



Regular Expressions

Operators



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.

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

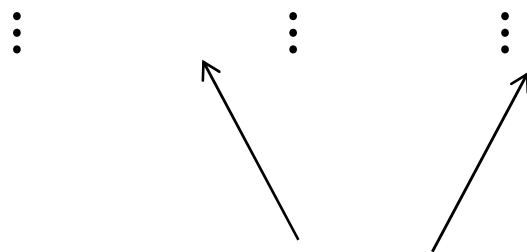
Pertwee/May 24, 2010/2103.8

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
:           :           :
```



month names and day numbers of varying length

Regular expressions are patterns that match text.

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`.

```
# get fields from Notebook #2 with simple string methods
record = 'Davison/May 22, 2010/1721.3'
site, date, reading = record.split('/')
month, day, year = date.split(' ')
if day[-1] == ',':
    day = day[:-1]
print year, month, day
```

2010 May 22

```
# get fields from Notebook #2 with simple string methods
record = 'Davison/May 22, 2010/1721.3'
site, date, reading = record.split('/')
month, day, year = date.split(' ')
if day[-1] == ',':
    day = day[:-1]
print year, month, day
```

2010 May 22

This is *procedural* (we tell the computer *how*)

```
# get fields from Notebook #2 with simple string methods
record = 'Davison/May 22, 2010/1721.3'
site, date, reading = record.split('/')
month, day, year = date.split(' ')
if day[-1] == ',':
    day = day[:-1]
print year, month, day
```

2010 May 22

This is *procedural* (we tell the computer *how*)

Regular expressions are *declarative* (we tell the computer *what*, and it figures out how)

```
# use '*' to break whole record into pieces
match = re.search('(.*)/(.*)/(.*)',
                  'Davison/May 22, 2010/1721.3')
print match.group(1)
print match.group(2)
print match.group(3)
```

Davison

May 22, 2010

1271.3

```
# use '*' to break whole record into pieces
match = re.search('(.*)/(.*)/(.*)',
                  'Davison/May 22, 2010/1721.3')
print match.group(1)
print match.group(2)
print match.group(3)
```

Davison

May 22, 2010

1271.3

'*' means "zero or more"

```
# use '*' to break whole record into pieces
match = re.search('(.*)/(.*)/(.*)',
                  'Davison/May 22, 2010/1721.3')
print match.group(1)
print match.group(2)
print match.group(3)
```

Davison

May 22, 2010

1271.3

'*' means "zero or more"

A *postfix* operator (like the 2 in x^2)

```
# use '*' to break whole record into pieces
match = re.search('(.*)/(.*)/(.*)',
                  'Davison/May 22, 2010/1721.3')
print match.group(1)
print match.group(2)
print match.group(3)
```

Davison

May 22, 2010

1271.3

'*' means "zero or more"

A *postfix* operator (like the 2 in x^2)

So $(.*)$ means "zero or more characters"


```
# use '*' to break whole record into pieces
match = re.search('(.*)/(.*)/(.*)',
                  'Davison/May 22, 2010/1721.3')
print match.group(1)
print match.group(2)
print match.group(3)
```

Davison

May 22, 2010

1271.3

'*' means "zero or more"

A *postfix* operator (like the 2 in x^2)

So $(.*)$ means "zero or more characters"

But the slashes must match exactly for it to work

```
# but this permits false positives
match = re.search('(.*)/(.*)/(.*)',
                  '///')
print '*', match.group(1)
print '*', match.group(2)
print '*', match.group(3)
```

```
*
*
*
```

```
# but this permits false positives
match = re.search('(.*)/(.*)/(.*)',
                  '///')
print '*', match.group(1)
print '*', match.group(2)
print '*', match.group(3)
```

*
*
*

.***** can match the empty string (zero characters)

```
# but this permits false positives
match = re.search('(.*)/(.*)/(.*)',
                  '///')
print '*', match.group(1)
print '*', match.group(2)
print '*', match.group(3)
```

*
*
*

.* can match the empty string (zero characters)

So this pattern will accept badly-formatted data

```
# force pattern to match _some_ characters  
print re.search('(.+)/(.+)/(.+)',  
                '///')
```

None

```
# force pattern to match _some_ characters
print re.search('(.+)/(.+)/(.+)',
                '///')
```

None

'+' is a postfix operator meaning "1 or more"

```
# force pattern to match _some_ characters
print re.search('(.+)/(.+)/(.+)',
                'Davison/May 22, 2010/1721.3')

print m.group(1)
print m.group(2)
print m.group(3)
```

Davison

May 22, 2010

1721.3

Always check that it still works with valid data...

write a function to show matched groups

```
def show_groups(pattern, text):
    m = re.search(pattern, text)
    if m is None:
        print 'NO MATCH'
        return
    for i in range(1, 1 + len(m.groups())):
        print '%2d: %s' % (i, m.group(i))

show_groups('(.+)/(.+)/(.+)',
            'Davison/May 22, 2010/1721.3')
```

1: Davison

2: May 22, 2010

3: 1721.3


```
# get the year, month, and day at the same time
show_groups('(.+)/(.+) (.+), (.+)/(.+)',
            'Davison/May 22, 2010/1721.3')
```

1: Davison

2: May

3: 22

4: 2010

5: 1721.3

why doesn't this work?

