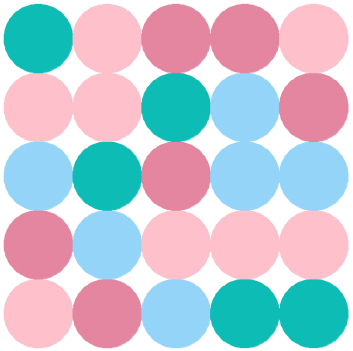
Iteration – Part 2



CS111 Computer Programming

Department of Computer Science
Wellesley College

Review: Iteration [Part 1]

- Iteration is the repeated execution of a set of statements until a stopping condition is reached.
- `while` loops are an iteration construct used when it is not known in advance how long execution should continue. `for` loops (an abstraction of `while` loops) are used when we have a fixed set of items in a sequence to iterate over.
- If the **stopping condition** is never reached, the loop will run **forever**. It is known in this case as an **infinite loop**.
- The stopping condition might involve one or more **state variables**, and we need to make sure that the body of the loop contains statements that continuously **update** these state variables.
- We can use the model of **iteration tables** to understand the inner workings of a loop. Its columns represent the state variables and the rows represent their values in every iteration.

Concepts in this slide: Comparing the syntax of both loop constructs.

Review: Syntax of loops

a boolean expression denoting whether to iterate through the body of the loop one more time.

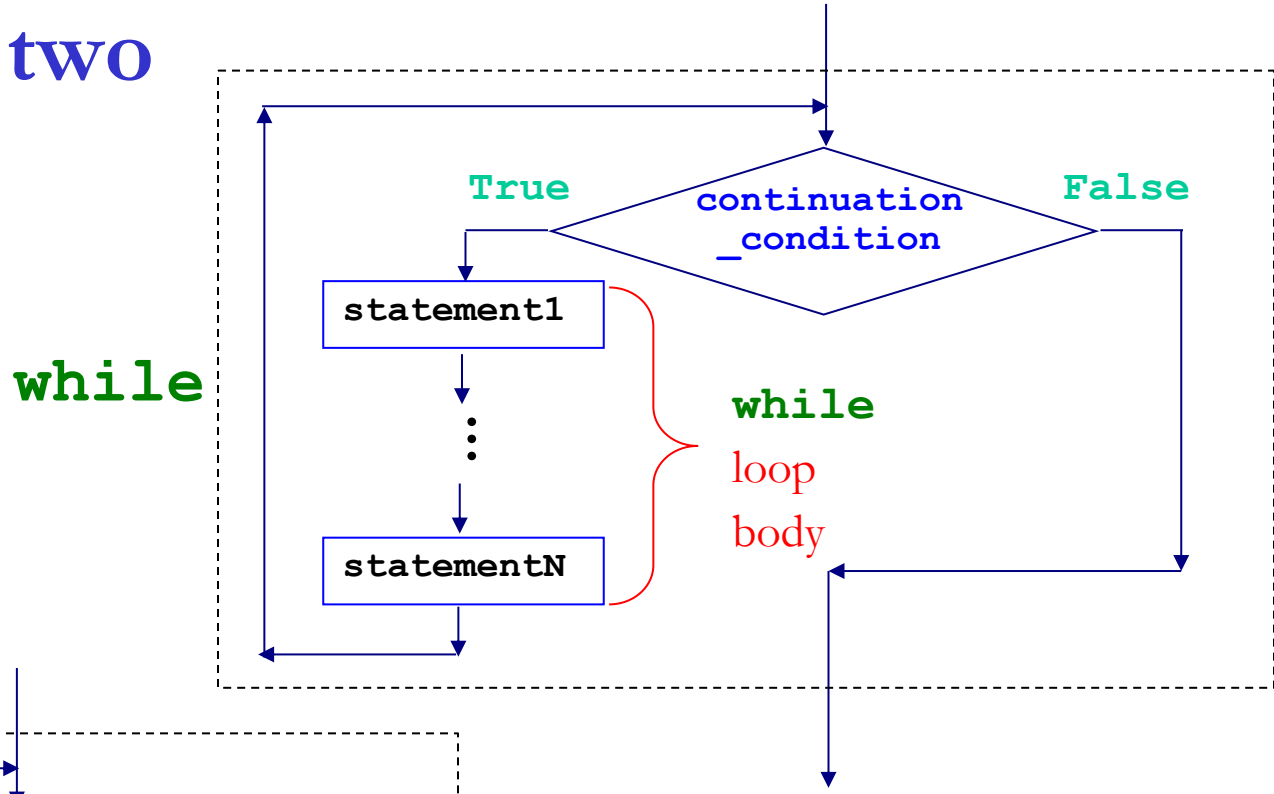
```
while continuation_condition :
    statement1
        ⋮
    statementN
```

