Python Programming

MODULE – III(Part-1)

Agenda:

- **Regular Expression (RE):** Introduction,
- Special Symbols and Characters,
- REs and Python.

Regular Expression (RE):

- A regular expression is a series of characters used to search or find a pattern in a string.
- In other words, a regular expression is a special sequence of characters that form a pattern.
- The regular expressions are used to perform a variety of operations like searching a substring in a string, replacing a string with another, splitting a string, etc.
- The Python programming language provides a built-in module re to work with regular expressions.
- There is a built-in module that gives us a variety of built-in methods to work with regular expressions. In Python, the regular expression is known as RegEx in short form.

Special Symbols and Characters

Creating Regular Expression:

The regular expressions are created using the following.

- Metacharacters
- Special Sequences
- Sets

Metacharacters

- Metacharacters are the characters with special meaning in a regular expression. The following table provides a list of metacharacters with their meaning.

Metacharacters	Meaning
[ ]	Square brackets specifies a set of characters you wish to match.
\	Backlash \ is used to escape various characters including all metacharacters.
.	A period matches any single character (except newline '\n')
^	The caret symbol ^ is used to check if a string starts with a certain character.
$	The dollar symbol $ is used to check if a string ends with a certain character.
*	The star symbol * matches zero or more occurrences of the pattern left to it.
+	The plus symbol + matches one or more occurrences of the pattern left to it.
{ }	Consider this code: {n,m}. This means at least n, and at most m repetitions of the pattern left to it.
|	Vertical bar | is used for alternation (or operator).
()	Parentheses () is used to group sub-patterns.
?	The question mark symbol ? matches zero or one occurrence of the pattern left to it.

[] - Square brackets

- **Square brackets specifies a set of characters you wish to match.**

. - Period

Expression	String	Matched?
[abcd]	Cse	1 match
	cse ds	2 matches
	MREC	No match
	cse ds mrec	3 matches

- **A period matches any single character (except newline '\n').**

Expression	String	Matched?
..	C	No match
	Ds	1 match
	Cse	1 match
	Cseds	2 matches
	cse ds	3 matches(including space)

^ - Caret

- **The caret symbol ^ is used to check if a string starts with a certain character.**

Expression	String	Matched?
^c	C	1 match
	Cse	1 match
	Ds	No match
^ds	cse ds	1 match
	Cse	No match

$ - Dollar

- **The dollar symbol $ is used to check if a string ends with a certain character.**

Expression	String	Matched?
c$	C	1 match
	Mrec	1 match
	Ds	No match

*** - Star**

- **The star symbol * matches zero or more occurrences of the pattern left to it.**

Expression	String	Matched?
abc*	ab	1 match
	abc	1 match
	abcabc	2 match
	cse	No match
	Dsabc	1 match

+ - Plus

- **The plus symbol + matches one or more occurrences of the pattern left to it.**

Expression	String	Matched?
ab+c	Ac	No match (no a character)
	Abc	1 match
	Abbbc	1 match
	Cse	No match (a is not followed by n)
	Dsabc	1 match

? - Question Mark

- **The question mark symbol ? matches zero or one occurrence of the pattern left to it.**

Expression	String	Matched?
ab?c	ac	1 match
	abc	1 match
	abbbc	No match
	abrc	No match
	cseabc	1 match

{} - Braces

- **Consider this code: {n,m}. This means at least n, and at most m repetitions of the pattern left to it.**

Expression	String	Matched?
a{2,3}	abc dat	No match
	abc data	1 match (at daat)
	aabc daaat	2 matches (at aabc and daaat)
	aabc daaaat	2 matches (at aabc and daaaat)

| - Alternation

- **Vertical bar | is used for alternation (or operator).**

Expression	String	Matched?
a|b	Cde	No match
	Ade	1 match (match at ade)
	acdbea	3 matches (at acdbea)

() - Group

- **Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz**

Expression	String	Matched?
(a|b|c)xz	ab xz	No match
	Abxz	1 match (match at abxz)
	axz cabxz	2 matches (at axzbc cabxz)

\ - Backslash

- **Backlash \ is used to escape various characters including all metacharacters. For example,**
- **\$a match if a string contains $ followed by a. Here, $ is not interpreted by a RegEx engine in a special way.**
- **If you are unsure if a character has special meaning or not, you can put \ in front of it. This makes sure the character is not treated in a special way.**

```
>>> print(re.findall(r'$a','dscse$a'))
[]
>>> print(re.findall(r'\$a','dscse$a'))
['$a']
```

Special Sequences

- A special sequence is a character prefixed with \, and it has a special meaning. The following table gives a list of special sequences in Python with their meaning.

