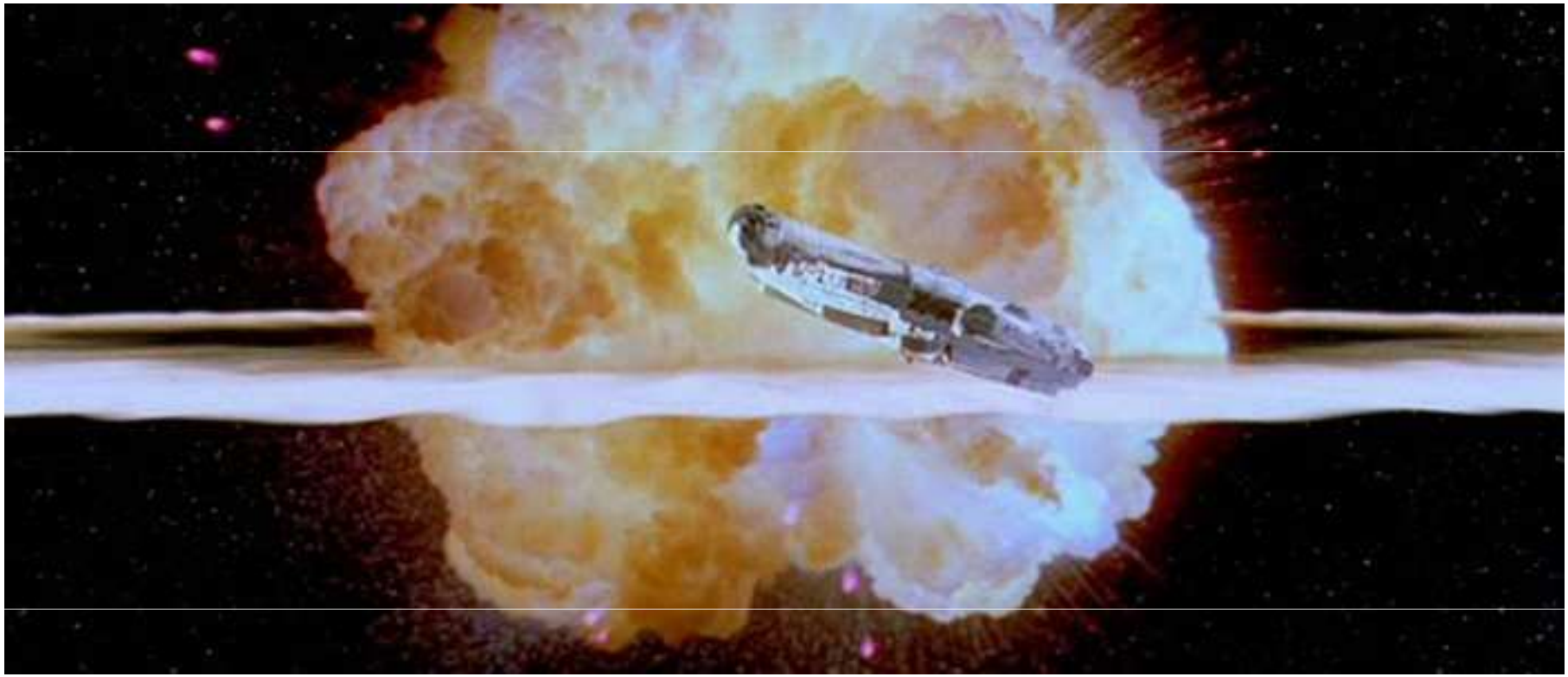
