Python - RegEx and Data Parsing

Eli the Computer Guy - November 27,2024
Version 1

Grep

Returns only the lines with the character or pattern in it. It makes it easier for Python to parse if you can limit the amount of data that it has to interact with.

Grep on OS Commands

We can use Grep with OS Commands to return specific lines that we are interested in. this can be used with the Python OS Module.

Run the Ping Command but only return lines with a **/** in them

```
ping -c 1 cnn.com | grep /
```

```
round-trip min/avg/max/stddev = 34.655/34.655/34.655/0.000 ms
```

Example: Find IP Address on MacOS Terminal

```
ifconfig | grep "inet "
```

Grep on Files

We output the results of a ping command to the file ping-result.txt. We then can run Grep against that file.

```
ping -c 1 cnn.com > ping-result.txt
```

```
grep / ping-result.txt
```

```
round-trip min/avg/max/stddev = 30.867/30.867/30.867/nan ms
```

Simple Matching with Split()

split() will turn a string into a list and use the separator that you assign.

- split(' ') - Splits at every space
- split(',') - Splits at commas for CSV, comma separated values
-

```
message = 'My email address is bob@bob.com.'
message = message.split(' ')
```

```
['My', 'email', 'address', 'is', 'bob@bob.com.']
```

We can then iterate through the new list using a f**or x** loop.

```
for word in message:
    if '@' in word:
        print(word)
```

```
bob@bob.com.
```

We can then use strip() to remove unwanted characters from the string.

```
for word in message:
    if '@' in word:
        word = word.strip('.(),')
        print(word)
```

```
bob@bob.com
```

split.py

```
string = '''hello this is bob at bob@bob.com.  I'm contacting you on
behalf of my friend tom who is at (tom@aol.com).  Cindy at cindy@gmail.com
gave me your email address to contact you. You can ping me on Linked in at
@bob, or @cool_dude.'''

string = string.split(' ')

#print(string)

for word in string:
    if '@' in word:
        print(word)
        print(word.strip('.(),'))
```

RegEx

RegEx stands for Regular Expressions and is used to find text that matches patterns within strings. You can use this to pull out IP Addresses, Email Addresses, Phone Numbers, etc.

All* languages have the ability to do egEx, but the patterns may be different.

With Python you use the **re** Module

Re Functions

With Python the RegEx module has 3 functions. The module is named **re** and should be installed by default.

findall()

Finds all instances and puts into list

re-findall.py
```
import re

pattern = r'[\w\.-]+@[\w\.-]+\.+[\w]{3}'

text = '''hello this is bob at bob@bob.com.  I'm contacting you on behalf
of my friend tom who is at tom@aol.com.  Cindy at cindy@gmail.com gave me
your email address to contact you. You can ping me on Linked in at @bob,
or @cool_dude.'''

response = re.findall(pattern, text)

print(response)
```

```
['bob@bob.com', 'tom@aol.com', 'cindy@gmail.com']
```

match()

Match checks if a pattern is at the beginning of a string. You use group() to print out the word that matches the pattern.

re-match.py
```
import re

pattern = r'hello'
text = 'hello world'

match = re.match(pattern, text)
if match:
    print('Match found:', match.group())
else:
    print('No match found')
```

search()

Searches the string and returns the first value that matches the pattern.
- group() - Shows the text that matches the pattern
- start() - Shows the position of the starting character for the pattern in the string

• end() - Shows the position of the ending character for the pattern in the string

re-search.py
```
import re

pattern = r'[\w\.-]+@[\w\.-]+\.[\w]{2,6}'

text = '''
        hello bob how are you?
        I was told your email is bob@aol.com.
        Is this right?
        My email is tim@gmail.com
        '''
response = re.search(pattern, text)

print(response)
print(response.group())
print(response.start())
print(response.end())
```

sub()

The sub function allows you to substitute characters in a string with other characters. This can be an easy way to strip out characters that you don't want to test against.

re-sub.py
```
import re

pattern = r'[- ]'
number = '111-222 3333'

result = re.sub(pattern, '', number)
print(result)
```

RegEx Characters and Patterns

When using RegEx you create patterns for the **re** module to test against.

When defining a pattern you use the r'' for Raw Strings. This disables certain text formatting rules Python follows.

When you need to test for a character that is used as a wild card in re you use the black slash to escape it. Such as if you want to test for a period you would use '\.'

Note: ChatGPT is wonderful and RegEx patterns

Pattern Examples

Mac Address:
```
([0-9A-Fa-f]{2}[:-]){5}[0-9A-Fa-f]{2}
```

IPv4 Address
```
\b(\d{1,3}\.){3}\d{1,3}\b
```

Email Address
```
\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b
```

Social Security Number
```
\b\d{3}-\d{2}-\d{4}\b
```

Meta-Character	Description
`.`	Matches **any single character** except a newline.
`^`	Matches the **start** of a string or line (in multiline mode).
`$`	Matches the **end** of a string or line (in multiline mode).
`*`	Matches **0 or more occurrences** of the preceding character or group.
`+`	Matches **1 or more occurrences** of the preceding character or group.
`?`	Matches **0 or 1 occurrence** of the preceding character or group.
`{n}`	Matches exactly **n occurrences** of the preceding character or group.
`{n,}`	Matches **n or more occurrences**.
`{n,m}`	Matches **between n and m occurrences**.
`[ ]`	Defines a **character set**. Matches any character inside the brackets.
`	`
`()`	Groups characters or expressions.

Class	Description	Equivalent
`\d`	Matches any **digit** (0-9).	`[0-9]`
`\D`	Matches any **non-digit**.	`[^0-9]`
`\w`	Matches any **word character**.	`[a-zA-Z0-9_]`
`\W`	Matches any **non-word character**.	`[^a-zA-Z0-9_]`
`\s`	Matches any **whitespace**.	`[ \t\n\r\f\v]`

`\S`	Matches any **non-whitespace** character.	`[^ \t\n\r\f\v]`

Anchor	Description
`^`	Matches the **start** of a string.
`$`	Matches the **end** of a string.
`\b`	Matches a **word boundary**.
`\B`	Matches **not at a word boundary**.