Let's import the regular expression library in python.

In [1]:
```
import re
```

Let's do a quick search using a pattern.

In [2]:
```
re.search('Ravi', 'Ravi is an exceptional student!')
```

Out[2]:
```
<re.Match object; span=(0, 4), match='Ravi'>
```

In [3]:
```
# print output of re.search()
match = re.search('Ravi', 'Ravi is an exceptional student!')
print(match.group())
```

```
Ravi
```

Let's define a function to match regular expression patterns

In [4]:
```
def find_pattern(text, patterns):
    if re.search(patterns, text):
        return re.search(patterns, text)
    else:
        return 'Not Found!'
```

Quantifiers

In [5]:
```
# '*': Zero or more
print(find_pattern("ac", "ab*"))
print(find_pattern("abc", "ab*"))
print(find_pattern("abbc", "ab*"))
```

```
<re.Match object; span=(0, 1), match='a'>
<re.Match object; span=(0, 2), match='ab'>
<re.Match object; span=(0, 3), match='abb'>
```

In [6]:
```
# '?': Zero or one (tells whether a pattern is absent or present)
print(find_pattern("ac", "ab?"))
print(find_pattern("abc", "ab?"))
print(find_pattern("abbc", "ab?"))
```

```
<re.Match object; span=(0, 1), match='a'>
<re.Match object; span=(0, 2), match='ab'>
<re.Match object; span=(0, 2), match='ab'>
```

In [7]:
```
# '+': One or more
print(find_pattern("ac", "ab+"))
print(find_pattern("abc", "ab+"))
print(find_pattern("abbc", "ab+"))
```

```
Not Found!
<re.Match object; span=(0, 2), match='ab'>
<re.Match object; span=(0, 3), match='abb'>
```

In [8]:
```
# {n}: Matches if a character is present exactly n number of times
print(find_pattern("abbc", "ab{2}"))
```

```
<re.Match object; span=(0, 3), match='abb'>
```

In [9]:
```
# {m,n}: Matches if a character is present from m to n number of times
print(find_pattern("aabbbbbbc", "ab{3,5}"))   # return true if 'b' is present
print(find_pattern("aabbbbbbc", "ab{7,10}"))  # return true if 'b' is present
print(find_pattern("aabbbbbbc", "ab{,10}"))   # return true if 'b' is present
print(find_pattern("aabbbbbbc", "ab{10,}"))   # return true if 'b' is present
```

```
<re.Match object; span=(1, 7), match='abbbbb'>
Not Found!
<re.Match object; span=(0, 1), match='a'>
Not Found!
```

Anchors

In [10]:
```
# '^': Indicates start of a string
# '$': Indicates end of string

print(find_pattern("James", "^J"))   # return true if string starts with 'J'
print(find_pattern("Pramod", "^J"))  # return true if string starts with 'J'
print(find_pattern("India", "a$"))   # return true if string ends with 'c'
print(find_pattern("Japan", "a$"))   # return true if string ends with 'c'
```

```
<re.Match object; span=(0, 1), match='J'>
Not Found!
<re.Match object; span=(4, 5), match='a'>
Not Found!
```

Wildcard

In [11]:
```
# '.': Matches any character
print(find_pattern("a", "."))
print(find_pattern("#", "."))
```

```
<re.Match object; span=(0, 1), match='a'>
<re.Match object; span=(0, 1), match='#'>
```

Character sets

In [12]:
```
# Now we will look at '[' and ']'.
# They're used for specifying a character class, which is a set of characters
# Characters can be listed individually as follows
print(find_pattern("a", "[abc]"))

# Or a range of characters can be indicated by giving two characters and separ
print(find_pattern("c", "[a-c]"))  # same as above
```

```
<re.Match object; span=(0, 1), match='a'>
<re.Match object; span=(0, 1), match='c'>
```

In [13]:
```
# '^' is used inside character set to indicate complementary set
print(find_pattern("a", "[^abc]"))  # return true if neither of these is prese
```

```
Not Found!
```

