

# CSC 221: Introduction to Programming

Fall 2011

## 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

1

## 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

2

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

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

note: range returns a list of numbers

4

## Exercises

grab a partner and go to [codingbat.com](https://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
```

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 float(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

```
from control import rollDie  
  
def sevenStats(numRolls):  
    sevenCount = 0  
    for i in range(numRolls):  
        roll = rollDie(6) + 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, desiredRolls):  
    rollCount = 0  
    for i in range(numRolls):  
        roll = rollDie(6) + rollDie(6)  
        if roll == desiredRolls:  
            rollCount += 1  
    return rollCount
```

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

```
2 : 281  
3 : 569  
4 : 790  
5 : 1144  
6 : 1337  
7 : 1660  
8 : 1411  
9 : 1151  
10 : 848  
11 : 508  
12 : 272
```

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 diceStats(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 = rollDie(6) + 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
    print 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 diceStats(numRolls):
    rollCount = [0]*13
    for i in range(numRolls):
        roll = rollDie(6) + rollDie(6)
        rollCount[roll] += 1
    return rollCount
```

```
>>> diceStats(100000)
[0, 0, 2823, 5479, 8187, 11048, 13950, 16992, 13706, 11132, 8211, 5678, 2794]
```

12

## Nicer output

simply returning the list of counts is not ideal

- shows the placeholder numbers in indices 0 and 1
- difficult to see which number goes with which dice total

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

```
>>> diceStats(100000)
Number of 2's = 2771
Number of 3's = 5534
Number of 4's = 8438
Number of 5's = 11010
Number of 6's = 13791
Number of 7's = 16840
Number of 8's = 13917
Number of 9's = 11067
Number of 10's = 8359
Number of 11's = 5512
Number of 12's = 2761
```

13

## Generalized version

finally, could generalize to work for different dice

- can even format the output to align the numbers

```
def diceStats(numRolls, dieSides):
    rollCount = [0]*(2*dieSides+1)
    for i in range(numRolls):
        roll = rollDie(dieSides) + 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)))
```

```
>>> diceStats(100000, 8)
Number of  2's = 1569
Number of  3's = 3120
Number of  4's = 4686
Number of  5's = 6248
Number of  6's = 7782
Number of  7's = 9327
Number of  8's = 10853
Number of  9's = 12454
Number of 10's = 11026
Number of 11's = 9443
Number of 12's = 7762
Number of 13's = 6302
Number of 14's = 4799
Number of 15's = 3075
Number of 16's = 1554
```

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 float(sum)/len(numList)

def avgGrades():
    gradeString = raw_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 = raw_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

```
from words import pigLatin

def pigPhrase(phrase):
    words = phrase.split()
    pigPhrase = ""
    for nextWord in words:
        pigPhrase += 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

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

the following methods actually modify the contents of the list

- unlike strings, lists are mutable objects

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

20

## More practice

grab a partner and go back to [codingbat.com](https://codingbat.com)

- solve the first 3 problems from the List-2 category
- these problems can be solved using a loop to traverse and inspect the list items