Constructing searches

Introduction to Regular Expressions

(*MENA Version*)

Maxim Romanov

To start: Regular Expressions Practicum File on *GitHub:* https://github.com/maximromanov/re_tutorial

- **On Windows or Mac**
  - Install **Sublime Text**
  - Open the practicum file in the editor

What are Regular Expressions?

- very small language for describing textual patterns
- not a programming language, yet a part of each one
- incredibly powerful tool for find/replace operations
- old (1950s-60s)
- arcane art
- ubiquitous

Why Use Regular Expressions?

To search:

- all spelling variations of the same word:
  - Col. Qadhdhafi or Kaddafi? Isfahan or Iṣbahān?
- words of specific morphological patterns:
  - *[root]er, [root]ed, [root]ing [root]s*: all derivatives from the same root
- entities that may be referred to differently:
  - references to Austria?
  - references to education in biographies

To search and replace:

- reformat "dirty"/inconsistent data

To tag:

- make texts navigable and more readable
- tag information relevant to your research

- and many other uses…

The Basics

- a **regular expression** is a pattern enclosed within delimiters
  - delimiters will differ depending on a programming language or software that you use; you may also not see them at all
  - most text editors that support RE do not display delimiters (*EditPad Pro*, *Sublime Text, TextMate*)
- most characters match themselves
- there are also special characters

Example:

- /**Hamburg**/ is a regular expression that matches "**Hamburg**"
  - *slash* is the delimiter enclosing the expression
  - "**Hamburg**" is the pattern

/at/

- Matches strings with “a” followed by “t”.

at	hat
that	atlas
aft	Athens

/at/

- Matches strings with “a” followed by “t”.

at	h***at***
th***at***	***at***las
aft	Athens

Characters & Special Characters

- most characters match themselves
- matching is case sensitive
- special characters: **`()^${}[]\|.+?*`**
- to match a special character in your text, you need to "escape it", i.e. precede it with "****" in your pattern:

  - **`/Baġḍād [sic]/`**
    does not match "**`Baġḍād [sic]`**"

  - **`/Baġḍād \[sic\]/`**
    matches "**`Baġḍād [sic]`**"

Character Classes: [ ]

- Characters within **[]** are choices for a single-character match.
- Think of a type of ***or***.
- Order within **[]** is unimportant.
- **/x[01]/** matches >>> "**x0**" and "**x1**".
- **/[10][23]/** matches >>>
  >>> "**02**", "**03**", "**12**" and "**13**".
- Initial **^** negates the class:
  - **/[^45]/** matches any character except 4 or 5.

/[ch]at/

- Matches strings with “c” or “h”, followed by “a”, followed by “t”.

that	at
chat	cat
fat	phat

/[ch]at/

- Matches strings with “c” or “h”, followed by “a”, followed by “t”.

t***hat***

at

c***hat***

cat

fat

p***hat***

Ranges (within classes)

- Ranges define sets of characters within a class.
  - **`/[1-9]/`** matches any non-zero digit
  - **`/[a-zA-Z]/`** matches any letter of the English alphabet
  - **`/[12][0-9]/`** matches numbers between 10 and 29

Ranges shortcuts

Shortcut	Name	Equivalent Class
\d	digit	**[0-9]**
\D	not digit	**[^0-9]**
\w	word	**[a-zA-Z0-9_]** (*actually more!*)
\W	not word	**[^a-zA-Z0-9_]**
\s	space	**[\t\n\r\f\v]**
\S	not space	**[^\t\n\r\f\v]**
.	everything	**[^\n]** (depends on mode)

/\d\d\d[-]\d\d\d\d/

- Matches strings with:
  - Three digits
  - Space or dash
  - Four digits

501-1234 234 1252

652.2648 713-342-7452

PE6-5000 653-6464x256

/\d\d\d[-]\d\d\d\d/

- Matches strings with:
  - Three digits
  - Space or dash
  - Four digits

501-1234

234 1252

652.2648

713-**342-7452**

PE6-5000

653-6464x256

Repeaters

- Symbols indicating that the preceding element of the pattern can repeat.
- **`/runs?/`** matches *runs* or *run*
- **`/1\d*/`** matches any number beginning with "**`1`**".

