Introduction to: Computers & Programming: Loops in Python

Adam Meyers
New York University
Outline

• What is a Loop?
• While Loops
• For Loops
• Examples
• Nested Loops
What is a Loop?

- Loops are control structures
  - A block of code repeats
  - The extent of the repetition is usually limited in some way

- Two kinds of Loops in Python
  - `while` loops
    - The evaluation of a boolean expression determines when the repetition stops
    - Changes in values of variables lead to different evaluations of the boolean expression on each repetition
    - When the expression is evaluated as `False`, the loop halts
    - If the expression can never evaluate as `False`, the loop is endless
  - `for` loops
    - The length of a sequence determines how many times the body executes
    - The loop uses one member of the sequence at a time, until there are no more members
An Endless Loop

• Example

    def endless_timer ():
        import time
        now = 0
        while (True):
            time.sleep(1)
            now = now + 1
        print(now)

• This loop will keep counting seconds until stopped with a Control-C
What is a *while* Loop?

• A while loop consists of:
  – The word *while*
  – A boolean expression (*True* on the last slide)
  – A colon :
  – The body: an indented block of instructions

• The body of the loop repeats
  – until the boolean expression is False

• The loop on the previous slide is endless
  – because boolean expression is never False.
  – Any program can be stopped using Control-C
What is a **while** Loop? 2

- A loop that iterates a limited number of times

  ```python
def seconds_stop_watch (total_seconds):
    import time
    now = 0
    while (now < total_seconds):
      time.sleep(1)
      now = now + 1
      print(now)
  ```

- If we call `seconds_stop_watch` with 5 as an argument
  - The variable `now` is initialized to 0
  - The loop iterates 5 times
  - Each time: a second passes, 1 is added to `now` and `now` is printed
  - In this way, 1 to 5 is printed over 5 seconds

- How many times would a loop beginning `while (False):` repeat?
A sample *for* loop

- This function simulates a 60 second timer

```python
def one_minute_timer():
    print(0)
    for second in range(60):
        time.sleep(1)
        print(second + 1)
```

- The function prints 0, then enters a *for* loop
  - The loop iterates through a list of numbers from 0 to 59
    - The variable `second` is assigned that number as a value
    - The system waits one second
    - The system prints `second + 1`
New Material Introduced in the `one_minute_timer` function

- The `range` function
  - `range` takes one or two integers \( m \) and \( n \) as an arguments
  - when \( m \) is left out, it is (by default) set to 0
  - creates a sequence of numbers from \( m \) to \( n \)

- A `for` loop
  - The first line – `for variable in sequence`:
    - `for` and `in` are keywords
    - `variable` can be any legal variable name
    - `sequence` is an ordered set of items
      - Python sequences includes data types like: `range, list, string, …`
  - The body of the loop repeats once for each item in the sequence
  - On each iteration, the variable is bound to the next item in the sequence
Looping Through a String

• Using a `for` loop
  
  ```python
def for_string_loop (string):
    for letter in string:
      print(letter)
  ```
  – `for-string-loop('Downward')`

• Using a `while` loop
  
  ```python
def while_string_loop (string):
    position = 0
    while(position < len(string)):
      print(string[position])
      position = 1 + position
  ```
Lengths and elements of Sequences

• The function `len` returns a sequence's length
  – The number of characters – `len('Downward')`
  – The number of integers in a range – `len(range(60))`
  – Etc.

• Elements in a range can be identified by their position, beginning with 0 and ending in one less than the length.
  – 'Downward'[0], range(5,10)[0]
  – 'Downward'[7], range(5,10)[4]
  – 'Downward'[8], range(5,10)[5] --- these are errors
**for loops vs. while loops**

- With some code modification, it is always possible to replace a *for* loop with a *while* loop, but not the other way around.
- *for* loops are used for situations where you know the number of iterations ahead of time
  - e.g., looping through sequences
- There is no significant efficiency difference
- The difference relates to ease in which humans can read/write code
Example: Drawing an asterisk triangle

• def draw_n_asterisks(n):
    for current_length in range(n):
        print('*', end='')
    print()

– print can take a named argument
  • End='' indicates what to print at the end of the string
  – the character in between the single quotes
  • In this case, nothing
  • The default is a newline character

• def asterisk_triangle(base_size):
    for current_length in range(base_size):
        draw_n_asterisks(current_length)
    print()
Drawing an asterisk triangle 2

• Nested Loops – a single function
  
  def asterisk_triangle2(base_size):
      for current_length in range(base_size):
          for n in range(current_length):
              print('*', end='')
          print()

• Python indicates depth of nesting via indentation
  – Suppose the last line was indented once
Printing a Multiplication table

• def multiplication_table (high_num):
  for num1 in range(1, 1+high_num):
    for num2 in range(1, 1+high_num):
      print(num1,'X',num2, '=' , num1*num2)

• How does this work?
Sample Problem for Class

• Write a function that:
  – Takes three arguments:
    • base_size
    • repetitions
    • hour_glass_or_diamond
  – This function makes a pattern of asterisks that repeats the number of times indicated by repetitions
  – Each cycle consists of two triangles, one the upside down version of each other, both of which have a base of size base_size
  – If hour_glass_or_diamond is in the 'hour glass' setting, the function draws an upside down triangle and then a right side up triangle
  – If hour_glass_or_diamond is in the 'diamond' setting, the function draws the right side up triangle first and the upside down one second