Introduction to: Computers & Programming: Sequences in Python Part 2

Adam Meyers
New York University
Outline

• Lists: Reviewing and Elaborating
• Lists for different types of problems
• Ordering Sequences
• Summary
• Homework
Lists: Overview

- Sequence of elements of any type (called arrays in other languages)
- Python Syntax: elements in square brackets separated by commas
- Indexes and Slices – like all sequences
  - my_list = ['the','big','cheese','sandwich']
  - my_list[0] → 'the'
  - my_list [-1] → 'sandwich'
  - my_list [2:] → ['cheese','sandwich']
  - my_list[:2] → ['the','cheese']
  - my_list[1:3] → ['big','cheese']

- Mutable (unlike previous data types we discussed)
  - my_list.append('is')
  - my_list.extend(['on','the','plate'])
  - my_list[0]='The'
  - my_list → ['The','big','cheese','sandwich','is','on','the','plate']

- To produce a copy of a list (not a pointer to the same list)
  - copy_list = my_list[:]  ## a slice that goes from the beginning to the end
Mutability: Test Lists to Predict Behavior

• Two variables can point to the same list (changing the list will change both values)
  – Abc = [1]
  – Abd = Abc
  – Abc.append(1)
  – Abd → [1,1]

• Functions can change lists
  – def add_2(inlist):
    # the variable inlist is local, but the list it points to is not
    inlist.append(2)
    add_2(Abd)
    Abd → [1,1,2]

• Be careful of the repetition operators repeat lists, they don't copy them
  – Abc = [1]*3
    • Abc → [1, 1, 1] # 3 copies of the integer 1
  – efg = [[1,2,3]]*3
    • Efg → [[1,2,3],[1,2,3],[1,2,3]] # 3 pointers to the same exact list
    • Efg[2][0] = 'banana'
    • Efg → [['banana', 2, 3], ['banana', 2, 3], ['banana', 2, 3]]
Other List-Like Structures in Python

- Tuples – dealt with peripherally in this class
  - Round brackets instead of square
  - Unmutable, act more like strings
  - Not used as much as lists

- Sets – dealt with peripherally in this class
  - Curly brackets instead of square
  - Not sequences, order doesn't matter
  - No duplicate elements
  - Standard set operations (subset, superset, etc.)

- Dictionaries – covered later in the semester
  - Like sets, except sets of keys and values
  - Values can be “looked up” by using keys
  - Implementation of hash tables
Program File: dog_instructions.py

- http://cs.nyu.edu/courses/fall15/CSCI-UA.0002-007/dog_instructions.py

- Imports functions & variables from http://cs.nyu.edu/courses/fall15/CSCI-UA.0002-007/turtle_dog.py
  - Put in same directory

- Var1, var2, var3, … = list_or_tuple

- 2 kinds of lists used for drawing dogs:
  - List of different variable values – a record of a sort
  - List of lists = sets of instructions
Examples with Lists as Records

• \([\text{name, id\_number, hw\_grade, test1\_grade, test2\_grade, test3\_grade}]\)
  – For each student, printout a statement including: their name, student\_number and average (weighted average)

• \([\text{item\_number, description, price, net\_weight, review\_score}]\)
  – Sort products once by review\_scores and once by price

• \([\text{team\_name, wins, losses, ties}]\)
  – Print each team name followed by a score equal to \((\text{wins} + 1/2 \text{ ties})/\text{total\_number\_of\_games}\)

• In order to use a list in this way, your programs must be consistent with certain positions in the list having certain purposes, as suggested by the variable names
Text Version of Tic Tac Toe: Slide 1

- [http://cs.nyu.edu/courses/fall15/CSCI-UA.0002-007/tic_tac_toe.py](http://cs.nyu.edu/courses/fall15/CSCI-UA.0002-007/tic_tac_toe.py)
- Initialize list of rows to represent board

  for num in range(3):
      row = ['_'] * 3  # creates a new list each iteration
      rows.append(row)

  # bad_copy = row*3 → same list 3 times – causes problems

  for row in rows:
      # ([_, _, _], [_, _, _], [_, _, _])
      print(row)  # looks like a tic tac toe board
      [_, _, _]
      [_, _, _]
      [_, _, _]

- Game: Take turns filling in X or O, trying to get 3 in a row
- Each position can be identified by indexes,
  - Rows[0][1] → 1st row, 2nd position (or 2nd column)
Text Version of Tic Tac Toe – Slide 2

• 'X' and 'Y' take turns
  – Each turn results in filling in one position
  – A turn can result in X or O winning
  – If 9 turns pass and nobody won, it is a draw

• `go_tic_tac_toe_turn(player,rows)`
  – Player is X or O
  – Player is queried about where he/she wants to go
  – The list of lists that row points to is modified

