10 REGULAR EXPRESSIONS

Introduction

A regular expression in a programming language is a special text string used for describing a search pattern. It is extremely useful for extracting information from text such as code, files, log, spreadsheets 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. Ascii or latin letters are those that are on your keyboards and Unicode is used to match the foreign text. It includes digits and punctuation and all special characters like $#@!%, etc.

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.

Simple Character Matches

The following regex functions are used in the python.

SN	Function	Description
1	match	This method matches the regex pattern in the string with the optional flag. It returns true if a match is found in the string otherwise it returns false.
2	search	This method returns the match object if there is a match found in the string.
3	findall	It returns a list that contains all the matches of a pattern in the string.
4	split	Returns a list in which the string has been split in each match.
5	sub	Replace one or many matches in the string.

Example :

```
import re
str = "How are you. How is everything"
matches = re.findall("How", str)
print(matches)
```

Output:

['How', 'How']

Special Characters

Metacharacter/Special character is a character with the specified meaning.

Metacharacter	Description	Example
[]	It represents the set of characters.	"[a-z]"
\	It represents the special sequence.	"\r"
.	It signals that any character is present at some specific place.	"Ja.v."
^	It represents the pattern present at the beginning of the string.	"^Java"
$	It represents the pattern present at the end of the string.	"point"
*	It represents zero or more occurrences of a pattern in the string.	"hello*"
+	It represents one or more occurrences of a pattern in the string.	"hello+"
{}	The specified number of occurrences of a pattern the string.	"java{2}"
|	It represents either this or that character is present.	"javalpoint"
()	Capture and group	

Example

```
import re
str = "The Computer Programming"
#Find all lower case characters alphabetically between "a" and "m":
x = re.findall("[a-m]", str)
print(x)
```

Output:

['h', 'e', 'm', 'e', 'g', 'a', 'm', 'm', 'i', 'g']

```
import re
str = "hello world"
#Check if the string starts with 'hello':
x = re.findall("^hello", str)
if (x):
```

```
  print("Yes, the string starts with 'hello'")
else:
  print("No match")
```

Output :

Yes, the string starts with 'hello'

Character Classes

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Example :

```
import re
str = "The Computer Programming"
#Check if the string starts with "The":
x = re.findall("\AThe", str)
print(x)
if (x):
  print("Yes, there is a match!")
else:
  print("No match")
```

utput:

['The']

Yes, there is a match!

Quantifiers

A quantifier after a token, which can be a single character or group in brackets, specifies ow often that preceding element is allowed to occur. The most common quantifiers are

- the question mark ?
- the asterisk or star character *
- and the plus sign +

```
import re
str = "The Computer Programming requires logic skills"
#Check if the string contains "pu" followed by 0 or more "c" characters:
x = re.findall("puc*", str)
print(x)
if (x):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output :

['pu']

Yes, there is at least one match!

The Dot Character

Dot character represents any character (except newline character)in expression.

```
import re
str = "hello world"
#Search for a sequence that starts with "he", followed by two (any) characters, and an "o":
x = re.findall("he..o", str)
print(x)
```

Output :

['hello']

Greedy Matches

The '*', '+', and '?' quantifiers are all greedy; they match as much text as possible. Adding ? after the quantifier makes it perform the match in non-greedy or minimal fashion; as few characters as possible will be matched.

Example:

```
importre
string="<h1>Heading</h1><p>HTML text.</p>"
match=re.search("<.*>",string)
ifmatch:
print("Greedy")
print("start:",match.start(0))
print("end:",match.end(0))
print("group:",match.group(0))
match=re.search("<.*?>",string)
ifmatch:
print("\nNon-greedy")
print("start:",match.start(0))
print("end:",match.end(0))
print("group:",match.group(0))
```

Output:

Greedy
start: 0
end: 33
*group: <h1>Heading</h1> HTML text. *
Non-greedy
start: 0
end: 4
group: <h1>

Grouping

Match groups match whatever regular expression is inside parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed.

Example:

```
importre
string="<p>HTML text.</p>"
match=re.match("<p>(.*)</p>",string)
ifmatch:
print("start:",match.start(1))
print("end:",match.end(1))
print("group:",match.group(1))
string="'cat': 'Frisky', 'dog': 'Spot', 'fish': 'Bubbles'"
match=re.search("'cat': '(.*?)', 'dog': '(.*?)', 'fish': '(.*?)'",string)
ifmatch:
print("groups:",match.group(1),match.group(2),match.group(3))
```

Output:

start: 3
end: 13
group: HTML text.
groups: Frisky Spot Bubbles

Matching at Beginning or End

^ searches string in the given expression from the beginning

```
import re
str = "hello world"
#Check if the string starts with 'hello':
x = re.findall("^hello", str)
if (x):
  print("Yes, the string starts with 'hello'")
else:
  print("No match")
```

Output :

Yes, the string starts with 'hello'

$ searches string in the given expression at the end.

