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
      - A sequence is an object that is made up of other objects (arranged in an order), e.g., a string is a sequence of characters: “duck” = “d”, “u”, “c”, “k”.
    - The loop uses one member of the sequence at a time, ending with the last one
An Endless Loop: keeping time

• Example

```python
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 *True* does not change in value
  – And *True* does not equal *False*.
  – Any program can be stopped using Control-C
A “normal” while Loop

- Normal loops iterate until some condition is True (endless loops are unusual)
  
def timer (total_seconds):
      import time
      now = 0
      while (now < total_seconds):
          time.sleep(1)
          now = now + 1
          print(now)

- If we call timer 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?
Loops for Guiding User Input

- `def get_yes_or_no_answer()`:  
  ```python
  ## User must respond “yes” or “no”
  output = 'initial_input' # initialize the variable output
  output = output.lower() # convert to lowercase
  while(not ((output == 'yes') or (output == 'no'))):
    if output != 'initial_input':
      print('Invalid Input!')
    output=(input("Please respond: 'yes' or 'no'"))
    output = output.lower() # convert to lowercase
  if output=='yes':
    return(True)
  elif output=='no':
    return(False)
  ```

- `def get_integer_from_user()`:  
  ```python
  ## User must respond with a single digit
  output = 'initial_input' # initialize variable output
  while (not (output in '0123456789')):
    if output != 'initial_input':
      print('Invalid Input!')
    output = input('choose an integer. ')
  output = input('choose an integer. ')
  return(output)
  ```
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`
The range function

- **range** takes three arguments:
  - **START**: An optional 1st argument (starts at START)
  - **MAXIMUM**: A required 2nd argument (ends before MAXIMUM)
  - **INCREMENT**: An optional 3rd argument

- When left out, START defaults to 0
- When left out, INCREMENT defaults to 1
- creates a sequence of numbers from **START** to **MAXIMUM-1** such that consecutive items in the sequence differ by **INCREMENT**

- Normal Examples:
  - `range(5) → [0,1,2,3,4]`
  - `range(2,5) → [2,3,4]`
  - `Range(0,10,2) → [0,2,4,6,8]`

- Examples with Negative Increments
  - **Numbers count down**
  - **Ends 1 before 2nd argument**
    - `range(10,2,-1) → [10,9,8,7,6,5,4,3]`
    - `range(10,2,-2) → [10,8,6,4]`
Looping Through a Range

- def factorial(number):
  
  total = 1  ## example of accumulator variable
  for num in range(1,number+1):
    total=total*num
  
  return(total)
Looping through String

• `def accum_spaces_and_add_stars(word):
  output = '* ' ## also an accumulator variable
  for letter in string:
    output = output+letter+' '
  output = output+'*'
  return(output)"
The **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
- Examples:
  - **for** character **in** 'multi-character':
    ```python
    print(character)
    ```
  - **for** number **in** range(5):
    ```python
    print(number)
    ```
Looping Through a String

• Using a \textit{for} loop

\begin{verbatim}
def for_string_loop (string):
    for letter in string:
        print(letter)
– for_string_loop('Downward')
\end{verbatim}

• Using a \textit{while} loop

\begin{verbatim}
def while_string_loop (string):
    position = 0 ## counter
    while(position < len(string)):
        print(string[position])
        position = 1 + position
\end{verbatim}
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))

• Elements in a sequence 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

• The number in square brackets refers positions around items in sequences
  – Example: 'cat' consists of 3 letters
    • starts at position 0 and ends at position 3
    •  c a t
      0 1 2 3
    • cat[2] → 't' --- because 't' starts at position 2
for loops vs. while loops

• With some code modification, it is always possible to replace a for loop with a while loop, but not always 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
Simple Examples of Nested Loops

- What do you expect to be returned from the following loop within a loop?
  - `def print_1_to_4_by_a_to_d():`
    - `for number in [1,2,3,4]:`
      - `for letter in 'abcd':`
        - `print(number,letter)`
  - **Answer on Etherpad**

- Function based on above example
  - `def print_two_item_combinations(seq1,seq2):`
    - `for item1 in seq1:`
      - `for item2 in seq2:`
        - `print(item1,item2)`

- Notice that embedded loops are indented under higher loops

- If there are Num1 items in the Upper loop sequence and Num2 items in the Lower loop sequence, the lower loop goes Num1 X Num2 times
  - e.g., `seq1 = 'cat' and seq2 = '123'`, then 3 X 3 = 9 items are printed out.
Example: 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 if high_num is 10?
  – when num1 is 1, there are 10 different values for num2 (1 to 10), there are 10 more when num1 is 2, and so on until num1 is 10 (high_num). So there are 10 X 10 combinations.
**break**: causes a loop to exit

• keep counting while num is less than 10
  ```python
def count_to_ten_with_while():
    num = 0
    while num < 10:
        num = num + 1
    print(num)
  ```

• Equivalently, break out of loop when num equals 10
  ```python
def count_to_ten_with_break():
    num = 0
    while True:
        if num >= 10:
            break
        num = num+1
    print(num)
  ```
**pass and continue**

- **pass** does nothing
  
  ```python
  def print_every_other_number(max):
      for num in range(1,max+1):
          if num%2==0:
              pass
          else:
              print(num)
  ```

- **continue** skips to next iteration of loop
  
  ```python
  def skip_every_fifth_number(max):
      for num in range(1,max+1):
          if num%5==0:
              continue
          print(num)
  ```
A Few More Details About “print”

- **print** takes 2 optional arguments:
  - `sep='*'` – prints asterisk between arguments
    - Default: prints spaces between arguments
  - `end='$'` – prints dollar sign at the end of line
    - Default – prints newline character at end of line

- **def test1():**
  ```python
  print('New','York','University',sep='*',end='$')
  print('Blah','Blah','Blah',sep='*',end='$')
  ```

- **def test2():**
  ```python
  print('New','York','University')
  print('Blah','Blah','Blah')
  ```
Example: Drawing an asterisk triangle

• **def draw_n_asterisks(n):**
  ```python
  for current_length in range(n):
      print('*', end='')
  ```
  Printing nothing (the empty string) at the end of each line, instead of the newline character

• **def asterisk_triangle(base_size):**
  ```python
  for current_length in range(1, base_size+1):
      draw_n_asterisks(current_length)
  print()
  ```
Drawing an asterisk triangle 2

• Nested Loops – a single function

```python
def asterisk_triangle2(base_size):
    for current_length in range(1, base_size + 1):
        for n in range(current_length):
            print('*', end='')
        print()
```

• Python indicates depth of nesting via indentation
  – Suppose the last line was indented once
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 indicates 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
• Contribute by either talking or typing possible code into Etherpad
Summary

• Loops provide a way to repeat blocks of instructions
• While loops are the most general
  – They require a condition for exiting the loop
    • If the condition is never false, the loop is endless
• For loops provide a simple way of repeating a block
  – once for each element in a sequence
  – or a fixed number of times
• A For loop can always be replaced by an equivalent While loop
• It is often useful to have nested loops (loops within loops)
Homework

• Go to HW part of website:
  – http://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/#Homework%20Schedule
  – See Homework due October 4
    • Modules/readings before class
    • Written assignment at midnight