Introduction to Python Programming

20 – Regular Expressions (Advanced Topic)

Josef van Genabith (Stefan Thater)
Dept. of Language Science & Technology
Universität des Saarlandes

WS 2022/23

Some common string operations in Python

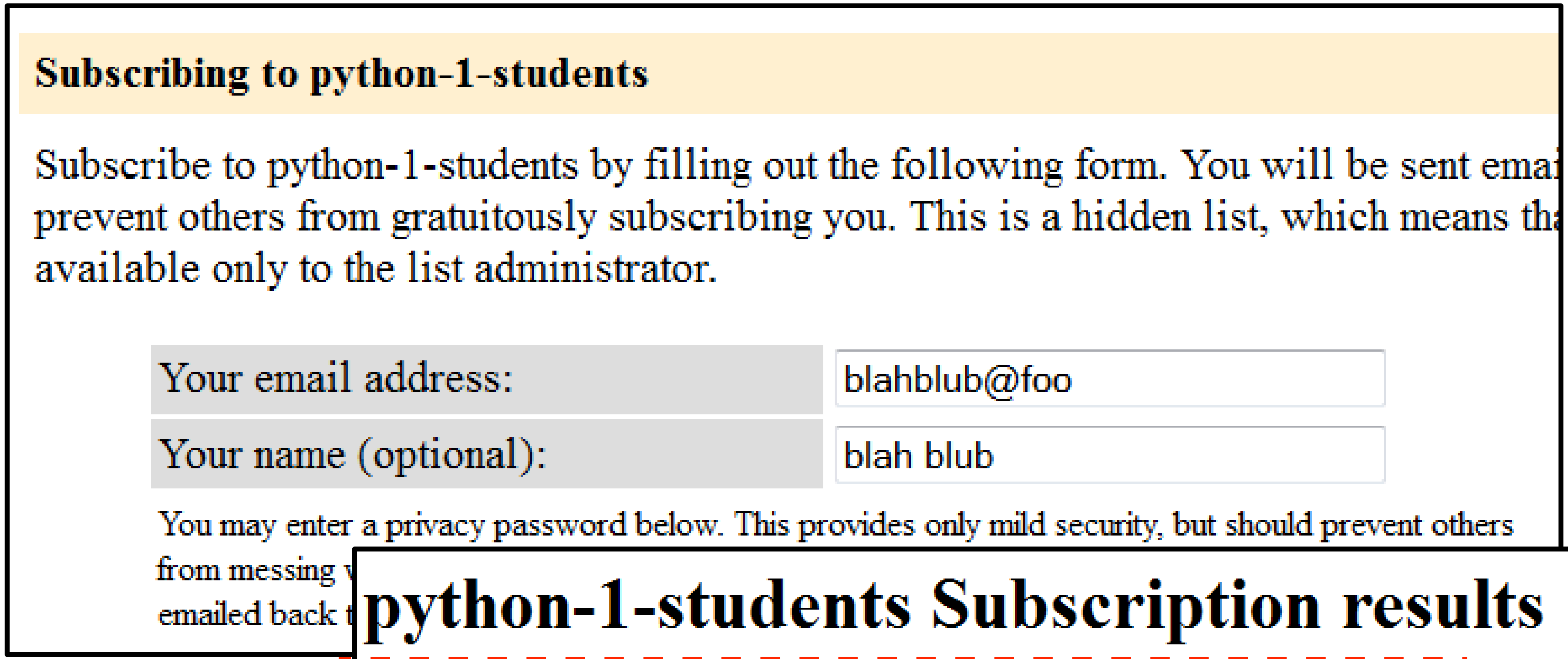
- s.startswith("The")
  - does string s start with "The"?
- s.endswith(".")
  - does string s end with "."?
- s.find("the")
  - does the string "the" occur in s?
  - returns the index or -1 if not found
- s.replace("the", "a")
  - replaces all occurrences of "the" in s by "a"

Text Search: why is this useful?

- Example: extraction of events & when they happened
- First step: find the dates in the text

```
Albert Einstein was born in Ulm, in the Kingdom of
Württemberg in the German Empire on 14 March 1879. In
September 1896, he passed the Swiss Matura with mostly
good grades, including a top grade of 6 in physics and
mathematical subjects, on a scale of 1-6, and, though
only seventeen, enrolled in the four- year mathematics
and physics teaching diploma program at the Zurich
Polytechnic. Einstein and Maric´married in January 1903.
In May 1904, the couple's first son, Hans Albert
Einstein, was born in Bern, Switzerland. Their second
son, Eduard, was born in Zurich in July 1910.
```

