Regular Expressions

Objectives

1 Regular Expressions

2 Patterns

Regular Expressions

Regular expression

- Regular expression is a **character sequence** used to describe a **textual pattern**.
- Using regular expressions we can **match/search** input data for certain patterns with **minimum amount of code.**
- Regular expressions provide a efficient set of string extraction and manipulation capabilities
- In Python, the regular expression capability is provided through the **re module**

The regular expression (re) module – (1/2)

- The first step to utilize the capabilities of re module is to import it into the namespace

```
Import re
```

- The re module provides interfaces to compile the patterns into objects and then perform matches

```
re.compile(pattern) - # return a pattern object
```

- re provides **match()** and **search()** methods to perform matches

```
re.match(pattern, string) -  # checks if the pattern matches at the beginning
                             # of the string; Return a match object,

re.search(pattern, string) - # Looks for a matching pattern anywhere in the string;
                                    # return match object
```

The regular expression (re) module – (2/2)

- Both search() and match() methods return *a match object* if a match is found else returns None
- Information can be extracted from the match object using the following methods

  >>> group() - return the matched string

  >>> start()   - return the starting position of match

  >>> end()   - return the end position of match

  >>> span() - return (start,end) position

- **re.findall(pattern,string)** – return the list of all matched strings
- **re.finditer(pattern,string)** – return an iterator of all matched strings
- Note: We will see couple of examples after understanding patterns in future slides….

Patterns

Patterns

- Characters match themselves

  e.g. an expression “python” will match with a string “python”

- Certain characters and sequences are use to generalize the pattern
- **Special sequences**

  - **\d** – matches any decimal digit [0-9]
  - **\D** – matches non-decimal character[^0-9]
  - **\s** – matches whitespace [" "\t\n\r\f\v]

    - \t-tab,\n-newline,\r-carriage return,\f-form feed,\v-vertical tab

  - **\S** – matches non-whitespace character[^" "\t\n\r\f\v]
  - **\w** - matches alpha-numeric [a-zA-Z0-9]
  - **\W** - non alpha-numeric characters [^a-zA-Z0-9]

Patterns – meta-characters – (1/2)

- **Metacharacters**

- **"."** - matches any character other than the newline
- **[ ]** - used to specify a character class; A set from which you would like to match
  - members can be specified individually - [123abc] or using range [a-zA-Z0-9]
- **‘+’** - the preceding character or the class can occur 1 or more times
- **‘*’** - the preceding character or class can occur o or more times
- **‘?’** - the preceding character can occur 0 or once(1)
- **{} - {m},{m,n},{,n},{m,}**
- **a{3}** - 'a' should occur thrice
- **a{2,4}** - 'a' can occur minimum twice and maximum 4 times
- a{2,} → min. 2 times, no upper limit
- a{,5} → max 5 times

Patterns – meta-characters – (2/2)

- **a{2,}** - 'a' can occur minimum twice and maximum no limits
- **a{,4}** - 'a' can occur minimum 0 and maximum 4 times
- **a{0,}** – ‘*’ , **a{1,}** – ‘+’ and **a{0,1}** – ‘?’

- () - specify substring of interest

- **^** - the match is expected at the beginning of the string or at the beginning of each line with re.MULTILINE flag

- **$** - the match is expected at the end of the string or at the beginning of each line with re.MULTILINE
- | - or test | Test - looks for a 'test' or a 'Test'

Program

re.match() and re.search()

```
Example1:
import re

pattern = '^S....i$'
test_string = 'Suzuki'
result = re.match(pattern,test_string)

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

```
#Output: Search successful.
```

```
Example2:
import re

string = "Play with Python"
# check if 'Python' is inside string
match = re.search('Python', string)

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

```
# Output: pattern found inside the string
```

Program :

- re.search() - Search for pattern anywhere in the target

```
import re

 s = "123-456-789"
    m = re.search("(\d+)-(\d+)-(\d+)", s)
    if m:
        print (m.groups()[0])
        print (m.groups()[1])
        print (m.groups()[2])
```

```
123

456

789
```

Program

start() and findall()

```
import re


txt = "Charles Babbage is father of computing"
x = re.search("\s", txt)
print("The first white-space located is in position:", x.start())
```

```
#Output:
The first white-space character is
located in position: 7
```

```
# Program to extract numbers from a string
import re
string = 'I was 10 then 20 now 40'
pattern = '\d+'
result = re.findall(pattern, string)
print(result)
```

```
# Output:
['10', '20', '40']
```