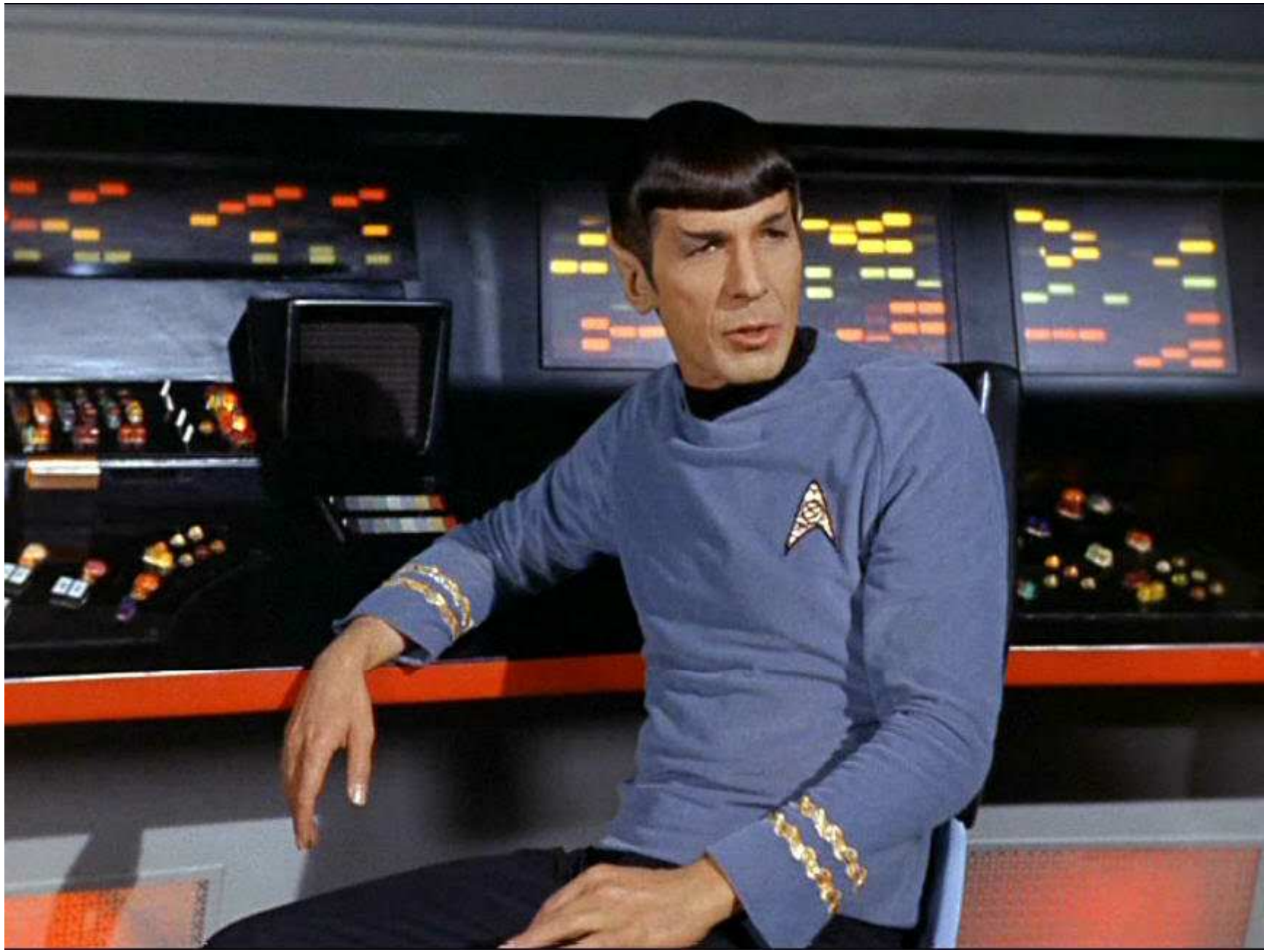
Regular Expressions

