Introduction to:
Computers & Programming:
Using Patterns with Strings
For Search and Modification

Adam Meyers
New York University
Outline

• Eliza – a famous AI program using patterns in strings
• What is a string pattern and why would we want to use it?
• What are regular expressions?
• Using regular Expressions in Python
Eliza: An Application of String Manipulation

• A famous program derived by matching patterns in string and altering sentences based on these patterns (re-implemented many times all over the internet).

• I haven't found a version for Python 3
  – But I am working on it

• It matches strings in your sentences and feeds them back to you in different forms, trying to simulate a psychiatrist

• http://www-ai.ijs.si/eliza-cgi-bin/eliza_script
Eliza 2

• Joseph Weizenbaum between 1964 to 1966
• The Turing Test:
  – If a program that passes the Turing Test
    • A human being will not be able to tell the difference between the output of the program and the response of a human being
• Elisa actually fooled some people
• Even people who knew that it was a program claimed that communicating with it was therapeutic and treated it as if it was a therapist
String Pattern Matching

• We have used loops to find patterns
  – For example, the plural program
• However, *regular expressions* are another way.
• Let's compare two versions of the plural program
  – The original one using loops
  – A new one using regular expressions
• Regular expressions are used for a variety of purposes in Computer Science
What is a Regular expression?

- A regular expression is a compact way to represent a fairly complex pattern.
- Examples “|” used to represent “or”
  - 'Dog|dog' means 'dog' or 'Dog'
- [ ] are used to list alternative characters
  - '[Dd]og' means 'dog' or 'Dog'
    - Inside [], ^ means not
- A period . is used to mean any character
- $ means end of string and ^ means beginning of string (note ambiguity for ^)
- pattern* – means 0 or more instances of pattern
- pattern+ – means 1 or more instances of pattern
- There are more conventions which we will not discuss
The Mathematics of Regular Expressions

• Regular expressions can be used to represent the set of strings that they match.

• Examples:
  – [AB]* – represents the empty string and all combinations of A and B
  – (AB)* represents: '', 'AB', 'ABAB', 'ABABAB', ...
  – ([^A]B)* represents sequences of one non A followed by B, e.g., XB, XBBB, XBCBRB, ...
Plural Rule

- `'([sxz]|[cs]h)$' matches one or two characters at the end of a string ($)`
  - s or x or z or ch or sh

- `'^[aeiou]y$' matches a non-vowel preceding a y
  - The bracketed part means "not" (\^) a member of the set {a,e,i,o,u}
  - This precedes a y and the end of string indicator $`

- Python has several functions using regular expressions, but we will focus on: re.search
Regular Expressions are Used for Many Computer Science Applications

• They are part of almost every scripting language (perl, sed, ruby, …) and some other languages as well.

• They are used to manipulate and search through text.

• They are used by various command line programs, e.g., “grep”
  – grep -e 'turtle.*turtle' *.py
A More Complicated Application:

• Approximating syllable boundaries for voice generation

• One version written using loops and one version written with regular expressions

• In python, the search function
  – Returns a search object
  – That object has 3 slots
    • search.start() → beginning of matching slice
    • search.end() → end of matching slice
    • search.group(0) → the matching slice
The Loop Version

![](image)

• Currently, a little more accurate than the regexp version
• Uses functions: is_vowel, is_consonant
• Assembles syllables one at a time, dealing with exceptions explicitly.
• Stores partial results along the way
• Records whether the syllable being assembled has a vowel yet (necessary condition for syllablehood).
The Regular Expression Version

• Uses the disjunction of 3 patterns (probably needs a few more)
  – Pattern1 or Pattern2 or Pattern3

• Finds the first pattern to match
  – Assumes that anything skipped over is part of the newest syllable

• Adds the matching syllable.

• Uses *While* loop that ends when no more patterns are found or we reach the end of the word
Regular Expression Definition Repeated

- A regular expression is a compact way to represent a fairly complex pattern.
- Examples “|” used to represent “or”
  - 'Dog|dog' means 'dog' or 'Dog'
- [  ] are used to list alternative characters
  - '[Dd]og' means 'dog' or 'Dog'
  - Inside [], ^ means not
- A period . is used to mean any character
- $ means end of string and ^ means beginning of string (note ambiguity for ^)
- pattern* – means 0 or more instances of pattern
- pattern+ – means 1 or more instances of pattern
- There are more conventions which we will not discuss
More Regular Expressions

• Character? – indicates that the character is optional
  – Mar[iy]a? – Mary or Maria or Marya (the a is optional)

• (expression){number} – that many times
  – '(ho){4}' – matches 'hohohoho'

• More info at:
  http://docs.python.org/dev/library/re.html
Regular Expression Examples

• '(ho)+' – one or more instances of 'ho'
• '(ho)*' – zero or more instances of 'ho'
  – Compare
    • re.search('(ho)+','The laugh sounded like 'hohoho'”
    • re.search('(ho)+','The laugh sounded like 'hahahoa”
    • The same searches with '(ho)*'

• ^ beginning of strings:
  – '^s[bcpt][rl]' – strings beginning with:
    • sbr, sbl, scr, scl, spr, spl, str, stl – except for the last one, possible 3 letter consonant strings in English
More Examples

• $ – the end of strings
  – [.?!]$ – period, question mark or exclamation mark at the end of a string

• . – any character
  – .*[.?!]$ – any string that ends in a period, question mark or exclamation mark
  – [ABCDEFGHIJKLMNOPQRSTUVWXYZ].*
    • A string beginning with a capital letter
Summary

- Regular expressions provide a compact way to do complex string matching (and string manipulation).

- A search with a single regular expression is equivalent to several different searches with simple strings combined with an 'Or'.
  - 'Mar[yi]a?' is equivalent to: Mary or Maria or Marya

- Useful for any programs involving matching and manipulating strings.
  - Computational linguistics, Text formatting, Data Mining, Web Development, etc.
Homework

• Write a function that add *er* to the end of a word with the following modifications:
  – If the last letter of the word is *y*, change the last letter to an *i* before adding *er*
  – Double the last letter before adding *er* if:
    • the last letter is *b,d,g,l,m,n,p,r,s or t*
    • and the second to last letter is not the same as the last letter
  – If the last letter of the word is *e*, just add an *r* instead of *er*
• Assume that there is a single parameter for your function, the word that you want to add *er* to.
• Examples: *bid* → *bidder*, *eat* → *eater*, *sin* → *sinner*, *silly* → *sillier*, *sprint* → *sprinter*, *bike* → *biker*
• Grading criteria
  – Does your program work?
  – Does it solve the problem?
  – Do you use regular expressions?
  – Is your code clearly written?
  – Is it elegant or clever?
  – Do you figure out ways to handle exceptions?
    • Please make a note using a comment

• You can ask any questions by email or in person and I will give you hints if you are having trouble. Some hints on next slide.
What you need to remember

- `re.search(pattern, string)`
  - Will return nothing if no pattern is found
  - This is treated like False, if used as a boolean expression

- If a pattern is found, a match object will be returned.

- There are 3 slots of that object:
  - `pattern.group(0)` is the matched substring
  - `pattern.start()` is the starting position of that substring
  - `pattern.end()` is the ending position
Homework Slide 4
Regular Expression Clues

- You need to use the square brackets to indicate a list of possible letters.
  - \[abcd\] means a or b or c or d
- You need to use $ to represent the end of the word.
- You need to create a new string by slicing and appending.