Repeater	Count
`?`	zero or one
`+`	one or more
`*`	zero or more
`{n}`	exactly *n*
`{n,m}`	between *n* and *m* times
`{,m}`	no more than *m* times
`{n,}`	at least *n* times

Repeaters

Strings:

```
1: "at"      2: "art"
3: "arrrrt"  4: "aft"
```

Patterns:

```
A: /ar?t/     B: /a[fr]?t/
C: /ar*t/     D: /ar+t/
E: /a.*t/     F: /a.+t/
```

Repeater	Count
`?`	zero or one
`+`	one or more
`*`	zero or more
`{n}`	exactly *n*
`{n,m}`	between *n* and *m* times
`{,m}`	no more than *m* times
`{n,}`	at least *n* times

Repeaters

```
1: "at"          2: "art"
3: "arrrrt"      4: "aft"
```

- **/ar?t/** matches “at” and “art” but not “arrrt”.
- **/a[fr]?t/** matches “at”, “art”, and “aft”.
- **/ar*t/** matches “at”, “art”, and “arrrrt”
- **/ar+t/** matches “art” and “arrrt” but not “at”.
- **/a.*t/** matches anything with an ‘a’ eventually followed by a ‘t’.

Lab: Intro (`in the practicum file`)

Repeater	Count
`?`	zero or one
`+`	one or more
`*`	zero or more
`{n}`	exactly *n* times
`{n,m}`	between *n* and *m* times
`{,m}`	no more than *m* times
`{n,}`	at least *n* times

Shortcut	Name
`\d`	digit
`\D`	not digit
`\w`	word
`\W`	not word
`\s`	space
`\S`	not space
`.`	any symbol

Anchors

- Anchors match between characters.
- Used to assert that the characters you are matching must appear in a certain place.
- `/\bat\b/` matches "`at work`" but not "`batch`".

Anchor	Matches
`^`	start of line
`$`	end of line
`\b`	*word boundary*
`\B`	not boundary
`\A`	start of string (rare)
`\Z`	end of string (rare)
`\z`	raw end of string (rare)

Alternation – "|" (pipe)

- In **RE**, "|" means "or".
- You can put a full expression on the left and another full expression on the right.
- Either can match.
- **`/seek|seeks|sought/`**
  - matches "seek", "seeks", or "sought".
- **`/seeks?|sought/`**
  - matches "seek", "seeks", or "sought".

Grouping

- Everything within **`(`** ... **`)`** is grouped into a single element for the purposes of *repetition* and *alternation*.
- The expression **`/(la)+/`** matches "**`la`**", "**`lala`**", "**`lalalala`**" but not "**`all`**".
- **`/schema(ta)?/`** matches "**`schema`**" and "**`schemata`**" but not "**`schematic`**".

Grouping Example

- What regular expression matches "**`eat`**", "**`eats`**", "**`ate`**" and "**`eaten`**"?

Grouping Example

- What regular expression matches "**`eat`**", "**`eats`**", "**`ate`**" and "**`eaten`**"?
- **`/eat(s|en)?|ate/`**

- Add word boundary anchors to exclude "**`sate`**" and "**`eating`**":
- **`/\b(eat(s|en)?|ate)\b/`**

Lab: Part I (in the practicum file)

Repeater	Count
`?`	zero or one
`+`	one or more
`*`	zero or more
`{n}`	exactly *n* times
`{n,m}`	between *n* and *m* times
`{,m}`	no more than *m* times
`{n,}`	at least *n* times

Shrtct	Name
`\d`	digit
`\D`	not digit
`\w`	word
`\W`	not word
`\s`	space
`\S`	not space
`.`	any symbol

Anchor	Matches
`^`	start of line
`$`	end of line
`\b`	word boundary
`\t`	TAB symbol
`\n`	new line
`|`	“or” alternation
`(...)`	capture group
`[...]`	class

Replacement

- Regex most often used for search/replace
- Text editors:
  - Search Window: **`pattern`**
  - Replace Window: **`replacement`**

Capture

- During searches, `(` … `)` groups capture patterns for use in replacement.
- Special variables `\1`, `\2`, `\3` etc. contain the capture
  - in *Sublime Text*: `$1`, `$2`, `$3`
- `/(\d\d\d)-(\d\d\d\d)/` "`123-4567`"
  - `\1` (`$1`) contains "`123`"
  - `\2` (`$2`) contains "`4567`"

Capture & Reformat

- How to convert “Schwarzenegger, Arnold” to “Arnold Schwarzenegger”?
-
-
-
-

Capture & Reformat

- How to convert “Schwarzenegger, Arnold” to “Arnold Schwarzenegger”?

