3 iterations.
- The provided range from 0 to 3 will provide values 0,1 , and 2 .
- However, all we need are 3 consecutive numbers that will be the loop iterations. Therefore we could use other ranges, for example these would work as well: range ( 1,4 ), range ( 2,5 ), range ( 100,103 )...

We can decompose the solution provided by adding commands to "check" the values of variables as the loop progress. This is an easy technique of "debugging:"

```
result = 1
print("initial result=", result)
for i in range(100, 103):
    print("current result =", result, "and is multiplied by 5:", result, "x 5")
    result = result * 5
    print("In the loop i = ", i)
    print("The new result is: result = ", result)
    print("")
print(result)
('initial result=', 1)
('current result =', 1, 'and is multiplied by 5:', 1, 'x 5')
('In the loop i = ', 100)
('The new result is: result = ', 5)
('current result =', 5, 'and is multiplied by 5:', 5, 'x 5')
('In the loop i = ', 101)
('The new result is: result = ', 25)
```

```
('current result =', 25, 'and is multiplied by 5:', 25, 'x 5')
('In the loop i = ', 102)
('The new result is: result = ', 125)
```


## 125

The solution provided only prints the final number without any remarks:

```
result = 1
for i in range(0, 3):
    result = result * 5
print(result)
```


## 125

### 5.3 Reverse a String

Write a loop that takes a string, and produces a new string with the characters in reverse order, so 'Newton' becomes 'notweN'.

The solution can be described in the following way:

- The first step in the solution is the creation of variables newstring and oldstring that can contain the provided word or phrase which are designated as string in Python.
- The second step is the creation of the string length_old that will contain the length or number of characters in the string we have to work on.
- The third step is a for loop that will start printing from the end using variables values.

The solution provided may need some clarification. We can use a "debugging" method to add code to the solution to understand the process by printing extra information along the for loop.
Note the mixing of informational text within quotes in the print () statements intermingled with the variable names so that their values are printed within the text.

Note: Just a reminder about ranges and length: The word Newton has 6 letters. Within the for loop we use the equivallent of range (6) which would represent numbers $0-5$. Therefore, in the loop indexing we need to subtract 1 to match the letter number wihtin the word. For example, the index for the 6 th letter would be index 5 and so on. Check the lines with -1 in the code.

```
newstring = ''
oldstring = 'Newton'
length_old = len(oldstring)
print("Initialize variables:")
print("`newstring` is initialized as an empty string:", newstring)
print("The string to work on is defined as `oldstring`:", oldstring)
print("The number of characters in `oldtring`", oldstring, "calculated with `len()` is:", length_old)
print("Note the -1 used to match the length of `oldstring`", oldstring, "with the range used:", range(l,
print("")
for char_index in range(length_old):
    print("Loop index `char_index`=", char_index)
    newstring = newstring + oldstring[length_old - char_index -1]
    print("length_old - char_index -1 =", length_old - char_index -1)
    print("`newstring` is now:", newstring)
    print("")
```

```
print("The final result is:")
print(newstring)
Initialize variables:
('`newstring` is initialized as an empty string:', '')
('The string to work on is defined as `oldstring`:', 'Newton')
('The number of characters in `oldtring`', 'Newton', 'calculated with `len()` is:', 6)
('Note the -1 used to match the length of 'oldstring`', 'Newton', 'with the range used:', [0, 1, 2, 3,
('Loop index `char_index`=', 0)
('length_old - char_index -1 =', 5)
('`newstring` is now:', 'n')
('Loop index `char_index`=', 1)
('length_old - char_index -1 =', 4)
('`newstring` is now:', 'no')
('Loop index `char_index`=', 2)
('length_old - char_index -1 =', 3)
('`newstring` is now:', 'not')
('Loop index `char_index`=', 3)
('length_old - char_index -1 =', 2)
('`newstring` is now:', 'notw')
('Loop index `char_index`=', 4)
('length_old - char_index -1 =', 1)
('`newstring` is now:', 'notwe')
('Loop index `char_index`=', 5)
('length_old - char_index -1 =', 0)
('`newstring` is now:', 'notweN')
The final result is:
notweN
```


## References and/or Footnotes

"Programming with Python." 2016. (Archived May 14, 2016 http://go.wisc.edu/db1733). http://swcarpentry. github.io/python-novice-inflammation.