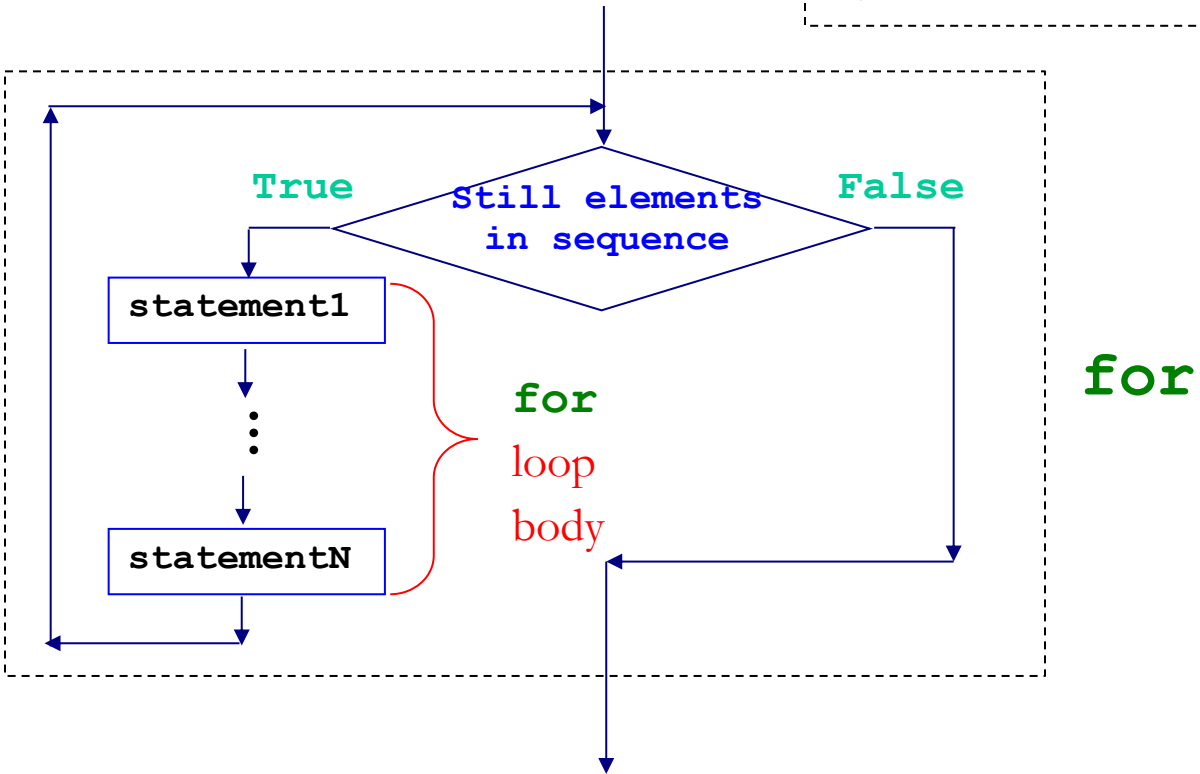
A sequence of items: characters in a string, items in a list, ranges, etc.

A variable that takes its values from the items of the sequence.

```
for var in sequence:
    statement1
        ⋮
    statementN
```

Flow charts for two loop constructs





Review: `sumBetween` with `while` loop

Concepts in this slide: Using the iteration table to reason about a problem.

```
In[6]: sumBetween(4,8)
Out[6]: 30 # 4 + 5 + 6 + 7 + 8
```

```
sumBetween(4,8) returns 30
sumBetween(4,4) returns 4
sumBetween(4,3) returns 0
```

step	lo	hi	sumSoFar
0	4	8	0
1	5	8	4
2	6	8	9
3	7	8	15
4	8	8	22
5	9	8	30

```
def sumBetween(lo, hi):
    '''Returns the sum of the integers from lo to hi
    (inclusive). Assume lo and hi are integers.'''
    sumSoFar = 0
    while lo <= hi:
        sumSoFar += lo
        lo += 1
    return sumSoFar
```

initialize accumulator

update accumulator

return accumulator

To notice:

- Row 0 in the table shows the initial values of all state variables.
- Row 1 shows values after the updates in the loop body.