Special Sequences	Meaning
\A	Matches if the specified characters are at the start of a string.
\b	Matches if the specified characters are at the beginning or end of a word.
\B	Opposite of \b. Matches if the specified characters are not at the beginning or end of a word.
\d	Matches any decimal digit. Equivalent to [0-9]
\D	Matches any non-decimal digit. Equivalent to [^0-9]
\s	Matches where a string contains any whitespace character. Equivalent to [\t\n\r\f\v].
\S	Matches where a string contains any non-whitespace character. Equivalent to [^ \t\n\r\f\v].
\w	Matches any alphanumeric character (digits and alphabets). Equivalent to [a-zA-Z0-9_]. By the way, underscore _ is also considered an alphanumeric character.
\W	Matches any non-alphanumeric character. Equivalent to [^a-zA-Z0-9_]
\Z	Matches if the specified characters are at the end of a string.

- **\A - Matches if the specified characters are at the start of a string.**

Expression	String	Matched?
\Acse	cse ds	Match
	ds cse	No match

- **\b - Matches if the specified characters are at the beginning or end of a word.**

Expression	String	Matched?
\bcse	Csemrec	Match
	ds csemrec	Match
	Dscsemrec	No match
ds\b	cse ds	Match
	cse ds mrec	Match
	cse dsmrec	No match

- **\B - Opposite of \b. Matches if the specified characters are not at the beginning or end of a word.**

Expression	String	Matched?
\Bcse	Csemrec	No match
	ds csemrec	No Match
	Dscsemrec	Match
ds\B	cse ds	No match
	cse ds mrec	No match
	cse dsmrec	Match

- **\d - Matches any decimal digit. Equivalent to [0-9]**

Expression	String	Matched?
\d	12mrec3	3 matches (at 12mrec3)
	cse ds	No match

- **\D - Matches any non-decimal digit. Equivalent to [^0-9]**

Expression	String	Matched?
\D	cseds123	5 matches (at cseds123)
	1345	No match

- **\s - Matches where a string contains any whitespace character. Equivalent to [\t\n\r\f\v].**

Expression	String	Matched?
\s	cse\tds\nmrec	2 match
	Csedsmrec	No match

- **\S - Matches where a string contains any non-whitespace character. Equivalent to [^ \t\n\r\f\v].**

Expression	String	Matched?
\S	a b	2 matches (at a b)
		No match

- **\w - Matches any alphanumeric character (digits and alphabets). Equivalent to [a-zA-Z0-9_]. By the way, underscore _ is also considered an alphanumeric character.**

Expression	String	Matched?
\w	12&": ;c	3 matches (at 12&": ;c)
	%"> !	No match

- **\W - Matches any non-alphanumeric character. Equivalent to [^a-zA-Z0-9_]**

Expression	String	Matched?
\W	1cse@ds	1 match (at 1cse@ds)
	Cseds	No match

- **\Z - Matches if the specified characters are at the end of a string.**

Expression	String	Matched?
ds\Z	cse ds	1 match
	cse ds mrec	No match
	ds cse.	No match

Sets

- A set is a set character enclosed in [ ], and it has a special meaning. The following table gives a list of sets with their meaning.

Set	Meaning
[aeiou]	Matches with one of the specified characters are present
[d-s]	Matches with any lower case character from d to s
[^aeiou]	Matches with any character except the specified
[1234]	Matches with any of the specified digit
[3-8]	Matches with any digit from 3 to 8
[a-zA-Z]	Matches with any alphabet, lower or UPPER

- **[aeiou] Matches with one of the specified characters are present**

```
>>> print(re.findall(r'[aeiou]','cse and ds dept'))
['e', 'a', 'e']
```

- **[d-s] Matches with any lower case character from d to s**

```
>>> print(re.findall(r'[a-h]','cse and ds dept'))
['c', 'e', 'a', 'd', 'd', 'd', 'e']
```

- **[^aeiou] Matches with any character except the specified**

```
>>> print(re.findall(r'[^aeiou]','cse and ds dept'))
['c', 's', ' ', 'n', 'd', ' ', 'd', 's', ' ', 'd', 'p', 't']
```

- **[1234] Matches with any of the specified digit**

```
>>> print(re.findall(r'[12345]','cse-1 cse-2 cse-3 cse-4 ds'))
['1', '2', '3', '4']
```

- **[a-zA-Z] Matches with any alphabet, lower or UPPER**

```
>>> print(re.findall(r'[a-zA-Z]','This is Cse and Ds Dept'))
['T', 'h', 'i', 's', 'i', 's', 'C', 's', 'e', 'a', 'n', 'd', 'D', 's', 'D', 'e', 'p', 't']
```

REs and Python

Built-in methods of re module

The re module provides the following methods to work with regular expressions.