Introduction






VENKMAN

Your mission: read 20-30 files containing several hundred measurements each of background evil levels (in millivaders) and convert them to a uniform format for further processing

Your mission: read 20-30 files containing several hundred measurements each of residual evil levels and convert them to a uniform format for further processing

Each reading has a site name, the date, and the level of background evil (in millivaders)

Your mission: read 20-30 files containing several hundred measurements each of residual evil levels and convert them to a uniform format for further processing

Each reading has a site name, the date, and the level of background evil (in millivaders)

Some use tabs to separate fields, others use commas

Your mission: read 20-30 files containing several hundred measurements each of residual evil levels and convert them to a uniform format for further processing

Each reading has a site name, the date, and the level of background evil (in millivaders)

Some use tabs to separate fields, others use commas

Dates are written in several different styles

Notebook #1

```
Site     Date     Evil (millivaders)
----     ----     ------------------
Baker 1  2009-11-17       1223.0
Baker 1  2010-06-24       1122.7
Baker 2  2009-07-24       2819.0
Baker 2  2010-08-25       2971.6
Baker 1  2011-01-05       1410.0
Baker 2  2010-09-04       4671.6
   ⋮        ⋮              ⋮
```

Notebook #1

```
Site     Date     Evil (millivaders)
----     ----     ------------------
Baker 1  2009-11-17       1223.0
Baker 1  2010-06-24       1122.7
Baker 2  2009-07-24       2819.0
Baker 2  2010-08-25       2971.6
Baker 1  2011-01-05       1410.0
Baker 2  2010-09-04       4671.6
  ⋮          ⋮               ⋮
```

single tab as separator

Notebook #1

```
Site     Date     Evil (millivaders)
----     ----     ------------------
Baker 1  2009-11-17       1223.0
Baker 1  2010-06-24       1122.7
Baker 2  2009-07-24       2819.0
Baker 2  2010-08-25       2971.6
Baker 1  2011-01-05       1410.0
Baker 2  2010-09-04       4671.6
   ⋮        ⋮              ⋮
```

spaces in site names

Notebook #1

```
Site     Date     Evil (millivaders)
----     ----     ------------------
Baker 1  2009-11-17       1223.0
Baker 1  2010-06-24       1122.7
Baker 2  2009-07-24       2819.0
Baker 2  2010-08-25       2971.6
Baker 1  2011-01-05       1410.0
Baker 2  2010-09-04       4671.6
   ⋮        ⋮                ⋮
```

dates in international standard format (YYYY-MM-DD)

Notebook #2

```
Site/Date/Evil
Davison/May 22, 2010/1721.3
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
  ⋮         ⋮       ⋮
```

Notebook #2

```
Site/Date/Evil
Davison/May 22, 2010/1721.3
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
  ⋮         ⋮       ⋮
```

slashes as separators

Notebook #2

```
Site/Date/Evil
Davison/May 22, 2010/1721.3
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
  ⋮          ⋮        ⋮
```

↑

site names don't appear to have spaces

Notebook #2

```
Site/Date/Evil
Davison/May 22, 2010/1721.3
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
  ⋮          ⋮        ⋮
```

month names and day numbers of varying length

Solution: regular expressions

Solution: regular expressions

A pattern that strings can match

Solution: regular expressions

A pattern that strings can match

Like `'*.txt'` matches filenames ending in `'.txt'`

Solution: regular expressions

A pattern that strings can match

Like `'*.txt'` matches filenames ending in `'.txt'`

Warning: notation is ugly

Solution: regular expressions

A pattern that strings can match

Like `'*.txt'` matches filenames ending in `'.txt'`

Warning: notation is ugly

Writing patterns for strings *as* strings...

Solution: regular expressions

A pattern that strings can match

Like `'*.txt'` matches filenames ending in `'.txt'`

Warning: notation is ugly

Writing patterns for strings *as* strings...

...using only the symbols on the keyboard (instead of inventing new symbols like mathematicians do)

```
# read the first half-dozen data records from two files
readings = []
for filename in ('data-1.txt', 'data-2.txt'):
  lines = open(filename, 'r').read().strip().split('\n')
  readings += lines[2:8]

for r in readings:
  print r
```

```
Baker 1   2009-11-17     1223.0
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
Baker 2   2010-08-25     2971.6
Baker 1   2011-01-05     1410.0
Baker 2   2010-09-04     4671.6
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
```

```
# select readings in month '06'
for r in readings:
  if '06' in r:
    print r
```

Baker 1 2010-06-24 1122.7

```
# select readings in month '06' or month '07'
for r in readings:
  if ('06' in r) or ('07' in r):
    print r
```

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

```
# what about readings in month '05'? (shouldn't be any)
for r in readings:
  if ('05' in r):
    print r
```

```
Baker 1   2011-01-05     1410.0
```

```
# what about readings in month '05'? (shouldn't be any)
for r in readings:
  if ('05' in r):
    print r
```

```
Baker 1   2011-01-05     1410.0
```

"in string" is a dangerously blunt tool

```
# try using regular expressions instead
import re
for r in readings:
  if re.search('06', r):
    print r
```

```
Baker 1   2010-06-24    1122.7
```

```
# try using regular expressions instead
import re
for r in readings:
  if re.search('06', r):
    print r
```

Baker 1 2010-06-24 1122.7

not much of an improvement so far...

```
# find records with '06' or '07' in one search
import re
for r in readings:
  if re.search('06|07', r):
    print r
```

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

```
# find records with '06' or '07' in one search
import re
for r in readings:
  if re.search('06|07', r):
    print r
```

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

first argument is

the pattern to search for

```
# find records with '06' or '07' in one search
import re
for r in readings:
  if re.search('06|07', r):
    print r
```

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

first argument is

the pattern to search for

written as a string

```
# find records with '06' or '07' in one search
import re
for r in readings:
  if re.search('06|07', r):
    print r
```

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

second argument is the data to search *in*

```
# find records with '06' or '07' in one search
import re
for r in readings:
  if re.search('06|07', r):
    print r
```