Today's topics

- Nested `for` loops
- Swapping two variable values
- Simultaneous assignment in Python

Concepts in this slide: Two nested loops: the outer and inner loop.

Nested loops for printing

A **for** loop body can contain a **for** loop.

Outer loop

```
for i in range(2, 6):
    for j in range(2, 6):
        print(i, 'x', j, '=', i*j)
```

Inner loop

```
# print the multiplication table from 2 to 5
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
...
```

To notice:

- Variable **i** in the outer loop is set initially to value 2.
  - Variable **j** in the inner loop is set initially to value 2.
  - Variable **j** keeps changing its value: 3, 4, 5; meanwhile **i** doesn't change.
- When the inner loop is done, **i** becomes 3.
  - Now the inner loop starts again, and **j** takes on the values 2, 3, 4 ,5
- Every time **j** reaches 5, the inner loop ends and **i** increments.
- The outer loop ends when both **i** and **j** are 5.

Nested Loop Exercises

Fill in the BLANK

Conditionals can appear anywhere in loops.

Predict the printed output of the following loops.
(Answers are in the notebook solutions.)

```
for i in range(2, 6):
  for j in range(2, 6):
    if i <= j:
      print(i, 'x', j, '=', i*j)
```

```
for i in range(2, 6):
  if i % 2 == 0:
    for j in range(2, 6):
      if i <= j:
        print(i, 'x', j, '=', i*j)
```

See notebook solutions for answers

Nested loops for accumulation

Concepts in this slide: Using nested loops for successive accumulations.