Character sets

Pattern	Matches
[abc]	Matches either an a, b or c character
[abcABC]	Matches either an a, A, b, B, c or C character
[a-z]	Matches any characters between a and z, including a and z
[A-Z]	Matches any characters between A and Z, including A and Z
[a-zA-Z]	Matches any characters between a and z, including a and z ignoring cases of the characters
[0-9]	Matches any character which is a number between 0 and 9

Meta sequences

Pattern	Equivalent to
\s	[\t\n\r\f\v]
\S	[^ \t\n\r\f\v]

Pattern	Equivalent to
\d	[0-9]
\D	[^0-9]

Greedy vs non-greedy regex

In [14]:
```
print(find_pattern("aabbbbbb", "ab{3,5}")) # return if a is followed by b 3-5
```

```
<re.Match object; span=(1, 7), match='abbbbb'>
```

In [15]:
```
print(find_pattern("aabbbbbb", "ab{3,5}?")) # return if a is followed by b 3-5
```

```
<re.Match object; span=(1, 5), match='abbb'>
```

In [16]:
```
# Example of HTML code
print(re.search("<.*>","<HTML><TITLE>My Page</TITLE></HTML>"))
```

```
<re.Match object; span=(0, 35), match='<HTML><TITLE>My Page</TITLE></HTML>'>
```

In [17]:
```
# Example of HTML code
print(re.search("<.*?>","<HTML><TITLE>My Page</TITLE></HTML>"))
```

```
<re.Match object; span=(0, 6), match='<HTML>'>
```

The five most important re functions that you would be required to use most of the times are

match() Determine if the RE matches at the beginning of the string

search() Scan through a string, looking for any location where this RE matches

findall() Find all the substrings where the RE matches, and return them as a list

finditer() Find all substrings where RE matches and return them as asn iterator

sub() Find all substrings where the RE matches and substitute them with the given string

In [18]:
```
# - this function uses the re.match() and let's see how it differs from re.sea
def match_pattern(text, patterns):
    if re.match(patterns, text):
        return re.match(patterns, text)
    else:
        return ('Not found!')
```

In [19]:
```
print(find_pattern("abbc", "b+"))
```

```
<re.Match object; span=(1, 3), match='bb'>
```

In [20]:
```
print(match_pattern("abbc", "b+"))
```

```
Not found!
```

In [21]:
```
## Example usage of the sub() function. Replace Road with rd.

street = '21 Ramakrishna Road'
print(re.sub('Road', 'Rd', street))
```

```
21 Ramakrishna Rd
```

In [22]:
```
print(re.sub('R\w+', 'Rd', street))
```

```
21 Rd Rd
```

In [23]:
```
## Example usage of finditer(). Find all occurrences of word Festival in given

text = 'Diwali is a festival of lights, Holi is a festival of colors!'
pattern = 'festival'
for match in re.finditer(pattern, text):
    print('START -', match.start(), end="")
    print('END -', match.end())
```

```
START - 12END - 20
START - 42END - 50
```

In [24]:
```
# Example usage of findall(). In the given URL find all dates
url = "http://www.telegraph.co.uk/formula-1/2017/10/28/mexican-grand-prix-2017
date_regex = '/(\d{4})/(\d{1,2})/(\d{1,2})/'
print(re.findall(date_regex, url))
```

```
[('2017', '10', '28')]
```

In [25]:
```
## Exploring Groups
m1 = re.search(date_regex, url)
print(m1.group())  ## print the matched group
```

```
/2017/10/28/
```

In [26]:
```
print(m1.group(1)) # - Print first group
```

```
2017
```

In [27]:
```
print(m1.group(2)) # - Print second group
```

```
10
```

In [28]:
```
print(m1.group(3)) # - Print third group
```

```
28
```

In [29]:
```
print(m1.group(0)) # - Print zero or the default group
```

```
/2017/10/28/
```