Python -RegEx

Saturday, August 13, 2022 2:37 PM

RegEx

- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern with a formal syntax. Regular expressions are typically used in applications that involve a lot of text processing.
- As a data scientist/engineer, having a solid understanding of Regex can help you perform various data preprocessing very easily.
- There are multiple open-source implementations of regular expressions, each sharing a common core syntax but with different extensions or modifications to their advanced features. Python has a built-in package called re, which can be used to work with Regular Expressions.

- A **Reg**ular **Ex**pression (RegEx) is a sequence of characters that defines a search pattern. For example:

```
^a...s$
```

- The above code defines a RegEx pattern. The pattern is: **any five letters string starting with a and ending with s**.
- A pattern defined using RegEx can be used to match against a string.

Expression	String	Matched?
^a...s$	abs	No match
	alias	Match
	abyss	Match
	Alias	No match
	An abacus	No match

Module ”Re”

- Python has a module named **re** to work with RegEx. Here's an example:

```
Import re
```

```
import re

pattern = "^a...s$"
test_string = "abyss"
result = re.match(pattern, test_string)

if result :
   print("Search successful.")
else:
   print("Search unsuccessful.")
```

re.findall()

- The re.findall() method returns a list of strings containing all matches.

Example 1: re.findall()

```
# Program to extract numbers from a string

import re

string = "hello 12 hi 89. Howdy 34"
pattern = "\d+"

result =  re.findall(pattern, string)
print(result)

# Output : ['12', '89, '34']
```

- If the pattern is not found, re.findall() returns an empty list.

re.split()

- The re.split method splits the string where there is a match and returns a list of strings where the splits have occurred.

Example 2: re.split()

re.split()

- The re.split method splits the string where there is a match and returns a list of strings where the splits have occurred.

Example 2: re.split()

```
import re

string = "Twelve: 12 Eighty nine: 89. "
pattern = "\d+"

result =  re.split(pattern, string)
print(result)

# Output : ['Twelve: ', ' Eighty nine: ', '. ']
```

- If the pattern is not found, re.split() returns a list containing the original string.

maxsplit

- You can pass maxsplit argument to the re.split() method. It's the maximum number of splits that will occur.

```
import re

string = "Twelve: 12 Eighty nine: 89 Nine:9. "
pattern = "\d+"

# maxsplit = 1
# split only at the first occurrence
result = re.split(pattern, string, 1 )
print(result)

# Output : ['Twelve: ', ' Eighty nine: 89 Nine:9. ']
```

- The default value of maxsplit is 0; meaning all possible splits.

re.sub()

- The syntax of re.sub() is:

```
re.sub(pattern, replace, string)
```

- The method returns a string where matched occurrences are replaced with the content of replace variable.

Example 3: re.sub()

```
import re
# multiline string
string = "abc 12\
de 23 \n f45 6"
# matches all whitespace characters
pattern = "\s+"
# empty string
replace = " "
new_string = re.sub(pattern, replace, string)
print(new_string)
# Output : abc 12de 23 f45 6
```

count

- You can pass count as a fourth parameter to the re.sub() method. If omited, it results to 0. This will replace all occurrences.

```
import re

# multiline string
string = "abc 12\
de 23 \n f45 6"

# matches all whitespace characters
pattern = "\s+"
replace = " "

new_string = re.sub(r'\s+', replace, string, 1)
print(new_string)

# Output :
abc 12de 23
 f45 6
```

re.subn()

- The re.subn() is similar to re.sub() expect it returns a tuple of 2 items containing the new string and the number of substitutions made.

Example 4: re.subn()

```
# Program to remove all whitespaces
import re

# multiline string
string = "abc 12\
de 23 \n f45 6"

# matches all whitespace characters
pattern = "\s+"
replace = " "

new_string = re.sub(r"\s+", replace, string, 1)
print(new_string)
# Output :
 abc 12de 23
 f45 6
```

re.search()

- The re.search() method takes two arguments: a pattern and a string. The method looks for the first location where the RegEx pattern produces a match with the string.

- If the search is successful, re.search() returns a match object; if not, it returns None.

```
match = re.search(pattern, str)
```

```
import re

string = "Python is fun"

# check if 'Python ' is at the beginning
match = re.search("\APython", string)

if match:
    print( "pattern found inside the string")
else:
    print( "pattern not found")

#Output: pattern found inside the string
```

Match.start(), match.end() and match.span()

- The start() function returns the index of the start of the matched substring. Similarly, end() returns the end index of the matched substring.

```
>>> match.start()
0
>>> match.end()
6
```

- The span() function returns a tuple containing start and end index of the matched part.

```
>>> match.span()
(0,6)
```

match.re and match.string

- The re attribute of a matched object returns a regular expression object. Similarly, stringattribute returns the passed string.

```
match.re
#Output
re.compile(r'\APython', re.UNICODE)
 match.string
#Output
'Python is fun'
```

- **Using r prefix before RegEx:**
  - When r or R prefix is used before a regular expression, it means raw string. For example, '\n' is a new line whereas r'\n' means two characters: a backslash \ followed by n.
  - Backlash \ is used to escape various characters including all metacharacters. However, using r prefix makes \ treat as a normal character.

```
import re
string = "\n and \r are escape sequences."

result = re.findall(r"[\n\r]", string)
print(result)
# Output :[ '\n', '\r' ]
```