Lab 6 - Repetition

1 ME 30 Lab 5 - Repetition

ME 30 ReDev Team

Description and Summary: This lab introduces the programming concept of *repetition*, also called looping, where some operations in a section of code are repeated multiple times. This lab exploration will focus on the two main repetition constructs in Python:

1. While loops (which are ***condition***-controlled)
2. FOR loops (which are ***sequence***-controlled)

Use the ME 30 Lab Report JupyterLab notebook template to write your lab report. ***

1.1 While Loops

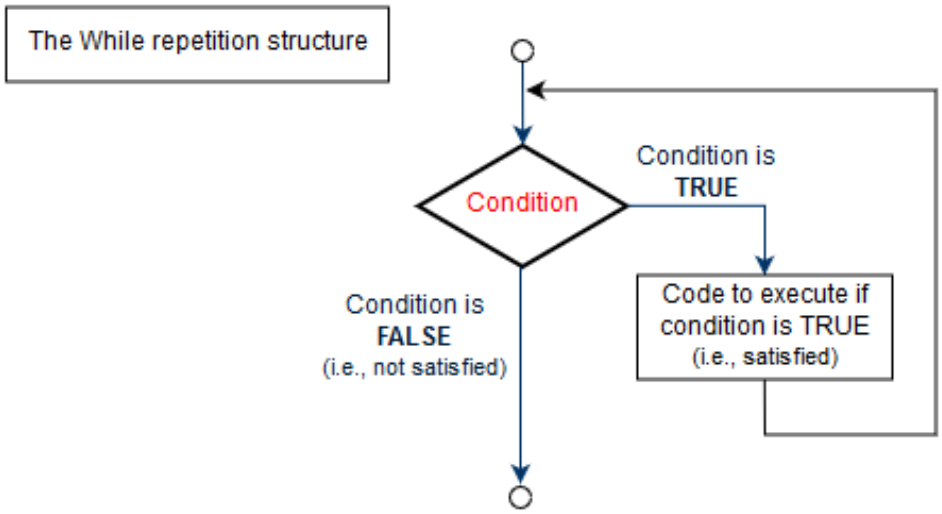
While loops have a conditional expression that is evaluated *prior* to the execution of the code to be repeated. If the condition is satisfied (True), the subsequent lines to be repeated are executed. After they are executed, the condition is checked again, and if satisfied, the block of code is executed again. If the condition is not satisfied (False), the code in the repetition block is skipped. While loops are especially useful when you want to repeat execution of code an unknown number of times, but where you can formulate a logical test for when the loop must end. The figure below graphically depicts how the while loop works:

The form of the while loop in Python code is shown below. [Note: the parentheses around the condition are actually NOT required! However, using parentheses probably will result in more readable code, and it is good practice, especially if you want to learn the C programming language (where they are required for a while loop.] Just like with functions and if/if-else/if-elif-else code blocks, the statements that the while condition controls are *indented* (customarily by 4 spaces):

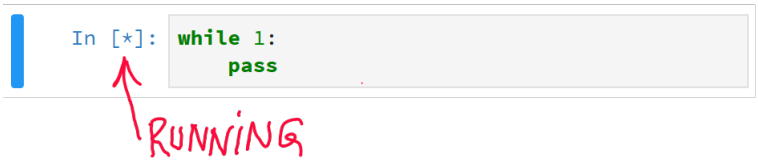
```
while (condition):
    statements to execute
```

In your lab report notebook, try this simple example in a Python code cell:

```
i = 1
while ( i <= 10 ):
    print( i )
    i = i + 1
```



While loop structure



Python running

```
print( "Done with the loop! " )
print( "i now has the value of", i )
print( "Can you see why?" )
```

The last line in the indented code block *increments* the value of i by 1 each time the code block is repeated. After that line executes, the new value of i is tested in the condition of being less than or equal to 10. If the value of i satisfies the condition (True), the indented code is executed again. If the condition is not satisfied (False), then control passes to the first non-indented print statement, and then continues to the last two print statements. A 'short-hand' for the the statement `i = i + 1` is:

```
i += 1
```

Try this statement in place of the earlier one, and see if you get the same result.

[Note: when you work with repetition structures, it is likely that at some point you will introduce a program 'bug', such that condition you are testing for never becomes false, and you end up being stuck in the loop. Python won't tell you that you made a mistake in this way, but you can get an indication that this might be happening in the JupyterLab notebook by looking just to the left of your code cell. While the code in the cell is running, you will see an asterisk appear inside the square braces:

If the asterisk never goes away, your code is still running. For simple programs that don't handle much data, the asterisk should disappear almost immediately. If it lingers for longer than

Restart the kernel

it should, you may be caught in an infinite loop. To break out of an infinite loop in the JupyterLab notebook, just restart the kernel:

If you are running Python code from a terminal window or within an IDE like Spyder, pressing CTL+C will usually stop execution of the code.

1.1.1 Breaking out of a repetition loop

Sometimes you might want to break out of the loop even if the condition controlling the loop is satisfied (True). For example, consider this snippet of code:

```
i = 1
while ( i <= 10 ):
    print( i )
    if ( i == 5 ):
        print( "i is 5; We're 'outta here!" )
        break
    i = i + 1
```

Try it out in a Python code cell in your lab report notebook. Report on what happens in a Markdown cell.

