III UNIT – REGULAR EXPRESSIONS

3.1 INTRODUCTION:-

- Manipulating text/data is a big thing.
- Look very carefully to what computers primarily do today. Word processing, "fill-out-form" Web pages, streams of information coming from a database dump, stock quote information, news feeds and the list goes on and on.
- Because we may not know the exact text or data that we have programmed our machines to process, it becomes advantageous.
- Regular expressions (REs) provide such an infrastructure for advanced text pattern matching, extraction, and/or search-and-replace functionality.
- REs are simply strings that use special symbols and characters to indicate pattern repetition or to represent multiple characters so that they can "match" a set of strings with similar characteristics described by the pattern.
- They supports for matching.
- Python has a module named **re** to work with RegEx.
- Throughout this chapter, you will find references to searching and matching.
- When we are strictly discussing regular expressions with respect to patterns in strings, we will say "matching,"
- Referring to the term pattern-matching. In Python terminology, there are **two** main ways to accomplish pattern-matching: i) **searching**, i.e., looking for a pattern match in any part of a string, and ii) **matching,** i.e., attempting to match a pattern to an entire string (starting from the beginning).
- Searches are accomplished using the **search()** function or method, and
- Matching is done with the **match()** function or method.
- A Regular Expression (RegEx) is a sequence of characters that defines a search pattern. For example:- ^a...s$
- Here the pattern is any five letter string starting with **a** and ending with **s**.
- A pattern defined using RegEx can be used to match against a string.

Edit with WPS Office

Expression	String	Matched?
^a...s$	abs	No match
	alias	Match
	abyss	Match
	Alias	No match
	An abacus	No match

Ex:- import re

```
pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)
if result:
        print("Search successful.")
else:
        print("Search unsuccessful.")
```

Note:- Here, we used 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.

3.2 Meta Characters:- Meta characters are characters that are interpreted in a special way by a RegEx engine. Here's a list of meta characters:

[ ] . ^ $ * + ? {} () \ |

Edit with WPS Office

Table Common Regular Expression Symbols and Special Characters

Notation	*Description*	*Example RE*
Symbols		
`literal`	Match literal string value `literal`	`foo`
`re1|re2`	Match regular expressions `re1` or `re2`	`foo|bar`
`.`	Match *any character* (except NEWLINE)	`b.b`
`^`	Match *start of string*	`^Dear`
`$`	Match *end of string*	`/bin/*sh$`
`*`	Match *0 or more* occurrences of preceding RE	`[A-Za-z0-9]*`
`+`	Match *1 or more* occurrences of preceding RE	`[a-z]+\.com`
`?`	Match *0 or 1* occurrence(s) of preceding RE	`goo?`
`{N}`	Match `N` occurrences of preceding RE	`[0-9]{3}`
`{M,N}`	Match from `M` to `N` occurrences of preceding RE	`[0-9]{5,9}`

. - Period

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

Expression	String	Matched?
..	a	No match
	ac	1 match
	acd	1 match
	acde	2 matches (contains 4 characters)

^ - Caret:-

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

Expression	String	Matched?
^a	a	1 match
	abc	1 match
	bac	No match
^ab	abc	1 match
	acb	No match (starts with a but not followed by b)

$ - Dollar:-

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

Edit with WPS Office

Expression	String	Matched?
a$	a	1 match
	formula	1 match
	cab	No match

*** - Star:-**

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

Expression	String	Matched?
ma*n	mn	1 match
	man	1 match
	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

+ - Plus

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

Expression	String	Matched?
ma+n	mn	No match (no a character)
	man	1 match
	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

? - Question Mark

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

Expression	String	Matched?
ma?n	mn	1 match
	man	1 match
	maaan	No match (more than one a character)
	main	No match (a is not followed by n)
	woman	1 match

Edit with WPS Office

{} - 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 daat	1 match (at daat)
	aabc daaat	2 matches (at aabc and daaat)
	aabc daaaat	2 matches (at aabc and daaaat)

- Let's try one more example. This RegEx [0-9]{2, 4} matches at least 2 digits but not more than 4 digits

Expression	String	Matched?
[0-9]{2,4}	ab123csde	1 match (match at ab123csde)
	12 and 345673	3 matches (12, 3456, 73)
	1 and 2	No match

| - 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:-

- Backslash \ is used to escape various characters including all meta characters.
- 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

Edit with WPS Office

front of it.

