Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

REGULAR EXPRESSIONS

Learning outcomes:

What is a *REGULAR EXPRESSION*?

Metacharacters

match() function

search() function

re.match() vs *re.search()*

findall() function

split() function

sub() function

What is a REGULAR EXPRESSION?

A *regular expression* RegEx is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.
The module ***re*** provides full support for regular expressions in Python. The ***re*** module raises the exception *re.error* if an error occurs while compiling or using a regular expression.

What is a REGULAR EXPRESSION?

It is extremely useful for extracting information from text such as code, files, log, spread sheets or even documents.
While using the ***regular expression*** the first thing is to recognize is that everything is essentially a character, and we are writing patterns to match a specific sequence of characters also referred as string.

What is a REGULAR EXPRESSION?

For instance, a regular expression could tell a program to search for specific text from the string and then to print out the result accordingly. Expression can include

Text matching

Repetition

Branching

Pattern-composition etc.

In Python, a regular expression is denoted as RE (REs, regexes or regex pattern) are imported through **re module**. Python supports regular expression through libraries. In Python regular expression supports various things like **Modifiers, Identifiers, and White space characters**.

What is a REGULAR EXPRESSION?

We would cover two important functions, which would be used to handle regular expressions. But a small thing first: There are various characters, which would have special meaning when they are used in regular expression. To avoid any confusion while dealing with regular expressions, we would use Raw Strings as **r'expression'**.

Metacharacters

Module Regular Expressions(RE) specifies a set of strings(pattern) that matches it.
To understand the RE analogy, MetaCharacters are useful, important and will be used in functions of module re.
There are a total of 14 metacharacters and will be discussed as they follow into functions:
\ :Used to drop the special meaning of character following it [] :Represent a character class
^ :Matches the beginning
$: Matches the end
. :Matches any character except newline

Metacharacters

? :Matches zero or one occurrence.

| :Means OR (Matches with any of the characters separated by it.

* :Any number of occurrences (including 0 occurrences)

+ :One or more occurrences

{} :Indicate number of occurrences of a preceding RE to match.

() :Enclose a group of REs

The match() function

This function attempts to match *Regular Expression pattern* to *string* with optional *flags*.

Here is the syntax for this function:

re.match(pattern, string, flags=0)

Here is the description of the parameters:

pattern :This is the regular expression to be matched.

String: This is the string, which would be searched to match the pattern at the beginning of string.

flags: You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The match() function

The *re.match* function returns a **match** object on success, **none** on failure. We use *group(num)* or *groups()* function of **match** object to get matched expression.

You can read here about match object methods.

group(num=0) : This method returns entire match (or specific subgroup num)

groups() : This method returns all matching subgroups in a tuple (empty if there weren't any)

The match() function

Example: Simple example of match() function.

```
import re
line = "Cats are smarter than dogs"
matchObj = re.match(r'(.*) are', line)
print(matchObj)
```

Output:

```
<re.Match object; span=(0, 8), match='Cats are'>
```

The match() function

Example:

```
line = "Learning Python Programming is easy"
matchObj = re.match(r'(.*) Programming', line)
if matchObj:
   print ("matchObj.group() : ", matchObj.group())
   print ("matchObj.group(1) : ",
matchObj.group(1))
else:
   print ("No match!!")
```

The search() function

The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned.

This function searches for first occurrence of RE *pattern* within *string* with optional *flags*.

Here is the syntax for this function:

re.search(pattern, string, flags=0)

The search() function

Example:

```
txt = "The rain 125 in London"
x = re.search("rain", txt)
y = re.search('[0-9]', txt)
print(x)
print(y)
```

Output:

```
<re.Match object; span=(4, 8), match='rain'>
<re.Match object; span=(9, 10), match='1'>
```

The search() function

Example:

```
line = "Cats are smarter than dogs";
searchObj = re.search( r'(.*) are (.*?) .*', line)
if searchObj:
   print ("searchObj.group() : ", searchObj.group())
   print ("searchObj.group(1) : ",
searchObj.group(1))
   print ("searchObj.group(2) : ",
searchObj.group(2))
else:
   print ("Nothing found!!")
```

re.match() vs re.search()

There is a difference between the use of both functions. Both return first match of a substring found in the string, but **re.match()** searches only in the first line of the string and return match object if found, else return none. But if a match of substring is found in some other line other than the first line of string (in case of a multi-line string), it returns none.

While **re.search()** searches for the whole string even if the string contains multi-lines and tries to find a match of the substring in all the lines of string.

re.match() vs re.search()

Example:

```
Substring ='Programming'
String ='''You are learning Python Programming
from expert trainer.'''

# Use of re.search() Method
print(re.search(Substring, String, re.IGNORECASE))

# Use of re.match() Method
print(re.match(Substring, String, re.IGNORECASE))
```

Uplatz

The findall() function

The *findall()* function returns a list containing all matches.
The list contains the matches in the order they are found. If no matches are found, an empty list is returned.
Example:

```
import re
txt = “My name is Imad Jaweed"
x = re.findall(“me", txt)
print(x)
```

The split() function

The split() function returns a list where the string has been split at each match:

Example:

```
txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
print(txt.split())
```

Output:

```
['The', 'rain', 'in', 'Spain']
['The', 'rain', 'in', 'Spain']
```

The sub() function

The sub() function replaces the matches with the text of your choice.

Example: Replace every white-space character with the number 9:

```
import re
txt = “I am Imad Jaweed"
x = re.sub("\s", "9", txt)
print(x)
```

Output:

```
I9am9Imad9Jaweed
```

Uplatz

Thank you

Uplatz