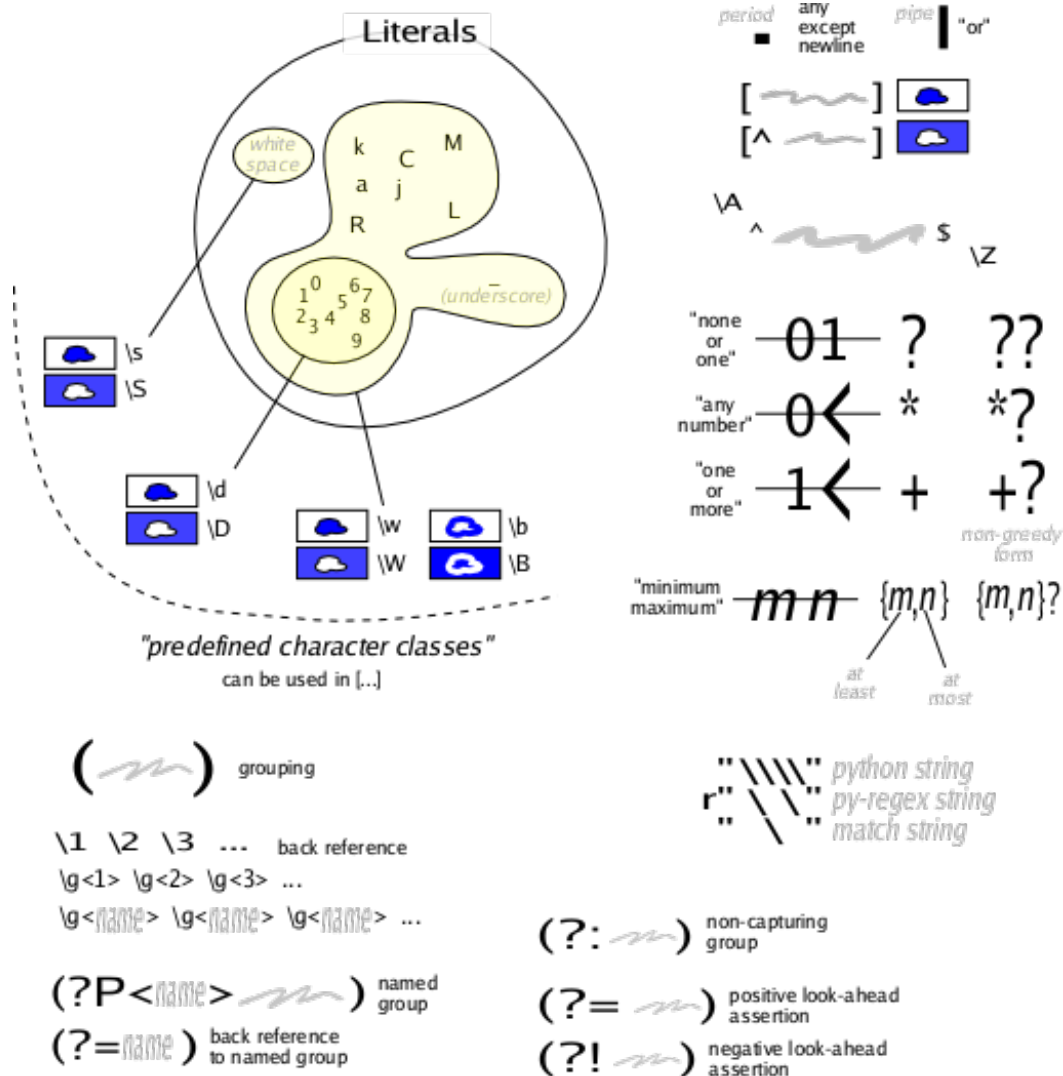


Introduction to Regular Expressions



Note: Image by [Lion Kimbro](#) downloaded from [Python Wiki](#) on Apr. 14, 2015.

What is RegEx?