Why is text search useful: validation

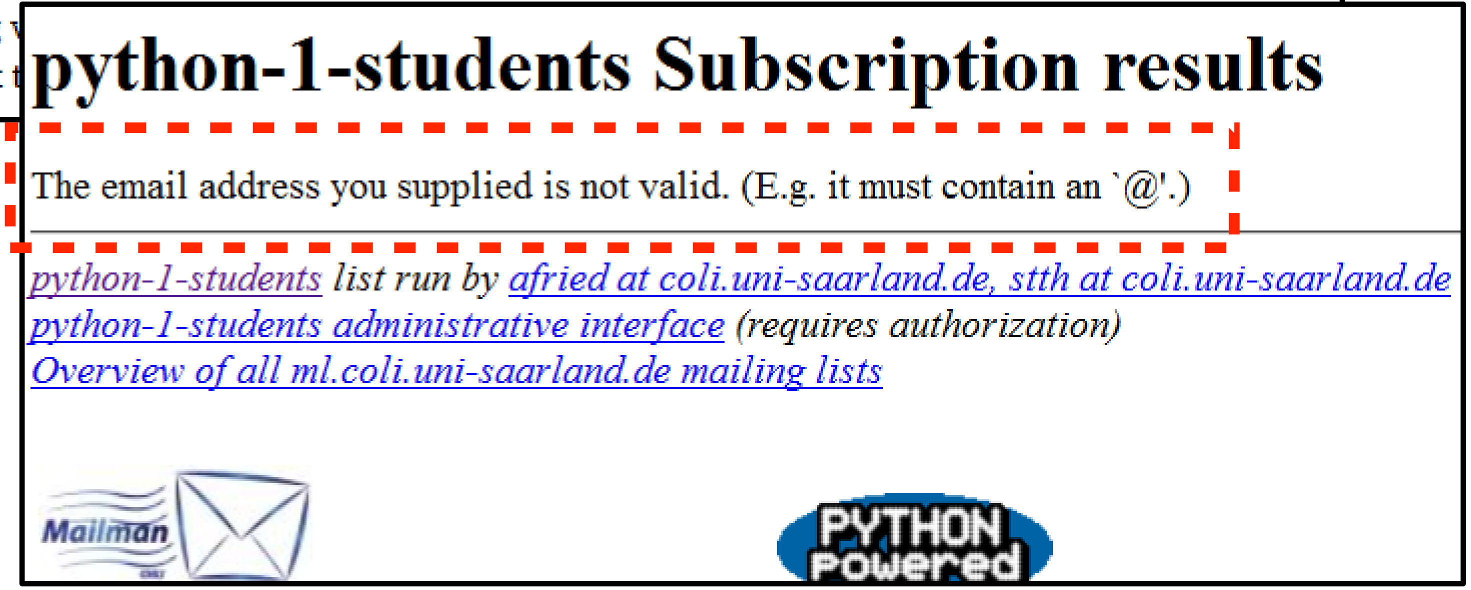


Subscribing to python-1-students

Subscribe to python-1-students by filling out the following form. You will be sent emai
prevent others from gratuitously subscribing you. This is a hidden list, which means th
available only to the list administrator.

Your email address:	blahblub@foo
Your name (optional):	blah blub

You may enter a privacy password below. This provides only mild security, but should prevent others
from messing v
emailed back t



python-1-students Subscription results

The email address you supplied is not valid. (E.g. it must contain an `@'.)

python-1-students list run by afried at coli.uni-saarland.de, stth at coli.uni-saarland.de
python-1-students administrative interface (requires authorization)
Overview of all ml.coli.uni-saarland.de mailing lists

We want …

- We want a way to describe (potentially infinite) sets of strings in a compact way as a pattern.
  - ⇒ Regular expressions, RegEx
  - ⇒ RegExs define search patterns

- An Example:
  - '(Jan|January)\s([1-9]|[12][0-9]|3[01])(\s|$)'
  - matches “Jan 1”, “Jan 2”, ..., “Jan 31”, “January 1”, ...
  - but not “Jan 01”, “Jan 32”

We want …

- An Example:
  - '(Jan|January)\s([1-9]|[12][0-9]|3[01])(\s|$)'
  - matches “Jan 1”, “Jan 2”, ..., “Jan 31”, “January 1”, ...
  - but not “Jan 01”, “Jan 32”

We want …

- An Example:
  - '(Jan|January)\s([1-9]|[12][0-9]|3[01])(\s|$)'
  - '(Jan|January)\s([1-9]|[12][0-9]|3[01])(\s|$)’
  - '(Jan|January)\s([1-9]|[12][0-9]|3[01])(\s|$)'
  - matches “Jan 1”, “Jan 2”, ..., “Jan 31”, “January 1”, ...
  - but not “Jan 01”, “Jan 32”

We want …

- An Example:
  - '(Jan|January)\s([1-9]|[12][0-9]|3[01])(\s|$)'
  - '(Jan|January)\s([1-9]|(1|2)[0-9]|3(0|1))(\s|$)’
  - … would do the same

Regular Expressions

- Regular expressions offer a flexibe way to match and manipulate strings.
- Regular expressions = patterns for strings that (can) express generalizations
- Typical applications:
  - text search – find a substring in a string
  - text transformation – replace substrings
  - validation – is the input well-formed?

