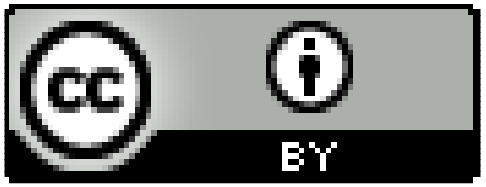
Regular Expressions

More Patterns



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 #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
  ⋮          ⋮          ⋮
```

```
# Get date from record.
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
```

```
# Get all fields from record.
def get_fields(record):
  '''Return (Y, M, D, site, reading) or None.'''

  patterns = (
    ('(.+)\t([0-9]{4})-([0-9]{2})-([0-9]{2})\t(.+)',
     2, 3, 4, 1, 5),
    ('(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)',
     4, 2, 3, 1, 5)
  )
  for p, y, m, d, s, r in patterns:
    m = re.search(p, record)
    if m:
      return m.group(y), m.group(m), m.group(d),
             m.group(s), m.group(r)

  return None
```

```
# Get all fields from record.
def get_fields(record):
  '''Return (Y, M, D, site, reading) or None.'''

  patterns = (
    ('(.+)\t([0-9]{4})-([0-9]{2})-([0-9]{2})\t(.+)',
     2, 3, 4, 1, 5),
    ('(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)',
     4, 2, 3, 1, 5)
  )
  for p, y, m, d, s, r in patterns:
    m = re.search(p, record)
    if m:
      return m.group(y), m.group(m), m.group(d),
             m.group(s), m.group(r)

  return None
```

```
# Get all fields from record.
def get_fields(record):
  '''Return (Y, M, D, site, reading) or None.'''

  patterns = (
    ('(.+)\t([0-9]{4})-([0-9]{2})-([0-9]{2})\t(.+)',
     2, 3, 4, 1, 5),
    ('(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)',
     4, 2, 3, 1, 5)
  )
  for p, y, m, d, s, r in patterns:
    m = re.search(p, record)
    if m:
      return m.group(y), m.group(m), m.group(d),
             m.group(s), m.group(r)

  return None
```

```
# Get all fields from record.
def get_fields(record):
  '''Return (Y, M, D, site, reading) or None.'''

  patterns = (
    ('(.+)\t([0-9]{4})-([0-9]{2})-([0-9]{2})\t(.+)',
     2, 3, 4, 1, 5),
    ('(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)',
     4, 2, 3, 1, 5)
  )
  for p, y, m, d, s, r in patterns:
    m = re.search(p, record)
    if m:
      return m.group(y), m.group(m), m.group(d),
             m.group(s), m.group(r)

  return None
```

```
# Get all fields from record.
def get_fields(record):
  '''Return (Y, M, D, site, reading) or None.'''

  patterns = (
    ('(.+)\t([0-9]{4})-([0-9]{2})-([0-9]{2})\t(.+)',
     2, 3, 4, 1, 5),
    ('(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)',
     4, 2, 3, 1, 5)
  )
  for p, y, m, d, s, r in patterns:
    m = re.search(p, record)
    if m:
      return m.group(y), m.group(m), m.group(d),
             m.group(s), m.group(r)

  return None
```

```
# Get all fields from record.
def get_fields(record):
  '''Return (Y, M, D, site, reading) or None.'''

  patterns = (
    ('(.+)\t([0-9]{4})-([0-9]{2})-([0-9]{2})\t(.+)',
     2, 3, 4, 1, 5),
    ('(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)',
     4, 2, 3, 1, 5)
  )
  for p, y, m, d, s, r in patterns:
    m = re.search(p, record)
    if m:
      return m.group(y), m.group(m), m.group(d),
             m.group(s), m.group(r)

  return None
```

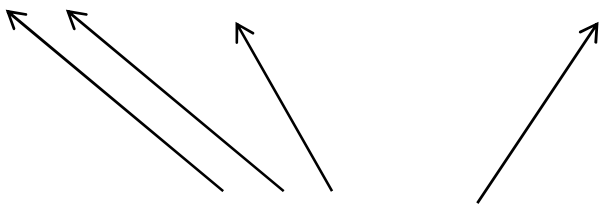
```
# Get all fields from record.
def get_fields(record):
  '''Return (Y, M, D, site, reading) or None.'''

