JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

Introduction to Python for Biologists - Regex in a nutshell

Katerina Taškova[1] Jean-Fred Fontaine[1,2]

[1]Faculty of Biology, Johannes Gutenberg-Universität Mainz, Mainz, Germany

[2]Genomics and Computational Biology, Kernel Press, Mainz, Germany

`https://cbdm.uni-mainz.de/mb17`

March 23, 2017

Regular Expressions

- Regular expressions (REs, regexes, or regex patterns) are a powerful language for searching text patterns in strings
- Classical example

```
1 import re                              # import re module
2 str = 'I can see 123 cats!'            # Example string
3
4 match = re.search(r'1..', str) # Search pattern 1..
5
6 if match:
7     print(match.group())               # if found print '123'
8 else:
9     print("Not found")                 # otherwise match == None
```

Basic Patterns

Pattern	Match
a, X, 9, <	ordinary characters match themselves exactly
.	a period matches any single character except newline
\w	matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_]
\W	matches any non-word character
\b	boundary between word and non-word
\s	a single whitespace character – space, newline, return, tab, form [\n \r \t \f]
\S	matches any non-whitespace character
\t	tab
\n	newline
\r	return
\d	decimal digit [0-9]
^	circumflex (top hat) matches the start of a string
$	dollar matches the end of a string
\	inhibits the "specialness" of a character. So, for example, use \. to match a period

Table: Regular expressions: basic patterns

Basic examples

The basic rules of RE search for a pattern within a string are:

- The search proceeds through the string from start to end, stopping at the first match found
- All of the pattern must be matched, but not all of the string

```
match = re.search(r'iii', 'piiig')                    # 'iii'
match = re.search(r'igs', 'piiig')                    # None
match = re.search(r'..g', 'piiig')                    # 'iig'
match = re.search(r'\d\d\d', 'p123g')                 # '123'
match = re.search(r'\w\w\w', '@@abcd!!')              # 'abc'
match = re.search(r'\w*H\w*', 'You Hello You') # 'Hello'
```

Repetitions I

Repetitions are defined using +, *, ? and { }

- **A+** for 1 or more A's
- **B*** for 0 or more B's
- **C?** for 0 or 1 C
- **D{5}** for 5 D's
- **E{6,10}** for 6 to 10 E's

Leftmost and Largest ("greedy")

- First the search finds the leftmost match for the pattern, and second it tries to use up as much of the string as possible

Repetitions II

```
# simple repetitions
re.search(r'pi+',    'piiig'    ).group() # piii
re.search(r'pi?',    'ap'       ).group() # p
re.search(r'pi?',    'apii'     ).group() # pi
re.search(r'pi*',    'ap'       ).group() # p
re.search(r'pi*',    'apii'     ).group() # pii
re.search(r'pi{3}', 'apiiiii' ).group() # piii
re.search(r'i+',     'piigiiii').group() # ii (1st hit only)

# 3 digits possibly separated by whitespaces (\s*)
re.search(r'\d\s*\d\s*\d', 'xx1 2   3xx').group() # "1 2   3"
re.search(r'\d\s*\d\s*\d', 'xx12  3xx'  ).group() # "12  3"
re.search(r'\d\s*\d\s*\d', 'xx123xx'    ).group() # "123"
```

Sets of characters I

Square brackets indicate a set of characters

- **[ABC]** matches 'A' or 'B' or 'C'.
  - The codes \w, \s etc. work inside square brackets too with the one exception that dot (.) just means a literal dot
- **[a-z]** for lowercase alphabetic characters
- **[a-zA-Z]** for alphabetic characters
- **[AB-]** for A, B or dash
- **[^AB]** for any character except A or B.

Sets of characters II

```
1  str = 'purple alice-b@google.com monkey dishwasher'
2
3
4  # Example 1
5  match = re.search(r'\w+@\w+', str)
6
7  if match:
8      print match.group()   ## 'b@google'
9
10
11
12 # Example 2
13 match = re.search(r'[\w.-]+@[\w.-]+', str)
14
15 if match:
16     print match.group()   ## 'alice-b@google.com'
```

Functions I

Selected functions:

- **re.search()**
  - if found returns a **Match object** of 1st occurrence
    - **match.start()** returns start index
    - **match.end()** returns end index
    - **match.span()** returns start and end index in a tuple
    - **match.group()** returns matched string
  - if not found returns **None**
- **re.findall()**
  - if found returns a **list** of all matched sub strings
  - if not found returns an **empty list**

Functions II

```
1  import re
2  seq = "RPAPPDRAPDQX" # A sequence
3  expr = 'A.{1,2}D'    # A and D separated by 1 or 2 characters
4
5  match = re.search(expr, seq)
6  if match:
7      print(
8          match.start(),          # start index
9          match.end(),            # end index
10         match.span(),           # start and end index
11         match.group(),          # the matched string
12         sep=' - '
13     )
14 # 2 - 6 - (2, 6) - APPD
15
16 matches = re.findall(expr, seq) # Found 2 occurrences
17 if match:
18     print(matches)                  # ['APPD', 'APD']
```

Group Extraction

- Groups are defined with parentheses
- On a successful search
  - match.group(): the whole match text
  - match.group(1): match text of 1st left parenthesis
  - match.group(2): match text of 2nd left parenthesis
  - ...

```
1 import re
2 str = 'purple alice-b@google.com monkey dishwasher'
3
4 match = re.search('([\w.-]+)@([\w.-]+)', str)
5 if match:
6     print(match.group())   ## 'alice-b@google.com'
7     print(match.group(1))  ## 'alice-b'
8     print(match.group(2))  ## 'google.com'
```

Group Extraction and Findall

- Pattern contains 1 set of parenthesis defining a group
  - findall returns a **list of strings** for that single group
- Pattern contains >1 sets of parenthesis defining groups
  - findall returns a **list of tuples**. Each tuple represents one match of the pattern, and inside the tuple is the data of **group(1)**, **group(2)** ...

```
1 str = 'alice@google.com, monkey bob@abc.com dishwasher'
2
3 matches = re.findall(r'([\w\.-]+)@', str)
4 print(matches)
5 # ['alice', 'bob']
6
7 tuples = re.findall(r'([\w\.-]+)@([\w\.-]+)', str)
8 print(tuples)
9 # [('alice', 'google.com'), ('bob', 'abc.com')]
```

Ignore case

- The IGNORECASE flag is added as an extra argument to the **search()** or **findall()** etc.
- e.g. **re.search(pat, str, re.IGNORECASE)**

Substitution

- **re.sub(expression, replacement, string)**

```
1 text1 = 'alice@google.com and bob@abc.net'
2 text2 = re.sub(r'\.\w+', r'.de', text1)
3 print (text2)
4 # alice@google.de and bob@abc.de
```

- **\1, \2** ... in replacement refer to match group(1), group(2) ...

```
1 text1 = 'alice@google.com and bob@abc.com'
2 text2 = re.sub(
3     r'([\w\.-]+)@([\w\.-]+)', # Expression
4     r'\2@\1',                 # Replacement string
5     str)                      # Input string
6 print (text2)
7 ## google.com@alice and abc.com@bob
```

Exercise

URL

- `https://cbdm.uni-mainz.de/mb17`

Jupyter Notebook

- File: Regex.ipynb
- Download the file into the notebooks folder

Data file

- File: sequences.tsv
- Download the file into the data folder