```
import re
str = "hello world"
#Check if the string ends with 'world':
x = re.findall("world$", str)
if (x):
  print("Yes, the string ends with 'world'")
else:
  print("No match")
```

Output :

Yes, the string ends with 'hello'

Match Objects

The match object contains the information about the search and the output. If there is no match found, the None object is returned.

Example

```
import re
str = "How are you. How many lines ?"
matches = re.search("How", str)
print(type(matches))
print(matches) #matches is the search object
```

Output:

<class '_sre.SRE_Match'>

<_sre.SRE_Match object; span=(0, 3), match='How'>

The Match object methods

There are the following methods associated with the Match object.

span(): It returns the tuple containing the starting and end position of the match.

string(): It returns a string passed into the function.

group(): The part of the string is returned where the match is found.

Example

```
import re
str = "How are you. How is everything"
matches = re.search("How", str)
print(matches.span())
print(matches.group())
print(matches.string)
```

Output:

(0, 3)
How
How are you. How is everything

Substituting

Substituting means replacing one character/string with another. The sub() function replaces the matches with the text of your choice.

Example:

```
import re
#Replace all white-space characters with the digit "9":
str = "The rain in Spain"
x = re.sub("\s", "9", str)
print(x)
```

Splitting a String

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

Example :

```
import re
#Split the string at every white-space character:
str = "The rain in Spain"
x = re.split("\s", str)
print(x)
```

Compiling Regular Expressions

If you want to use the same regexp more than once in a script, it might be a good idea to use a regular expression object, i.e. the regex is compiled.

The general syntax:

re.compile(pattern[, flags])

compile returns a regex object, which can be used later for searching and replacing. The expressions behaviour can be modified by specifying a flag value.

Compiled regular objects usually are not saving much time, because Python internally compiles AND CACHES regexes whenever you use them with re.search() or re.match(). The only extra time a non-compiled regex takes is the time it needs to check the cache, which is a key lookup of a dictionary.

Flags

Compilation flags let you modify some aspects of how regular expressions work. Flags are available in the re module under two names, a long name such as IGNORECASE and a short, one-letter form such as I.

Sr.No.	Flag & Meaning
1	**ASCII, A** Makes several escapes like \w, \b, \s and \d match only on ASCII characters with the respective property.
2	**DOTALL, S** Make, match any character, including newlines
3	**IGNORECASE, I** Do case-insensitive matches
4	**LOCALE, L** Do a locale-aware match
5	**MULTILINE, M** Multi-line matching, affecting ^ and $
6	**VERBOSE, X (for 'extended')** Enable verbose REs, which can be organized more cleanly and understandably

Example:

```
import re
xx = """guru99
careerguru99
selenium"""
k1 = re.findall(r"^\w", xx)
k2 = re.findall(r"^\w", xx, re.MULTILINE)
print(k1)
print(k2)
```

EXERCISE

VERY SHORT ANSWER QUESTIONS:

Q.1. What is regular expression?

Ans. A regular expression in a programming language is a special text string used for describing a search pattern.

Q.2. What is match function?

Ans. This method matches the regex pattern in the string with the optional flag. It returns true if a match is found in the string otherwise it returns false

Q.3. What is quantifiers?

Ans. A quantifier after a token, which can be a single character or group in brackets, specifies how often that preceding element is allowed to occur.

Q.4. What is the function of dot character?

Ans. Dot character represents any character (except newline character)in expression.

Q.5. What characters are used for searching in beginning and at end?

Ans. ^, $

Q.6. What is flag?

Ans. Compilation flags let you modify some aspects of how regular expressions work. Flags are available in the re module under two names, a long name such as IGNORECASE and a short, one-letter form such as I.

SHORT ANSWER QUESTIONS :

1. What are special characters used in regular expression?
2. Explain character classes used in regular expression.
3. Explain different quantifiers.
4. What is greedy match?