UNIT 10: REGULAR EXPRESSIONS

In computing, a regular expression, also referred to as "regex" or "regexp", provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor. Really clever "wild card" expressions for matching and parsing strings - special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. In this example we will learn how to use it using examples.

Things to know before we start:

- Very powerful and quite cryptic
- Fun once you understand them
- Regular expressions are a language unto themselves
- A language of "marker characters" - programming with characters
- It is kind of an "old school" language – compact

This shows the power of Regular Expression!!!

Quick Guide: Regular Expression Patterns

Except for control characters, (+ ? . * ^ $ ( ) [ ] { } | \), all characters match themselves. You can escape a control character by preceding it with a backslash. Some of the popular regular expression syntax that is available in Python:

Modifier	Description
^	Matches beginning of line.

Modifier	Description
$	Matches end of line.
.	Matches any single character except newline. m option allows it to match newline.
\s	Matches whitespace
\S	Matches any non-whitespace character
*	Repeats a character zero or more times
*?	Repeats a character zero or more times (non-greedy)
+	Repeats a chracter one or more times
+?	Repeats a character one or more times (non-greedy)
[aeiou]	Matches a single character in the listed set
[^XYZ]	Matches a single character not in the listed set
[a-z0-9]	The set of characters can include a range
(	Indicates where string extraction is to start
)	Indicates where string extraction is to end
[^…]	Matches any single character not in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.

- Before you can use regular expressions in your program, you must import the library using "import re"
- You can use re.search() to see if a string matches a regular expression similar to using the find() method for strings
- You can use re.findall() extract portions of a string that match your regular expression similar to a combination of find() and slicing: var[5:10]
- You can use re.match() function to search pattern within the test_string. The method returns a match object if the search is successful. If not, it returns None.

Modifiers: Regular expression literals may include an optional modifier to control various aspects of matching. The modifiers are specified as an optional flag.

Modifier	Description
re.I	Performs case-insensitive matching.
re.L	Interprets words according to the current locale. This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior (\b and \B).
re.M	Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string).

Modifier	Description
re.S	Makes a period (dot) match any character, including a newline.
re.U	Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
re.X	Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash) and treats unescaped # as a comment marker.

APPLICATION 1: THE MATCH() FUNCTION

```
import re
line = "A love story of a King and Queen upon their thrown"
matchObj = re.match(r"(.*) and (.*?) .*", line, re.M|re.I)
if matchObj:
    print("matchObj.group(): ",matchObj.group())
    print("matchObj.group(1): ", matchObj.group(1))
    print("matchObj.group(2): ", matchObj.group(2))
else:
    print("No match found!")
```

Output:

matchObj.group(): A love story of a King and Queen upon their thrown

matchObj.group(1): A love story of a King

matchObj.group(2): Queen

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.

Another example

Check if the entered password value matches the given criteria. Criteria we have is:

- At least one digit [0-9]
- At least one lowercase character [a-z]
- At least one uppercase character [A-Z]
- At least one special character [*.!@#$]
- At least 8 characters in length, but no more than 32.
- Ensure that we have at least **three** out of four password criteria met.

```
import re

def password_check(test_str):
    pattern = "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-
Z])(?=.*[*.!@#$]).{8,32}$"
    count=0
    if len(test_str) >=8 and len(test_str) <=32:
        if re.match(".*\\d.*", test_str):
            count += 1
        if re.match(".*[a-z].*",test_str):
            count +=1
        if re.match(".*[A-Z].*", test_str):
            count += 1
        if re.match(".*[*.!@#$].*",test_str):
            count +=1
    #print("Count: ",count)
```

```
    return count >=3

ret = password_check("Pass@w5ord")
if ret:
    print("Password accepted")
else:
    print("Password didnt meet the criteria, enter again!")
```

Dissecting the pattern:

- ^ - Match the beginning of the string
- (?=.*[0-9]) - Require that at least one digit appear anywhere in the string
- (?=.*[a-z]) - Require that at least one lowercase letter appear anywhere in the string
- (?=.*[A-Z]) - Require that at least one uppercase letter appear anywhere in the string
- (?=.*[*.!@#$]) - Require at least one special character appear anywhere in the string.
- .{8,32} - The password must be at least 8 characters long, but no more than 32
- $ - Match the end of the string.

APPLICATION 2: THE SEARCH() FUNCTION

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

```
import re
line = "A love story of a King and Queen upon their thrown"
matchObj = re.search(r"(.*) and (.*?) .*", line, re.M|re.I)
if matchObj:
    print("matchObj.group(): ",matchObj.group())
    print("matchObj.group(1): ", matchObj.group(1))
    print("matchObj.group(2): ", matchObj.group(2))
else:
    print("No match found!")
```

Output:

matchObj.group(): A love story of a King and Queen upon their thrown

matchObj.group(1): A love story of a King

matchObj.group(2): Quee

Matching Versus Searchings

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.

```
# import re module
import re

substring ='window'
String ='''A love story of a King and Queen upon their thrown
    The eyes are said to be the window to the soul
    And within hers shun with a soul of gold '''

# Use of re.search() Method
print(re.search(substring, String, re.I))
# Use of re.match() Method
```

```
print(re.match(substring, String, re.I))
```

Output

<_sre.SRE_Match object; span=(84, 90), match='window'>

None

Let's look at some more programs to practice

```
# Program to extract numbers from a string - findall
import re
string = 'Hello, wish you a happy 40th birthday on 27 day of Month 6 in
2020 year.'
pattern = '\d+'

result = re.findall(pattern, string)
print(result)

# Output: ['40', '27', '6', '2020']
```

```
#If the pattern is not found, re.split() returns a list
# containing the original string.
import re

string = 'June 27 Saturday year 2020 thats twenty twenty.'
pattern = '\d+'

result = re.split(pattern, string)
print("Result after splitting the pattern: ",result)
```

```
# re.sub() returns a string where matched occurrences are
# replaced with the content of replace variable.
# Syntax: re.sub(pattern, replace, string)
#If the pattern is not found, re.sub() returns the original string
import re

string ='''A love story of a King and Queen upon their thrown
    The eyes are said to be the window to the soul
    And within hers shun with a soul of gold '''

# matches all whitespace characters
pattern = '\s+'

# empty string
replace = ''

new_string = re.sub(pattern, replace, string)
print(new_string)

#Output:
#AlovestoryofaKingandQueenupontheirthrownTheeyesaresaidtobethewindowtotheso
ulAndwithinhersshunwithasoulofgold
```