- Search: **`/(\w+), (\w+)/`**
- Replace (a): **`/\2 \1/`**
- Replace (b): **`/$2 $1/`**

- **(!) Before hitting “Replace”, make sure that your match does not catch what you do NOT want to change**

Lab: Part II (in the practicum file)

Repeater	Count
`?`	zero or one
`+`	one or more
`*`	zero or more
`{n}`	exactly *n* times
`{n,m}`	between *n* and *m* times
`{,m}`	no more than *m* times
`{n,}`	at least *n* times

Shrtct	Name
`\d`	digit
`\D`	not digit
`\w`	word
`\W`	not word
`\s`	space
`\S`	not space
`.`	any symbol

Anchor	Matches
`^`	start of line
`$`	end of line
`\b`	word boundary
`\t`	TAB symbol
`\n`	new line
`|`	“or” alternation
`(...)`	capture group
`[...]`	class

Finding Toponyms

- *Very Simple*: Construct regular expressions that finds references all Austrian cities.

Finding Toponyms

- *Very Simple*: Construct regular expressions that finds references all Austrian cities.

- Simply connect all toponyms from the list with a pipe symbol "|"

Finding Toponyms

- *A Bit Tricky*: Construct regular expression that finds only cities from 1) Lower Austria; 2) Salzburg.

Finding Toponyms

- *A Bit Tricky*: Construct regular expression that finds only cities from 1) Lower Austria; 2) Salzburg.

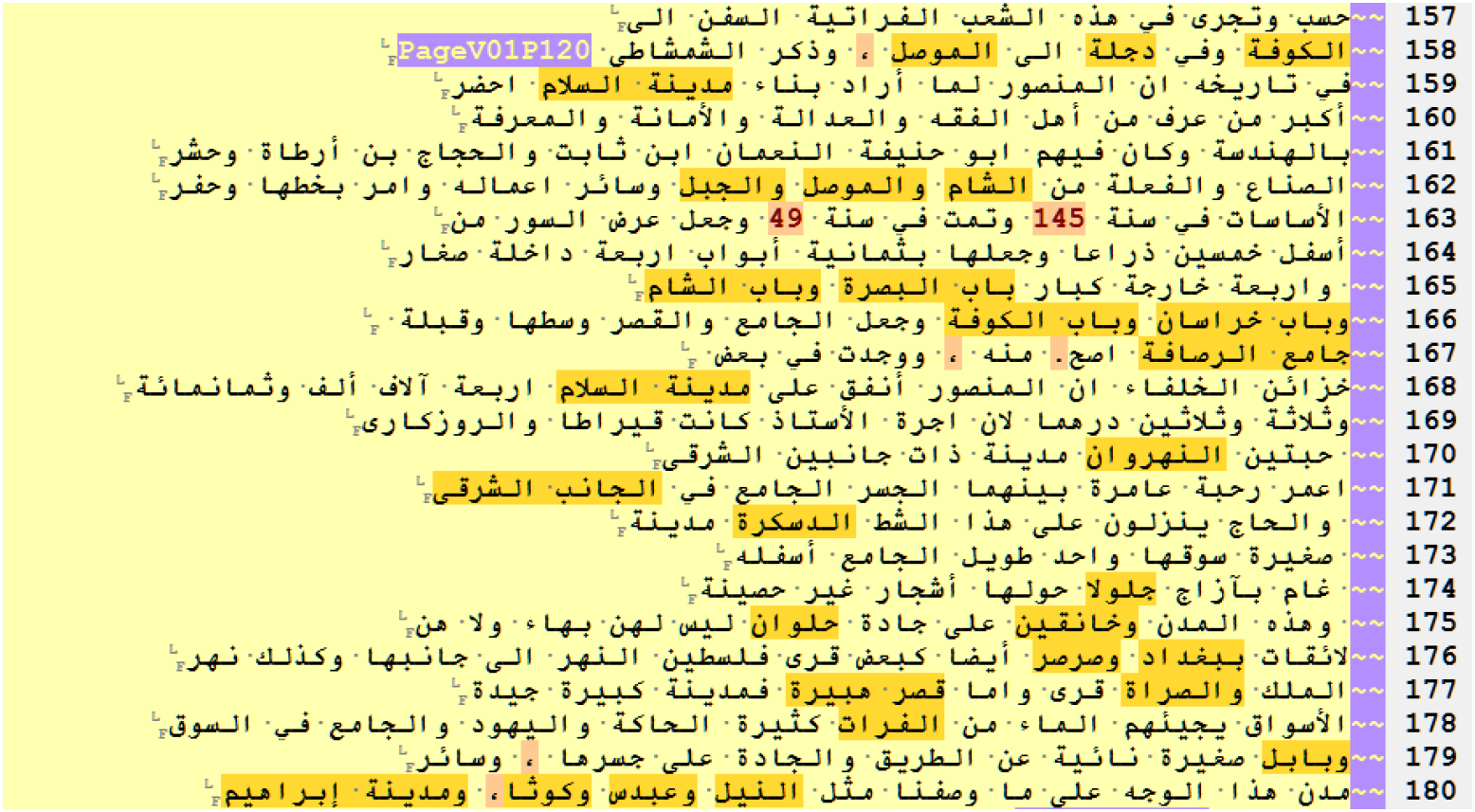
- Option I:

```
\b([\w ]+) \(Lower Austria\)
\b([\w ]+) \(Salzburg\)
```