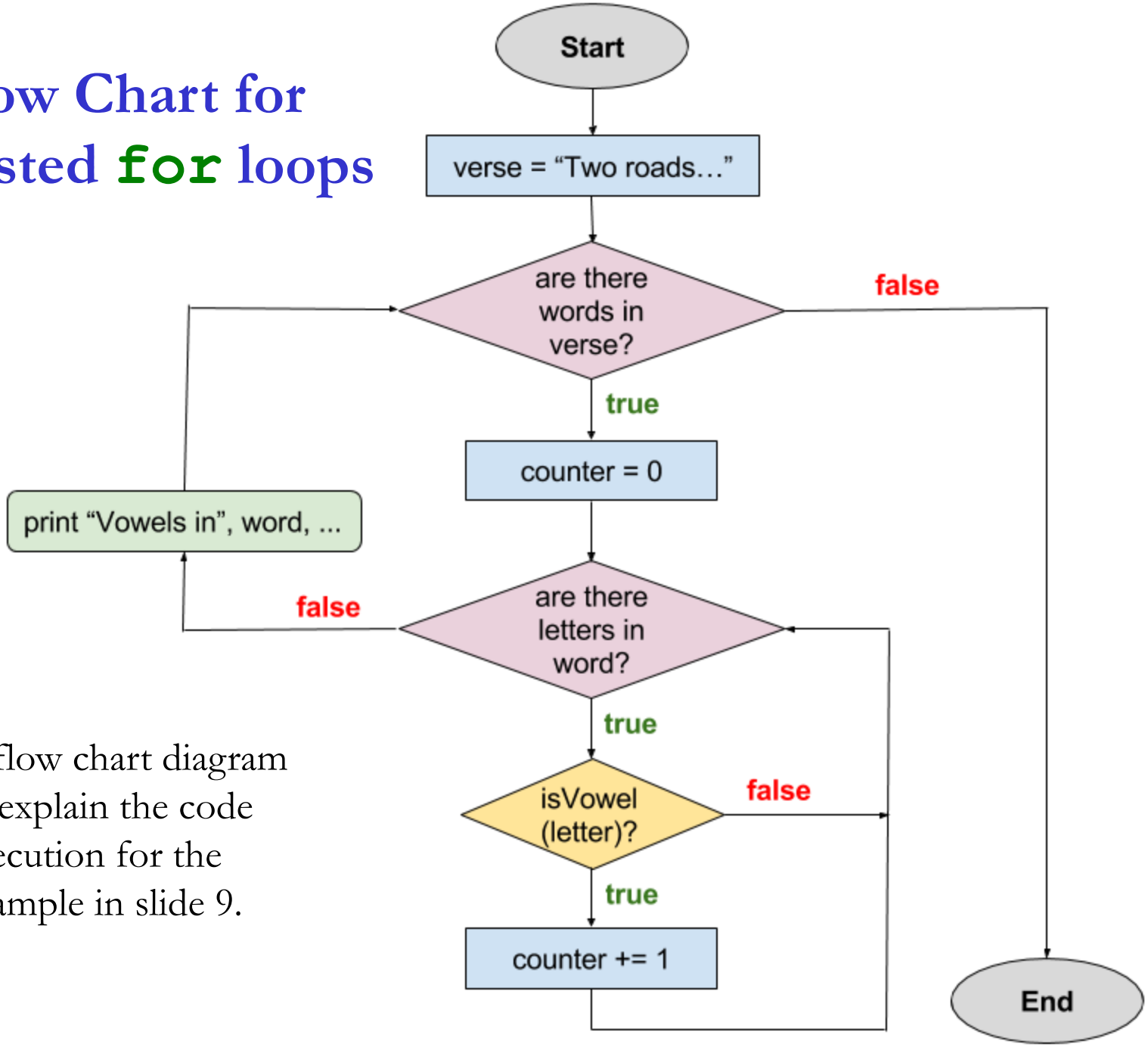
```
def isVowel(char):
    return char.lower() in 'aeiou'

verse = "Two roads diverged in a yellow wood"
for word in verse.split():
    counter = 0
    for letter in word:
        if isVowel(letter):
            counter += 1
    print('Vowels in "'+ word + '" =>', counter)
```

```
Vowels in "Two" => 1
Vowels in "roads" => 2
Vowels in "diverged" => 3
Vowels in "in" => 1
Vowels in "a" => 1
Vowels in "yellow" => 2
Vowels in "wood" => 2
```

To notice:

- The accumulator variable **`counter`** is set to 0 every time the inner loop starts.
- Outer loop iterates over a list of words.
- Inner loop iterates over characters in a string.

Flow Chart for nested `for` loops



A flow chart diagram to explain the code execution for the example in slide 9.

Avoiding Nested Loops with Functions

Encapsulating the inner loop into a separate function eliminates the loop nesting and can make programs easier to read.

```
# Our old friend countVowels from Lec 08 encapsulates
# the inner loop of the nested loop example with vowels
def countVowels(word):
    counter = 0
    for letter in word:
        if isVowel(letter):
            counter += 1
    return counter

# We can now use countVowels to avoid an explicit nested loop
# (though at runtime the for loop within countVowels still
# executes within the for loop within countVowelsInVerse
def countVowelsInVerse(verse):
    for word in verse.split():
        print('Vowels in "' + word + '" =>', countVowels(word))

countVowelsInVerse("Two roads diverged in a yellow wood")
```

Exercise: print words

Fill in the BLANK

What is printed below? (Answers are in the notebook solutions).