Matching regular expressions

- Strings are regular expressions that match themselves
  - “match” matches “match”
- Alternations:
  - “reali(s|z)e” matches exactly “realize” and “realise”
- Match any characters (".“ dot, also called the wildcard)
  - “.atch” matches “match”, “catch”, “watch”, ...
- Optional characters (“?“ question mark):
  - “modell?” matches “model” and “modell”
  - Makes previous character or regex optional

RegEx Quantifiers

- Match zero or more characters (Kleene star *)
  - “co*l” matches “cl”, “col”, “cool”, “coool”, ...
- Match one or more characters (Kleene plus +)
  - “co+l” matches “col”, “cool”, “coool”, …
- Match between n and m characters ({n,m})
  - “co{2,4}l” matches “cool”, “coool”, and “cooool”

Nested Regular Expressions

- Regular expressions can be nested:
  - “match(ed|ing)?” matches “match”, “matched”, “matching”
  - “(un)?friendly matches “unfriendly” and “friendly”
  - “(ha+)+” matches “haha”, “haaha”, “haahaa”, …

Character Sets

- Character sets (special case of alternation)
  - “[mc]atch” matches exactly “match” and “catch”   (“(m|c)atch”)
  - “[1-9][0-9]*” matches all (natural) numbers
- Character sets can be “negated”
  - “[^mc]atch” matches “watch” but not “match” or “catch”
- Some special character sets:
  - \w = word character (A, …, Z, a, …, z and underscore)
  - \W = not a word character
  - \s = white space (blank, tabulator, newline)
  - \d = digit (0, …, 9)

^ and $ (also called anchors)

- ^
  - matches the start of a string
  - for instance, ^abc matches “abcd” but not “dabc”
- $
  - matches the end of a string
  - for instance, abc$ matches “dabc” but not “abcd”

Summary RegEx meta-characters

.	Wildcard, matches any character
^	Matches the start of a string
$	Matches the end of a string
[abc]	Matches one of a set of characters
[A-Z0-9]	Matches one of a range of characters
end|ing|s	Matches one of the specified strings (disjunction)
*	Zero or more of previous item
+	One or more of previous item
?	Zero or one of the previous item
{n}	Exactly n repeats where n is a non-negative integer
{n,}	At least n repeats
{,n}	No more than n repeats
{m,n}	At least m and no more than n repeats
a(b|c)+	Parentheses that indicate the scope of the operators

Exercise: does the regex match?

(1) ab+c?	(1) abc
(2) (ab)+c?	(2) ac
(3) [ab]+c?	(3) abbb
(4) ^ab*a.*	(4) abab
(5) a?b*c	(5) bbc
(6) b+c*	(6) aabcd
(7) a.+b?c	(7) b
(8) b{2,}c?	

Exercise

- Design regular expressions matching
  - typical email addresses (like stth@coli.uni-saarland.de)
  - prices (like $99.99)
  - years between 1984 and 2009

Using regular expressions in Python

```
import re

...

if re.search(pattern, text):
   # pattern matches (text contains pattern)
   ...
else:
   # pattern doesn't match
   ...
...
```

regular expression
represented as a string

re.search returns “match objects”

```
import re

...

mo = re.search(pattern, text)
if mo:
   # pattern matches
   print(mo.group())
else:
   print("not found")

...
```

Match and search methods

- mo = re.search(pattern, string)
  - tests whether pattern matches **some substring** of string
  - returns a “match object” if successful, None otherwise
  - match objects count as True in conditionals (if ... then)
  - mo.group() returns the matched string
- mo = re.match(pattern, string)
  - tests whether pattern matches zero or more characters **at the beginning of string**

Exercise

- Implement a python program that reads in a file and prints all lines containing a form of „werden“:
    - werde, werden, werdest, werdet, wird, wirst, wurde, wurden, wurdest, wurdet
- Use the „chefkoch-sample.txt“ file from Moodle.

