UNIT 2 – Regular Expression and Pattern Matching

Python Regular Expressions

- **Def –** a special text string for describing a search pattern
- Can be used by importing ‘re’ package as → **import re**
- **Eg –** if NameAge = ‘Janice is 22 and Theon is 33’ → ‘Janice’ : 22, ‘Theon’ : 33

  Expressions – ages = re.findall (r ‘| d {1, 3}’, NameAge)

  name = re.findall (r ‘[A – Z][a – z]’, NameAge)

Features of Regex

- Hundreds of code could be reduced to a one line elegant regular expression
- Used to construct compilers, interpreters and text editors
- Used to search and match text patterns
- Used to validate text data formats especially input data
- Popular programming languages have regex capabilities. Python, PERL, Javascript, Ruby, C++, C#

General Uses of Regular Expressions

1. Search a string (search and match)
2. Replace parts of a string (sub)
3. Break string into small pieces (split)
4. Finding a string (findall)

General Concepts

Alternative	|		“cat | mat” == “cat” or “mat” “python | jython” == “python” or “jython”
Grouping	( )	more than 1 pattern embedded in a single line	“gr (e | a) y” == “grey” or “gray” “ra (mil | n (ny | el)” == “ramil” or “ranny” or “ranel”
Quantification	?	0 or 1 of the preceding element	“rani?el” == “raniel” or “ranel”
	*	0 or more of the preceding element	“fo*ot” == “foot” or “foooot” or “fooooot”
	+	1 or more of the preceding element	“too+fan” == “toofan” or (REMAINING)
	{m, n}	**m to n** times of the preceding element	“go{2, 3}gle” == “google” or “gooogle” “6{3}” == “666” “s{2, }” == “ss” or “sss” or “ssss”…
Anchors	^ (start / begin anchors)	Matches the starting position within the string	“^obje” == “object” or “object – oriented” “^2014” == “2014” or “2014/20/07”
	$ (end operators)	Matches the ending position within the string	“gram$” == “program” or “kilogram” “2014$” == “20/07/2014” or “2013 – 2014”

Metacharacters	. (dot)	Matches any single character	“bat.” == “bat” or “bats” or “bata” “87.1” == “8741” or “8751” or “8761”
	[ ]	Matches a single character that is contained within the brackets	“[xyz]” == “x” or “y” or “z” “[aeiou]” == any vowel “[0123456789]” == any digit
	[-]	Matches a single character that is contained within the brackets and the specified range	“[a – c]” == “a” or “b” or “c” “[a – z A – Z]” == all letters (both lowercase and uppercase) “[0 – 9]” == all digits
	[^]	Matches a single character that is not contained within the brackets	“[^aeiou]” == any non – vowel “[^0 – 9]” == any non – digit “[^xyz]” == any character but not “x” or “y” or “z”
Character class	\d	Matches a decimal digit	[0 – 9]
	\D	Matches non – digits	
	\s	Matches any white space character	\t == tab \n == new line \f == form
	\S	Matches any non – white space character	
	\w	Matches alphanumeric character class	[a – z A – Z 0 – 9_]
	\W	Matches non – alphanumeric character class	[^a – z A – Z 0 – 9_]
	\w+	Matches 1 or more words / characters	
	\A	Matches the beginning of the string	
	\z	Matches the end of the string	
	\b	Returns a match where the specified characters are at the beginning or at the end of a word	
	\B	Returns a match where the specified characters are present, but not at the beginning or at the end of a word	

<u>The search ( ) function</u>

- Scans through the input string and tries to match at any location
- Searches for the 1st occurrence of RE pattern within the string with optional flags

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

pattern → the regular expression to be matches
string → the string that would be searched to match the pattern anywhere in the string
flags → also called **modifiers** → option flags that can be specified using bitwise OR (|) operator

Modifier / Option Flags	Description
re.I	Performs case – insensitive matching
re.L	Interprets words according to the current locale. This interpretation affects the alphabetic group (\w, \W, \b, \B)
re.M	Makes $ match the end of a line (not just the end of the string) Makes ^ match the start of any line (not just the start of the string)
re.S	Makes a . (period / dot) match any character, including a new line
re.X	Allows comments in Regex

Example –

# Program 1	**# Program 2**

```
import re
txt = “The rain in Spain”
x = re.search (“ ^The .* Spain$ ”, txt)
if x:
        print (“Match”)
else:
        print (“Mismatch”)
```
	```
	import re      # importing RE built – in module
	text = “This is my First Regular Expression Program”
	patterns = [‘first’, ‘that’, ‘program’]

	for pattern in patterns:
	        if re.search (pattern, text, re.I):
	                print (“Match”)
	        else:
	                print (“Mismatch”)
	```