```
show_groups('(.+)/(.+) (.+), (.+)/(.+)',  
            'Davison/May 22 2010/1721.3')
```

None

why doesn't this work?

```
show_groups('(.+)/(.+) (.+), (.+)/(.+)',
            'Davison/May 22 2010/1721.3')
```


None



no comma in the data

make the comma optional

```
show_groups('(.+)/(.+) (.+),? (.+)/(.+)',
            'Davison/May 22 2010/1721.3')
```



1: Davison

2: May

3: 22


4: 2010

5: 1721.3

'?' is a postfix operator meaning "0 or 1"

make the comma optional

```
show_groups('(.)/(.) (.+),? (.)/(.)',
            'Davison/May 22 2010/1721.3')
```



1: Davison

2: May

3: 22

4: 2010

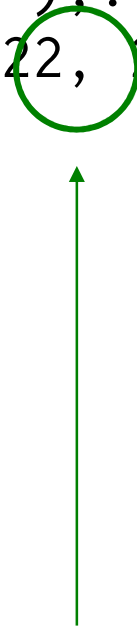
5: 1721.3

'?' is a postfix operator meaning "0 or 1"

I.e., "optional"

make the comma optional

```
show_groups('(.+)/(.+) (.+),? (.+)/(.+)',
            'Davison/May 22, 2010/1721.3')
```



1: Davison

2: May

3: 22

4: 2010

5: 1721.3

Still works on data with a comma

```
# we _don't_ want to match this
show_groups('(.+)/(.+) (.+),? (.+)/(.+)',
            'Davison/May 22, 201/1721.3')
```

1: Davison

2: May

3: 22

4: 201

5: 1721.3

years should have four digits, shouldn't they?

```
show_groups('(.+)/(.+) (.+),? (.+)/(.+) ',
            'Davison/May 22, 201/1721.3')
```

1: *Davison*

2: *May*

3: *22*

4: *201*

5: *1721.3*

Nobody's prefect


```
# force the pattern to match four characters
show_groups('(.+)/(.+) (.+),? (....)/(.+)',
            'Davison/May 22, 201/1721.3')
```

None

```
# force the pattern to match four characters
show_groups('(.+)/(.+) (.+),? (....)/(.+)',
            'Davison/May 22, 2017/1721.3')
```

None

Won't win any awards for readability

```
# force the pattern to match four characters
show_groups('(.+)/(.+) (.+),? (.{4})/(.+)',
            'Davison/May 22, 201/1721.3')
```

None

```
show_groups('(.+)/(.+) (.+),? (.{4})/(.+)',
            'Davison/May 22, 2010/1721.3')
```

1: Davison

2: May

3: 22

4: 2010

5: 1721.3

force the pattern to match four characters

```
show_groups('(.+)/(.+) (.+),? ({4})/(.+)',
            'Davison/May 22, 2017/1721.3')
```

None

```
show_groups('(.+)/(.+) (.+),? ({4})/(.+)',
            'Davison/May 22, 2010/1721.3')
```

1: Davison

2: May

3: 22

4: 2010

5: 1721.3

'{N}' is a postfix operator
meaning "match N times"

force the day to match to width

```
tests = (
    'Davison/May , 2010/1721.3',
    'Davison/May 2, 2010/1721.3',
    'Davison/May 22, 2010/1721.3',
    'Davison/May 222, 2010/1721.3',
    'Davison/May 2, 201/1721.3',
    'Davison/ 22, 2010/1721.3',
    '/May 22, 2010/1721.3',
    'Davison/May 22, 2010/'
)
pattern = '(.+)/(.+) (.{1,2}),? (.{4})/(.+)'
show_matches(pattern, tests)
```

force the day to match to width

```
tests = (
    'Davison/May , 2010/1721.3',
    'Davison/May 2, 2010/1721.3',
    'Davison/May 22, 2010/1721.3',
    'Davison/May 222, 2010/1721.3',
    'Davison/May 2, 201/1721.3',
    'Davison/ 22, 2010/1721.3',
    '/May 22, 2010/1721.3',
    'Davison/May 22, 2010/'
)
pattern = '(.+)/(.+) (.{1,2}),? (.{4})/(.+)'
show_matches(pattern, tests)
```

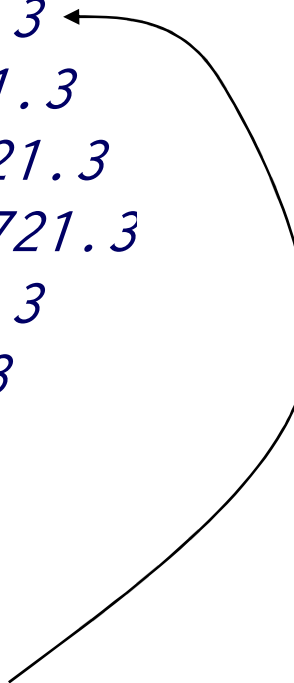
'{M,N}' matches from M to N times

matching against `'(.+)/(.+) (.{1,2}),? (.{4})/(.+)'`

