ECS 32A - `for` Loops

Aaron Kaloti

UC Davis - Summer Session #1 2020

UC DAVIS
COMPUTER SCIENCE

Introducing `for` Loops

Two Categories of `while` Loops

1. Iterate a set number of times.
2. Iterate until a certain user input.

Where `for` Loops Come In

- `for` loops are a tool for iteration, just like `while` loops. `for` loops are often more convenient.
- **Everything doable with a `for` loop is doable with a `while` loop and vice-versa**.
  - `for` loops are excellent for #1 above and horrible for #2.

Syntatically, Two Kinds of `for` Loops

1. `for var in range(...)`: iterate over range of integers.
2. `for var in X` (where X is a list or string): iterate over elements/characters of `X`.

My Goal in Teaching `for` Loops

- I don't really care about you learning the specific reasons a `for` loop may be better than a `while` loop or vice-versa.
- I will show you how to use `for` loops (so prepare for many examples again), and honestly, you will naturally prefer `for` loops in the majority of cases.

Example: Print from 3 to 8[1]

Code

```
for i in range(3,9):
    print(i)
```

Execution

```
3
4
5
6
7
8
```

Example: Print from 3 to 8

Comparison with `while` Loop Version

`while` Loop Version

```
x = 3
while x <= 8:
    print(x)
    x += 1
```

`for` Loop Version

```
for i in range(3,9):
    print(i)
```

Example: Print from 7 to 24

Comparison with `while` Loop Version

`while` Loop Version

```
y = 7
while y <= 24:
    print(y)
    y += 1
```

`for` Loop Version

```
for i in range(7, 25):
    print(i)
```

Example: Print from 6 to User-Chosen Integer

Comparison with `while` Loop Version

`while` Loop Version

```
end_bound = int(input("Enter end bound: "))
counter = 6
while counter <= end_bound:
    print(counter)
    counter += 1
# Random ending message...
print("After while loop!")
```

`for` Loop Version

```
end_bound = int(input("Enter end bound: "))
for counter in range(6, end_bound + 1):
    print(counter)
# Random ending message...
print("After for loop!")
```

Example: Sum of Integers from 0 to N

Prompt

- Write a program that prompts the user for an integer N and prints the sum of all integers from 1 up to and including N, i.e. prints $\sum_{i=1}^{N} i$. If the user enters a negative integer, then an error message should be printed instead.

`while` Loop Solution

```
n = int(input("Enter N: "))
if n < 0:
    print("Error: n can't be negative.")
else:
    i = 1
    sum = 0
    while i <= n:
        sum += i
        i += 1
    print("The sum is {}.".format(sum))
```

`for` Loop Solution

```
n = int(input("Enter N: "))
if n < 0:
    print("Error: n can't be negative.")
else:
    sum = 0
    for i in range(1, n+1):
        sum += i
    print("The sum is {}.".format(sum))
```

`range()`

Parameters: `for i in range(a, b, c)`

- `range()` generates a "pattern" of integers starting at `a` and being incremented by `c` repeatedly until greater than `b`.
- On the first iteration, `i` will be `a`.
- On the last iteration, `i` will *not* be greater than or equal to `b`.
- Before the next iteration, `i` is *set to* the next value in the pattern.
- `i` doesn't need to exist before loop.
- `c` can be negative, in which case `a` must exceed `b`.
- No floating-point numbers.

Example

```
for y in range(2, 10, 3):
    print(y)
```

```
2
5
8
```

`range()`

Technical Aspects

- Changing the loop variable *within* the loop doesn't affect what it is set to before each iteration.

Example

```
for i in range(8,13):
    print("Before mid-loop change, i is", i)
    i = 2
    print("After mid-loop change, i is", i)
```

```
Before mid-loop change, i is 8
After mid-loop change, i is 2
Before mid-loop change, i is 9
After mid-loop change, i is 2
Before mid-loop change, i is 10
After mid-loop change, i is 2
Before mid-loop change, i is 11
After mid-loop change, i is 2
Before mid-loop change, i is 12
After mid-loop change, i is 2
```

Example: Print Even Numbers[1]

Prompt

- Write a program that asks the user for an integer N and prints all even numbers x such that $0 \leq x \leq N$, in ascending order.
  - For example, if the user enters $N = 7$, then the program should print 0, 2, 4, and 6.

Solution

```
N = int(input("Enter N: "))
for i in range(0, N + 1, 2):  # has to be N + 1, not N + 2
    print(i)
```

1. See slide #24 of the `while` loop slides for the `while` loop versions.

Example: Print Odd Numbers in Reverse[1]

Prompt

- Write a program that asks the user for an integer N and prints all ***odd*** numbers x such that $0 \leq x \leq N$ in ***descending*** order instead of ascending order.
  - For example, if the user enters $N = 13$, then the program should print 13, 11, 9, 7, 5, 3, and 1.

