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.

Review: Syntax of loops
- **while** `continuation_condition:`
  - `statement1`
  - `;` statementN
  - A boolean expression denoting whether to iterate through the body of the loop one more time.
  - A variable that takes its values from the items of the sequence.

- **for** `var in sequence:`
  - `statement1`
  - `;` statementN
  - A sequence of items: characters in a string, items in a list, the result of `range`, etc.
  - Still elements in sequence

Flow charts for two loop constructs
Review: `sumBetween` with `while` loop

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

<table>
<thead>
<tr>
<th>step</th>
<th>lo</th>
<th>hi</th>
<th>sumSoFar</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>8</td>
<td>30</td>
</tr>
</tbody>
</table>

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

```
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
```

Today’s topics

- Nested `for` loops
- How to interrupt loops with code?
- Swapping two variable values
- Simultaneous assignment in Python

## Nested loops for printing

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

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

# print the multiplication table from 2 to 5

<table>
<thead>
<tr>
<th>i</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2*2</td>
<td>2*3</td>
<td>2*4</td>
<td>2*5</td>
<td>2*6</td>
</tr>
<tr>
<td>3</td>
<td>3*2</td>
<td>3*3</td>
<td>3*4</td>
<td>3*5</td>
<td>3*6</td>
</tr>
<tr>
<td>4</td>
<td>4*2</td>
<td>4*3</td>
<td>4*4</td>
<td>4*5</td>
<td>4*6</td>
</tr>
<tr>
<td>5</td>
<td>5*2</td>
<td>5*3</td>
<td>5*4</td>
<td>5*5</td>
<td>5*6</td>
</tr>
</tbody>
</table>

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 `i` becomes 3, `j` restarts its cycle: 2, 3, 4, 5, and so it repeats, until `i` has taken values over all items of the list 2, 3, 4, 5.

## Nested loops for accumulation

```
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 11-8.

Exercise: print words

What is printed?

```python
for letter in ['g', 'p', 'd', 's']:
    for letter2 in ['ib', 'ump']:
        print letter + letter2
```

Exercise: Nested Loops with graphics

Here’s a picture involving a grid of randomly colored circles with radius = 50 on a 800x600 canvas.

This picture is created using two nested for loops and the `Color.randomColor()` function. How would you do that?

Interrupting Loops

- Sometimes we want to interrupt a loop without iterating over all elements of a sequence. Examples:
  - When we have found an element we're looking for
  - When we're accumulating a value through a `for` loop and reached some desired value
- There are two situations when we can do this:
  - Within a function, via a `return` statement
  - Within a block of code, via a `break` statement

Important:
- We will not cover `break` in this lecture.
- `return` always exits the body of a function.
- We’ll use `break` when we want to exit a loop, but not the function.
Returning early from a loop

In a function, `return` can be used to exit the loop early (e.g., before it visits all the elements in a list).

```python
def isElementOf(val, elts):
    '''Returns True if val is found in elts; False otherwise'''
    for e in elts:
        if e == val:
            return True # return (and exit the function) # as soon as val is encountered
    return False # only get here if val is not in elts
```

In [1]: sentence = 'the cat that ate the mouse liked the dog that played with the ball'

In [2]: isElementOf('cat', sentence.split())
Out[2]: True
   # returns as soon as 'cat' is encountered

In [3]: isElementOf('bunny', sentence.split())
Out[3]: False
   # only get here if 'bunny' is not in sentence

Premature return done wrong (1)

def isElementOfBroken(val, elts):
    '''Faulty version that returns True if val is found in elts; False otherwise'''
    for e in elts:
        if e == val:
            return True
        else:
            return False

In [4]: isElementOfBroken(2, [2, 6, 1])
Out[4]: True
   # returns after the 1st element without examining the rest of the list.

In [5]: isElementOfBroken(6, [2, 6, 1])
Out[5]: False

Premature return done wrong (2)

def sumHalvesBroken2(n):
    '''Broken version of returns sum of halves of n'''
    sumSoFar = 0
    while n > 0:
        sumSoFar = sumSoFar + n # or sumSoFar += n
        n = n/2
    return sumSoFar # wrong indentation!
                    # exits function after first # loop iteration. Sometimes we # want this, but not here!

Wrong indentation within the loop. Function returns after first iteration

In [4]: sumHalvesBroken2(22)
Out[4]: 22

Exercises [in the notebook]

In the notebook we'll write the following functions that return early.

```python
containsDigits
containsDigit('The answer is 42')
containsDigit('76 trombones')
containsDigit('the cat ate the mouse')
containsDigit('one two three')

areAllPositive
areAllPositive([17, 5, 42, 16, 31])
areAllPositive([17, 5, -42, 16, 31])
areAllPositive([-17, 5, -42, -16, 31])
areAllPositive([])

indexOf
indexOf(8, [8,3,6,7,2,4])
indexOf(7, [8,3,6,7,2,4])
indexOf(5, [8,3,6,7,2,4])
```

In the notebook we'll try to write the functions that return early.
longestConsonantSubstring

```python
def longestConsonantSubstring(s):
    '''Returns the longest substring of consecutive consonants. If more than one such substring has the same length, returns the first to appear in the string.'''

    Note: This is hard! Draw iteration tables first!
    What state variables do you need?
```

Swapping Values in Python

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

```python
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.

Variable update order matters

```python
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    |
```

This table is the solution to slide 9-18.

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

- The greatest common divisor (gcd) of integers $a$ and $b$ is the largest integer that divides both $a$ and $b$
  - Example: $\text{gcd}(84, 60)$ 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.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>84</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>60</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>0</td>
</tr>
</tbody>
</table>

### Fixing simultaneous update

#### 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
    return a
```

#### To notice:
- Functions 1&2 use temporary variables to store values before updates.
- The third function assigns multiple values in one step.

### Test your knowledge

1. The `sumBetween` solution in 11-5 has an iteration table with three state variables. How will the iteration table look like if the solution is written with a for loop (see Notebook Lecture 9)?
2. If we want to print out the entire multiplication table for 1 to 10, how many times will the print statement be executed?
3. What would be the value of `counter` in 11-8, if we move the assignment statement before the outer `for` loop?
4. What results will be printed in 11-8 if the counter assignment statement moves within the inner loop?
5. For the exercise in 11-11, try to draw a flowchart diagram as the one in 11-9, before writing code to solve the problem.
6. What is an alternative way of writing the function in 11-4, which leads to the same gotcha?
7. Only by reasoning about the problems in 11-16 (no need to write code yet), which of them needs to be solved with the accumulator pattern?
8. If you write `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?