```
** Davison/May , 2010/1721.3
** Davison/May 2, 2010/1721.3
** Davison/May 22, 2010/1721.3
   Davison/May 222, 2010/1721.3
   Davison/May 2, 201/1721.3
   Davison/ 22, 2010/1721.3
   /May 22, 2010/1721.3
   Davison/May 22, 2010/
```

matching against `'(.+)/(.+) (.{1,2}),? (.{4})/(.+)'`

```
** Davison/May , 2010/1721.3
** Davison/May 2, 2010/1721.3
** Davison/May 22, 2010/1721.3
   Davison/May 222, 2010/1721.3
   Davison/May 2, 201/1721.3
   Davison/ 22, 2010/1721.3
   /May 22, 2010/1721.3
   Davison/May 22, 2010/
```



Why does this match?

look at that test case more closely

```
show_groups('(.+)/(.+) (.{1,2}),? (.{4})/(.+)',
            'Davison/May , 2010/1721.3')
```

1: Davison

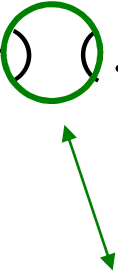
2: May

3: ,

4: 2010

5: 1721.3

'(.+)/(.+) (.{1,2}),? (.{4})/(.+)'



'Davison/May , 2010/1721.3'



space matches space

'(.+)/(.+) (.{1,2}),? (.{4})/(.+)'

'Davison/May,2010/1721.3'

space matches space

'.{1,2}' matches ','

'(.+)/(.+) (.{1,2}),? (.{4})/(.+)'

'Davison/May , 2010/1721.3'

space matches space

'.{1,2}' matches ','

',' matches nothing

(it's optional)

'(.+)/(.+) (.{1,2}),? (.{4})/(.+)'

'Davison/May , 2010/1721.3'

space matches space

'.{1,2}' matches ','

',?' matches nothing

(it's optional)

space matches space
again

force a match against digits

```
show_groups('(.+)/(.+) ([0-9]{1,2}),? (.{4})/(.+)',
            'Davison/May , 2010/1721.3')
```

None

```
show_groups('(.+)/(.+) ([0-9]{1,2}),? (.{4})/(.+)',
            'Davison/May 22, 2010/1721.3')
```

1: Davison

2: May

3: 22

4: 2010

5: 1721.3

force a match against digits

```
show_groups('(.+)/(.+) ([0-9]{1,2}),? (.{4})/(.+)',
            'Davison/May , 2010/1721.3')
```

None

```
show_groups('(.+)/(.+) ([0-9]{1,2}),? (.{4})/(.+)',
            'Davison/May 22, 2010/1721.3')
```

1: Davison

2: May

3: 22

4: 2010

5: 1721.3

'[...]' matches any character in a set

force a match against digits

```
show_groups('(.+)/(.+) ([0-9]{1,2}),? (.{4})/(.+)',
            'Davison/May , 2010/1721.3')
```

None

```
show_groups('(.+)/(.+) ([0-9]{1,2}),? (.{4})/(.+)',
            'Davison/May 22, 2010/1721.3')
```

1: Davison

2: May

3: 22

4: 2010

5: 1721.3

'[...]' matches any character in a set

E.g., '[aeiou]' matches vowels

match everything against characters and width

```
p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'
```

match everything against characters and width

```
p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'
```



Month name begins with upper-case letter...

match everything against characters and width

```
p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'
```



Month name is an upper-case letter...

...followed by one or more lower-case letters

match everything against characters and width

p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'



Day is one or two digits

match everything against characters and width

p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'



Day is one or two digits

This format allows '0', '00', '99', and so on

match everything against characters and width

```
p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'
```



Day is one or two digits

This format allows '0', '00', '99', and so on

Easiest to check that after converting to integer...

match everything against characters and width

p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'



Day is one or two digits

This format allows '0', '00', '99', and so on

Easiest to check that after converting to integer...

...since valid ranges depend on the month

match everything against characters and width

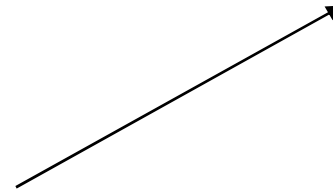
```
p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'
```

Year is exactly four digits



match everything against characters and width

```
p = '(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)'
```



Year is exactly four digits

Again, check for '0000' and the like after conversion

Put it all together.

```
def get_date(record):
```

```
    '''Return (Y, M, D) as strings, or None.'''
```

```
    # 2010-01-01
```

```
    m = re.search('([0-9]{4})-([0-9]{2})-([0-9]{2})',
                  record)
```

```
    if m:
```

```
        return m.group(1), m.group(2), m.group(3)
```

```
    # Jan 1, 2010 (comma optional, day may be 1 or 2 digits)
```

```
    m = re.search('/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/',
                  record)
```

```
    if m:
```

```
        return m.group(3), m.group(1), m.group(2)
```

```
    return None
```



created by

Greg Wilson

June 2010



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.