Matching patterns

Regular expressions are complicated mini-language. They rely on special characters to match unknown strings, but let's start with literal characters, such as letters, numbers, and the space character, which always match themselves. Let's see a basic example:

```
#Need module 're' for regular expression
import re
#
search_string = "HelloWorld"
pattern = "Hello"
match = re.match(pattern, search_string)
#If-statement after search() tests if it succeeded
if match:
  print("regex matches: ", match.group())
else:
  print('pattern not found')
```

Result

regex matches: Hello

Matching a string

The “re” module of python has numerous method, and to test whether a particular regular expression matches a specific string, you can use re.search(). The re.MatchObject provides additional information like which part of the string the match was found.

Syntax

matchObject = re.search(pattern, input_string, flags=0)

Example

```
#Need module 're' for regular expression
import re
# Lets use a regular expression to match a date string.
regex = r"([a-zA-Z]+) (\d+)"
if re.search(regex, "Jan 2"):
  match = re.search(regex, "Jan 2")
  # This will print [0, 5), since it matches at the beginning and end of the
  # string
  print("Match at index %s, %s" % (match.start(), match.end()))
  # The groups contain the matched values. In particular:
  # match.group(0) always returns the fully matched string
  # match.group(1), match.group(2), ... will return the capture
```

```
  # groups in order from left to right in the input string
  # match.group() is equivalent to match.group(0)
  # So this will print "Jan 2"
  print("Full match: %s" % (match.group(0)))
  # So this will print "Jan"
  print("Month: %s" % (match.group(1)))
  # So this will print "2"
  print("Day: %s" % (match.group(2)))
else:
  # If re.search() does not match, then None is returned
  print("Pattern not Found! ")
```

Result

```
Match at index 0, 5
Full match: Jan 2
Month: Jan
Day: 2
```

As the above method stops after the first match, so is better suited for testing a regular expression than extracting data.

Capturing Groups

If the pattern includes two or more parenthesis, then the end result will be a tuple instead of a list of string, with the help of parenthesis() group mechanism and finall(). Each pattern matched is represented by a tuple and each tuple contains group(1), group(2).. data.

```
import re
regex = r'([\w\.-]+)@([\w\.-]+)'
str = ('hello john@hotmail.com, hello@Helloworld.com, hello python@gmail.com')
matches = re.findall(regex, str)
print(matches)
for tuple in matches:
  print("Username: ",tuple[0]) #username
  print("Host: ",tuple[1]) #host
```

Result

```
[('john', 'hotmail.com'), ('hello', 'Helloworld.com'), ('python', 'gmail.com')]
Username: john
Host: hotmail.com
Username: hello
Host: Helloworld.com
Username: python
```

Host: gmail.com

Finding and replacing string

Another common task is to search for all the instances of the pattern in the given string and replace them, the re.sub(pattern, replacement, string) will exactly do that. For example to replace all instances of an old email domain

Code

```
# requid library
import re
#given string
str = ('hello john@hotmail.com, hello@Helloworld.com, hello python@gmail.com, Hello World!')
#pattern to match
pattern = r'([\w\.-]+)@([\w\.-]+)'
#replace the matched pattern from string with,
replace = r'\1@XYZ.com'
  ## re.sub(pat, replacement, str) -- returns new string with all replacements,
  ## \1 is group(1), \2 group(2) in the replacement
print (re.sub(pattern, replace, str))
```

Result

hello john@XYZ.com, hello@XYZ.com, hello python@XYZ.com, Hello World!

Re options flags

In the python regular expression like above, we can use different options to modify the behavior of the pattern match. These extra arguments, optional flag is added to the search() or findall() etc. function, for example re.search(pattern, string, re.IGNORECASE).

- IGNORECASE −
  As the name indicates, it makes the pattern case insensitive(upper/lowercase), with this, strings containing ‘a’ and ‘A’ both matches.
- DOTALL
  The re.DOTALL allows dot(.) metacharacter to match all character including newline (\n).
- MULTILINE
  The re.MULTILINE allows matching the start(^) and end($) of each line of a string. However, generally, ^ and & would just match the start and end of the whole string.