- Regular expressions (REs, regex or regex patterns) are an essential tool for processing text data.
- It is a tiny, specialized (programming) language embedded inside other general purpose languages like Perl, Python, R, and Stata.
- The `re` module in Python provides regex related functions and objects.

What can we do with a RegEx?

- To search *text* for a part that matches a regex *pattern*

You can also do:

- To match the beginning of the *text* with a *pattern* (`re.match()`)
- To get all that match a *pattern* (`re.findall()`, `re.finditer()`)
- To substitute matches with replacement (`re.sub()`)
- To split *text* at the matches (`re.split()`)

An Example

In [1]:

```
# show the opr faculty page
from IPython.display import HTML

url = "http://opr.princeton.edu/faculty/"
HTML('<iframe src=' + url + ' width=700 height=350></iframe>')
```

Out[1]:

The screenshot shows the OPR (Office of Population Research) website at Princeton University. The header includes the OPR logo and the text ".princeton.edu" and "The Office of Population Research at Princeton University". A Google Custom Search bar is visible in the top right. The main content area is divided into a left sidebar and a main section. The sidebar has a "People" section with links to Administration, Faculty, Staff, Students, and Jobs, and a "Research" section with links to Projects, Seminars, Working Papers and Publications, and Dissertations. The main section has a "Faculty and Research Scholars" heading and a navigation bar with "Faculty", "Postdocs", and "Visitors" tabs. Below the navigation bar, there are two profiles of researchers: Alicia Adsera and Jeanne Altmann, each with a photo and a brief biography.

In [2]:

```
# get the html for the page
import urllib2

url = "http://opr.princeton.edu/faculty/"
text = urllib2.urlopen(url).read()
print text[:8000], "\n\n ... (many more lines) ... \n"
```

<!DOCTYPE html>

```

<head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" /><link
rel="stylesheet" type="text/css" href="/opr.css" /><title>
Office of Population Research, Princeton University
</title>
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script>
<script src='/scripts/hitch.js'></script>
</head>
<body style='margin:0;position:relative'>
<a name="top"></a>
<form style='position:absolute;top:12px;left:578px'
action="http://opr.princeton.edu/cse/" id="cse-search-box">
  <div>
    <input type="hidden" name="cx" value="011944897329676384373:yr8ayly85_0" /
  >
    <input type="hidden" name="cof" value="FORID:11" />
    <input type="hidden" name="ie" value="UTF-8" />
    <input type="text" name="q" size="31" /><br/>
    <input type="submit" name="sa" value="Search" style='float:right' />
  </div>
</form>
<script type="text/javascript" src="http://www.google.com/cse/brand?form=cse-
search-box&lang=en"></script>

<map name='logo'> <area shape='rect' coords='0,0,166,76' href='/'
alt='logo' /></map>
<a href='/75'><img src='/images/opr75.jpg' style="position:absolute;top:4px;
left:820px" border='0' title='Click to visit our 75th anniversary page' /></a>

<table cellpadding="0" cellspacing="0" style="border:0;width:900px;border-coll
apse:collapse">
<tr><td width="160" valign="top">
<div style="padding: 0 12px 12px 12px">

<h2>People</h2>

<p class='indent'>
<a class='nav' href='/admin/' title="OPR's administration">Administration</a>
<br/>
<a class='nav' href='/faculty/' title="Faculty and research associates, post-
docs and visitors">Faculty</a><br/>
<a class='nav' href='/staff/' title="Administrative, computing, library, and
research staff">Staff</a><br/>
<a class='nav' href='/students/' title="OPR graduate students, with links to
individual pages">Students</a><br/>
<a class='nav' href='/jobs/' title="Current job openings at OPR">Jobs</a>
</p>
<h2>Research</h2>
<p class='indent'>
<a class='nav' href='/research/' title="List of research projects by
topic">Projects</a><br/>
<a class='nav' href='/seminars/' title="Notestein seminar
schedule">Seminars</a><br/>
<a class='nav' href='/papers/' title="List of downloadable OPR working

```

[illegible]

</style>

<div id='tabs'>

Faculty

Postdocs

Visitors

</div>

<div >

<div id='Faculty' class='tab'>

<h2>Faculty and Research Scholars</h2>

<table><tr><td></td>

<td>Alicia Adsera,</td>

<i>Research Scholar, Woodrow Wilson School. Lecturer in Economics and International Affairs</i>.

