Effective Programming Practices for Economists

17. Regular Expressions

Hans-Martin von Gaudecker

Department of Economics, Universität Bonn

universitätbonn

Motivation

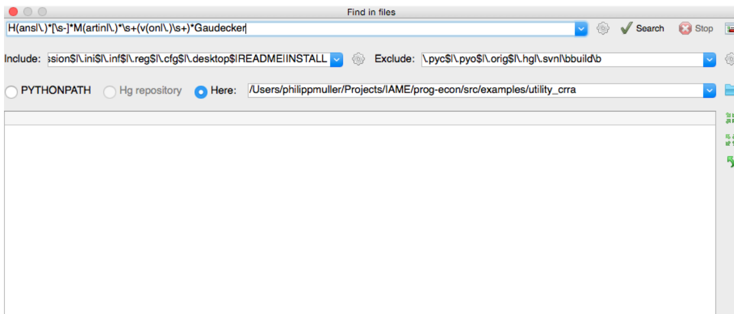
- Replace all occurences of my name in the project template
  - Hans-Martin von Gaudecker
  - HM Gaudecker
  - Convex combinations
  - Sloppy on whitespace?
- Search by hand, one-by-one for all potential patterns?

Licensed under the Creative Commons Attribution License

Motivation

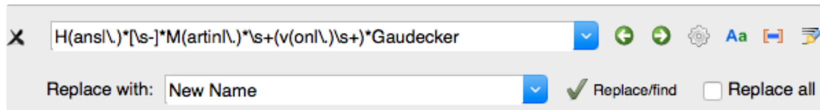
Press `Ctrl+F` or `Ctrl+Shift+F` in Spyder and type

```
H(ans|\.)*[\s-]*M(artin|\.)*\s+(v(on|\.)\s+)*Gaudecker
```



Licensed under the Creative Commons Attribution License

Motivation



Licensed under the Creative Commons Attribution License

At the end of this lecture you are able to . . .

- Appreciate usefulness of pattern matching in text
  - Everyday text editing
  - Data management
- Understand basic concepts of regular expression engines
- Apply the most important features of regular expressions
  - Everyday text editing
  - Data management

Licensed under the Creative Commons Attribution License

More applications of regular expressions

- Poorly formatted data files – limits to `str.split().strip()`
- Cleaning up or aggregating categorical string data: ICD codes, string answers in surveys, ...
- Experiments: Categorise chat messages
- Classification of newspaper articles
- Encoding of text-based legislative information from (semi-)standardized databases

- **Important:** Ability to extract substrings from matches $\Rightarrow$ homogenise data immediately

Licensed under the Creative Commons Attribution License

Warning

- Notation is ugly, even by standards of programming
- Match character patterns using the very same characters
  - No invention of new symbols, like in mathematics
  - Characters take on special meanings, need for escaping
  - Same as `'\n'`, `'*.txt'` etc., but much more to keep in mind
- `http://pyparsing.wikispaces.com/` invents “symbols”
  - Much more readable and powerful
  - Orders of magnitude slower
  - Not available in editor search field

Licensed under the Creative Commons Attribution License

The good news

- Regular expressions are more or less standardised
- All derive from UNIX' `grep` utility
- Only small differences in special characters between
  - Editor search & replace fields
  - Python
  - Stata
  - Matlab
  - R
  - etc.
- Understand concept, get manual for specific implementation

Licensed under the Creative Commons Attribution License

Example: Similar data from different sources

- In Gaudecker, Soest, and Wengström (2012), we compare the results from the same experiment in different populations
  - Students in Tilburg University's lab
  - Respondents to the CentERpanel, a Dutch household survey conducted via the Internet
- Data on lottery choices
- Here: Consider only the timing data, delivered separately

Licensed under the Creative Commons Attribution License

Example: `cp.csv`

```
nohhold,nomem,datum1,datum2,begin_h,begin_m,begin_s,end_h,end_m,end_s
4,2,12/9/2005,12/9/2005,19,6,57,19,42,40
6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
6,2,11/25/2005,,18,40,55,,,
21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
38,1,11/26/2005,11/26/2005,19,8,45,19,28,28
```

