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 collection of elements
    • 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 `chr` to convert numbers to characters
    • Use `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 `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 *print* Function

- Two named arguments (which occur after all unnamed arguments):
  - `sep='string'`
    - Default = ' ' (space)
    - Identifies the string that occurs between normal arguments
  - `end='string'`
    - Default = '/n' (newline)
    - Identifies the string that occurs at the end of print command

- 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 letters are 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 do something more complex, e.g.:
  - Use more variables to store intermediate solutions
  - Go through the number list more than once, e.g.,
    - On the first pass combine all numbers less than 1000
    - On the second pass multiply instances of consecutive pairs of numbers such than number1 < number2
    - Add the remaining numbers together
A Short Discussion of Getting the Right Input

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

```python
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

• 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 goal level of proficiency for this time in the term

• 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
Several Slides Listing String Functions

- Go to example-string-functions.py
- Also Listed on the next few slides
- I will do a quick overview, but will not really focus on these until the next talk about strings
- These all take the form:
  string.functioname(arguments)
- Examples,
  - 'abc'.islower()
    - Evaluates as True
  - 'Hello World'.center(20,'*')
    - Evaluates as '****Hello World*****'
string.functions(): Case/Format

- Case-Changing Functions
  - s.captitalize()  --- s[0] only
  - s.title()  – similar except capital after space
  - s.lower(), s.upper(), s.swapcase()

- Format Functions
  - s.center(LENGTH, ch)  – e.g., *** string ***
  - s.ljust(length, ch), s.rjust(length, ch)  – similar
  - s.format(vars)
    - '{whose} {thing} is nice'.format(pet = 'John\'s', thing = 'code')
string.function(): Tests and Search

- **Testing (Boolean)**
  - `endswith(suffix)`
  - `startswith(prefix)`
  - `isalnum()`, `isalpha()`, `isdigit()`, `isnumeric()`, `isidentifier()`, `islower()`, `isupper`, `istitle()`, `isprintable()`, `isspace()`

- **Search functions**
  - `find(substring)`, `rfind(substring)`
    - return index or -1
  - `index(substring)`, `rindex(substring)`
    - return index or error
string.functions(): Stripping off Characters

• Stripping Functions
  – Remove unwanted characters from edges of string
• s.strip(optional_arg)
  – If left out all white space characters are stripped
    • (tab, space, newline, …)
  – Otherwise all characters in optional_arg string
• s.lstrip and s.rstrip (left or right only)
Split and Partition functions

• Partition
  – s.partition(arg), s.split(arg)
  – create a list of substrings, partitioned by arg

• Split **** Useful for Homework ****
  – Example: “five hundred thirty”.split(' ') → ['five','hundred','thirty']
  – Split does not include the separators, but partition does
    • Try “five hundred thirty”.partition(' ')

• Rightward Versions
  – rpartition and rsplit variants: search for separators from right side
Summary I

• Sequences are Data Structures in which items are combined together in a pre-described order.
• Sequences share certain properties in Python, but many also have special functions and operators specific to them.
• We have so far focused on strings and we will continue to do so next time.
• Strings are sequences of Characters.
• Strings are important for the print function, as well as other processing involving text.
Summary II

• String manipulation involves
  – slicing and concatenating strings
  – converting characters to other characters
  – looping through sequences and making regular changes

• String manipulation is important for several applications
  – Applications involving linguistics: morphology, spell-checking, information extraction, machine translation, search, etc.
Homework Slide 1: Due in 2 classes

1. Read ½ of Chapter 6 in Donaldson Book

2. Secret Code: write a function that takes a string as an argument and prints out a new string consisting of numbers divided by spaces.
   - These numbers should be derived by using the `ord` function on each character.
   - It is suggested that you use a `for` loop to solve the problem. The output could begin as an empty string and be built up to the final solution.
   - For example 'cat' should be printed as: 99 97 116
3. Download the functions for converting number strings like 'four hundred and fifty three” into numbers like 453. (It does not matter which version you use.)

- Listed on the class website as materials for classes 14 & 15
- Write parts 1 and 2 of that code and incorporate the function we wrote in class or use the corrected one, it doesn't matter). Use the string.split function
Homework Slide 4

• 3 (continued)
  – Write two additional functions
    • `string_multiply`
      – Takes two strings as input, converts them to numbers and then multiplies them together (and returns the resulting value)
    • `string_add`
      – Takes two strings as input, convert them to numbers and then multiplies them together (and returns the resulting value)
Homework Grading Criteria

• Does the program work?

• Does it solve the problem described in the question?
  – If the question asks to print something out, does your program print it?
    If it asks to return something, do you return it?

• Is your code well written and clear?
  – Are the variable names and function names understandable?
  – Do you have adequate comments?
  – Do you encapsulate functions for reuse that clarify what you are doing? For example, a function “make_upper_case” is clearer than a loop that adds a certain number to a character code.

• Did you do anything clever?
  – Did you solve a more complex version of the problem?
  – Is your code elegant?
  – Etc.