Ph.D., Economics, Boston University, 1996.

<i>Interests: </i>fertility, household formation and labor market institutions, migration, income distribution and political economy, international and regional development, and press freedom.

</td></tr></table>

<table><tr><td></td>

<td>Jeanne Altmann,</td>

<i>Eugene Higgins Professor of Ecology and Evolutionary Biology, Emeritus</i>.

Ph.D., Behavioral Sciences, University of Chicago, 1979.

<i>Interests: </i>non-experimental research design and analysis, ecology and evolution of family relationships and of behavioral development; primate demography and life histories, parent-offspring relationships; infancy and the ontogeny of behavior and social relationships, conservation education and behavioral aspects of conservation.

</td></tr></table>

<table><tr><td></td>

<td>Elizabeth Armstrong,</td>

<i>Associate Professor of Public Affairs and Sociology</i>.

Ph.D., Sociology and Demography, University of Pennsylvania, 1998.

<i>Interests: </i>public health, the history and sociology of medicine, risk in obstetrics, and biomedical ethics.

</td></tr></table>

<table><tr><td></td>

<td>João Biehl,</td>

<i>Susan Dod Brown Professor of Anthropology. Co-Director, Program in Global Health and Health Policy</i>.

Ph.D., Anthropology, University of California, Berkeley, 1999.

<i>Interests: </i>Socio-cultural and Medical Anthropology; Social Studies of Science and Technology; Global Health; Culture and Mental Illness; HIV/AIDS; Pharmaceuticals; Health and Human Rights; Religion and Society; Subjectivity; Ethnographic Methods; Contemporary Social Theory; Latin American Societies; Brazil.

</td></tr></table>

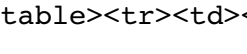
<table><tr><td></td>

<td><span

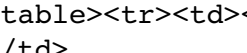
class='em'>Anne C. Case,</td>

<i>Alexander Stewart 1886 Professor of Economics and Public Affairs</i>.

Ph.D., Economics, Princeton University, 1988.
Interests: microeconomic foundations of development, health economics, public finance and labor economics.



Janet M. Currie,
 Henry Putnam Professor of Economics and Public Affairs, Woodrow Wilson School. Chair, Department of Economics. Director, Center for Health and Wellbeing.
 Ph.D., Economics, Princeton University, 1988.
Interests: health and well-being of children.



Rafaela Dancygier,
 Assistant Professor of Politics and Public and International Affairs.
 Cyril E. Black U

... (many more lines) ...

In [3]:

```
# extracting names of the faculty, visitors, and postdocs
import re

pattern = r"<span class='em'>(.*?)</span>"
for name in re.findall(pattern, text):
    print name
```

Alicia Adsera
 Jeanne Altmann
 Elizabeth Armstrong
 João Biehl
 Anne C. Case
 Janet M. Currie
 Rafaela Dancygier
 Angus S. Deaton
 Elisabeth Donahue
 Thomas J. Espenshade
 Patricia Fernández-Kelly
 Susan Fiske
 Ana Maria Goldani
 Noreen Goldman
 Bryan Grenfell
 Jean Grossman
 Tod G. Hamilton
 Jeffrey S. Hammer
 Douglas S. Massey
 Sara S. McLanahan
 C. Jessica E. Metcalf
 Daniel Notterman
 Elizabeth L. Paluck

Alejandro Portes
Germán Rodríguez
Matthew Salganik
Edward Telles
Marta Tienda
James Trussell
Tom S. Vogl
Charles F. Westoff
Jeanne Brooks-Gunn
Sharon H. Bzostek
Pamela Klebanov
Abigail Aiken
Michelle DeKlyen
Rachel E. Goldberg
Nicole K. Smith
Katherine M. Tumlinson
Brandon G. Wagner
Postdocs
Visitors