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

second argument is the data to search *in*

reversing these is a common mistake (and hard to track down)

```
# find records with '06' or '07' in one search
import re
for r in readings:
  if re.search('06|07', r):
    print r
```

vertical bar '|' means OR

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

```
# find records with '06' or '07' in one search
import re
for r in readings:
  if re.search('06|07', r):
    print r
```

```
Baker 1   2010-06-24     1122.7
Baker 2   2009-07-24     2819.0
```

vertical bar '|' means OR

match either what's on the left, *or* what's on the right, in a single search

```
# we're going to be trying out a lot of patterns,
# so let's write a function
def show_matches(pattern, strings):
  for s in strings:
    if re.search(pattern, s):
      print '**', s
    else:
      print '  ', s
```

```
# test our function right away
show_matches('06|07', readings)
```

```
   Baker 1  2009-11-17  1223.0
** Baker 1  2010-06-24  1122.7
** Baker 2  2009-07-24  2819.0
   Baker 2  2010-08-25  2971.6
   Baker 1  2011-01-05  1410.0
   Baker 2  2010-09-04  4671.6
   Davison/May 23, 2010/1724.7
   Pertwee/May 24, 2010/2103.8
   Davison/June 19, 2010/1731.9
   Davison/July 6, 2010/2010.7
   Pertwee/Aug 4, 2010/1731.3
   Pertwee/Sept 3, 2010/4981.0
```

```
# why doesn't this work?
show_matches('06|7', readings)
```

```
** Baker 1  2009-11-17  1223.0
** Baker 1  2010-06-24  1122.7
** Baker 2  2009-07-24  2819.0
** Baker 2  2010-08-25  2971.6
   Baker 1  2011-01-05  1410.0
** Baker 2  2010-09-04  4671.6
** Davison/May 23, 2010/1724.7
   Pertwee/May 24, 2010/2103.8
** Davison/June 19, 2010/1731.9
** Davison/July 6, 2010/2010.7
** Pertwee/Aug 4, 2010/1731.3
   Pertwee/Sept 3, 2010/4981.0
```

In mathematics, "ab+c" means (a×b) + c

In mathematics, "ab+c" means (a×b) + c

Multiplication is *implied*, and has *higher precedence*

In mathematics, "ab+c" means (a×b) + c

Multiplication is *implied*, and has *higher precedence*

To force the other meaning, write "a(b+c)"

In mathematics, "ab+c" means (a×b) + c

Multiplication is *implied*, and has *higher precedence*

To force the other meaning, write "a(b+c)"

In regular expressions, "06|7" means '06' or '7'

In mathematics, "ab+c" means (a×b) + c

Multiplication is *implied*, and has *higher precedence*

To force the other meaning, write "a(b+c)"

In regular expressions, "06|7" means '06' or '7'

...and there are a lot of 7's in our file

In mathematics, "ab+c" means (a×b) + c

Multiplication is *implied*, and has *higher precedence*

To force the other meaning, write "a(b+c)"

In regular expressions, "06|7" means '06' or '7'

...and there are a lot of 7's in our file

To force the other meaning, parenthesize "0(6|7)"

In mathematics, "ab+c" means (a×b) + c

Multiplication is *implied*, and has *higher precedence*

To force the other meaning, write "a(b+c)"

In regular expressions, "06|7" means '06' or '7'

...and there are a lot of 7's in our file

To force the other meaning, parenthesize "0(6|7)"

But "06|07" is more readable anyway

```
# still matching days when we want to match months
show_matches('05', readings)
```

```
   Baker 1  2009-11-17  1223.0
   Baker 1  2010-06-24  1122.7
   Baker 2  2009-07-24  2819.0
   Baker 2  2010-08-25  2971.6
** Baker 1  2011-01-05  1410.0
   Baker 2  2010-09-04  4671.6
   Davison/May 23, 2010/1724.7
   Pertwee/May 24, 2010/2103.8
   Davison/June 19, 2010/1731.9
   Davison/July 6, 2010/2010.7
   Pertwee/Aug 4, 2010/1731.3
   Pertwee/Sept 3, 2010/4981.0
```

```
# could rely on context
show_matches('-05-', readings)
```

```
Baker 1  2009-11-17  1223.0
Baker 1  2010-06-24  1122.7
Baker 2  2009-07-24  2819.0
Baker 2  2010-08-25  2971.6
Baker 1  2011-01-05  1410.0
Baker 2  2010-09-04  4671.6
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
```

```
# could rely on context
show_matches('-05-', readings)
```

