CS 111: Program Design I

Lecture 10: while concluded, list basics, Midterm

Robert H. Sloan & Richard Warner
University of Illinois at Chicago
October 1, 2019

UIC

ITERATION, ITERATION, ITERATION. …

How many times will this loop print

```
x = "Register now"
while len(x) > 9:
    print(x)
    x = x + "!"
```

A. 12
B. 9
C. 3
D. 4
E. It will print forever

What value is stored in variable z when code finishes executing?

```
x = 1
y = 2
z = 0
while x <= 3:
    z = z + y
    x = x + 1
```

A. 2

B. 3

C. 4

D. 6

E. 8

What value is stored in variable z when code finishes executing?

```
x = 1
z = 1
while x <= 3:
    z = z + x
    x = x + 1
```

A. 2

B. 3

C. 4

D. 7

E. 9

What value is stored in variable z when code finishes executing?

```
x = 5
y = 1
z = 3
while x > 0:
    z = z + y
    x = x - 1
```

A. 2

B. 3

C. 4

D. 8

E. 9

What value is stored in variable z when code finishes executing?

```
x = 0
z = 0
while x < 3:
    x = x + 1
    z = z + 2
```

A. 2

B. 3

C. 4

D. 6

E. 8

RELATING TO MIDTERM

functions and the midterm

- Writing small Python functions: master skill for CS 111 Law
- docstring requirements
  - Exactly 1
  - In quotes, by convention """inside 3 double"""
  - Must appear immediately after def line
  - Is distinct from comments
  - Universally considered good style in Python; required in current lab

functions in our class and most places

- Very rarely or never use the input() function
  - Exception: Zybooks is fond of style that makes heavy use of Python
- Usually but not always have formal parameters / (input) arguments / inputs
  - Number of them and their intended meaning should be part of problem specification on test; specification or your design on lab or project

Python types

- Basic understanding of Python types also a master skill for CS 111 Law

Python types

- Basic understanding of ~~Python~~ types ~~also~~ a master skill for CS ~~111 Law~~
- Can check type of expression at console with built-in function type()
  - E.g., type(3 % 17) → int

Midterm: Hold Harmless if #2 ok

```
if midterm2 > midterm1 and midterm2 >= 70:
      midterm1 = midterm2
```

MIDTERM: THE CODING QUESTIONS

PYTHON BASICS CONCLUDED : LISTS, RANGE

Lists: First peek

```
>>> evens = [2, 4, 6, 8, 10, 12]

>>> justices = ['Marshall', 'Brandeis', 'Brennan']

>>> mixed_grill = ['beef', 'shrimp']

>>> really_mixed = [2, 'beef', 3.5, 'shrimp']

>>> empty = []

>>> ls = [1, 'Brennan', ['Joe', 'Donald'] ]
```

Literal notation

- *Literal notation* means how we can write down specific object in our code, without having to calculate it
- 1, 2.0, 'three', and False are literal notations for an int, float, string, and bool, respectively

List literal notation is [ ]

So:

```
In [1]: evens = [2, 4, 6, 8, 10, 12]
```

And:

```
In [2]: x = 'Please'
In [3]: y = 'vote'
In [4]: ls = [x, y]
In [5]: ls
Out[5]: ['Please', 'vote']
```

RANGE

for loops are more general

```
for loop_variable in something_okay:
    <body>
    # in which loop_variable is each
    # element of something_okay in
    # turn
    # Not just for string characters!
        # List Elements
        # range of numbers!
```

range function

- range(start, stop) gives *something for loop can walk over*, going from integer start *up to but not including* integer stop (like slices!). E.g.:

```
for i in range(0, 25):
    print(i)

# prints out 0, 1, 2, …, 24
```

Getting a list out of range

- Technically *type* of output of range is range, a kind of *iterator* (take CS 341 for more…)
- If you want the list, use list(range(start, stop))
- E.g.,

```
>>> list(range(0, 25))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24]
```

types and type conversion

- Note that the name of a type is an operator to convert items to that type
- list(range(1,4)) → [1, 2, 3]
- list() must be given a very special kind of object to give back a list: an iterable. Output of range() is only one we'll see. But
- int(17.17) → 17
- str(17.17) → '17.17'

And can get type with type()

- type('cat') → str
- type([1, 2, 3]) → list
- Note:
  - type('cat') == str True
  - type('cat') == 'str' False!
  - str, list, int, float are Python keywords

Back home on the Range: more examples

```
>>> list(range(0, 25))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]

list(range(25))       # Gives same thing
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]

list(range(3, 9))
[3, 4, 5, 6, 7, 8]
```

range arguments

- 1 argument: start with 0
- 3 arguments: 3rd is skip. E.g.,

```
>>> list(range(3,15,3))
[3, 6, 9, 12]
```

Why range?