Another Example

In [4]:

```
import re

text = """
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA
CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT CTA
GAA GAC GAG GAT GCA GCC GCG GCT GGA GGC GGG GGT GTA
TAA TAC TAG TAT TCA TCC TCG TCT TGA TGC TGG TGT TTA
"""

# mach and highlight any DNA codons that code for Arginine
pattern = r"\s(CG\S|AG[AG])\s"
replacement = r"<\g<1>>"

text = re.sub(pattern, replacement, text)
print text
```

AAA	AAC	AAG	AAT	ACA	ACC	ACG	ACT	<AGA>	AGC	<AGG>	AGT	ATA
CAA	CAC	CAG	CAT	CCA	CCC	CCG	CCT	<CGA><CGC><CGG><CGT>				CTA
GAA	GAC	GAG	GAT	GCA	GCC	GCG	GCT	GGA	GGC	GGG	GGT	GTA
TAA	TAC	TAG	TAT	TCA	TCC	TCG	TCT	TGA	TGC	TGG	TGT	TTA

Basic Search

In [5]:

```
import re

text = "Most letters and characters will match themselves"
pattern = r"mat\w\w"
```

```
match = re.search(pattern, text) # returns a match obj or None
```

```
if match:
    print "found:", match.group()
else:
    print "not found"
```

```
found: match
```

- `re.search(pattern, text)` returns a match object if found or `None` if not.
- It is a good habit to put your regex pattern in a raw string
- The pattern matches "mat" followed by any two *word characters*. A word character is a letter, digit, or an underscore character.

Simple Patterns

- Letters and characters will match themselves
- Meta-characters (i.e., `.`, `^`, `$`, `*`, `+`, `?`, `{`, `}`, `[`, `]`, `(`, `)`, `\`, `|`) don't match themselves -- they have special meanings

Matches a Single Character

- Basic patterns that match a single character:

Pattern	(Description)	Matches
<code>.</code>	(dot)	any single character except newline
<code>\d</code>	(\ lower case d)	a digit, [0-9]
<code>\D</code>	(\ upper case D)	a non-digit
<code>\s</code>	(\ lower case s)	a "whitespace" character, [\n\r\t\f]
<code>\S</code>	(\ upper case S)	a non-writespace character
<code>\w</code>	(\ lower case w)	a "word" character
<code>\W</code>	(\ upper case W)	a non-word character
<code>[abc]</code>		matches character a, b or c
<code>[a-z]</code>		matches any single lower-cased characater
<code>[2468]</code>		matches one digit which is either 2, 4, 6, or 8

- The following matches not a character, but a boundary:

Pattern	(Description)	Matches
<code>\b</code>	\ lower case b	a boundary between a word and a non-word character
<code>^</code>	(caret)	the beainning of a text strina

\$		the end of a text string

In [6]:

```
import re

text = "This is my phone number 555-1234. Call me at 9:00am!"
pattern = r"\d\d\d-\d\d\d\d"

match = re.search(pattern, text)

if match:
    print "found:", match.group()
else:
    print "not found"
```

found: 555-1234

Quiz. Write a regex pattern that matches 9:00 am

In [7]:

```
import re

text = "This is my phone number, 555-1234. Call me at 9:00 am!"
pattern = r"" # replace this with your pattern

match = re.search(pattern, text)

if match:
    print "found:", match.group()
else:
    print "not found"
```

found:

Examples

In [8]:

```
import re

text = "Lunch will be served at 12:00 pm."

match = re.search(r"..nch", text)           # match.group() == "Lunch"
match = re.search(r"..unc[a-z]", text)      # match.group() == "Lunch"
match = re.search(r"lunch", text)           # match == None
match = re.search(r"L\w\w\w\w", text)       # match.group() == "Lunch"

match = re.search(r"\d\d:\d\d [ap]m", text) # match.group() == "12:00 pm"
match = re.search(r"s...ed", text)          # match.group() == "served"

match = re.search(r"^", text)               # match.group() == ""
```

```

match = re.search(r"...", text)          # match.group() == Lun
match = re.search(r".e\b", text)         # match.group() == "be"
match = re.search(r"...\W$", text)       # match.group() == "0 pm."