  • The variable `rows` is local, but the list it points to can be changed because lists are mutable
Text Version of Tic Tac Toe – Slide 3
go_tic_tac_toe

• Player Input
  – Player provides row and column (1, 2 or 3)
  – '1', '2', '3' are the only permitted answers
  – A player cannot chose a position already occupied by 'X' or 'O'

• Program prints the board after player goes

• Checks for winning condition and returns 'X' or 'O' if the player wins
  – All items in row match 'X' (or 'O')
  – The same index in each row match 'X' or 'O' (a column)
  – A diagonal: the middle plus opposing corners (2 cases)
The Turtle Version of Tic Tac Toe

- The text version plus additional elements
- Draws the board on turtle.Screen and records in a list the X,Y coordinates of the centers of each of the 9 board positions on the board
- `go_turtle_tic_tac_toe_turn`
  - Like `go_tic_tac_toe_turn`, except also moves turtle to center of position on board and draws X or O
- Writes win or draw message on turtle.Screen
List of Lists Representing Structure
(AKA 2 dimensional Lists)

- Tic Tac Toe rows[row_num][col_num]
  - A way of representing X and Y in a grid
  - Could also be used to represent X and Ys for turtle programs
- NYCStreets[X,Y] – for numbered Streets and Avenues in NYC (simplified)
  - Streets[4][6] → 'Waverly Theater'
- Wins = [['Mets','win','lose','win','win'],
  ['lose','Yankees','win','win','win'],
  ['win','lose','Bears','lose','lose'],
  ['lose','lose','win','Senators','win'],
  ['lose','lose','win','lose','Clowns']]
  - Wins[X][X] – team name of X
  - Wins[X][Y] – did team X win or lose against Y
More 2-D Lists

• Paragraph1 = [['John','saw','Mary','.'],['Mary','saw','John','.'] …]
  – Paragraph1[0][2] → 'Mary'

• The first sentence, second word

• Playing_Cards = [[[A', 'C'], [A', 'D'],[A' 'H'],[A','S']].
  [['2','C'],['2','D'],['2','H'],['2','S']]]
  [['3','C'],['3','D'],['3','H'],['3','S']]]
  …

  [['K','C'],['K','D'],['K','H'],['K','S']]]
  – Playing Cards[1][3] → ['2','S']
Comparing Objects

- Operators <, >, <=, >= are defined for numbers, strings and some lists (or tuples)
- For numbers, they have the expected interpretations
- Characters ordered based on unicode numbers:
  - \([\text{ord('1')}, \text{ord('A')}, \text{ord('a')}\] → [49, 65, 97]
  - '1'< 'A' → True
  - 'A' 'a' → True
  - 'a' '1' → False
- True > False
- Orders of Sequences based on the order of their parts
Sequences Ordered Lexographically

- The > operator is defined approximately as follows:
  - `def seq_greater_than(seq1, seq2):
    length = min(len(seq1), len(seq2))
    for index in range(length):
      if seq1[index] < seq2[index]:
        return(True)
      elif seq2[index] < seq1[index]:
        return(False)
    if len(seq1) > len(seq2):
      return(True)
    else:
      return(False)
  `

- Compares sequences one item at a time from the beginning
- Alphabetic order is a type of lexicographic order
Comparing Sequences

- '456' > '1234' → True
  - '4' (52) comes after '1' (49) in the unicode table
- 'Zebra' < 'ant' → True
  - 'Z' (90) comes before 'a' (97) in the unicode table
- 'ch ick' < 'chick' → True
  - ' ' (32) comes before 'c' (99) in the unicode table
- [1,3,4]<[1,5,6] → True
  - Based on the first different item
- [55, 'chicken', 'blah', 'blah', 'blah'] > [55, 'chicken', 'blah'] → True
  - Sequential elements are the same, but first is longer
- [True,True, False] < [True,True,True] → True
  - The left list has the first False matching a True
- [55, 'chicken'] > [55, 45] → Error
  - First items are the same, second items are not compatible
Sorting lists

- Sorting implies comparison using $\leq$
- **List1** = [-20, 45, 13, 1, 25.1]
  - List1.sort()
  - List1 $\rightarrow$ [-20, 1, 13, 25.1, 45]
  - Numbers sorted from lowest to highest number (integers and floats can be compared)
- **List2** = ['abc', '123', 'efg', 'ABC', 'EFG']
  - List2.sort()
  - ['123', 'ABC', 'EFG', 'abc', 'efg']
- **List3** = [[2, 'chicken'], [45, 'duck'], [-50, 'ox'], [150, 'emu'], [50, 'calf']]
  - List3.sort()
  - [[-50, 'ox'], [2, 'chicken'], [45, 'duck'], [50, 'calf'], [50, 'emu']]
  - Lists of lists are sorted by first elements, than second elements, etc.
- **List4** = [1, 'chicken', ['abc', 5]]
  - List4.sort() ### causes an error because string 'chicken' cannot be compared to number 5
Max and Min

