Using Regular Expressions
What are “Regular Expressions?”

- Power text-matching tools
- Let you search strings for patterns; manipulate or chop up strings based on patterns
  - Patterns can be based on “normal” characters (e.g., the alphabet)
  - Can also include “special” symbols that give more expressive power
    - Match only numbers
    - Match only letters
    - Require that a string have zero or more (or one or more, or ...) occurrences of a given pattern before it counts as a match
    - Require that a string have a certain pattern at the beginning, or the end, of it in order for it to match
Understanding Regular Expressions

- You define a *pattern* string that can contain normal characters as well as characters that represent special conditions like the ones on the earlier slide.
- Test this against a *target* string to determine if the pattern *matches* that string.
  - Meaning: that the pattern, including any special conditions, exists in that target string.
  - Normal characters must match exactly.
  - Special characters let you make the match more flexible.
Regular Expressions in Python

- **Use the “re” module:**
  - import re

- **Most important methods:**
  - search(pattern, string)
    - Tests to see if the pattern matches anywhere in the target string; returns a MatchObject corresponding to the first one found
  - split(pattern, string)
    - Breaks apart the string by finding occurrences of the pattern (in other words, treating the pattern as the delimiter). Matched pattern elements are *not* returned in the strings
import re
str = “Hello, Allan”
match = re.search(“ll”, str)

• match.start() - returns 2, the index of the start of where the pattern occurs
• match.end() - returns 4, the index of the end of where the pattern occurs

• To search for the next occurrence, one easy way is use the returned indices to create a substring of the original string that excludes the matched part:
  • substr = str[4:]
  • substr now refers to a string containing all the characters after index 4 (“o, Allan”) which can be searched again to find the next occurrence of the pattern
More Examples

\[\text{str} = \text{“Hello, Allan”}\]
\[\text{re.split(“ll”, str)} \text{ returns [‘He’, ‘o, A’, ‘an’]}\]
Special Characters

- Backslashes are frequently used in regular expression patterns
- ...but the backslash character itself has special meaning in Python, so normally you’d have to put another backslash in front of it
  - Results in really unreadable patterns!
- Alternative: use Python “raw” strings:
  - Preface string with lowercase \r
  - Lets you get away without the extra backslash
  - Example: r’\w\w’
### Special Characters

<table>
<thead>
<tr>
<th>Special Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>. (a single period)</td>
<td>Matches any character except a newline</td>
</tr>
<tr>
<td>^ or \A</td>
<td>Limits the match to occur at the beginning of the string</td>
</tr>
<tr>
<td>$ or \Z</td>
<td>Limits the match to occur at the end of the string</td>
</tr>
<tr>
<td>* (asterisk)</td>
<td>Matches zero or more of the preceding character. Example: s* means zero or more of the letter “s”</td>
</tr>
<tr>
<td>+ (plus)</td>
<td>Matches one or more of the preceding character</td>
</tr>
<tr>
<td>[ ]</td>
<td>Defines a character set. For example, to match against any of the vowels, use [aeiou]. To match against any number of numerals, use [0123456789-]</td>
</tr>
<tr>
<td>\s</td>
<td>Matches any whitespace character (space, tab, newline)</td>
</tr>
<tr>
<td>\n</td>
<td>Matches newline</td>
</tr>
<tr>
<td>\w</td>
<td>Matches any alphabetic or numeric character. Equivalent to [a-zA-Z0-9]</td>
</tr>
</tbody>
</table>
Resources, Tutorials, and Examples

- http://www.amk.ca/python/howto/regex/
- http://diveintopython.org/regular_expressions/index.html
- http://www.regular-expressions.info/python.html