<u>The “match object”</u>

Used for information about the matching strings
“match object” instances also have several methods and attributes –

Method	Purpose
group ( )	Return the string matched by the RE
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

Example –

```
import re
txt = “The rain in Spain”
x = re.search (“r”, txt)
print (“The matched string are : ”, x.group( ) )
```

Findall

Returns all non – overlapping matches of pattern in string, as a list of strings
The string is scanned left – to – right → matches are returned in the order found
If 1 or more groups are present in the pattern → returns a list of groups → this will be a list containing each element as a tuple if the pattern has more than 1 group
Empty matches are included in the result unless they touch the beginning of another match
Syntax – re.findall (pattern, strings, flags = 0)

Example –	#Print list of all matches
	import re
	txt = "The rain in Spain"
	x = re.findall ("ai", txt)
	print (x)
		#Returns an empty list if no match was found
		import re
		txt = "The rain in Spain"
		x = re.findall ("Portugal", txt)
		print (x)

Python Raw Strings

String literals prefixed with a 'r' or 'R'

Example – r "Hello"

Do not treat backslashes ('\') as part of an escape sequence
Will be printed normally as a result
This feature helps pass string literals which cannot be decoded using normal ways → like the sequence '\x'

Example –	#Print using raw string
	import re
	s = r "Hello \t from AskPython \n Hi"
	print (s)
		#Without using raw string
		import re
		s = "Hello \t from AskPython \n Hi"
		print (s)

Output –	Hello \t from AskPython \n Hi	Hello from AskPython Hi

The re.compile ( ) method

Combines a regular expression pattern into pattern objects, which can be used for pattern matching
Also helps to search a pattern again without rewriting it

Syntax – re.compile (pattern)

Example	**Output**

import re
pattern = re.compile ('TP')
result = pattern.findall ('TP Tutorialspoint TP')
print(result)
result2 = pattern.findall ('TP is most popular tutorials site of India')
print(result2)
	['TP', 'TP']
	['TP']

Modifying strings

RE → compiled into pattern objects, which have methods for various operators such as searching for pattern matches or performing string substitutions.

RE are also commonly used to modify strings in various ways using the following pattern methods –

Method	Purpose	Syntax	Example	Output
split ( )	Split the string into a list, splitting it wherever the RE matches	re.split (string, [maxsplit])	txt = “hello, my name is Peter, I am 26 years old” x = txt.split (“,”) print (x)	[‘hello’, ‘my name is Peter’, ‘I am 26 years old’]

			txt = “apple # banana # cherry # orange”
			x = txt.split (“#”, 1)
			print(x)
				[‘apple’, ‘# banana # cherry # orange’]

sub ( )	Find all substrings where the RE matches and replace them with a different string.
	
	Replaces all occurrences of the RE pattern in string with **repl,** substituting all occurrences unless **max** provided.
	
	Returns the modified string.
		re.sub (pattern, repl, string, max = 0)	import re
			DOB = “25-01-1991 # This is Date of Birth”
			
			# Delete Python – style commands
			Birth = re.sub (r ‘#.*$’, “ ”, DOB)
			print (“Date of Birth : ”, Birth)
			
			# Remove anything other than digits
			Birth1 = re.sub(r ‘\D’, “ ”, Birth)
			print (“Before substituting DOB : ”, Birth1)
			
			# Substituting the ‘ – ’ with ‘.’ (dot)
			New = re.sub (r ‘\W’, “.”, Birth)
			print (“After substituting DOB : ”, New)	

subn ( )	Does the same thing as sub ( ), but returns the new string and the no. of replacements			

Finditer

Return an iterator yielding MatchObject instances over all non – overlapping matches for the RE pattern in string

The string is → scanned from left – to – right → matches are returned in the order found

Empty matches → are included in the result → unless they touch the beginning of another match

Syntax – re.finditer (pattern, string, flags = 0)

Example	Output
import re	
string = “Python java c++ perl shell ruby tcl c#”	
print (re.findall (r “\bc [\W+]* ”, string, re.M | re.I))	[‘c++’, ‘c#’]
print (re.findall (r “\bp [\w]* ”, string, re.M | re.I))	[‘Python’, ‘perl’]
print (re.findall (r “\bs [\w]* ”, string, re.M | re.I))	[‘shell’]
print (re.sub (r ‘ \W+ ’, “ ”, string))	Pythonjavacperlshellrubytclc
it = re.finditer (r “\bc [(\W \s)]* ”, string)	‘c++’ was found between the indices (12, 16)
	‘c#’ was found between the indices (37, 39)
for match in it:	
print (“ ‘{gh}’ was found between the indices	
{st}”.format (gh = match.group ( ),	
st = match.span ( )))	