Regular Expressions & List Comprehension

The last of the Python Crash Course lectures

http://eddiema.ca/py

Eddie Ma

I

Regular Expressions?

- A regular expression is a **string literal**.
  - “Yay!”
- We use them as **patterns** to match other strings.

2

String Searching

- The **pattern** "oak" will match the following **strings**: "I eat oak", "oakville", "oak tree".
- When a pattern matches, a regular expression match object is returned.
- Internally called a "_sre.SRE_Match" object.
- The **None** object is returned otherwise.

3

literally searching

```
>>> import re
>>> patternString = “oak”
>>> patternObject = re.compile(patternString)
>>> patternObject.search(“oakville”)
<_sre.SRE_Match object at 0x362c98>
>>> print patternObject.search(“cows”)
None
```


4

More meta, less literal

- We can specify patterns with metacharacters that match degenerately
- The pattern "[abc]" matches these strings "a", "b", "c"
- The '[' and ']' characters are used to specify a regular expression **class**.
- Use "-" specifies a range in a class: "[a-c]".

5

matching with class

```
>>> import re
>>> pattern = “[0-9]”
>>> sentence = “Ten is 10 in base ten.”
>>> matchObject = re.compile(pattern).search(sentence)
>>> print matchObject
<_sre.SRE_Match object at 0x362d40>
```


6

More Metacharacters

- We include these characters in a pattern string to mean different things.
- The period "." matches any single character once.
- The wildcard "?" matches zero or one of an item.
- The Kleene plus "+" matches one or more items.
- The Kleene star "*" matches zero or more items.
- The parentheses "(" and ")" encloses a literal.

7

Metachar Examples

Characters	Pattern	Matching String	Another Match	Non-Matching
.	eleph.nt	elephznt	eleph.nt	elephoont
?	neighbou?r	neighbour	neighbor	neighbouur
+	quantu+m	quantuuuum	quantum	quantm
*	joh*n	johhhhn	jon	jn
(,), *	thes(is)*	thesisisisisis	thes	theses
[,], +	ID[0-9]+	ID0329	ID427	IDten

8

Extracting

- Now that we know how to get a match object, let's crack one open to extract useful information.

9

The first match

```
>>> import re
>>> pattern = “(their|there|they’re)”
>>> string = “they’re cat and there dog are over their.”
>>> firstMatch = re.compile(pattern).search(string)
>>> print firstMatch.start()
0
>>> print firstMatch.end()
7
>>> print firstMatch.group()
they’re
```


10

Extracting all matches

```
>>> import re
>>> pattern = “(their|there|they’re)”
>>> string = “they’re cat and there dog are over their.”
>>> matches = re.finditer(pattern, string)
>>> for m in matches:
...     print m.start(), m.end(), m.group()
0 7 they’re
16 21 there
35 40 their
```

More on re.finditer()

- re.finditer() actually returns an iterable object -- an object that you can use a for loop on, but one that isn't really a sequence.
  - That means it can't be indexed :(
- A related function is re.findall() which returns an actual list of all the matching strings (without start() and end() indices).

12

More metachars

- The pipe "|" enclosed inside "(" and ")" separates matching options.
- The caret as the first item "^" **inside a class** means "nothing in this class" (escape it with a backslash "\").
- The caret "^" in a pattern matches start of the string.
- The dollar-sign "$" in a matches the end.
- The "{" and "}" enclose an integer for how many times to match an item.
  - better explained in an example...

13

Metachar examples...

Characters	Pattern	Matching String	Another Match	Non-Matching
(, |,)	(Eddie|John)	Eddie will.	John will.	Fred will.
[, ^,]	gr[^ae]y	groy	grpy	grey
^	^Pop	Popsicle	Popstand	A Pop
$	halves$	two halves	eight halves	2 halves of 3
{, }	a{4}	aaaa	whaaaat	caat
[,], {, }	th[ae]{3}n	thaaen	theeen	than
(,), {, }	th(ou|aw){2}	thouaw	thawou	thaw

14

...match()

- re.compile("^" + pattern).search(string) is the same as
- re.compile(pattern).match(string)
- i.e. match only looks at the start of the string
- returns an _sre.SRE_Match object

15

re.split()

- re.split(pattern, string, count?)
  - breaks a string apart at the pattern
  - similar to string.split(pattern)
  - count is an optional argument indicating the maximum number of times to break
  - returns a list of strings

16

re.sub()

- re.sub(pattern, replacement, original, count?)
  - Substitute a **replacement** at each **pattern** in the **original** string.
  - Count is optional: the number of substitutions going left to right before stopping.
  - Returns a modified string.