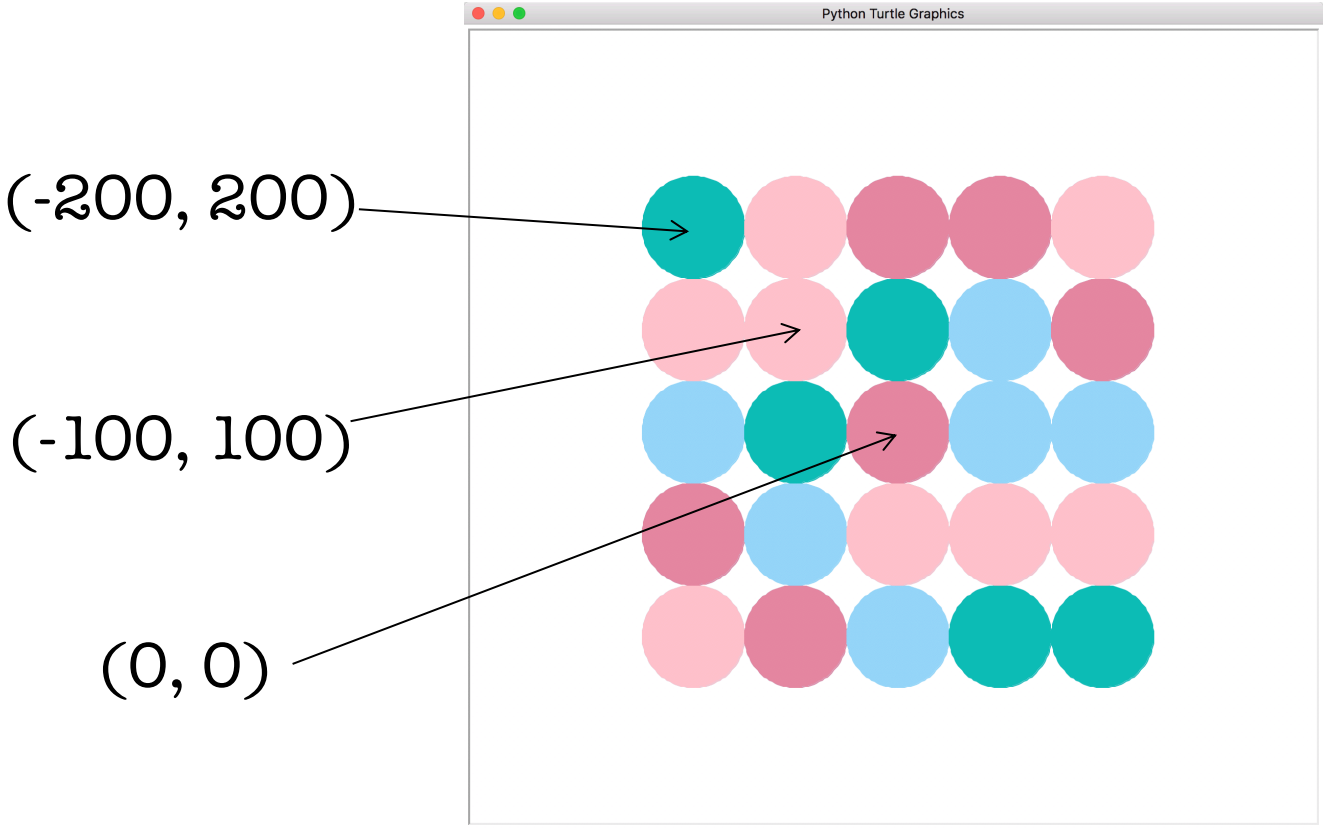
```
for letter in ['b','d','r','s']:
    for suffix in ['ad', 'ib', 'ump']:
        print(letter + suffix)
```

Exercise: Nested Loops with graphics

it's your turn

Here's a picture involving a grid of randomly colored circles with radius = 50.

This picture is created using two nested **`for`** loops. How would you do that? You can find the answers in the notebook!



Exercise: Triangles of Circles

it's your turn

Which of the 4 triangular patterns of circles is created by the following function?

```
colors = ["LightSkyBlue", "LightPink",
"LightSeaGreen", "PaleVioletRed"]

reset()
noTrace()

for x in coords:
    for y in coords:
        if y <= x:
            teleport(x, y)
            color(random.choice(colors))
            begin_fill()
            drawCircle(radius)
            end_fill()
showPicture()
```

How can you change the function to make the other 3 patterns?

Swapping Values in Python

Concepts in this slide: To swap the values of two variables, a third variable is needed.

Imagine you have a list of numbers that you want to sort by swapping two adjacent (neighbor) items every time one is smaller than the other. This is a famous algorithm known as the "bubble sort", and is usually implemented via nested for loops. If you're curious, read this page. You'll learn how to implement bubble sort in CS 230.

```
Start of list
nums = [3, 2, 1, 4]
After 1st swap
nums = [2, 3, 1, 4]
After 2nd swap
nums = [2, 1, 3, 4]
After 3rd swap
nums = [1, 2, 3, 4]
```

If we want to do the first swap of 3 and 2, can we write the following?

```
nums[0] = nums[1]
nums[1] = nums[0]
```

Try it out to see what happens. The solution in this case would look like this:

```
tempVal = nums[0]
nums[0] = nums[1]
nums[1] = tempVal
```

Simultaneous assignment in Python*

Concepts in this slide: It is possible to assign values to multiple variables in one statement.