- Separated by commas
- Some missing data
- Days have different length

Licensed under the Creative Commons Attribution License

Example: `lab.txt`

```
login datum1 datum2 tijd1 tijd2 datum3 tijd3
600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 2006" "14:22:26"
58225 "Sept 22, 2005" "Sept 22, 2005" "10:05:10" "10:17:43" "" ""
200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 2006" "11:09:52"
27928 "Sept 21, 2005" "Sept 21, 2005" "10:05:09" "10:23:48" "" ""
68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
38012 "Sept 21, 2005" "Sept 21, 2005" "11:04:38" "11:12:04" "" ""
```

- Separated by spaces
- But spaces also appear in string variables
- String variables are enclosed in quotes, others are not
- Months, days have different length

Licensed under the Creative Commons Attribution License

Reading in the data

```
times = []
for filename in ('cp.csv', 'lab.txt'):
    times += open(filename, 'r').readlines()[1:6]
```

Content of the variable `times`

```
4,2,12/9/2005,12/9/2005,19,6,57,19,42,40
6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
6,2,11/25/2005,,18,40,55,,,
21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 20
58225 "Sept 22, 2005" "Sept 22, 2005" "10:05:10" "10:17:43" "" ""
200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 20
27928 "Sept 21, 2005" "Sept 21, 2005" "10:05:09" "10:23:48" "" ""
68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

Licensed under the Creative Commons Attribution License

Matching experiments from November

```
# Look for experiments in November using list search.
for t in times:
    if '11' in t:
        print(t)
```

```
6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
6,2,11/25/2005,,18,40,55,,,
21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 20
68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

- Okay at the top, but we match too much
- Will need to explore structure, come back to this later

Licensed under the Creative Commons Attribution License

Introducing regular expressions

```
# Look for experiments in November using regular expressions.
import re
for t in times:
    if re.search('11', t):
        print(t)
```

- `re.search()` takes two required arguments
  - Pattern we are searching for, written as a string
  - String we are searching in
- Common to get these reversed, hard to track down
  - Both arguments are strings … — be careful!

Licensed under the Creative Commons Attribution License

Introducing regular expressions

```
# Look for experiments in November using regular expressions.
import re
for t in times:
    if re.search('11', t):
        print(t)
```

```
6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
6,2,11/25/2005,,18,40,55,,,
21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 20
68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

- Letters, digits, underscores always match themselves
- Nothing gained yet ...

Licensed under the Creative Commons Attribution License

Matching experiments from Nov. / Dec.

- Solution leading to very long expressions

```
if '11' in t or '12' in t:
```

- Instead, use `|` as `or` operator in regular expressions:

```
# Look for experiments in November or December.
for t in times:
    if re.search('11|12', t):
        print(t)
```

```
4,2,12/9/2005,12/9/2005,19,6,57,19,42,40
6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
6,2,11/25/2005,,18,40,55,,,
21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 20
68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

Licensed under the Creative Commons Attribution License

Abbreviating our code examples

- Make code examples shorter and the output more readable using a function that
  - Takes a pattern and a list of strings as inputs
  - Searches for the pattern in each of the strings
  - Prints each string, indicator of whether pattern was found

```
def show_matches(pattern, strings):
    for s in strings:
        if re.search(pattern, s):
            print('** ', s)
        else:
            print('   ', s)
```

Licensed under the Creative Commons Attribution License

Abbreviating our code examples

```
# Demonstrate the last bit again with new function.
pattern = '11|12'
show_matches(pattern, times)
```

```
** 4,2,12/9/2005,12/9/2005,19,6,57,19,42,40
** 6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
** 6,2,11/25/2005,,18,40,55,,,
** 21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
** 21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
   600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 200
   58225 "Sept 22, 2005" "Sept 22, 2005" "10:05:10" "10:17:43" "" ""
** 200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 200
   27928 "Sept 21, 2005" "Sept 21, 2005" "10:05:09" "10:23:48" "" ""
** 68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

Licensed under the Creative Commons Attribution License

Abbreviating our code examples

