Utrecht
Bioinformatics
Center

Regular Expression

Day 4b – Introduction to Python

UMC Utrecht

Universiteit Utrecht

Hubrecht Institute

Utrecht
Bioinformatics
Center

Regular Expression

- Wikipedia:
- “a regular expression (regex) is a sequence of characters that forms a search pattern.”

- What can we do with them?

- 1. Find matches in target (strings)
- 2. Substitute patterns

Regular Expression

regularexpressions

Anchors

^	Start of line +
\A	Start of string +
$	End of line +
\Z	End of string +
\b	Word boundary +
\B	Not word boundary +
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\xhh	Hexadecimal character hh
\Oxxx	Octal character xxx

POSIX Character Classes

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion +
?!	Negative lookahead +
?<=	Lookbehind assertion +
?!= or ?<!	Negative lookbehind +
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Note Items marked + should work in most regular expression implementations.

Sample Patterns

([A-Za-z0-9-]+)	Letters, numbers and hyphens
(\d{1,2}\/\d{1,2}\/\d{4})	Date (e.g. 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	jpg, gif or png image
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Any number from 1 to 50 inclusive
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Valid hexadecimal colour code
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	8 to 15 character string with at least one upper case letter, one lower case letter, and one digit (useful for passwords).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Email addresses
(\<(/?[^\>]+)\>)	HTML Tags

Note *These patterns are intended for reference purposes and have not been extensively tested. Please use with caution and test thoroughly before use.*

Quantifiers

*	0 or more +
*?	0 or more, ungreedy +
+	1 or more +
+?	1 or more, ungreedy +
?	0 or 1 +
??	0 or 1, ungreedy +
{3}	Exactly 3 +
{3,}	3 or more +
{3,5}	3, 4 or 5 +
{3,5}?	3, 4 or 5, ungreedy +

Special Characters

\	Escape Character +
\n	New line +
\r	Carriage return +
\t	Tab +
\v	Vertical tab +
\f	Form feed +
\a	Alarm
[\b]	Backspace
\e	Escape
\N{name}	Named Character

String Replacement (Backreferences)

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string
$_	Entire input string
$$	Literal "$"

Ranges

.	Any character except new line (\n) +
(a|b)	a or b +
(...)	Group +
(?:...)	Passive Group +
[abc]	Range (a or b or c) +
[^abc]	Not a or b or c +
[a-q]	Letter between a and q +
[A-Q]	Upper case letter + between A and Q +
[0-7]	Digit between 0 and 7 +
\n	nth group/subpattern +

Note *Ranges are inclusive.*

Pattern Modifiers

g	Global match
i	Case-insensitive
m	Multiple lines
s	Treat string as single line
x	Allow comments and white space in pattern
e	Evaluate replacement
U	Ungreedy pattern

Metacharacters (must be escaped)

^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Available free from AddedBytes.com

RegEx - Intro

- Keep it simple: Operators

- Example: Find EcoRI restriction enzyme site in sequence.

```
EcoRI = “GAATTC”
DNAseq = "TGCATAGCGAATTCGGACGT"
EcoRI in DNAseq
True
```

RegEx - Intro

- Find Eco13kl restriction site CCNGG

```
Eco13kl = “CCNGG”
DNAseq = "CCTGGAGCCCAGGGGACGT"
Eco13kl in DNAseq
False
```

- CCNGG = CCAGG or CCTGG or CCCGG or CCGGG

```
Eco13kl =[“CCAGG”,”CCTGG”,”CCCGG”,”CCGGG”]
DNAseq = "CCTGGAGCCCAGGGGACGT"
Eco13kl[0] in DNAseq          True
Eco13kl[1] in DNAseq          True
Eco13kl[2] in DNAseq          False
Eco13kl[3] in DNAseq          False
```

RegEx - module

- For complicated patterns use Regex module “re”

```
import re
```

- Background information:

- https://docs.python.org/3/library/re.html

RegEx – re.findall

- re.findall(pattern, string, flags=0)

- Return all non-overlapping matches of pattern in string, as a list of strings.