Using regular expressions

```python
import re
import sys

def main():
    pat = '(^|\s)(w[eu]rde(n|t|st)|wir(d|st))($|\s)'
    with open(sys.argv[1]) as f:
        for line in f:
            if re.search(pat, line):
                print(line)

if __name__ == '__main__':
    main()
```

More RE Methods/Functions

- re.findall(pattern, string)
  - returns a list of all (non-overlapping) matches in string
- re.finditer(pattern, string)
  - returns an iterator of all (non-overlapping) matches in string
- re.split(pattern, string)
  - Splits the source string by the occurrences of the pattern
  - Returns a list containing the resulting substrings

Extract substrings

- re.match(pattern, string)
  - returns a “match object” if pattern matches string
- match objects can be used to extract (matching) substrings (“groups”)
  - groups are indicated by parenthesis

```
>>> mo = re.match("it (matche[sd])", "it matched")
>>> mo.groups()
('matched',)
>>> mo.group(1)
'matched'
```

Non-greedy Quantifiers

- Quantifiers are greedy
  - they match as much of the string as possible/longest match
- If a question mark follows a quantifier, the quantifier becomes non-greedy

```
>>> input = 'name="Bill" age="23"'
>>> mo = re.search('"(.*)"', input)
>>> mo.group(1)
'Bill" age="23'
>>>
>>> mo = re.search('"(.*?)"', input)
>>> mo.group(1)
'Bill'
```

Lookahead

- Positive Lookahead: (?=regex)
  - matches without “consuming” the input

```
>>> mo = re.match('(.*)(?=(abc))(.*)', 'abcabc')
>>> mo.groups()
('abc', 'abc', 'abc')
```

- Negative lookahead: (?!regex)

Meta-Characters

- Most characters in a regular expression match themselves
  - “match” matches “match”
- Special characters (regex meta-characters) need to be “escaped” with the backslash character “\“ if we want to use them as normal characters in a regex.
  - “letter\.” matches “letter.” but not “letters”
- Special (regex meta-) characters:
  - ., ^, $, *, +, ?, {,}, [,], \, |, (,)

re.split & lookahead

```
import re

text = "The U.S. constitution is the fundamental
framework of America's system of government. The
Constitution bla bla bla."

sntncs = re.split('\.(?=\s[A-Z]|$)', text)

# ["The U.S. constitution is the fundamental framework
of America's system of government", ' The Constitution
bla bla bla']
```

Raw String Notation

- Regular expressions use the backslash character “\“
  1. to indicate special forms (like \s)
  2. or to allow special characters to be used without invoking their special meaning (like \.).
- This collides with Python’s usage of the same character for the same purpose in string literals
- For instance:

\break	– string
\\\\break	– representation of the string in Python
\\\\\\\\break	– regular expression matching “\\\\break”

Raw String Notation

- Raw strings = string literals prefixed by an r
  - r"this is a raw string"
  - Note: Just notation, not a different type of strings
- Within raw strings, backslashes are not handled in any special way.
  - r"\break" == "\\break"
  - r"\\break" == "\\\\break"

Exercise #1

```
<HEAD TYPE="MAIN" TEIFORM="head">
  <S N="1" P="N" TEIFORM="s">
    <W TYPE="NN1" TEIFORM="w">FACTSHEET </W>
    <W TYPE="DTQ" TEIFORM="w">WHAT </W>
    <W TYPE="VBZ" TEIFORM="w">IS </W>
    <W TYPE="NN1" TEIFORM="w">AIDS</W>
    <C TYPE="PUN" TEIFORM="c">?</C>
  </S>
</HEAD>
<P TEIFORM="p">
  <S N="2" P="N" TEIFORM="s">
    <HI REND="bo" TEIFORM="hi">
      <W TYPE="NN1" TEIFORM="w">AIDS </W>
      <C TYPE="PUL" TEIFORM="c">(</C>
...
```

```
FACTSHEET NN1
WHAT DTQ
IS VBZ
AIDS NN1
...
```

Exercise #2

```
Pierre/NNP Vinken/NNP ,/, 61/CD years/NNS old/JJ ,/, will/MD
join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN
Nov./NNP 29/CD ./.
Mr./NNP Vinken/NNP is/VBZ chairman/NN of/IN Elsevier/NNP
N.V./NNP ,/, the/DT Dutch/NNP publishing/VBG group/NN ./.
...
```

$\Downarrow$

```
the/DT board/NN
a/DT nonexecutive/JJ director/NN
the/DT Dutch/NN
...
```

RegEx learning resources on the web
Learning to learn …

- https://www.programiz.com/python-programming/regex
- https://regex101.com/
- Search for tutorials/videos …
  - Regular expressions for beginners
  - Regular expressions explained
  - Regular expressions for dummies
  - Regular expressions gentle explanation
  - Regular expressions made simple
  - …