```
# Trying a different pattern.
pattern = '11|2'
show_matches(pattern, times)
```

```
** 4,2,12/9/2005,12/9/2005,19,6,57,19,42,40
** 6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
** 6,2,11/25/2005,,18,40,55,,,
** 21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
** 21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
** 600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 200
** 58225 "Sept 22, 2005" "Sept 22, 2005" "10:05:10" "10:17:43" "" ""
** 200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 200
** 27928 "Sept 21, 2005" "Sept 21, 2005" "10:05:09" "10:23:48" "" ""
** 68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

Licensed under the Creative Commons Attribution License

Operator precedence

- `or` has lower precedence than connection of two characters
- Think of no operator in between two characters as an `and`

```
A and B or C
```

is true when `A` and `B` are true; or when `C` is true (or both)

- Parenthesis enforce grouping for custom precedence

```
A and (B or C)
```

is true when `A` is true and at least one of `B` and `C` is true

Licensed under the Creative Commons Attribution License

Operator precedence

```
# Just for demonstration -- less readable than 11|12.
pattern = '1(1|2)'
show_matches(pattern, times)
```

```
** 4,2,12/9/2005,12/9/2005,19,6,57,19,42,40
** 6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
** 6,2,11/25/2005,,18,40,55,,,
** 21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
** 21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
   600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 200
   58225 "Sept 22, 2005" "Sept 22, 2005" "10:05:10" "10:17:43" "" ""
** 200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 200
   27928 "Sept 21, 2005" "Sept 21, 2005" "10:05:09" "10:23:48" "" ""
** 68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

Licensed under the Creative Commons Attribution License

Using context to match only months

- Structure implies comma before, slash after the month

```
# Making use of context to match only months.
pattern = ',(11|12)/'
show_matches(pattern, times)
```

```
** 4,2,12/9/2005,12/9/2005,19,6,57,19,42,40
** 6,1,11/26/2005,11/26/2005,6,52,7,7,2,44
** 6,2,11/25/2005,,18,40,55,,,
** 21,1,11/26/2005,11/26/2005,12,11,2,12,30,58
** 21,2,11/27/2005,11/27/2005,13,55,43,14,5,38
   600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 200
   58225 "Sept 22, 2005" "Sept 22, 2005" "10:05:10" "10:17:43" "" ""
   200132 "May 4, 2006" "May 4, 2006" "11:05:29" "11:12:43" "May 4, 200
   27928 "Sept 21, 2005" "Sept 21, 2005" "10:05:09" "10:23:48" "" ""
   68303 "Sept 20, 2005" "Sept 20, 2005" "11:05:16" "11:21:47" "" ""
```

Licensed under the Creative Commons Attribution License

Extracting data

- `re.search` returns a match object if successful
  - Else it returns `None`
  - Used before that a match object evaluates to `True`
- Match objects have a `group()` method

```
month_match = re.search(',(11|12)/', times[0])
print(month_match.group(0))
```

- Takes group index (comma-separated indices) as argument
- The first group is the entire match

```
,12/
```

Licensed under the Creative Commons Attribution License

Extracting data

- Second function of parentheses: Delimit groups

```
print(month_match.group(1))
```

```
12
```

- **Note on “Replace” in most editors:**

  Access the groups by \$0, \$1, \$2, …

Licensed under the Creative Commons Attribution License

A quick look backwards ...

- Letters and digits match themselves
- `|` means `or`
- Parentheses enforce grouping
  - Change precedence of operators
  - Delimit groups for substring extraction
- `re.search` returns a match object or `None`
- `match.group(k)` is the text that matched group k

Licensed under the Creative Commons Attribution License

. . . and a look at the workflow for building up patterns

- Start with something simple that matches part of the data
- Test it against data that it should match
- Test it against data that it **should not** match
  - Tracking down false positives can be **very** hard
- Iterate over the previous steps until complete

Licensed under the Creative Commons Attribution License

Procedural vs. declarative revisited

- Extract lab experiments' starting date using string methods