- Pattern → pattern to search
- String → string to search in
- Flags → optional flags/modifiers

RegEx – metacharacters

- represent one or multiple characters you want to search for in a string

- Some examples of metacharacters:

^	Matches beginning of line
$	Matches end of line
.	Matches any single character except newline
[...]	Matches any single character in brackets
[^...]	Matches any single character not in brackets
a | b	Matches either a or b

RegEx – metacharacters

^	**Matches beginning of line.**
$	**Matches end of line.**
.	**Matches any single character except newline. Using m option allows it to match newline as well.**
[...]	**Matches any single character in brackets.**
[^...]	**Matches any single character not in brackets**
re*	**Matches 0 or more occurrences of preceding expression.**
re+	**Matches 1 or more occurrence of preceding expression.**
re?	**Matches 0 or 1 occurrence of preceding expression.**
re{ n}	**Matches exactly n number of occurrences of preceding expression.**
re{ n,}	**Matches n or more occurrences of preceding expression.**
re{ n, m}	**Matches at least n and at most m occurrences of preceding expression.**
a| b	**Matches either a or b.**
(re)	**Groups regular expressions and remembers matched text.**
(?imx)	Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?-imx)	Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?: re)	Groups regular expressions without remembering matched text.
(?imx: re)	Temporarily toggles on i, m, or x options within parentheses.
(?-imx: re)	Temporarily toggles off i, m, or x options within parentheses.

(?#...)	Comment.
(?= re)	Specifies position using a pattern. Doesn't have a range.
(?! re)	Specifies position using pattern negation. Doesn't have a range.
(?> re)	Matches independent pattern without backtracking.
\w	**Matches word characters.**
\W	**Matches non-word characters.**
\s	**Matches whitespace. Equivalent to [\t\n\r\f].**
\S	**Matches nonwhitespace.**
\d	**Matches digits. Equivalent to [0-9].**
\D	**Matches nondigits.**
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\G	Matches point where last match finished.
\b	Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
\B	Matches nonword boundaries.
\n, \t, etc.	**Matches newlines, carriage returns, tabs, etc.**
\1...\9	Matches nth grouped subexpression.
\10	Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

RegEx – FLAGS options

- Regex are case sensitive (by default)!

- re.I IGNORECASE
  → perform case-insensitive matching

- re.S DOTALL
  → make the '.' special character match any character at all, including a newline
  Default without this flag= '.' will match anything except a newline.

- re.M MULTILINE
  → the pattern character '^' matches at the beginning of the string and at the beginning of each line (immediately following each newline); and the pattern character '$' matches at the end of the string and at the end of each line (immediately preceding each newline).
  → Default (without this flag)= '^' matches only at the beginning of the string, and '$' only at the end of the string and immediately before the newline (if any) at the end of the string.

- re.X (VERBOSE), re.L (LOCALE), re.U (UNICODE)

RegEx – re.split

- Split string by the occurrences of pattern

re.split(pattern, string, maxsplit=0, flags=0)

pattern	→ pattern to search
string	→ string to search in
maxsplit	→ maximum number of splits (0 = infinite)
flags	→ optional flags/modifiers

RegEx – re.sub

- re.sub(pattern, repl, string, count=0, flags=0)

- Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl.

pattern	→ pattern to search
repl	→ string to substitute the pattern with
string	→ string to search in
count	→ maximum number of substitutions (0 = infinite)
flags	→ optional flags/modifiers

RegEx – re.sub - Groups

- Groups are very handy for substitutions

- Use the following grouping on the line below:

```
re.sub("(gr[ae]y)", "\g<1>blue", line)

line = "My computer should be grey and my car should also be gray"
```

\g<1> stands for group 1 = the first group between ()
\g<2> stands for group 2, etc..