Compilation flags

- Compilation flags can amend the way regular expression work
- Flags have a long name and a short name

e.g re.IGNORECASE or re.I # does case insensitive match

Flag	Action
re.DOTALL (or) re.S	"." matches any character including '\n'
re.MULTILINE (or) re.M	With '^' and '$' enables multiline match
re.ASCII (or) re.A	Special sequences like \w,\b,\s,\d matches only with ASCII characters
re.LOCALE (or) re.L	Matching characters from other languages depending on locale settings
Re.VERBOSE (or) re.X	Helps in organizing complex expression in a comprehendible fashion

Program

Finditer() -- returns an iterator that produces Match instances

Multiple Lines - matches the first or last word of the input

```
# Ex: finditer
text = 'abbaaabbbbaaaaa'
pattern = 'ab'

for match in re.finditer(pattern, text):
    s = match.start()
    e = match.end()
    print('Found "%s" at %d:%d' % (text[s:e], s, e))
```

```
#Output:
Found "ab" at 0:2
Found "ab" at 5:7
```

```
# Ex:multiline
 import re

text = 'This is some text -- with punctuation.\nAnd a second line.'
pattern = '(^\w+)|(\w+\S*$)'
single_line = re.compile(pattern)
multiline = re.compile(pattern, re.MULTILINE)

#print('Text        :', repr(text))
#print('Pattern     :', pattern)
print('Single Line :', single_line.findall(text))
print('Multline    :', multiline.findall(text))
```

```
#Output:
Single Line : [('This', ''), ('', 'line.')]
Multline    : [('This', ''), ('', 'punctuation.'),
('And', ''), ('', 'line.')]
```

Program

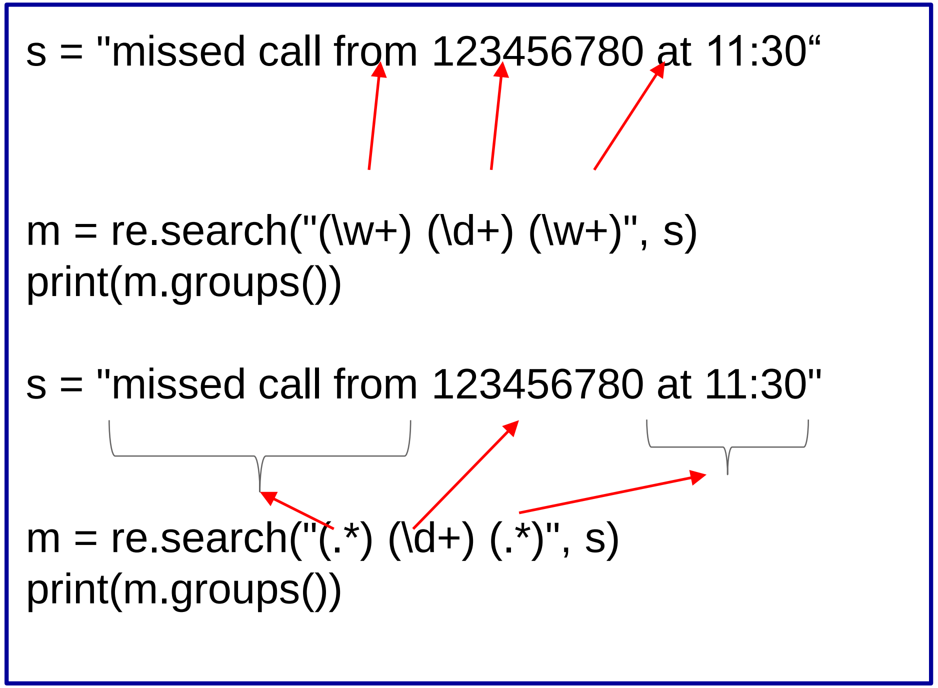
re.IGNORECASE

```
import re


text = 'This is Python textdemo'
pattern = r'\bT\w+'
with_case = re.compile(pattern)
without_case = re.compile(pattern, re.IGNORECASE)


print('Text              :', text)
print('Pattern           :', pattern)
print('Case-sensitive    :', with_case.findall(text))
print('Case-insensitive:', without_case.findall(text))
```

```
#Output:
Text             : This is Python textdemo
Pattern          : \bT\w+
Case-sensitive   : ['This']
Case-insensitive: ['This', 'textdemo']
```

Quiz : what is output of search()

- Search for pattern anywhere in the target



```
s = "missed call from 123456780 at 11:30“



m = re.search("(\w+) (\d+) (\w+)", s)
print(m.groups())

s = "missed call from 123456780 at 11:30"



m = re.search("(.*) (\d+) (.*)", s)
print(m.groups())
```

Thank you