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','big']
  – 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 in 2 weeks
  - 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

- [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)

- [team_name, wins, losses, ties]
  - Print each team name followed by a score equal to \((\text{wins} + \frac{1}{2} \text{ties}) / \text{total_number_of_games}\)

- [item_number, description, price, net_weight, review_score]
  - Sort list of lists in 2 ways – list.sort sorts the list
  - Sort by products by review_scores – put review_score first in list before sorting
  - Sort by price – put price first before sorting

- 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
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:
  - $\text{[ord('1'),ord('A'),ord('a')]} \rightarrow [49, 65, 97]$
  - '1'<'A' $\rightarrow$ True
  - 'A' < 'a' $\rightarrow$ True
  - 'a' < '1' $\rightarrow$ 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 'l' (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 'i' (105) 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 → \([-20, 1, 13, 25.1, 45]\)
  - 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 1
Max and Min

- Like sort, they use the $\leq$ and $\geq$
- Like sort, they use lexicographic order for sequences
- $\text{max('abcd Ghhau &')} \rightarrow 'u'$
- $\text{min('012345')} \rightarrow '0'$
- $\text{max(['$%^&*^%$','abc','XYZ']} \rightarrow 'abc'$
- $\text{max([True,True,False,True,True])} \rightarrow \text{True}$
- $\text{min(['abc',99,True])} \rightarrow \text{error}$
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, as in the file:
  – homework_sort.py
2 Ways to Use Your Score for Sort

1 – Put your score first in the lists to be sorted
   - `list_to_sort.append([score_homework(homework), homework])`
   - `[[0.75, ['ArtHist', 1, 'Min', 1]], [3.0, ['IntroCS', 2, 'Maj', 3]], ...]`
   - sort puts the lowest score first

2 – Use lambda notation (more compact)
   - `homework_list.sort(key=lambda homework: score_homework(homework))`
   - The term after lambda is a variable representing each item in the list
   - A function is listed after the colon
   - The sort compares values of the function for purposes of sorting
   - The output is the sorted list
   - The output of each call to score_homework is not saved as part of the list as in the other version
Text Version of Tic Tac Toe: Slide 1

• The file: 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)

  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']"
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

• http://cs.nyu.edu/courses/spring17/CSCI-UA.0002-004/hw7.html
Homework

- [http://cs.nyu.edu/courses/fall16/CSCI-UA.0002-007/hw7.html](http://cs.nyu.edu/courses/fall16/CSCI-UA.0002-007/hw7.html)