- Option II (cooler):

```
\b([\w ]+)(?=( \(Lower Austria\)))
\b([\w ]+)(?=( \(Salzburg\)))
```

RE for Highlighting Toponyms



RE for Finding Date Statements

- Download settings and examples for EditPad Pro: https://tinyurl.com/tutorial-re

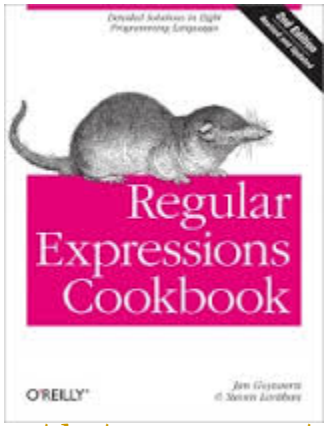
- In EditPad Pro:
  - Open: "**`0748Dhahabi.TarikhIslam.Shamela0035100-ara1.sample`**"
  - Open: "**`year_Statement_re.txt`**"
    - Copy/paste the regular expression into the search window
  - Switch back to "**`0748Dhahabi.TarikhIslam.Shamela0035100-ara1.sample`**"
- The date statements in the excerpt from al-Ḏahabī's *Taʾrīḫ al-islām* should appear highlighted with a different color

RE for Finding Date Statements

To keep in mind

- RE are "greedy," i.e. they tend to catch more than you may need. Always test!
- Test before applying! (In text editors *Ctrl+Z* (*Win*), *Cmd+Z* (*Mac*) can help to revert changes)
- Check the language/application-specific documentation: some common shortcuts are not universal (**`\1`** vs **`$1`**, for example)

Some readings



- Amazon.com
  - http://www.amazon.com/Regular-Expressions-Cookbook-Jan-Goyvaerts/dp/1449319432/
  - http://www.amazon.com/Mastering-Regular-Expressions-Jeffrey-Friedl/dp/0596528124/
- Free Online Readings
  - http://www.regular-expressions.info/
  - http://ruby.bastardsbook.com/chapters/regexes/
- Cheat Sheets
  - http://krijnhoetmer.nl/stuff/regex/cheat-sheet/
  - http://www.rexegg.com/regex-quickstart.html
- Interactive tutorial
  - http://regexone.com/