1.1.2 Skipping some code in a repetition loop without breaking out of the loop

Sometimes you might want to *skip* executing a portion of the code block being repeated when some condition occurs, but not completely terminate the loop. This can be done using the `continue` keyword. Consider the snippet of code below:

```
i = 1
while ( i <= 10 ):
    if ( i > 3 and i < 8 ):
        i = i + 1
        continue
    print( i )
    i = i + 1
```

Try this snippet in a Python code cell in your lab report notebook. Report on what happens in a markdown cell.

1.2 For Loops

For loops are the other way to repeat code in a controlled manner, and are especially useful for iterating over a sequence that you have defined or where you can specify the number of times the loop should operate. The form of a for loop is shown below:

```
for variable in some_sequence:
    statements
```

Consider the snippet of code below:

```
names = [ 'John', 'Roberto', 'Hua', 'Emma' ]
for name in names:
    print( name )
```

Try this snippet in a Python code cell in your lab report notebook. Report on what happens in a markdown cell.

The first line of the code snippet created a list of strings (first names). The second line essentially creates a variable, called 'name', that is then used in the for loop to index through the list of names and print them out. The indexing and printing loop repeats until the sequence has been exhausted.

The keywords **`break`** and **`continue`** work similarly in for loops as they do in while loops, and it is also possible to use an **`else`** alternative like was done with decision structures to be executed when the condition in the for loop is not satisfied.

1.3 Exercise 1 - Using repetition

Build on the last example and what was explained earlier, to write a short program that will print out all the names in the list ***except*** Roberto's.

1.3.1 Ranges

It is often useful to have a numeric index for many applications, especially in a for loop. This is such a common need that Python as a built-in function called

`range()`

that generates a *sequence* of numbers, and is extremely useful in for loops. For example, consider the code snippet below:

```
for i in range(10):
    print( i )
```

Copy this snippet into a Python code cell and run it. What number does the sequence generated by range(10) *start* with? What does it *end* with? Note your observations in a Markdown cell below your code. You will get more instruction about sequences later in the course, but check out the Python documentation" (section 4.6.6) to learn a bit more about it now. As the documentation explains, range() can alternatively take a start value, a stop value, and a step value.

```
*****
****
***
**
*
```

54321 asterisk pattern

1.4 Exercise 2 - Using a for loop to print a 'countdown'

In a new code cell, explore the full potential of the range() function by modifying the range() code you were just exploring, but make it so it will count *down* (i.e., print sequentially the values from 10 to 1), and then print "Blast off!" ***

1.4.1 Nesting for loops

Just like with decision structures, **for** loops can be 'nested', where the outermost loop will control the number of times that the inner loop is repeated. Nesting **for** loops is often done to index data structures such as matrices, where rows are accessed using one index (say the outer **for** loop), and the columns are accessed using another index (the inner **for** loop). Consider the following example:

```
rows = 5
for i in range( rows ):
    for j in range( i + 1 ):
        print('*', end='')
    print('')
```

Enter this snippet of code into a Python code cell and run it. Note what happens: 1. A variable named row is defined and given the value 5

2. The first **for** statement defines a loop for indexing the number of times the inner loop will be repeated.
3. The second **for** loop (indented) defines a loop for how many asterisks to print. Note in the print statement, `end=''` causes the printing of the asterisks to be done on the same line. Leaving off this optional argument will automatically insert a linefeed at the end of each print statement.
4. The last statement doesn't print anything, but since the termination character was not specified, it just adds a linefeed.

1.5 Exercise 3 - Using nested for loops - pt 1

In a new code cell, modify the code from the example above, but make it so it will print the asterisk pattern like this:

1.6 Exercise 4 - Using nested for loops - pt 2

In a new code cell, use nested for loops to print the asterisk pattern, so it looks like this:

12345 asterisk pattern

1.7 Exercise 5 - Approximate Pi

Using one of the Series Representations of Pi, write a program which will ask a user for the precision required and will then approximate Pi to the requested precision using one of the Series representations for Pi Wolfram Alpha approximation for Pi

What is the greatest precision which your program can generate?

---

2 What to Turn In

Ask your lab instructor what he or she wants you to submit to Canvas for this lab session.

---

3 References

+ Control Structures - Repetition from FSU: http://ww2.cs.fsu.edu/~nienaber/teaching/python/lectures/contro repetition.html + Essential Python For Loop Command Examples: https://www.t: https://stackoverflow.com/questions/493386/how-to-print-without-newline-or-spacehegeekstuff.com/2017/07/python-for-loop-examples/ + How to print without newline or space: https://stackoverflow.com/questions/493386/how-to-print-without-newline-or-space + Jupyter Notebook Users Manual (from Bryn Mawr College) https://jupyter.brynmawr.edu/services/public/dblank/Jupyter%20Notebook%20Users%20Manual.ipynb + Adam Pritchard's Markdown Cheatsheet https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet