Regular Expressions

CS 1111
Introduction to Programming

Spring 2019

[Ref: https://docs.python.org/3/library/re.html]

Overview

- What are regular expressions?
- Why and when do we use regular expressions?
- How do we define regular expressions?
- How are regular expressions used in Python?

What is Regular Expression?

- Special string for describing a pattern of characters
- May be viewed as a form of pattern matching
- Examples (we’ll discuss in details -- ”how to define”)

Regular expression	Description
[abc]	One of those three characters
[a-z]	A lowercase
[a-z0-9]	A lowercase or a number
.	Any one character
\.	An actual period
*	0 to many
?	0 or 1
+	1 to many

Why and When ?

Why ?

- To find all of one particular kind of data
- To verify that some piece of text follows a very particular format

When ?

- Used when data are unstructured or string operations are inadequate to process the data

Example unstructured data: [2012debate.txt]

Example structured data: [fake-111x-officehour-queue]

How to Define Regular Expressions

- Mark regular expressions as raw strings r""

- Use square brackets "[" and "]" for "any character"

r"[bce]"	matches either "b", "c", or "e"

- Use ranges or classes of characters

r"[A-Z]"	matches any uppercase letter
r"[a-z]"	matches any lowercase letter
r"[0-9]"	matches any number

Note: use "-" right after [or before] for an actual "-"

r"[-a-z]"	matches "-" followed by any lowercase letter

How to Define Regular Expressions(2)

- Combine sets of characters

  r"[bce]at"   starts with either “b”, “c”, or “e”, followed by “at”

  This regex matches text with “bat”, “cat”, and “eat”.
  How about “con**cat**enation”?

- Use "." for “any character”

  r".at"   matches three letter words, ending in “at”

- Use "\." for an actual period

  r"at\."   matches “at.”

How to Define Regular Expressions(3)

- Use "*" for 0 to many

  r"[a-z]*" matches text with any number of lowercase letter

- Use "?" for 0 or 1

  r"[a-z]?" matches text with 0 or 1 lowercase letter

- Use "+" for 1 to many

  r"[a-z]+" matches text with at least 1 lowercase letter

- Use "|" for option

  r"[ab|12]" matches either ab or 12

How to Define Regular Expressions(4)

- Use "^" for negate

  r"[^a-z]" matches anything except lowercase letters

  r"[^0-9]" matches anything except decimal digits

- Use "^" for “start” of string

  r"^[a-zA-Z]" must start with a letter

- Use "$" for “end” of string

  r".*[a-zA-Z]$" must end with a letter

- Use "{" and "}" to specify the number of characters

  r"[a-zA-Z]{2,3}" must contain 2-3 letters

  r"[a-zA-Z]{3}" must contain 3 letters

Predefined Character Classes

\d	matches any decimal digit – [0-9]
\D	matches any non-digit character – [^0-9]
\s	matches any whitespace character – [\t\n]
\S	matches any non-whitespace – [^\t\n]
\\	matches a literal backslash
\w	matches any alphanumeric character – [a-zA-Z0-9_]
\W	matches any non-alphanumeric character – [^a-zA-Z0-9_]

Exercise

Defining regular expressions describing the following information / pattern

- Names

  ```
  r"[A-Z][a-z]+"
  ```

- Phone numbers

  ```
  r"[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]"
  ```

- UVA Computing ID

  ```
  r"[a-z][a-z][a-z]?[0-9][a-z][a-z][a-z]?"
  ```

- Different patterns?

Use Regular Expressions in Python

- Import re module

```
import re
```

- Define a regular expression (manual or use a tool http://regexr.com/ , https://regex101.com/)

- Create a regular expression object that matches the pattern

```
regex = re.compile(r"[A-Z][a-z]*")
```

- Search / find the pattern in a given text

```
        results = regex.search(text)
   or
        results = regex.findall(text)
   or
        results = regex.finditer(text)
```

re.compile(*pattern*)

- Compile a regular expression pattern into a regular expression object

```
regex = re.compile(r"[A-Z][a-z]*")
```

re.search(*pattern*, *string*)

- Scan through *string* looking for the first location where the *pattern* matches and return a **match object**

- Otherwise, return `None` if a match is not found

- A match object contains `group()`-return the match object, `start()`-return first index of the match, and `end()`-return last index of the match

```
regex = re.compile(r"[A-Z][a-z]*")
results = regex.search(text)
```

↕ =

```
results = re.search(r"[A-Z][a-z]*"), text)
```

`re.findall(pattern, string)`

- Return a **list of strings** of all non-overlapping matches of *`pattern`* in *`string`*
- Otherwise, return an empty list if a match is not found
- The *`string`* is scanned left-to-right
- The matches are returned in the order found

```
regex = re.compile(r"[A-Z][a-z]*")
results = regex.findall(text)
```

- Note: a list does not support `group()`

re.finditer(*pattern*, *string*)

- Return a **collection of match objects** in *string*
- Otherwise, return an empty collection if a match is not found
- The *string* is scanned left-to-right
- The matches are returned in the order found

```
regex = re.compile(r"[A-Z][a-z]*")
results = regex.finditer(text)
```

- Note: a match object supports `group()`

match.group(), match.group(n), match.groups()

group()

- Return the matched object ≈ **group(0)**

group(n)

- Return the nth subgroup (n=1,2,…, number of subgroups)

groups()

- Return all matching subgroups in a tuple

```
regex = re.compile(r"([A-Z])([a-z]*)")
results = regex.finditer(text)
for m in results:
    print(m.group(), m.group(0), m.group(1), m.group(2))
    print(m.groups())
```

Summary

- Must know (based on exam3 topic list, as of 04/10/2019)
  - `import re`
  - `re.compile(r'...')`,
    - including the use of `.`, `[]`, `()`, `+`, `*`, and `?`
  - `compiled_re.search(text)`
  - `compiled_re.finditer(text)`
  - `match.group()`
  - `match.group(n)`
  - `match.groups()`