In Python, we can assign values to many variables at once, here are some examples, that you should try in the console:

```
a, b = 0, 1
a, b, c = 1, 2, 3
a, b = "AB"
a, b = [10, 20]
a, b = (15, 25)
a, b, c, d = [1, 2, 3, 4]
```

The reason that these assignments work is that there is an equal number of variables and values on each side. Even the string "AB" is a sequence of two characters.

Try a different number of variables or values on both sides to see what errors you get.

Swapping through simultaneous assignment

```
a, b = b, a
num[0], num[1] = num[1], num[0]
```

Do these statements work?

*This is also known as **tuple assignment**.

Variable update order matters

```
def sumHalvesBroken(n):
    sumSoFar = 0
    while n > 0:
        n = n//2 # updates n too early!
        sumSoFar += n
    return sumSoFar
```

```
In [3]: sumHalvesBroken(22)
Out[3]: 19
```

step	n	sumSoFar
0	22	0
1	11	11
2	5	16
3	2	18
4	1	19
5	0	19

Important:

If update rules involve rules where state variables are dependent on one another, be very careful with the order of updates.

Simultaneous update example: Greatest Common Divisor algorithm

Digging Deeper

- The greatest common divisor (gcd) of integers a and b is the largest integer that divides both a and b
  - Eg: **`gcd(84, 60)`** is 12
- Euclid (300 BC) wrote this algorithm to compute the GCD:
- Given **a** and **b**, repeat the following steps until **b** is 0.
  - Let the new value of **b** be the remainder of dividing **a** by **b**
  - Let the new value of **a** be the old value of **b**
- … this is a perfect opportunity for a while loop.

step	**a**	**b**
0	84	60
1	60	24
2	24	12
3	12	0

Simultaneous update (e.g., gcd)

step	a	b
0	84	60
1	60	24
2	24	12
3	12	0

step	a	b
0	33	24
1	24	9
2	9	6
3	6	3
4	3	0

step	a	b
0	33	7
1	7	5
2	5	2
3	2	1
4	1	0

Neither of the following two gcd functions works. Why?

```
# Assume a >= b > 0
def gcdBroken1(a, b):
    while b != 0:
        a = b
        b = a % b
    return a
```

```
# Assume a >= b > 0
def gcdBroken2(a, b):
    while b != 0:
        b = a % b
        a = b
    return a
```

Fixing simultaneous update

Digging Deeper

```
# Assume a >= b > 0
def gcdFixed1(a, b):
    while b != 0:
        prevA = a
        prevB = b
        a = prevB
        b = prevA % prevB
    return a
```

```
# Assume a >= b > 0
def gcdFixed2(a, b):
    while b != 0:
        prevA = a
        prevB = b
        b = prevA % prevB
        a = prevB
    return a
```

Python's simultaneous assignment is an even more elegant solution!

```
# Assume a >= b > 0
def gcdFixed3(a, b):
    while b != 0:
        a, b = b, a % b # simultaneous assignment of a
                        # pair of values to a pair
                        # of variables (parens optional)
    return a
```

To notice:

- Functions 1&2 use temporary variables to store values before updates.
- Function 3 assigns multiple values in one step.

Test your knowledge

1. The **`sumBetween`** solution in slide 5 has an iteration table with three state variables. How will the iteration table look like if the solution is written with a for loop?
2. If we want to print out the entire multiplication table for 1 to 10, how many times will the print statement in slide 7 be executed?
3. What would be the final value of **`counter`** in slide 9, if we move the assignment statement before the outer **`for`** loop?
4. What results will be printed in slide 9 if the counter assignment statements moves within the inner loop?
5. For the exercise in slide 12, try to draw a flow chart diagram like the one in slide 10, before writing code to solve the problem.
6. What is an alternative way of writing the function in slide 17, which leads to the same gotcha?
7. If you type 0, 1, 2 in the Python console, what kind of type will Python assign to this sequence of numbers? How does that help for simultaneous assignments?