Solution #1

```
n = int(input("Enter N: "))
for i in range(n, 0, -1):  # n=5 -> 5,4,3,2,1
    if i % 2 == 1:
        print(i)
```

Solution #2

```
n = int(input("Enter N: "))
if n % 2 == 0:  # if n is even
    n -= 1
for i in range(n, 0, -2):
    print(i)
```

1. See slide #26 of the `while` loop slides for the `while` loop version.

Example: Product of Range[1]

Prompt

- Write a program that prompts the user for two integers M and N and prints the product of all integers between M and N (exclusive). In other words, the program should print $\prod_{i=M+1}^{N-1} i$.

Solution

```
M = int(input("Enter M: "))
N = int(input("Enter N: "))
product = 1
for i in range(M + 1, N):
    product *= i
print("The product is", product)
```

1. See slide #28 of the `while` loop slides for the `while` loop version.

`break` and `continue`

- These do work in `for` loops, but are rare.

Harder `while` Loop Examples

- `for` loops struggle when the condition for ending depends on user input.

Ones I Didn't Mention (from `while` loop slides)[1]

- Example from slide #16: Write a program that asks the user for three integers -- a, b, and c -- and keeps prompting the user for these three integers until the sum of the first two integers equals the product of the last two integers.
- Example from slide #34: Write a program that keeps prompting the user to enter an integer until they enter 0. The program should then tell the user the sum of every *other* integer that the user has entered. For example, if the user enters 7, -3, 4, 2, 8, and -1, then the program should output the sum $7 + 4 + 8 = 19$.

1. There were others I didn't mention (e.g. the example on slide #30 of the `while` loop slides) that are still easily convertible to `for` loops.

String Traversal

- A `for` loop can also be used to iterate over the characters of a string.

```
for ch in "abcd":
    print(ch)
```

```
a
b
c
d
```

- `ch` is set to each of the characters in `"abcd"`, *one at a time*.
- In this case, we no longer need a `range()` statement. However, we still need the words `for` and `in`.

String Traversal: Long String

- Note that any string can be traversed, regardless of length or number of "words".[1]

```
for x in "My name is Aaron":
    print(x)
```

```
M
y

n
a
m
e

i
s

A
a
r
o
n
```

1. A program has no idea what a "word" is, by the way.

Reassignment Within a `for` Loop

- As with the `range()` statement, `ch` *takes on* each of the values, and the old value of `ch` is overwritten before the start of each iteration. Moreover, the string that was traversed *is not changed*.
- What does the following program print?

```
chars = "wxyz"
for ch in chars:
    print("ch:", ch)
    ch = "a"
print(chars)
```

```
ch: w
ch: x
ch: y
ch: z
wxyz
```

- Notice that "a" is nowhere to be found.

Reassignment Within a `for` Loop

- Note, however, that changing `ch` *within* the loop does affect `ch` for the remainder of the iteration.

```
chars = "wxyz"
for ch in chars:
    print("ch:", ch)
    ch = "a"
    print("new ch:", ch)
print(chars)
```

```
ch: w
new ch: a
ch: x
new ch: a
ch: y
new ch: a
ch: z
new ch: a
wxyz
```

- Notes:
  - The original string (`chars`) is unchanged.
  - `ch` is still successfully *overwritten* by the next character when the next iteration starts.

Example: Count Letter 'a'[1]

Prompt

- Write a program that asks the user for a string and prints the number of occurrences of the lowercase "a" letter.
  - For example, if the user enters "banana", the program should print 3.

Solution

```
s = input("Enter: ")
counter = 0
for c in s:
    if c == "a":
        counter += 1
print("The letter \"a\" occurs {} times in {}.".format(
      counter, s))
```

1. We have not yet learned to do this with a `while` loop, but you will learn how to once we talk exclusively about strings.

Example: Counting Capital Letters

Prompt

- Write a program that asks the user for a string and prints the number of capital letters contained in that string.
  - For example:
    - if the user enters "HI AARON", the program should print 7.
    - if the user enters "aBcDe", the program should print 2.
    - if the user enters "", the program should print 0.

Solution

```
s = input("Enter: ")
counter = 0
for c in s:
    if c.isupper():
        counter += 1
print("There are {} uppercase letters in {}.".format(
      counter, s))
```

Exercise: Filtering Out Uppercase Letters

Prompt

- Write a program that asks the user for a string and then prints out that string with the uppercase letters removed. You may assume that the user will only enter strings that only contain letters. Do not use any string methods (e.g. `isupper()`), or you will make me unhappy.
- Create two approaches: one that involves string concatenation, and one that does not.

Examples

```
Input: abcDe
Output: abce
```

```
Input: ABCdE
Output: d
```

```
Input: GHJ
Output:
```

Exercise: Filtering Out Uppercase Letters

Solution #1

```
s = input("Enter: ")
new_s = ""
for c in s:
    # if c is NOT uppercase, then append c to new_s.
    # if not c.isupper():
    if "a" <= c <= "z":
        new_s += c
print("Output: {}".format(new_s))
```

Solution #2

```
s = input("Enter: ")
print("Output: ", end="")
for c in s:
    # if c is NOT uppercase:
    # if not c.isupper():
    if "a" <= c <= "z":
        print(c, end="")
print()
```