CSC 221: Introduction to Programming

Fall 2013

Lists

- lists as sequences
- list operations
  +, *, len, indexing, slicing, for-in, in
- example: dice stats
- from strings to lists
- list methods
  index, count, sort, reverse, append, extend, insert, remove

String review

recall that a Python string is a *sequence* of characters

basic operations on strings:

- `+` concatenates strings
- `*` concatenates multiple copies
- `len` function
- indexing via [i]
- slicing via [s:e], [s:e:n], [:e], [s:], …

string methods:

- isupper, islower, isalpha, isdigit, isspace
- capitalize, upper, lower
- find, rfind, count, replace
- strip, lstrip, rstrip
- center, ljust, rjust

Lists

closely related: a list is a *sequence* of arbitrary items

- can specify the contents of a list in brackets, separated by commas

```
>>> words = ["foo", "bar", "biz", "baz"]
>>> words
['foo', 'bar', 'biz', 'baz']
>>> nums = [1, 2, 3, 4]
>>> nums
[1, 2, 3, 4]
>>> mixed = ["foo", 9, 3+5, ["a", "b"], True]
>>> mixed
['foo', 9, 8, ['a', 'b'], True]
```

the same sequence operations apply to lists:

- `+` concatenates lists
- `*` concatenates multiple copies
- `len` function
- indexing via [i]
- slicing via [s:e], [s:e:n], [:e], [s:], …

3

List operation examples

```
>>> mixed = ["foo", 9, 3+5, ["a", "b"], True]
>>> mixed
['foo', 9, 8, ['a', 'b'], True]
>>>
>>> mixed[0]
'foo'
>>> mixed[1:3]
[9, 8]
>>> len(mixed)
5
>>> [1, 2] + [3, 4]
[1, 2, 3, 4]
>>> [0] * 6
[0, 0, 0, 0, 0, 0]
```

```
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> import random
>>> for i in range(10):
        nums += [random.randint(1, 4)]


>>> nums
[4, 2, 1, 4, 1, 2, 2, 2, 3, 4, 1]
```

note: a `range` is sequence similar to a list

can be converted into a list using `list`

4

Exercises

grab a partner and go to codingbat.com

- solve problems (in order) from the List-1 category
- all of these problems can be solved using list indexing, slicing, +, and *

5

Traversing a list sequence

as with strings (character sequences), can traverse a list using for-in

```
>>> nums = [3, 9, 2.7, 500, -1]
>>> for nextValue in nums:
        print(nextValue)


3
9
2.7
500
-1
```

likewise, can test for list membership using in

```
>>> 2.7 in nums
True
>>> 2.8 in nums
False
>>> 1000/2 in nums
True
>>> 1000//2 in nums
True
```

6

Example: averaging numbers

suppose you have a list of numbers (e.g., grades)

- want to calculate the average of those numbers

```
def average(numList):
    sum = 0
    for num in numList:
        sum += num
    return sum/len(numList)
```

any potential problems?

- i.e., will this function work correctly on all possible lists of numbers?

modify this function so that it returns 0.0 on an empty list

7

Dice stats

recall earlier examples involving simulated dice rolls

- suppose we wanted to perform repeated rolls, keep track of the number of 7's

```
import control

def sevenStats(numRolls):
    sevenCount = 0
    for i in range(numRolls):
        roll = control.rollDie(6) + control.rollDie(6)
        if roll == 7:
            sevenCount += 1
    return sevenCount
```

```
>>> sevenStats(1000)
173
>>> sevenStats(1000)
157
>>> sevenStats(10000)
1711
>>> sevenStats(10000)
1647
>>> sevenStats(100000)
16738
>>> sevenStats(100000)
16679
```

8

Dice stats (cont.)

we could similarly define functions for the other dice totals

- or, could generalize by adding a parameter for the desired total

```
def rollStats(numRolls, desiredRoll):
    rollCount = 0
    for i in range(numRolls):
        roll = control.rollDie(6) + control.rollDie(6)
        if roll == desiredRoll:
            rollCount += 1
    return rollCount
```

```
>>> for i in range(2, 13):
        print(i, ":", rollStats(10000, i))


2 : 272
3 : 565
4 : 885
5 : 1090
6 : 1362
7 : 1711
8 : 1422
9 : 1075
10 : 913
11 : 579
12 : 275
```

do these totals add up to 10,000?

should they?

9

Dice stats (cont.)

the problem with separate calls for each dice total is that each call is independent of the others

- actually simulating 10,000 x 11 different rolls
- WASTEFUL, and leads to inconsistent results

what we need is to be able to count all the totals at the same time

- could do it by having 11 variables, each keeping count for a dice total
- would need an 11-case if-elif-else to check for each dice total
- would then need to return or print each count

10

UGLY!

extremely tedious

what if we wanted to extend this example to 8-sided dice?

what would have to change?

```
def diceStats1(numRolls):
    twoCount = 0
    threeCount = 0
    fourCount = 0
    fiveCount = 0
    sixCount = 0
    sevenCount = 0
    eightCount = 0
    nineCount = 0
    tenCount = 0
    elevenCount = 0
    twelveCount = 0
    for i in range(numRolls):
        roll = control.rollDie(6) + control.rollDie(6)
        if roll == 2:
            twoCount += 1
        elif roll == 3:
            threeCount += 1
        elif roll == 4:
            fourCount += 1
        elif roll == 5:
            fiveCount += 1
        elif roll == 6:
            sixCount += 1
        elif roll == 7:
            sevenCount += 1
        elif roll == 8:
            eightCount += 1
        elif roll == 9:
            nineCount += 1
        elif roll == 10:
            tenCount += 1
        elif roll == 11:
            elevenCount += 1
        else:
            twelveCount += 1
    return [twoCount, threeCount, fourCount, fiveCount, \
            sixCount, sevenCount, eightCount, nineCount, \
            tenCount, elevenCount, twelveCount]
```