• Like sort, they use the $\leq$ and $\geq$
• Like sort, they use lexicographic order for sequences
• max('abcd Ghhau &') → 'u'
• min('012345') → '0'
• max(['$%^&*%^&%$','abc','XYZ']) → 'abc'
• max([True,True,False,True,True]) → True
• min(['abc',99,True]) → error
Controlling Sequence Comparisons by Using a Score

• You classified homework as follows:
  – [SUBJECT,Days_until_due,Maj_Min_Elect,hours]
• Now you want to order it favoring
  – Major < Minor < Elective
  – Fewer hours < More hours
  – Fewer Days_until_due < More Days_until_due
• You can make up a score based on these attributes and sort according to that score
• http://cs.nyu.edu/courses/fall15/CSCI-UA.0002-007/homework_sort.py
Summary

• Python's List Data structure
  – Is mutable
  – list any kind of element in a sequence
• Lists are used to represent records, tables of data, and other multi-dimensional objects
• Python defines <, >, <=, >= for many data types.
• Sequences use these definitions lexicographically
• Sort, max, min and other functions assume these orderings as well
• A Score Function provides 1 means to sort data according to your own definition.
Homework Part 1 (Due Nov 9): Question 1

• Given input data in the form
  – [team_name, wins, losses, ties]

• Write a function that computes the following score:
  – (wins+1/2 ties)/total_number_of_games

• 1a) Write a function that prints each team name followed by the score

• 1b) Write a function that uses this Score to sort the players from highest to lowest score. See the homework-sort.py program we discussed in class

• Use the following as sample list of input:
  – Input_Data = [['Mets',10,5,5],
                   ['Yankees',11,2,2],
                   ['Bears',7,15,0],
                   ['Senators',5,30,1],
                   ['Clowns',10,50,1]]

• Hint: You will probably need to use the list.reverse() method for 1b
Homework Part 1 – Question 2

• Write a turtle graphics program that given 2 sets of coordinates will draw a complete bipartite graph. In a complete bipartite graph, lines connect all the points in one set with all the points in the other set. In the following example, there are lines connecting all the dots on the left with all the dots on the right.
Homework Part 1 – Question 2 Continued

• Test your program using the following 2 lists of X and Y coordinates:
  – coord_list_1 = [[-200,200],[-200,90],[-50,-20]]
  – coord_list_2 = [[50,200],[50,170],[300,90],[50,30],[50,0],[50,-13],[50,-73]]

• It is suggested that you use the turtle.setposition(X,Y) method to draw each line
Homework Part I: Question 3

- Write a program to play a game blackjack (text only – no graphics)
- Overview of the game:
  - First the program selects hands of two cards each, one for the dealer and one for the player
  - If the dealer scores black jack (to be defined), the game is over and the dealer wins.
  - Elif the dealers cards are worth 17 points or more, it is the player's turn.
  - Else: the dealer must draw one card at a time until the dealer's cards are worth 17 or more.
  - If the dealer's cards are worth 22 or more, the game is over and the player wins. If the dealer's hand is 21 or less (and not black Jack), the player does not see the dealer's cards until the game ends
  - Continued on Next Slide
Homework Part I: Question 3 (Continued)

- If the dealer is still in the game, it is now the player's turn.
- If the player has blackjack, he wins and the game is over.
- Else the player looks at his/her hand
- While his/her hand is valued at under 21, he/she has the option of getting an additional card.
  - At any time, if the player's hand is valued at 22 or more the game ends and the player loses.
- After the player stops drawing cards, if the player's hand is more valuable than the dealers' hand, the player wins. Otherwise the dealer wins.
Homework Part I: Question 3  (Continued)

• Some Assumptions
  – Suits (Clubs, Diamonds, Hearts, Spades) will be ignored
  – We will assume that there are only 13 different cards:
    ['A',2,3,4,5,6,7,8,9,10,'J','Q','K']
    • There can never be more than 4 of the same card type in play at any
      time, e.g., if there are 4 10s in play, and a fifth ten is randomly selected,
      a new card must be drawn in its place
  – 'A' can be worth either 1 or 11 points
  – 2-10 are worth their face value (2 is worth 2, 3 is worth 3, etc.)
  – 'J','Q' and 'K' are each worth 10 points
  – Black Jack is a hand consisting of two cards: an 'A' and a card worth 10
    points

• What you need:
  – You should use the random module and lists to write this program
Homework Part 2
(Due Monday Nov 16)

• Read Chapter 6 in Gaddis Book
• Do Module 9
• Do Quiz 9 on NYUClasses