```
600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 20
```

```
start_date = times[5].split('"')[1]
m, d, y = start_date.split(' ')
d = d.rstrip(', ')
print(m, '-', d, '-', y)
```

```
May - 3 - 2006
```

- **Procedural** way of solving the problem
- Regular expressions are **declarative**

Licensed under the Creative Commons Attribution License

Operators and patterns

- Operators change the meaning of other characters
  - `()` delimits groups and changes the scope of `|`
  - `gray|grey` and `gr(e|a)y` match the same pattern
- Postfix operator `*`
  - "Match the previous expression zero or more times"
  - Postfix: Operator appears after the expression it applies to
  - Like the 2 in x^2
  - Prefix, infix analogously defined

Licensed under the Creative Commons Attribution License

Operators and patterns

- Complete patterns are then made up of
  - Characters
  - Metacharacters
  - Operators
- Metacharacters are also called special characters
- E.g. a dot `.` is a wildcard matching any one character

Licensed under the Creative Commons Attribution License

Overview of metacharacters

Character	Meaning
`.`	Any one character (except newline, usually)
`^`	Match at the start of string / prefix operator for complementing set inside `[ ]`
`$`	Match at the end of string
`\s`	Whitespace character
`\S`	Complement of `\s`
`\w`	Word characters (valid Python variable name characters – alphanumerics, underscore)
`\W`	Complement of `\w`
`\n`, `\t`, etc.	As usual in Python strings

Licensed under the Creative Commons Attribution License

Use raw strings(i.e. `r'abc'`) to define regular expressions

From `http://docs.python.org/3/library/re.html`:

`\` either escapes special characters (permitting you to match characters like `*`, `?`, and so forth), or signals a special sequence [...]

If you're not using a raw string to express the pattern, remember that Python also uses the backslash as an escape sequence in string literals; if the escape sequence isn't recognized by Python's parser, the backslash and subsequent character are included in the resulting string. However, if Python would recognize the resulting sequence, the backslash should be repeated twice. **This is complicated and hard to understand, so it's highly recommended that you use raw strings for all but the simplest expressions.**

Licensed under the Creative Commons Attribution License

Overview of operators

Character	Meaning
`*`	Match zero or more occurrences of the preceding expression (greedy)
`+`	Match one or more occurrences of the preceding expression (greedy)
`?`	Match zero or one occurrences of the preceding expression (greedy)
`*?`, `+?`, `??`	Lazy versions of previous operators
`{m}`	Match `m` occurences of the preceding expression
`{m, n}`	Match between `m` and `n` occurences of the preceding expression

Licensed under the Creative Commons Attribution License

Greedy vs. lazy

- Back to extracting start date of the online experiments

```
600514 "May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May 3, 20
```

```
# Extract date using regular expressions.
pattern = r'\w\s"(.+) (.+), (.+)"'
print(re.search(pattern, times[5]).group(1))
```

```
May 3, 2006" "May 3, 2006" "14:09:43" "14:26:30" "May
```

- Operators are greedy by default – match maximum possible

Licensed under the Creative Commons Attribution License

Greedy vs. lazy

1. Make operator lazy

```
# Extract date using regular expressions, lazy operators.
pattern = r'\w\s"(.+?) (.+?), (.+?)"'
d = re.search(pattern, times[5])
print(d.group(1), '-', d.group(2), '-', d.group(3))
```

```
May - 3 - 2006
```

2. Use more information about structure

```
# Extract date using regular expressions, more structure.
pattern = r'\w\s"(\w+) (\w{1,2}), (\w{4})"'
d = re.search(pattern, times[5])
print(d.group(1), '-', d.group(2), '-', d.group(3))
```