11

Better solution

11 separate variables are unwieldy – no way to test/update uniformly

alternatively, could place all 11 counters in a single list

- actually, to simplify things will create a list with 13 numbers
  - `twoCount` will be stored in `rollCount[2]`
  - `threeCount` will be stored in `rollCount[3]`
  - ...
  - `twelveCount` will be stored in `rollCount[12]`

```
def diceStats2(numRolls):
    rollCount = [0]*13
    for i in range(numRolls):
        roll = control.rollDie(6) + control.rollDie(6)
        rollCount[roll] += 1
    return rollCount[2:]
```

```
>>> diceStats2(100000)
[2750, 5583, 8351, 11031, 13878, 16808, 13921, 11027, 8302, 5619, 2730]
```

12

Nicer output

simply returning the list of counts is not ideal

- difficult to see which number goes with which dice total

```
def diceStats3(numRolls):
    rollCount = [0]*13
    for i in range(numRolls):
        roll = control.rollDie(6) + control.rollDie(6)
        rollCount[roll] += 1
    for r in range(2,13):
        print("Number of", str(r)+"\'s =", rollCount[r])
```

```
>>> diceStats3(100000)
Number of 2's = 2859
Number of 3's = 5538
Number of 4's = 8332
Number of 5's = 11186
Number of 6's = 13806
Number of 7's = 16600
Number of 8's = 13877
Number of 9's = 11260
Number of 10's = 8329
Number of 11's = 5377
Number of 12's = 2836
```

13

Generalized version

finally, could generalize to work for different dice

- can even format the output to align the numbers

```
def diceStats4(numRolls, dieSides):
    rollCount = [0]*(2*dieSides+1)
    for i in range(numRolls):
        roll = control.rollDie(dieSides) + \
               control.rollDie(dieSides)
        rollCount[roll] += 1
    for r in range(2,2*dieSides+1):
        print("Number of", str(r).rjust(2)+"\'s =", \
              str(rollCount[r]).rjust(len(str(numRolls))))
```

```
>>> diceStats4(100000, 8)
Number of  2's =   1575
Number of  3's =   3103
Number of  4's =   4672
Number of  5's =   6137
Number of  6's =   7697
Number of  7's =   9461
Number of  8's =  10911
Number of  9's =  12605
Number of 10's =  10994
Number of 11's =   9306
Number of 12's =   7775
Number of 13's =   6312
Number of 14's =   4807
Number of 15's =   3111
Number of 16's =   1534
```

14

Strings & lists

there is a useful string method that converts a string into a list

- `str.split()` returns a list of the separate words in str
- by default, it uses whitespace to separate the words, but can provide an optional argument to specify different separators

```
>>> sentence = "This sentence is made up of eight words"
>>> sentence.split()
['This', 'sentence', 'is', 'made', 'up', 'of', 'eight', 'words']
>>> csv = "Hoover, J. Edgar"
>>> csv.split(",")
['Hoover', ' J. Edgar']
```

15

Adding user interaction

since it is easier for the user to enter multiple items in a string, reading input as a string and then splitting into a list for processing is common

- will the following work for averaging grades?

```
def average(numList):
    if numList == []:
        return 0.0
    else:
        sum = 0
        for num in numList:
            sum += num
        return sum/len(numList)

def avgGrades():
    gradeString = input("Enter your grades: ")
    gradeList = gradeString.split()
    avg = average(gradeList)
    print("The average is", avg)
```

16

Adding user interaction (cont.)

not quite – `split` produces an array of strings

- but, `average` is expecting a list of numbers

- could go through the list and convert each string with the corresponding number
- simpler, could have `average` convert each element to a number when processing

```
def average(numList):
    if numList == []:
        return 0.0
    else:
        sum = 0
        for num in numList:
            sum += float(num)
        return sum/len(numList)

def avgGrades():
    gradeString = input("Enter your grades: ")
    gradeList = gradeString.split()
    avg = average(gradeList)
    print("The average is", avg)
```

17

Mystery function

what does this function do?

```
def mystery(phrase):
    words = phrase.split()
    letters = ""
    for nextWord in words:
        letters += nextWord[0].upper()
    return letters
```

18

Pig Latin revisited

if we want to convert a phrase into Pig Latin

- split the string into individual words
- use the `pigLatin` function to convert
- append the Pig Latin words back together

```
import text

def pigPhrase(phrase):
    words = phrase.split()
    pigPhrase = ""
    for nextWord in words:
        pigPhrase += text.pigLatin(nextWord) + " "
    return pigPhrase.strip()
```

why the call to strip?

19

List methods

similar to string, the list type has useful methods defined for it

index(x)	Return the index of the first occurrence of x (error if not in the list)
count(x)	Return the number of times x appears in the list
sort()	Sort the items of the list, in place
reverse()	Reverse the elements of the list, in place

the following methods actually modify the contents of the list

- unlike strings, lists are mutable objects

append(x)	Add x to the end of the list
extend(L)	Extend the list by appending all the items in L to end
insert(i, x)	Insert x at a given position i
remove(x)	Remove the first occurrence of x (error if not in the list)

20