Python: regular expressions

Christoph Schubert

School of Information, Zhejiang University of Finance and Economy

April 17, 2017

Before we start: Python raw strings

- Regular expressions often make use of the \ character to express special *character classes*
- but \ is also used inside Python strings to as an escape character
- so if we want to write a character class we will need to write \\ inplace of \
- is quickly becomes complicated, one solution: use raw strings
- raw strings are prefixed by an `r`, for example:

  ```
  r"raw s\tring"
  ```

  observe what happens when we drop to `r` in front
- we often use raw strings when writing regular expressions in Python.

Regular expressions

- specifically formatted strings
- to describe types of strings
- we say a string *matches* a regular expression if it is described by the expression

Regular expression: examples 1

```
Regex                Matches any string that
=============================================================
hello                contains {hello}
gray|grey            contains {gray, grey}
gr(a|e)y             contains {gray, grey}
gr[ae]y              contains {gray, grey}
b[aeiou]bble         contains {babble, bebble, bibble,
                               bobble, bubble}
[b-chm-pP]at|ot      contains {bat, cat, hat, mat, nat, oat, pat,
                               Pat, ot}
colou?r              contains {color, colour}
rege(x(es)?|xps?)    contains {regex, regexes, regexp, regexps}
go*gle               contains {ggle, gogle, google,
                               gooogle, goooogle, ...}
go+gle               contains {gogle, google, gooogle,
                               goooogle, ...}
g(oog)+le            contains {google, googoogle, googoogoogle,
                               googoogoogoogle, ...}
z{3}                 contains {zzz}
```

Regular expressions in Python

- Use `re`-module.
- Commonly used functions:
  - `re.search(pattern, string, flags=0)` look in string for first matching place
  - `re.match(pattern, string, flags=0)` look at beginning of string
  - `re.fullmatch(pattern, string, flags=0)` check if entire string matches the pattern

  All these functions return `None` when no match was found, and a `match` object otherwise:

```
import re
m = re.search("ab", "xyabc")
m.start()
```

More regular expression functions in Python

- `re.split(pattern, string)` split the string as occurences of pattern:

  ```
  re.split('\W+', 'Words, words, words.')

  ['Words', 'words', 'words', '']
  ```

- `re.findall(pattern, string)` find all (non-overlapping) occurrences of pattern in string
- `re.sub(pattern, repl, string)` replaces non-overlapping occurences of pattern by repl.

Regular expression: examples 2

Regex	Matches any string that
`z{3,6}`	contains {zzz, zzzz, zzzzz, zzzzzz}
`z{3,}`	contains {zzz, zzzz, zzzzz, ...}
`[Rr]ains\*\*`	contains {Rains**, rains**}
`\d`	contains {0,1,2,3,4,5,6,7,8,9}
`\d{6}`	contains a Chinese postal code
`\d{5}(-\d{4})?`	contains a United States postal code
`1\d{10}`	contains 11-digit string starting with a 1
`[2-9]|[12]\d|3[0-6]`	contains an integer in the range 2..36 inclusive
`Hello\nworld`	contains Hello followed by a newline followed by world
`b..b`	contains a four-character (sub)string beginning and ending with a b
`\d+(\.\d\d)?`	contains a positive integer or a floating point number with exactly two characters after the decimal point.

Regular expression: examples 3

Regex	Matches any string that
sh[^io]t	contains sh followed by any character other than an i or o, followed by t
//[^\r\n]*[\r\n]	contains a Java or C++ slash-slash comment
^dog	begins with "dog"
dog$	ends with "dog"
^dog$	is exactly "dog"

Notation

Many different way of writing regular expressions, general rules:

- Most characters stand for themselves
- Certain characters, called metacharacters, have special meaning and must be escaped (usually with \) if you want to use them as characters. The metacharacters (a.k.a. the dirty dozen) are:

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

Using Regular Expressions

- Validate that a piece of text (or a portion of that text) matches some pattern
- Find fragments of some text that match some pattern
- Extract fragments of some text
- Replace fragments of text with other text

Parts of regular expressions (1/2)

- dot character `.`
  - matches any single character except new line
- several characters enclosed in square brackets `[]`
  - matches any single listed character
- ranges using a dash `-` placed between 2 characters
  - match any character lexicographically within range
  - example: `[0-35-9]`
    - 0-3 means a number between 0 and 3
    - 5-9 means a number between 5 and 9
    - both between [and] means either of the choices
    - thus: any number except 4
- carrot `^` in character set
  - if first character in set is a `^` characters not listed in the set are matched
  - example: `[^b]at` matches `cat` and `hat`, but *not* `bat`

Parts of regular expressions (2/2)

- | match expression to the left or to the right
  - Example: `grey|gray`
    - matches `grey` or `gray`,
    - does not match any other sting
- () allow sub-expressions to be grouped
- Quantifiers:

Quantifier	meaning
*	matches zero or more occurences
+	matches one or more occurences
?	matches zero or one occurences
{n}	matches exactly *n* occurences
{m-n}	matches between *m* and *n* occurences

- A+ versus A*
  - both match strings A, AA,AAA, etc
  - only A* matches empty string

Substring versus entire string

- any matching substring may be returned
  - ^ character matches beginning of string
  - $ character matches end of string

  regular expression enclosed between ^ and $ determines whether entire string matches
- Quantifiers
  - applied to sub-expressions enclosed in parentheses
  - match sub-expression multiple times
  - Greedy behaviour
    - match as many occurrences as possible for successful match
    - set by default: all quantifiers are greedy
  - Lazy behaviour
    - invoked by following any quantifier with ? characters
    - match as few occurrences as possible for successful match

Examples for regular expressions

- `^[A-Z][a-zA-Z]*$`

  one upper-case letter followed by zero or more upper or lower-case letter
- `^[0-9]+\s+([a-zA-Z]+|[a-zA-Z]+\s[a-zA-Z]+)$`

  check (North American) phone number
- `^([a-zA-Z]+|[a-zA-Z]+\s[a-zA-Z]+)$`

  check city name
- `^\d{5}$`

  check zip code format: 5 digits
- `^[1-9]\d{2}-[1-9]\d{2}-\d{4}$`

  check phone number format

Character classes

- `.` Any character (may or may not match line terminators)
- `\d` A digit: `[0-9]`
- `\D` A non-digit: `[^0-9]`
- `\s` A whitespace character: `[ \t\n\x0B\f\r]`
- `\S` A non-whitespace character: `[^\s]`
- `\w` A word character: `[a-zA-Z_0-9]`
- `\W` A non-word character: `[^\w]`

Useful regular expressions

No need to understand them now!

- Internet E-Mail Address
  `\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*`
- Internet URL
  `http://([\w-]+\.)+[\w-]+(/[\w- ./?\%\&=]*)?`