17

Shorthands for classes

Pattern	Equivalent Class	What?
\d	[0-9]	digits
\D	[^0-9]	not digits
\s	[\t\r\n\f\v]	white space
\S	[^ \t\r\n\f\v]	not space
\w	[a-zA-Z0-9_]	alphanumeric
\W	[^a-zA-Z0-9_]	not alphanum

Low Budget Halftime Show...

:)

19

Low Budget Halftime Show...

:D

20

Low Budget Halftime Show...

:O

21

Low Budget Halftime Show...

XD

22

Low Budget Halftime Show...

List Comprehension!

23

List Comprehension

- List comprehension is a suite of syntactic sugar useful for dealing with sequences or Python lists.
- With this, we can perform the filter, map and fold operations that are familiar to functional programming languages.

24

Trivial Example

- The list [0, 1, 2, 3, 4, 5] can be created with the expression:
- “[x for x in xrange(6)]”
- Where xrange is a function that generates the integers from 0 until before 6.

25

xrange()

- The xrange() function is great for this kind of thing.
- xrange(start, before, increment)
  - Start is the integer to start on (inclusive)
  - Before is the integer to stop before (exclusive)
  - Increment is the size of the step between integers generated

26

General Syntax

- *[resultant for element in original if condition]*
- Where resultant is an element that enters the final list
- Element is a raw item from the original list
- Original is the original list
- Condition is some optional constraints for what to consider from the original list
- Naturally a filter!

27

Simple filter examples...

```
>>> from math import pi
>>> from math import e
>>> constants = [22, 13, 3.1, 3.53, 2, 1, 3345, 8, e, 0, 56.13, 3231, 1, pi, -1]
>>> [Eye for Eye in constants if Eye % 2 == 0]
[22, 2, 8, 0]
>>> [Jay for Jay in constants if Jay % 1 == 0]
[22, 13, 2, 1, 3345, 8, 0, 3231, 1, -1]
>>> [Kay for Kay in constants if Kay < 30 if Kay > 20]
[22]
```


28

Simple Map Example...

```
>>> from math import pi
>>> from math import e
>>> constants = [22, 13, 3.1, 3.53, 2, 1, 3345, 8, e, 0, 56.13, 3231, 1, pi, -1]
>>> rooted = [Emm ** (1.0/2.0) for Emm in constants if Emm > 0]
>>> print rooted
[4.6904157598234297, 3.6055512754639891, ...
>>> squished = [log(Enn) for Enn in constants if Enn > 0]
>>> print squished
[3.0910424533583161, 2.5649493574615367, ...
```


29

Operating on Two Lists

- Pairwise means that each element in each list are paired together so that their indices are the same
- Crosswise means that each element in one list is paired with every element in the other list

30

Continuing...

```
>>> product = [Ooo * Pee for Ooo, Pee in zip(rooted, squished)]
>>> print len(product)
13
>>> combo = [Ooo + Pee for Ooo in rooted for Pee in squished]
>>> print len(combo)
169
```


31

Functions are objects

- Before going to folding sequences--
- Everything in Python is an object.
- Functions are objects too and can be passed like variables--
- You simply drop the parentheses!
- Example:
  - blah = len([1, 2, 3])
  - “blah” gets the value 3 for the length of the list
  - blah = len
  - blah([1, 2, 3])
  - “blah” becomes a reference for the length function

32

Folding Lists

- In Python, the reduce function is used.

33

reduce() example...

```
>>> morbid = ["The ", "cat ", "ate ", "a ", "mouse."]
>>> def catcat(x, y):
...     return x + y
reduce(catcat, morbid)
The cat ate a mouse.
```


34

filter(), map()

- Along with list comprehension, the functions filter() and map() also exist.
- semantics:
  - filter(function, sequence)
  - map(functions, sequence, ...)
    - map takes additional sequences and operates on them pairwise-like.

35

You have just been introduced to Python!
Go home!

Halftime redux...

:) :D :O XD