Introduction to:
Computers & Programming:
Strings and Other Sequences in Python
Part I

Adam Meyers
New York University
Outline

• What is a Data Structure?
• What is a Sequence?
• Sequences in Python
• All About Strings
What is a Data Structure?

- A Structure for Storing Data
- Formally defined parts
- Formally defined relations between parts
- Particular algorithms are designed to run with particular data structures
- We will focus on some data structures that are implemented in Python
  - Note that other programming languages may use the same names for different structures
What is a Sequence? What is a Sequence in Python?

- An ordered set of elements (math, e.g., permutations)
- In computer science, there are more than one way for elements to be arranged in a sequence. Python Examples:
  - Lists, Strings, Ranges, Tuples
    - different syntax
    - different functions designed for handling them
  - A string is a sequence of characters
  - Ranges are defined by start and end numbers
  - A list must contain a single type of element
    - It is possible to alter a list, once created
  - Tuples:
    - Can consist of multiple types
    - Cannot be changed once created
Strings in Python

• A String is a sequence consisting of characters
  – Characters also have special properties
• Special syntax allows the identification of subsequences or “slices”
• Special Python functions operate on the data structure “string”
  – testing, searching, changing case, formatting, stripping, splitting, etc.
New Data Type: Character

• Character
  – The smallest part of a string
  – Typically represented by one byte

• Coercion Functions:
  – chr(number) ## Number to ASCII/unicode character
  – ord(character) ## ASCII to number

• We can use these to write our own case changing functions
Using Characters

• Convert Upper Case to Lower Case
  – Let's try to figure this out logically by trying out the type conversions on the previous slide
    • ord('a')
    • ord('A')
    • Use \texttt{chr} to convert numbers to characters
    • Use \texttt{for} loop to convert words
  – Do the reverse: convert Lower Case to Upper Case

• Convert Number Characters 1-9 to corresponding letters using a similar strategy

• Convert whole strings using a \texttt{for loop}
Printing, Characters and Strings

• Special Characters can be part of strings
  – \n = newline character
  – \t = tab character

• Try
  – print('Hello\nWorld')
  – print('Hello\tWorld')

• Unicode Characters
  – Python supports both ASCII and Unicode
  – \uxxxx = 4 digit unicode character
  – Print('\u0770') ## Asian character
  – http://www.utf8-chartable.de/unicode-utf8-table.pl?number=1024&utf8=string-literal
Common Escape Characters

• \ backslash
• ' single quote
• ” double quote
• \n newline
• \r (carriage) return
• \t tab
Other Aspects to \textit{print} Function

- Two named arguments (which occur after all unnamed arguments):
  
  \begin{itemize}
  \item \textit{sep}=\texttt{'string'}
    \begin{itemize}
    \item Default = \texttt{'}\texttt{'} (space)
    \item Identifies the string that occurs between normal arguments
    \end{itemize}
  \item \textit{end}=\texttt{'string'}
    \begin{itemize}
    \item Default = \texttt{'/n'} (newline)
    \item Identifies the string that occurs at the end of print command
    \end{itemize}
  \end{itemize}

- String can be any string, even the empty string " (two single quotes with no space between them)
Indices from Either Direction

• An Index allows access to items in a sequence numbered from 0 to length - 1
  – 'Hello'[0] == 'H'
  – 'Hello'[1] == 'e'
  – ...
  – 'Hello'[4] == 'o'

• An Index allows access to items in a sequence counting in reverse.
  – 'Hello'[-1] == 'o'
  – 'Hello'[-2] == 'l'
  – ...
  – 'Hello'[-5] == 'H'
Slices: Parts of Strings (and some other sequences)

- 'dishes'[0:2] == 'di'
- 'dishes'[4:6] == 'es'
- 'dishes'[:2] == 'di'
- 'dishes'[-2:] == 'es'
- 'dishes'[:] == 'dishes'

SEQUENCE[start:end]

- **start** and **end** can be positive integers from 0 to the length of the sequence or negative integers up to -1 X the string length
- If start is left out, the string starts from the beginning
- If end is left out, the string goes all the way to the end
Example: Regular Plurals in English

• This is for “normal” words, not exceptions
  – Not *sheep, oxen, octopi, aircraft, men, women*, …
  – These could be handled by a separate dictionary
• If final letter is a vowel, add 's'
• Else if final letter is “y”
  – If second-to-last letter is vowel, add 's'
  – Else remove “y” and add “ies”
• Else if final letter is a member of (x, s, z, ch, sh)
  – Add “es”
• Else add 's'
Morphological Rules in Linguistics

• Morphological rules include
  – Rules that add suffixes and/or prefixes
    • noun + -s
  – Other regular sound changes that result in different forms of the same word
    • 'sit' + past → 'sat'

• Irregular morphology
  – Depends on the grammar, one assumes
    • 'sit' → 'sat' is either irregular or a regular instance of an irregular paradigm
  – Some cases would be irregular for all grammars
    • 'go' + past → 'went'
Implementing the Plural Rule in Python

• morphology.py

• Uses the member operator `in`
  – A boolean operator which tests whether an item is a member of a sequence

• Uses another kind of sequence: the list
  – Delimiters = square brackets
  – Members = python objects
  – Separators = commas

• Structure of program: Decision tree using logical operators
Example: Converting Spelled Out Numbers

- What is “two hundred sixty two”? 
  two + hundred + sixty + two
- Convert
  - two → 2, hundred → 100, sixty → 60, two → 2
- Combining numbers in a sequence
  - Lower Higher: multiplication
    - two hundred → 200
  - Higher Lower: addition
    - two hundred sixty → 260
  - Equal Equal: Error
    - two two ???
Class Exercise: Implement Program to convert string numbers to numbers

• We will assume that steps 1 and 2 are done and we will start with input for step 3:
• Example input: ['one','hundred','thirty','five']

1. Convert string to lower case
2. Tokenize string (split at spaces)
3. Given a list of such strings, implement algorithm on previous page
Difficulties with Solution to String → Number Conversion

• Solutions which compare two numbers at a time are difficult when we try to convert large numbers.
• We may need to either:
  – Use more variables to store intermediate solutions
  – Use Recursion
    • When Current < total, we may have to convert the remaining substring to a number before comparing.
    – Process as follows until we only have a single number
      • Process sequences that are one power of ten apart
        – Multiply if first < second, add if first > second
      • Process sequences that are two powers of ten apart
      • Etc.
A Short Discussion of Getting the Right Input

• For example, suppose you want to make sure that the user responds 'yes' or 'no'

  output = "

  while(not (output == 'yes') or (output == 'no')):
      output=(input("Please respond: 'yes' or 'no')

  if (output == 'yes'):
      return(True)

  Else:
      return(False)
An In-Class Problem

• The next Midterm is in 1 ½ weeks.
• Let's do a 20 minute test problem.
  – Everyone should do it individually
  – You should try it out and make sure it works
  – This is a minimum level of proficiency for the next midterm
• Write a program that does the following:
  – Queries the user to provide 2 strings that are the same length. For example, “abcdefghij' and '0123456789'
  – Create a new string that alternates between them, producing 'a0b1c2d3e4f5g6h7i8j9'
  – return that string
• If you have questions, that's OK, but make sure that you really understand what you are doing in the end
• If you can't do this, you need to tell me or email me today