1. match()
2. search()
3. findall()
4. finditer()
5. sub()
6. split()
7. compile()

1. match() in Python:

- We can use match function to check the given pattern at beginning of target string. If the match is available then we will get Match object, otherwise we will get None.
- The re.match() method will start matching a regex pattern from the very first character of the text, and if the match found, it will return a re.Match object. Later we can use the re.Match object to extract the matching string.

Syntax of re.match(): re.match(pattern, string, flags=0)

- The regular expression pattern and target string are the mandatory arguments, and flags are optional.
- **pattern:** The regular expression pattern we want to match at the beginning of the target string. Since we are not defining and compiling this pattern beforehand (like the compile method). The practice is to write the actual pattern using a raw string.
- **string:** The second argument is the variable pointing to the target string (In which we want to look for occurrences of the pattern).
- **flags:** Finally, the third argument is optional and it refers to regex flags by default no flags are applied.

Eg:

```
>>> str='This is MREC'
>>> print(re.match(r'\w{4}',str))
<re.Match object; span=(0, 4), match='This'>
```

- This re.Match object contains the following items.

- A span attribute that shows the locations at which the match starts and ends. i.e., is the tuple object contains the start and end index of a successful match.Save this tuple and use it whenever you want to retrieve a matching string from the target string
- Second, A match attribute contains an actual match value that we can retrieve using a group() method.
- The Match object has several methods and attributes to get the information about the matching string. Let's see those.

Method	Description
group()	Return the string matched by the regex
start()	Return the starting position of the match
end()	Return the ending position of the match
span()	Return a tuple containing the (start, end) positions of the match.

```
>>> res=re.match(r'\w{4}',str)
>>> res
<re.Match object; span=(0, 4), match='This'>
>>> res.group()
'This'
>>> res.start()
0
>>> res.end()
4
>>> res.span()
(0, 4)
```

search( ) in Python:

- Python regex re.search() method looks for occurrences of the regex pattern inside the entire target string and returns the corresponding Match Object instance where the match found.
- **Syntax:re.search(pattern, string, flags=0)**

```
>>> import re
>>> print(re.search('cse','cse and ds depts in MREC'))
<re.Match object; span=(0, 3), match='cse'>
>>> print(re.search('ds','cse and ds depts in MREC'))
<re.Match object; span=(8, 10), match='ds'>
```

findall( ) in Python:

- The RE module's re.findall() method scans the regex pattern through the entire target string and returns all the matches that were found in the form of a Python list.
- **Syntax:re.findall(pattern, string, flags=0)**

```
>>> print(re.findall('abc','abc abcde bchkdhk abc'))
['abc', 'abc', 'abc']
>>> print(re.findall('cse','cse-A cse-B cse-C cse-D'))
['cse', 'cse', 'cse', 'cse']
>>> print(re.findall('ds','This is ds dept'))
['ds']
```

finditer() in python:

- The re.finditer() works exactly the same as the re.findall() method except it returns an iterator yielding match objects matching the regex pattern in a string instead of a list. It scans the string from left-to-right, and matches are returned in the iterator form. Later, we can use this iterator object to extract all matches.
- In simple words, finditer() returns an iterator over MatchObject objects.

```
import re
res=re.finditer(r'\b\w{3}\b','cse ds raj')
for match in res:
   print(match.group())
print(re.findall(r'\b\w{3}\b','cse ds raj'))
```

outPut:

```
cse
raj
['cse', 'raj']
```

sub( ) in Python:

- The sub( ) method of re object replaces the match pattern with specified text in a string.
- The syntax of sub( ) method is **sub(pattern, text, string).**
- The sub( ) method does not modify the actual string instead, it returns the modified string as a new string.

```
>>> print(re.sub('depts','DEPTS','cse and ds depts') )
>>>cse and ds DEPTS
>>> id='rajasekhar.v86@gmail.com'
>>> print(re.sub('.com','.in',id))
rajasekhar.v86@gmail.in
```

split( ) in Python

- The Pythons re module's re.split() method split the string by the occurrences of the regex pattern, returning a list containing the resulting substrings.

- **Syntax:re.split(pattern, string, maxsplit=0, flags=0)**
- The regular expression pattern and target string are the mandatory arguments. The maxsplit, and flags are optional.
- **pattern:** the regular expression pattern used for splitting the target string.
- **string:** The variable pointing to the target string (i.e., the string we want to split).
- **maxsplit:** The number of splits you wanted to perform. If maxsplit is nonzero, at most maxsplit splits occur, and the remainder of the string is returned as the final element of the list.
- **flags:** By default, no flags are applied.

```
print(re.split('\.','http://www.google.com/'))
['http://www', 'google', 'com/']

str=" cse and ds depts in mrec'
print(re.split(r'\s+',str))
['cse', 'and', 'ds', 'depts', 'in', 'mrec']

str='123-45-678-9'
print(re.split(r'\D',str,maxsplit=1))
['123', '45-678-9']
```

compile() in python:

- Python's re.compile() method is used to compile a regular expression pattern provided as a string into a regex pattern object (re.Pattern).
- Later we can use this pattern object to search for a match inside different target strings using regex methods such as a re.match() or re.search().
- In simple terms, We can compile a regular expression into a regex object to look for occurrences of the same pattern inside various target strings without rewriting it.

