Tuples
Named Tuples
Loops contd

How comfortable are you with using for loops in your programs?

A. Very comfortable

B. Need more practice

C. A little confused

D. Very confused

Tuples

- Similar to lists: store a sequence of elements

lst = [10, 20] //ex of a list

tup = (10, 20) //ex of a tuple

- Elements are ordered an can be accessed using the appropriate index

tup[0]

tup[1]

- Different from lists in the following ways
  - Can't change an element in the tuple
  - Can't sort the elements in a tuple

Named Tuples

- Used to package data with multiple attributes: e.g. representing a student in your program
- A student's attributes may be: name, perm number, major etc.
- Named tuples make it easier to access each attribute

```
from collections import namedtuple

#Design your named tuple object
Student = namedtuple('Student', 'name perm major gpa')

# Create objects of type Student
s1 = Student("Jack", 123443, CS, 3.8)
s2 = Student("Mary", 8932737, CE, 3.9)

# Access the elements of the objects
print(s1.name, s1.perm)
```

The accumulator pattern

Useful for calculating something from repeated smaller computations

Example: find the sum of a series

```
def sumGeometric series(n):
  '''returns the sum of the series
   1 + 2**1 + 2**2 + 2**3 + …+ 2**n'''
   Assume n>=0 '''
```

More on the accumulator pattern

```
def countWords(sentence):
  "returns the number of words in the sentence"


def countWords(sentence, len):
  "returns the number of words in the sentence with
  length greater than len"
```

Concept Test

```
def containsOddNumber(lst):
    '''return True if any element in lst is
    odd, otherwise return False'''
    for x in lst:
        if (x % 2 == 1):
            return True
        else:
            return False
```

Put this statement outside the for loop

The loop runs only once and returns True or False depending on the value of only the first number in the list

Is the above implementation correct?(Why or Why not)

A. Yes

B. No

lst = [3, 4, 10] | True

lst = [2, 4, 10] | False

lst = [2, 4, 7] | True

Returns False incorrectly in the last input

Index vs value

```
def largestOddNumber(lst):
  "return the maximum odd number in the list,
  return -1 if the list has no odd numbers”
```

```
def indexOfLargestOdd(lst):
  "return the index of the largest odd number in
  the list, return -1 if there are no odd numbers
  in the list”
```

Concep Question

```
def hasVowels(word):
    if type(word) == str:
        for letter in word:
            if letter in 'aeiou':
                return True
    else:
        return False
```

What is the return value for hasVowels("")?

A. True

B. False

C. None

Motivating While Loops

- So far, we know about one type of loop: `for` loop
  - It requires a sequence (e.g. a `range` sequence or a string) to loop over
- Another type of loop is the `while` loop: it repeatedly tests a condition, executing the entire body of the loop if it is True, and terminating the loop if it is False
  - Useful when there is no sequence to loop over
  - Commonly used when we don't know how many times the loop will run

ConcepTest

What is printed by the following code? (Output is on one line to save space.)

```
x = 6
while x > 4:
  print(x)
  x = x - 1
```

- A. 6 5
- B. 6 5 4
- C. 5 4
- D. 5 4 3
- E. 6 5 4 3

ConcepTest

What is printed by the following code? (Output is on one line to save space.)

```
x = 6
while x > 4:
  x = x - 1
  print(x)
```

- A. 6 5
- B. 6 5 4
- C. 5 4
- D. 5 4 3
- E. 6 5 4 3

For vs. While

Use `for` when:

- You want to loop through an entire sequence without stopping
- The number of iterations does not depend on user input
- The increment to the loop variable is the same on every iteration

```
s = 'abc'
for count in range(len(s)):
  print('Index {0} is {1}'.format(count, s[count]))

count = 0
while count < len(s):
  print('Index {0} is {1}'.format(count, s[count]))
  count += 1
```

ConcepTest

```
valid = False
while not valid:
  s = input ("Enter a password: ")
  valid = len(s) == 5 and s[:2] == 'xy'
```

Which of the following passwords gets us out of the loop?

- A. `xyz`
- B. `abcxy`
- C. `xyabc`
- D. More than one of the above passwords get us out of the loop
- E. None; the loop never executes and no passwords are obtained

True and break

- There are several ways to write a loop whose body is required to run at least once
  1. Artificially make the condition true before the loop starts (like `inputloop.py`)
  2. Copy some loop code above the loop to make the condition true
  3. Use `True` as the condition and `break` to exit the loop
- `break` causes immediate termination of the loop
- `break` can make code difficult to read if used improperly
- We frequently do not allow `break` on exams or assignments

ConcepTest

A valid password is one that is length 5 and starts with `xy`. Such passwords should get us out of the loop. Which of these does this?

- A.

```
while True:
  s = input ("Enter a password: ")
  if len(s) == 5 and s[:2] == 'xy':
    break
```

- B.

```
s = input ("Enter a password: ")
while len(s) == 5 and s[:2] == 'xy':
  s = input ("Enter a password: ")
```

- C. Both are correct
- D. None is correct

ConcepTest

What is the output of this code? (Output is on one line here to save space.)

```
n = 3
while n > 0:
  if n == 5:
    n = -99
  print(n)
  n = n + 1
```

- A. 3 4
- B. 3 4 5
- C. 3 4 -99
- D. 3 4 5 -99