```

In [9]:

```

# let's just take a break. unquote to open the web page
import webbrowser
# webbrowser.open("http://tinyurl.com/mh9olpo")

```

Repetition

pattern	matches	equivalent to
*	0 or more occurrences of the pattern to its left	{0,}
+	1 or more occurrences of the pattern to its left	{1,}
?	0 or 1 occurrences of the pattern to its left	{0,1}

The search finds the earliest (leftmost) matching sub-text for the pattern then it tries to use up as many characters as possible for the repetition. (The + and * are said to be *greedy*.)

You can also specify repetitions with curly brackets ({ })

pattern	matches
{m}	exactly m occurrences of the pattern to its left
{m,n}	from m occurrences (default 0) to n times (default $+\infty$) of the pattern to its left

In [10]:

```

import re

text = "Vroooooooooom Vroom Vroom"
pattern = r"o+"

match = re.search(pattern, text)
if match:
    print "match:", match.group()
else:
    print "no match"

```

match: oooooooooo

Repetition Examples

In [11]:

```

import re

text = "Vroooooooooom Vroom Vroom"

```

```

match = re.search(r"Vro+m", text)      # match.group() == "Vroooooooooom"
match = re.search(r"A+", text)          # match == None
match = re.search(r"\w+\s+\w+\s+\w+", text) # three words separated by 1+ spaces

match = re.search(r"A*", text)          # len(match.group()) == 0
match = re.search(r"\w*m", text)        # match.group() == "Vroooooooooom"
match = re.search(r"\w*$", text)        # match.group() == "Vroom"

match = re.search(r"Vroo?m", text)      # match.group() == "Vroom"
match = re.search(r"s\s+\w+", text)     # match.group() == "Vroom"
match = re.search(r"s\s*\w+", text)     # match.group() == "Vroom"

match = re.search(r"\b\w+", text)        # match.group() == "Vroooooooooom"
match = re.search(r"^\b\w+", text)      # match.group() == "Vroooooooooom"

match = re.search(r"Vro{3}m", text)     # match.group() == "Vroom"
match = re.search(r"Vro{0,7}m", text)   # match.group() == "Vroom"
match = re.search(r"Vro{4,7}m", text)   # match == None

```

Character Classes or Square Brackets

- matches a single character
- you can list a set of characters to match. [aeiou] matches any vowel
- complement set can be specified by a caret (^). For example, [^aeiou] matches anything but a vowel. Notice that the caret should be the first character in within the brackets. Outside the brackets, the caret (^) matches the beginning of the text.
- ranges can be specified. [0-9] matches a digit
- you can use special sequences like \w, \d, \s and the like.
- dot (.) within the brackets represents a dot, not any character.
- in fact, there are very few special characters within the character classes, including: ^ (complement), - (range), and] (signals the end of the character class)
- an example of matching email addresses.

In [12]:

```

import re

text = "This is my email address: funny.changarilla@princeton.edu"
pattern = r"[\w.]+\@[ \w. ]+"

match = re.search(pattern, text)

if match:
    print "match:", match.group()
else:
    print "no match"

match: funny.changarilla@princeton.edu

```

Groups or parentheses

- Groups let you pick out parts of the matching string
- Use parentheses to make groups
- Let's group the email pattern by username and hostname. See below example

In [13]:

```
import re

text = "This is my email address: funny.changarilla@princeton.edu"
pattern = r"([\w.]+)@([\w.]+)"

match = re.search(pattern, text)

if match:
    print "match.group():", match.group()
    print "match.group(1):", match.group(1)
    print "match.group(2):", match.group(2)
else:
    print "no match"
```

```
match.group(): funny.changarilla@princeton.edu
match.group(1): funny.changarilla
match.group(2): princeton.edu
```

What Else Does the Match Object Return?