```
May - 3 - 2006
```

Licensed under the Creative Commons Attribution License

Defining sets of characters with `[]`

Use square brackets to indicate a set of characters. Characters can be listed individually, or a range of characters can be indicated by giving two characters and separating them by a `-`. Special characters are not active inside sets. For example, `[akm$]` will match any of the characters `a`, `k`, `m`, or `$`; `[a-z]` will match any lowercase letter, and `[a-zA-Z0-9]` matches any letter or digit. Character classes such as `\w` or `\S` (defined below) are also acceptable inside a range.

You can match the characters not within a range by complementing the set. This is indicated by including a `^` as the first character of the set; `^` elsewhere will simply match the `^` character. For example, `[^5]` will match any character except `5`, and `[^^]` will match any character except `^`.

Note that inside `[]` the special forms and special characters lose their meanings and only the syntaxes described here are valid. For example, `+`, `*`, `(`, `)`, and so on are treated as literals inside `[]`.

Licensed under the Creative Commons Attribution License

Extract the starting date using a character set

3. Use character sets

```
# Extract date using character set.
pattern = r'\w\s"([^"]+)'
d = re.search(pattern, times[5])
print(d.group(1))
```

```
May 3, 2006
```

- Not as powerful as the previous solutions in this case
- But often extremely useful, e.g. matching digit months

```
1[0-2]|[1-9]
```

Licensed under the Creative Commons Attribution License

A minimal amount of theory . . .

- RegEx implementation uses **finite state machines**
- Can take one of several states, but no sense of history
- Similar to a Markov Chain
  - Transition to next state does not depend on history
  - E.g. probability of exiting current health state only depends on the current level of health, not on past ups and downs

Licensed under the Creative Commons Attribution License

… (example) …

- Regular expression `'ab'`, string `'acabd'`
- Hit `'a'`, move to state “matched an a”
- Hit `'c'`, move back to initial state
- Hit `'a'`, move to state “matched an a”
- Hit `'b'`, move to state “done”

Licensed under the Creative Commons Attribution License

… for an important implication

- No sense history means you cannot match nested expressions
- E.g. extract the correct mathematical structure of the following expressions (in general)

```
z ** (1.0 / (1.0 - gamma))
b_star ** ((1.0 - rho) / rho)
```

- Use pyparsing instead

Licensed under the Creative Commons Attribution License

Wrapping up

- Regular expressions indispensable for serious text editing
  - Simple operations in the editor
  - Parsing complicated data
- Slower than standard string operations (large datasets)
- Use it at a small level, even if you're slower for now
  - Keep Friedl (2006) under your pillow
  - Watch the `http://software-carpentry.org/4_0/regexp/` screencasts for transforming data
- You'll soon be annoyed by amateur web programmers . . .

Licensed under the Creative Commons Attribution License

At the end of this lecture you are able to . . .

- Appreciate usefulness of pattern matching in text
  - Everyday text editing
  - Data management
- Understand basic concepts of regular expression engines
- Apply the most important features of regular expressions
  - Everyday text editing
  - Data management

Licensed under the Creative Commons Attribution License

References I

Friedl, Jeffrey E. F. (2006). *Mastering Regular Expressions*. 3rd ed. O'Reilly Media.

Gaudecker, Hans-Martin von, Arthur van Soest, and Erik Wengström (2012). "Experts in Experiments: How Selection Matters for Estimated Distributions of Risk Preferences". In: *Journal of Risk and Uncertainty* 42.2, pp. 159–190.

Licensed under the Creative Commons Attribution License

Acknowledgements

- This course is designed after and borrows a lot from the `Software Carpentry` course designed by `Greg Wilson` for scientists and engineers.
- The Software Carpentry course material is made available under a `Creative Commons Attribution License`, as is this course's material.

License for the course material

[Links to the full `legal` text and the source text for `this` page.] You are free:

- **to Share** to copy, distribute and transmit the work
- **to Remix** to adapt the work

Under the following conditions:

- **Attribution** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

- **Waiver** Any of the above conditions can be waived if you get permission from the copyright holder.
- **Public Domain** Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- **Other Rights** In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
  - The author's moral rights;
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to `this` web page.