- Try to understand what happens in the following example:

```
re.sub("(gr[ae]y)(\D*)(gr[ae]y)", "\g<1>blue\g<2>not \g<3>black", line)
```

RegEx – re.search

- re.search (pattern, string, flags=0)

Scan through string looking for the first location (only) where the regular expression pattern produces a match, and return a corresponding **MatchObject instance**.

```
import re
line = "TGCATAGCGAATTCGAGCGT"
match_output = re.search("GAATTC",line)
print match_output

<_sre.SRE_Match object at 0x10c2827e8>

if match_output:
print "GAATTC site found!"

print match_output.group()        # GAATTC
print match_output.start()        # 8
print match_output.end()          # 14
print match_output.span()         # (8, 14)
```

RegEx – other methods

- re.subn(pattern, repl, string, count=0, flags=0)

  Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl. If the pattern isn't found, string is returned unchanged. Returns a tuple with the number of changes.

- re.escape(string)

  Return string with all non-alphanumerics backslashed

- Returns Objects
- re.match(pattern, string, flags=0)

  If zero or more characters at the beginning of string match the regular expression pattern, return a corresponding MatchObject instance

- re.finditer(pattern, string, flags=0)

  Return an iterator yielding MatchObject instances over all non-overlapping matches for the RE pattern in string

- Compile regex (generally for speed purposes)
- re.compile(pattern, flags=0)

  Compile a regular expression pattern into a regular expression object

Regular Expression

regularexpressions

Anchors

^	Start of line +
\A	Start of string +
$	End of line +
\Z	End of string +
\b	Word boundary +
\B	Not word boundary +
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\xhh	Hexadecimal character hh
\Oxxx	Octal character xxx

POSIX Character Classes

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion +
?!	Negative lookahead +
?<=	Lookbehind assertion +
?!= or ?<!	Negative lookbehind +
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Note Items marked + should work in most regular expression implementations.

Sample Patterns

([A-Za-z0-9-]+)	Letters, numbers and hyphens
(\d{1,2}\/\d{1,2}\/\d{4})	Date (e.g. 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	jpg, gif or png image
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Any number from 1 to 50 inclusive
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Valid hexadecimal colour code
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	8 to 15 character string with at least one upper case letter, one lower case letter, and one digit (useful for passwords).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Email addresses
(\<(/?[^\>]+)\>)	HTML Tags

Note *These patterns are intended for reference purposes and have not been extensively tested. Please use with caution and test thoroughly before use.*

Quantifiers

*	0 or more +
*?	0 or more, ungreedy +
+	1 or more +
+?	1 or more, ungreedy +
?	0 or 1 +
??	0 or 1, ungreedy +
{3}	Exactly 3 +
{3,}	3 or more +
{3,5}	3, 4 or 5 +
{3,5}?	3, 4 or 5, ungreedy +

Special Characters

\	Escape Character +
\n	New line +
\r	Carriage return +
\t	Tab +
\v	Vertical tab +
\f	Form feed +
\a	Alarm
[\b]	Backspace
\e	Escape
\N{name}	Named Character

String Replacement (Backreferences)

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string
$_	Entire input string
$$	Literal "$"

Ranges

.	Any character except new line (\n) +
(a|b)	a or b +
(...)	Group +
(?:...)	Passive Group +
[abc]	Range (a or b or c) +
[^abc]	Not a or b or c +
[a-q]	Letter between a and q +
[A-Q]	Upper case letter + between A and Q +
[0-7]	Digit between 0 and 7 +
\n	nth group/subpattern +

Note *Ranges are inclusive.*

Pattern Modifiers

g	Global match
i	Case-insensitive
m	Multiple lines
s	Treat string as single line
x	Allow comments and white space in pattern
e	Evaluate replacement
U	Ungreedy pattern

Metacharacters (must be escaped)

^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Available free from AddedBytes.com

Exercises