  patterns = (
    ('(.+)\t([0-9]{4})-([0-9]{2})-([0-9]{2})\t(.+)',
     2, 3, 4, 1, 5),
    ('(.+)/([A-Z][a-z]+) ([0-9]{1,2}),? ([0-9]{4})/(.+)',
     4, 2, 3, 1, 5)
  )
  for p, y, m, d, s, r in patterns:
    m = re.search(p, record)
    if m:
      return m.group(y), m.group(m), m.group(d),
             m.group(s), m.group(r)

  return None
```

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```



spaces

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

But how to match parentheses?

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

But how to match parentheses?

The '()' in '(.+)' don't actually match characters

Put '\(' and '\)' in regular expression to match parenthesis characters '(' and ')'

Put '\(' and '\)' in regular expression to match parenthesis characters '(' and ')'

Another *escape sequence*, like '\t' in a string for tab

Put '\(' and '\)' in regular expression to match parenthesis characters '(' and ')'

Another *escape sequence*, like '\t' in a string for tab

In order to get the '\' in the string, must write '\\'

Put '\(' and '\)' in regular expression to match parenthesis characters '(' and ')'

Another *escape sequence*, like '\t' in a string for tab

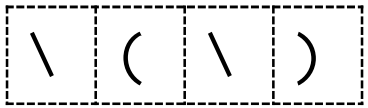
In order to get the '\' in the string, must write '\\'

So the strings representing the regular expressions that match parentheses are '\\(' and '\\)'

```
# find '()' in text
m = re.search('\\(\\)', text)
⋮      ⋮         ⋮      ⋮
```

program text

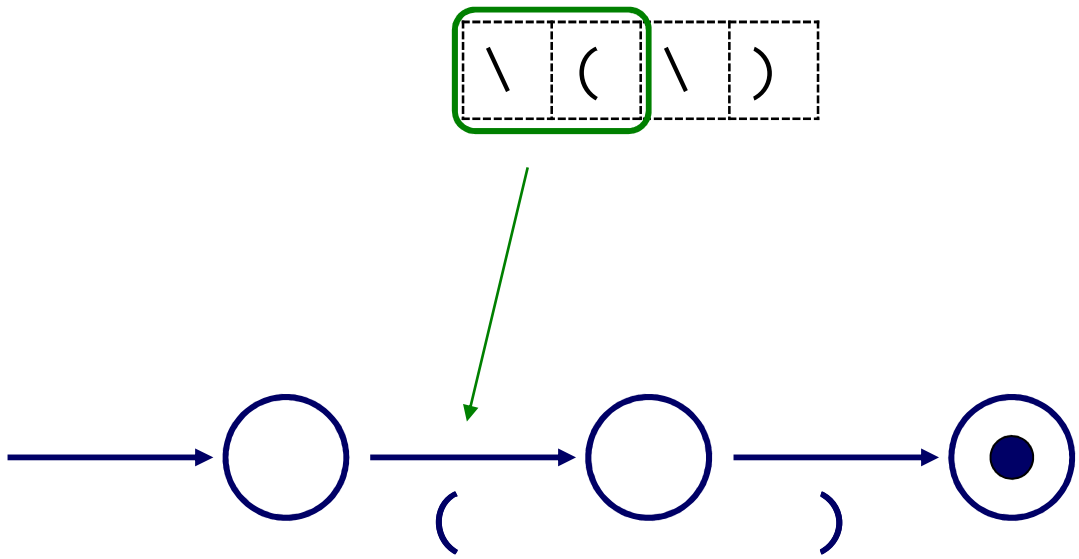
```
# find '()' in text
m = re.search('\\(\\)', text)
```



program text

Python string