- Useful for looping over (coming)
  - Working with numbers
  - Working with strings, but want to know index not just character. I.e.,
    - `for i in len(str):`
- Lists for free

FOR (CONT.): THE FINAL PYTHON BASIC CONSTRUCT

iteration: More for

- Control can flow 3 ways
  1. sequentially
  2. conditionally (if elif else)
  3. repeatedly: iteration
     - iteration either while or for
     - for typically easier when appropriate
     - Many things for can do; so far really only saw
       - `for character in a_string:`
     - Can also iterate over items in list, or in a range()

Advantage of for over while

- loop variable created automagically and automagically gets proper values in sequence:
  - Let the computer do the work instead of you whenever possible!

iterating over a list

```
for faculty in ['Bob', 'Shanon', 'Dave']:
    print("Hi ", faculty,
          "Building finishes 2020 so no
              new office yet.")
```

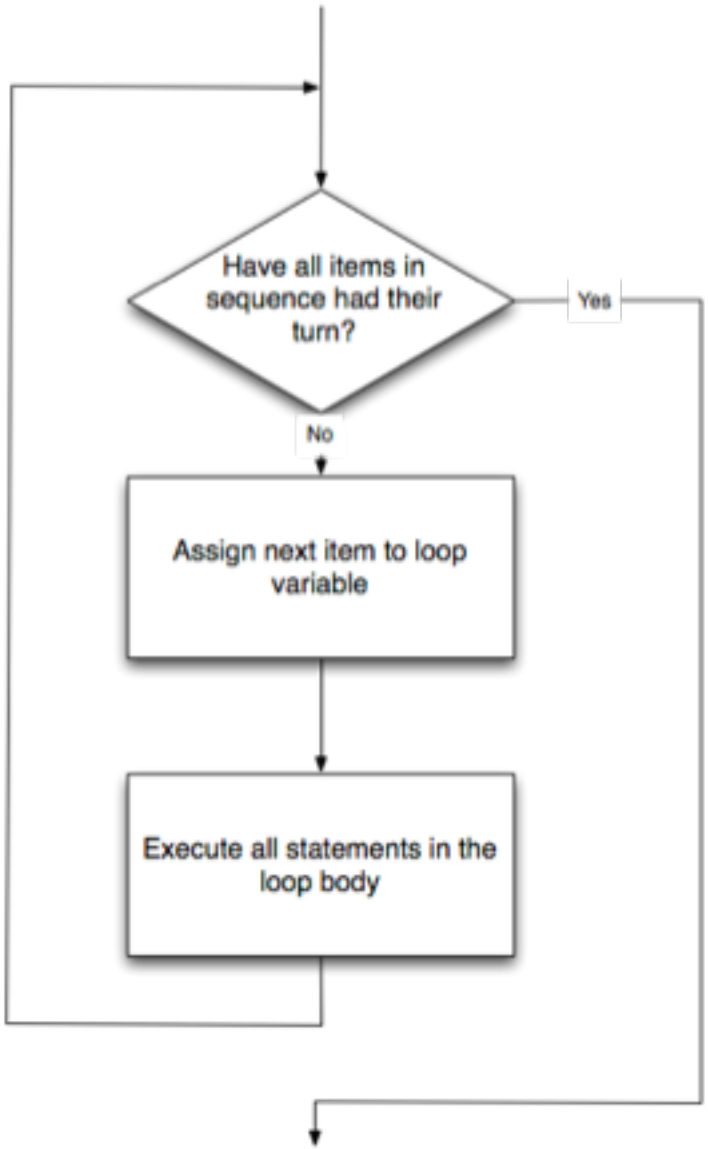
iterating over a list

loop body

```
for faculty in ['Bob', 'Shanon', 'Dave']:
      print("Hi ", faculty,
            "Building finishes 2020 so no
             new office yet.")
```

loop variable

Flow of execution



(from *How To Think Like a Computer Scientist)*

for to count

```
for i in range(10):
    print("Happy Homemade Cookies Day!")

# Yes, today really is
# National Homemade Cookies Day!!
```

Which will print numbers 1 through 3?

```
for x in range(1, 3):        # A
    print(x)
print(x)

for x in range(1, 4):        # B
    print(x)
```

A. A
B. B
C. Both A and B
D. Neither

Which this print #'s 1 through 3?

```
for x in range(1, 4):
    print(x)
    x = x + 1
```

A. Yes
B. No
C. Error
D. **Who cares which of A, B, or C because its style is so awful**