Syntax of re.compile()

- re.compile(pattern, flags=0)
- **pattern:** regex pattern in string format, which you are trying to match inside the target string.
- **flags:** The expression's behavior can be modified by specifying regex flag values. This is an optional parameter

```
>>> pattern=re.compile(r'\b\w{4}\b')
>>> result=patten.findall('abcd raaj')
>>> result=pattern.findall('abcd raaj')
>>> result
['abcd', 'raaj']
>>> print(re.findall(pattern,'abcd raaj'))
['abcd', 'raaj']
>>>
```

Examples

Example 1: Write a regular expression to search digit inside a string

```
import re
str="The total no of students are120"
res=re.findall(r'\d',str)
print(res)
```

output:

```
['1', '2', '0']
```

Example2: match 3-letter word anywhere in the string

```
str='MREC civil cse eee ece ds iot ai&ml cs mech'
 print(re.findall(r'\w{3}',str))

['MRE', 'civ', 'cse', 'eee', 'ece', 'iot', 'mec']
```

Example 3 : Extract all characters from the paragraph using Python Regular Expression.

```
import re
str="The total no of students are120"
print(re.findall(r'.',str))
```

['T', 'h', 'e', ' ', 't', 'o', 't', 'a', 'l', ' ', 'n', 'o', ' ', 'o', 'f', ' ', 's', 't', 'u', 'd', 'e', 'n', 't', 's', ' ', 'a', 'r', 'e', '1', '2', '0']

Example 4: Extract all of the words and numbers

```
import re
str="The total no of students are120"
print(re.findall(r'\w+',str))
```

['The', 'total', 'no', 'of', 'students', 'are120']

Example 5: Extract only numbers

```
import re
str="The total no of students are120"
print(re.findall(r'\d+',str))
```

['120']

Example 6: Extract the beginning word

```
import re
```

```
str="The total no of students are120"
print(re.findall(r'^\w+',str))
```

['The']

Example 7: Extract first two characters from each word (not the numbers)

```
import re
str="The total no of students are120"
print(re.findall(r'\b[a-zA-Z].',str))
```

['Th', 'to', 'no', 'of', 'st', 'ar']

Example 8: Find out all of the words, which start with a vowel.

```
import re
str="The total no of students are120"
print(re.findall(r'\b[aeiou]\w+',str))
```

['of', 'are120']

Example 9:Extract date from the string

```
import re
str='Today date is June 09, 2021.'
pattern=r'(\w+)(\s)(\d+)([,]\s)(\d+)'
print(re.findall(pattern,str))
```

[('June', ' ', '09', ', ', '2021')]

```
import re
str='Today date is 06-09-2021'
pattern=r'(\d+)(.)(\d+)(.)(\d+)'
print(re.findall(pattern,str))
```

[('06', '-', '09', '-', '2021')]

```
import re
str='Today date is 06-09-2021'
match=re.search(r'\d{2}-\d{2}-\d{4}',str)
```

```
print(match.group())
```

06-09-2021

Example 10:Extract date from the string

```
import re
str = "Please contact us at rajasekhar.v86@gmail.com for further information."
email = re.findall(r"[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+", str)
print(email)
```

['rajasekhar.v86@gmail.com']

Example 11:Write a Python program that matches a string that has an a followed by zero or more b's.

```
import re
def match(str):
    pattern = 'ab*?'
    if re.search(pattern,str):
        return 'Found a match!'
    else:
        return('Not matched!')

print(match("ac"))
print(match("abc"))
print(match("abbc"))
print(match("abbbc"))
print(match("$12"))
```

Found a match!
Found a match!
Found a match!
Found a match!
Not matched!

Example 12:Replace maximum 2 occurrences of space, comma, or dot with a colon

```
import re
text = 'CSE DS, MREC .'
print(re.sub("[ ,.]", ":", text, 2))
```

Example 13:Develop a Python program to match a string that contains only upper and lowercase letters, numbers, and underscores.

```
import re
str = 'Raj_1254'
pattern='^[a-zA-Z0-9_]*$'
print(re.findall(pattern,str))
```

['Raj_1254']