```
# find '()' in text
m = re.search('\\(\\)', text)
```



program text

Python string

finite state machine

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

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

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

match actual '(' and ')'

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

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

match actual '(' and ')'

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

create a group

Notebook #3

```
Date Site Evil(mvad)
May 29 2010 (Hartnell) 1029.3
May 30 2010 (Hartnell) 1119.2
June 1 2010 (Hartnell) 1319.4
May 29 2010 (Troughton) 1419.3
May 30 2010 (Troughton) 1420.0
June 1 2010 (Troughton) 1419.8
 ⋮          ⋮          ⋮
```

match actual '(' and ')'

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

create a group

save the match

Some character sets come up frequently enough to be given abbreviations

Some character sets come up frequently enough to be given abbreviations

\d	digits	'0', '4', '9'

Some character sets come up frequently enough to be given abbreviations

\d	digits	'0', '4', '9'
\s	white space	' ', '\t', '\r', '\n'

Some character sets come up frequently enough to be given abbreviations

\d	digits	'0', '4', '9'
\s	white space	' ', '\t', '\r', '\n'
\w	word characters	'[A-Za-z0-9_]'

Some character sets come up frequently enough to be given abbreviations

\d	digits	'0', '4', '9'
\s	white space	' ', '\t', '\r', '\n'
\w	word characters	'[A-Za-z0-9_]'

actually the set of characters that can appear in a variable name in C (or Python)

Some character sets come up frequently enough to be given abbreviations

\d	digits	'0', '4', '9'
\s	white space	' ', '\t', '\r', '\n'
\w	word characters	'[A-Za-z0-9_]'

actually the set of characters that can appear in a variable name in C (or Python)

Need to double up the '\' when writing as string

And now for an example of really bad design

And now for an example of really bad design

\S non-space characters

And now for an example of really bad design

\S non-space characters

that's an upper-case 'S'

And now for an example of really bad design

\S non-space characters

that's an upper-case 'S'

\W non-word characters

And now for an example of really bad design

\S non-space characters

that's an upper-case 'S'

\W non-word characters

that's an upper-case 'W'

And now for an example of really bad design

\S non-space characters

that's an upper-case 'S'

\W non-word characters

that's an upper-case 'W'

Very easy to mis-type

And now for an example of really bad design

\S non-space characters

that's an upper-case 'S'

\W non-word characters

that's an upper-case 'W'

Very easy to mis-type

Even easier to mis-read

Can also match things that aren't actual characters

Can also match things that aren't actual characters

^ At the start of a pattern, matches the beginning of the input text

Can also match things that aren't actual characters

^ At the start of a pattern, matches the beginning of the input text

re.search('^mask', 'mask size') => match

Can also match things that aren't actual characters

^ At the start of a pattern, matches the beginning of the input text

re.search('^mask', 'mask size') => match

re.search('^mask', 'unmask') => None

Can also match things that aren't actual characters

^ At the start of a pattern, matches the beginning of the input text

re.search('^mask', 'mask size') => match

re.search('^mask', 'unmask') => None

$ At the end of a pattern, matches the end of the input text

Can also match things that aren't actual characters

^ At the start of a pattern, matches the beginning of the input text

re.search('^mask', 'mask size') => match

re.search('^mask', 'unmask') => None

$ At the end of a pattern, matches the end of the input text

re.search('temp$', 'high-temp') => match

Can also match things that aren't actual characters

^ At the start of a pattern, matches the beginning of the input text

re.search('^mask', 'mask size') => match

re.search('^mask', 'unmask') => None

$ At the end of a pattern, matches the end of the input text

re.search('temp$', 'high-temp') => match

re.search('temp$', 'temperature') => None

\b Boundary between word and non-word characters

\b Boundary between word and non-word characters

re.search('\\bage\\b', 'the age of') => match

\b Boundary between word and non-word characters

re.search('\\bage\\b', 'the age of') => match

re.search('\\bage\\b', 'phage') => None

created by

Greg Wilson

June 2010