In [14]:

```
import re

text = "This is my email address funny.changarilla@princeton.edu, email me!"
#-----1-----2-----3-----4-----5-----
pattern = r"([\w.]+)@([\w.]+)"

match = re.search(pattern, text)
#print dir(match)

if match:
    print "match.start():", match.start()
    print "match.end():", match.end()
    print "match.span():", match.span()
#    print "text[match.start():match.end()]:", text[match.start():match.end()]
```

```
match.start(): 25
match.end(): 56
match.span(): (25, 56)
```

Alternation (|)

pattern A|pattern B matches either pattern A or pattern B

In [15]:

```
import re

text = "I love a dog. You love a cat."
pattern = r"dog|cat"

match = re.search(pattern, text)

if match:
    print "match:", match.group()
else:
    print "no match"
```

match: dog

Escaping Meta-characters

In [16]:

```
import re

text = "re.search() returns the 1st match. re.findall() returns a [list]"
pattern = r"\w+\(\)"      # use the slash to escape

match = re.search(pattern, text)

if match:
    print "match:", match.group()
else:
    print "no match"
```

match: search()

Options or Compilation Flags

- I or IGNORECASE performs case-insensitive matching
- M or MULTILINE does multi-line matching, affecting ^ and \$
- S or DOTALL makes the dot . match any character including the newline \n
- Multiple options can be specified by concatenating with pipes (|)

In [17]:

```
import re

text = """Substitution can be done with re.sub(pattern, text).
and splits with re.split(pattern, text).
"""

pattern = r"sub.+"

match = re.search(pattern, text, re.IGNORECASE | re.DOTALL)
```

```
if match:
    print "match:", match.group()
else:
    print "no match"
```

match: Substitution can be done with `re.sub(pattern, text)`.
and splits with `re.split(pattern, text)`.

Quiz: Write a regex pattern to match any adverb

Hint: Let's define an adverb as a word that ends with "ly". You can use any text, or use the text given below.

In [18]:

```
import re

text = """
Once upon a time, there was a beautiful princess who had a golden ball.
She lived in a palace with her father, the King, and her seven sisters.
Every day she played with her ball in the garden of the palace.
At the end of the garden there was a deep, dark lake. Unfortunately,
one day she dropped her golden ball into the water. She was very unhappy
and she sat on the grass and started to cry. Suddenly, she heard a voice:
"Don't cry, princess".
She opened her eyes and saw a large green frog. "Oh, please help me!"
she said, "I can't get my ball."
"I'll help you", said the frog, "if I can com and live with you in the
palace!"
"Yes, yes, of course. I promise," said the princess.
So the frog jumped into the water and cam back with the ball.
The princess laghed and took the ball. She ran quickly back to the palace
and forgot all about the frog.
"""
```

In []:

Quiz. Find all the adverbs in the text

Hint: The function `re.findall(pattern, text)` returns all the matching sub-text as a list.

In []:

Quiz. Validate a poker hand

Suppose that you are writing a poker program where a player's hand is represented as a five-character string with each character representing a card, "a" for ace, "k" for king, "q" for queen, "j" for jack, "t" for 10, and "2" through "9" representing the card with that value.

Write a pattern to validate a hand. It does not have to be very thorough. Just to pass the test cases shown below.

In [19]:

```
import re

pattern = r"" # replace this with your pattern

match = re.search(pattern, "245") # match == None
match = re.search(pattern, "a 334") # match == None
match = re.search(pattern, "akt5q") # match.group() == "akt5q"
match = re.search(pattern, "akt5e") # match == None
match = re.search(pattern, "727ak") # match.group() == "727ak"
match = re.search(pattern, " akt34") # match == None
match = re.search(pattern, "23456 ") # match == None
```

Learning Resources

- Official [Reference](#) and Kuchling's [HowTo](#)
- Friedl's book, [Mastering Regular Expressions](#)
- Google Developers site Python Course [Chapter](#)
- Tartley (Jonathan Hartley)'s [Cheatsheet](#)