month has '-' before and after

```
Baker 1  2009-11-17  1223.0
Baker 1  2010-06-24  1122.7
Baker 2  2009-07-24  2819.0
Baker 2  2010-08-25  2971.6
Baker 1  2011-01-05  1410.0
Baker 2  2010-09-04  4671.6
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
```

```
# could rely on context
show_matches('-05-', readings)
```

```
Baker 1  2009-11-17  1223.0
Baker 1  2010-06-24  1122.7
Baker 2  2009-07-24  2819.0
Baker 2  2010-08-25  2971.6
Baker 1  2011-01-05  1410.0
Baker 2  2010-09-04  4671.6
Davison/May 23, 2010/1724.7
Pertwee/May 24, 2010/2103.8
Davison/June 19, 2010/1731.9
Davison/July 6, 2010/2010.7
Pertwee/Aug 4, 2010/1731.3
Pertwee/Sept 3, 2010/4981.0
```

so no matches

Matching is enough: we need to *extract* data

Matching is enough: we need to *extract* data

When a regular expression matches, the library remembers what matched against every parenthesized sub-expression

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

```
2009
```

pattern to match years is in parentheses

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

first record from data

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

first record from data

(remember '\t' is a tab)

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

`re.search` returns a *match object* if a match is found

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

`re.search` returns a *match object* if a match is found

returns `None` if there is no match

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

`match.group(k)` returns the text that matched the kth sub-expression in the regular expression

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

`match.group(k)` returns the text that matched the kth sub-expression in the regular expression

k goes from 1 to N (number of matches), not 0 to N-1

```
# extract the year from the match
match = re.search('(2009|2010|2011)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1)
```

2009

`match.group(k)` returns the text that matched the kth sub-expression in the regular expression

k goes from 1 to N (number of matches), not 0 to N-1

because `match.group(0)` is *all* the text that was matched

Regular expression to match month would be

"(01|02|03|04|05|06|07|08|09|10|11|12)"

Regular expression to match month would be

"(01|02|03|04|05|06|07|08|09|10|11|12)"

Expression to match day would be three times longer

Regular expression to match month would be

"(01|02|03|04|05|06|07|08|09|10|11|12)"

Expression to match day would be three times longer

Use '.' (period) to match *any single character*

Regular expression to match month would be "(01|02|03|04|05|06|07|08|09|10|11|12)"

Expression to match day would be three times longer

Use '.' (period) to match *any single character*

So '....-..-..' matches four characters, a dash, two more characters, another dash, and two more characters

Regular expression to match month would be "(01|02|03|04|05|06|07|08|09|10|11|12)"

Expression to match day would be three times longer

Use '.' (period) to match *any single character*

So '....-..-..' matches four characters, a dash, two more characters, another dash, and two more characters

And '(....)-(..)-(..)' remembers those matches individually

```
# match and extract YYYY-MM-DD date formats
match = re.search('(....)-(..)-(..)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1), match.group(2), match.group(3)
```

2009 11 17

```
# match and extract YYYY-MM-DD date formats
match = re.search('(....)-(..)-(..)',
                  'Baker 1\t2009-11-17\t1223.0')
print match.group(1), match.group(2), match.group(3)
```

2009 11 17

Try doing *that* with substring searches...

```
# let's write another function to show match groups
def show_groups(pattern, text):
    m = re.search(pattern, text)
    if m is None:
      print 'NO MATCH'
      return
    print 'all:', m.group(0)
    for i in range(1, 1 + len(m.groups())):
      print '%2d: %s' % (i, m.groups(i))

show_groups('(....)-(..)-(..)',
            'Baker 1\t2009-11-17\t1223.0')
```

```
01: 2009
02: 11
03: 17
```

1. Letters and digits match themselves.

1. Letters and digits match themselves.
2. '|' means OR.

1. Letters and digits match themselves.
2. '|' means OR.
3. '.' matches any single character.

1. Letters and digits match themselves.
2. '|' means OR.
3. '.' matches any single character.
4. Use '()' to enforce grouping.

1. Letters and digits match themselves.
2. '|' means OR.
3. '.' matches any single character.
4. Use '()' to enforce grouping.
5. re.search returns a match object or None.

1. Letters and digits match themselves.
2. '|' means OR.
3. '.' matches any single character.
4. Use '()' to enforce grouping.
5. re.search returns a match object or None.
6. match.group(k) is the text that matched group k.

created by

Greg Wilson

June 2010