Object Oriented Programming (Using Python)

UNIT- IV

Python to access Web Data :

- Regular Expressions
- Extracting data
- Sockets
- Using the Developer Console to Explore HTTP
- Retrieving Web Page, and Passing Web Pages

Prof. R. MADANA MOHANA
Professor, Artificial Intelligence & Data Science
http://rmadanamohana.com/

Reference

https://www.coursera.org/learn/python-network-data

https://github.com/I-am-Harsh/Using-Python-to-Access-Web-Data

https://docs.python.org/3/howto/regex.html

https://www.py4e.com/code3/

https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Book%3A_Python_for_Everybody_(Severance)

https://www.w3schools.com/python/python_regex.asp

Regular Expressions

- In computing, a regular expression, also referred as “regex” or “regexp”, provides a concise and flexible means for matching of strings of text, such as particular characters, words, or patterns of characters.
- A regular expression is written in a formal language that can be interpreted by a regular expression processor.
- Really clever “wild card” expressions for matching and parsing strings.
- Really smart “Find” or “Search”
- Regular expressions are widely used in various programming languages, including python for tasks such as pattern matching, string searching, data validation and text manipulation.

Understanding Regular Expressions

- Very powerful and quite cryptic (confusing)
- Fun once we understand them
- Regular expressions are a language unto (something continued until a particular time) themselves
- A language of “marker characters” – programming with characters
- It is kind of an “old school” language - compact

Python Regular Expression Quick Guide

Regular Expression	Description
`^`	Matches the beginning of a line
`$`	Matches the end of the line
`.`	Matches any character
`\s`	Matches whitespace
`\S`	Matches any non-whitespace character
`*`	Repeats a character zero or more times
`*?`	Repeats a character zero or more times (non-greedy)
`+`	Repeats a character one or more times
`+?`	Repeats a character one or more times (non-greedy)
`[aeiou]`	Matches a single character in the listed set
`[^XYZ]`	Matches a single character not in the listed set

Python Regular Expression Quick Guide

Regular Expression	Description
`[a-z0-9]`	The set of characters can include a range
`(`	Indicates where string extraction is to start
`)`	Indicates where string extraction is to end
`d`	Matches any digit character
`\D`	Matches any non-digit character
`\w`	Matches any alphanumeric character (word character)
`\W`	Matches any non-alphanumeric character
`?`	Matches zero or one occurrence of the preceding pattern
`{n}`	Matches exactly n occurrences of the preceding pattern
`{n,}`	Matches n or more occurrences of the preceding pattern
`{n,m}`	Matches between n and m occurrences of the preceding pattern

Python Regular Expression Quick Guide

Regular Expression	Description
`[]`	Matches any single character within the brackets
`[^]`	Matches any single character not within the brackets
`(pattern)`	Groups multiple patterns together
`(?:pattern)`	Non-capturing group. Matches the pattern but does not capture it
`\b`	Matches a word boundary
`\B`	Matches a non-word boundary
`[0-9]+`	One or more digits
`^F.+:`	It will look for lines or strings that start with 'F' followed by one or more characters and ending with a ':'. (Greedy Matching)
`^F.+?:`	It will look for lines or strings that start with 'F' followed by one or more characters (non-greedily) and ending with a ':'.

The Regular Expression Module

- Before we can use regular expression in our program, we must import the library using "**`import re`**"

```
>>> import re
```

Metacharacters

Metacharacters are characters with a special meaning.

Regular Expression	Description
[ ]	Matches any single character within the brackets
\	Signals a special sequence (can also be used to escape special characters)
.	Matches any character
^	Matches the beginning of a line
$	Matches the end of the line
*	Repeats a character zero or more times
+	Repeats a character one or more times
?	Matches zero or one occurrence of the preceding pattern
{ }	Matches exactly the specified number of occurrences of the preceding pattern
|	Either or
( )	Capture and group

Special Sequences

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Regular Expression	Description
`\A`	Returns a match if the specified characters are at the beginning of the string
`\b`	Returns a match where the specified characters are at the beginning or at the end of a word
`\B`	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word
`\d`	Returns a match where the string contains digits (numbers from 0-9)
`\D`	Returns a match where the string DOES NOT contain digits
`\s`	Returns a match where the string contains a white space character
`\S`	Returns a match where the string DOES NOT contain a white space character
`\w`	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)

Special Sequences

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning: *cont'd.*

Regular Expression	Description
\W	Returns a match where the string DOES NOT contain any word characters
\Z	Returns a match if the specified characters are at the end of the string

Sets

A set is a set of characters inside a pair of square brackets [ ] with a special meaning:

Regular Expression	Description
`[arn]`	Returns a match where one of the specified characters (a, r, or n) is present
`[a-n]`	Returns a match for any lower case character, alphabetically between a and n
`[^arn]`	Returns a match for any character EXCEPT a, r, and n
`[0123]`	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
`[0-9]`	Returns a match for any digit between 0 and 9
`[0-5][0-9]`	Returns a match for any two-digit numbers from 00 and 59
`[a-zA-Z]`	Returns a match for any character alphabetically between a and z, lower case OR upper case
`[+]`	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

Metacharacters - Examples

[] - A set of characters

Example: "[a-m]"

Program:

```
>>>import re
>>> txt = "The rain in Hyderabad"
#Find all lower case characters alphabetically between "a" and "m"
>>> x = re.findall("[a-m]", txt)
>>> print(x)
['h', 'e', 'a', 'i', 'i', 'd', 'e', 'a', 'b', 'a', 'd']
```

Metacharacters - Examples

\ : Signals a special sequence (can also be used to escape special characters)

Example: "\d"

Program:

```
>>>import re
>>> txt = "That will be 83 dollars"
#Find all digit characters
>>> x = re.findall("\d", txt)
>>> print(x)
['8', '3']
```

Metacharacters - Examples

`.` : Any character (except newline character)

Example: `"he..o"`

Program:

```
>>>import re
>>> txt = "hello planet"
#Search for a sequence that starts with "he", followed by two (any) characters, and an "o":
>>> x = re.findall("he..o", txt)
>>> print(x)
['hello']
```

Metacharacters - Examples

For other Metacharacters, Special Sequences and Sets refer the following link:

https://www.w3schools.com/python/python_regex.asp