- ❖ This makes sure the character is not treated in a special way.

Special Sequences:-

- ❖ Special sequences make commonly used patterns easier to write.
- ❖ 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"
\b	Returns a match where the specified characters are at the beginning or at the end of a word	r"\bain"
	(the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word	r"\Bain"
	(the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\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"

Edit with WPS Office

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

Expression	String	Matched?
\Athe	the sun	Match
	In the sun	No match

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

Expression	String	Matched?
\bfoo	football	Match
	a football	Match
	afootball	No match
foo\b	the foo	Match
	the afoo test	Match
	the afootest	No match

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

Expression	String	Matched?
\Bfoo	football	No match
	a football	No match
	afootball	Match
foo\B	the foo	No match
	the afoo test	No match
	the afootest	Match

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

Expression	String	Matched?
\d	12abc 3	3 matches (at 12abc3)
	Python	No match

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

Edit with WPS Office

Expression	String	Matched?
\D	1ab34"50	3 matches (at 1ab34"50)
	1345	No match

\s - Matches where a string contains any whitespace character

Expression	String	Matched?
\s	Python RegEx	1 match
	PythonRegEx	No match

S - Matches where a string contains any non-whitespace character

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

\w - Matches any alphanumeric character (digits and alphabets and Under score). i,e a-z, A-Z, 0 - 9 & '_'

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

\W - Matches any non-alphanumeric character

Expression	String	Matched?
\W	1a2%c	1 match (at 1a2%c)
	Python	No match

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

Expression	String	Matched?
Python\Z	I like Python	1 match
	I like Python Programming	No match
	Python is fun.	No match

3.3 REs & PYTHON:-

- Python has a module named re to work with regular expressions.
- It is a sequence of characters that forms a search pattern. can be used to check if a string contains the specified search pattern.

Edit with WPS Office

- To use it, we need to import the module. i.e import re
- The module defines several functions and constants to work with RegEx.
- The different Functions/Methods that are present
  1. Compile (pattern, flags=0)
  2. Match(pattern, string, flags=0)
  3. Search(pattern, string, flags=0)
  4. Findall(pattern, string)
  5. Split(pattern, string, max=0)
  6. Sub(pattern, repl, string, max=0)
  7. Subn(pattern, repl, string)

1. Compile (pattern, flags=0):-

- Complies RE pattern with any optional flags and return a regex object

2. re.match (pattern, string, flags=0):-

- Attempt to match RE pattern to string with optional flags; return match object on success, none on failure.

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

- The re.search() method takes two arguments: a pattern and a string.
- The method looks for the first location (first occurrence) where the RegEx pattern produces a match with the string.
- If the search is successful, re.search() returns a match object; if not, it returns None.
- Syntax is: match = re.search(pattern, str)

Ex:

```
import
  re
string = "Python is fun"
 # check if 'Python' is at the beginning
 match = re.search('\APython', string)
    if
match:
 print("pattern found inside the string")
  else:
    print("pattern not found")

  # Output: pattern found inside the
```


Edit with WPS Office

4. re.findall (pattern, string)

- The findall() method returns a list of strings containing all matches.
- It looks for all occurrences of pattern in string.

Ex:

```
# Program to extract numbers from a string
import re
string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+'
result = re.findall(pattern, string)
print(result)
# Output: ['12', '89', '34']
```

5. re.split(pattern, string, max=0)

- The re.split method splits the string where there is a match and returns a list of strings where the splits have occurred.

Ex:

```
import re
string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'
result = re.split(pattern, string)
print(result)
# Output: ['Twelve:', ' Eighty nine:', '.']
```

6. Sub(pattern, repl, string, max=0)

- The method returns a string where matched occurrences are replaced with the content of replace variable.

Ex:

```
    # Program to remove all
whitespaces
    import re
    # multiline string
    string = 'abc 12\
    de 23 \n f45 6'
    # matches all whitespace
characters
```


Edit with WPS Office

```
pattern = '\s+'
# empty string
replace = ''
new_string = re.sub(pattern, replace,
string)
print(new_string)
# Output: abc12de23f456
```

Note:- If the pattern is not found, **re.sub()** returns the original string.

- ❖ We can control the number of replacements by specifying the count parameter:
- ❖ For this you have to pass **count** as a fourth parameter to the **re.sub()** method.

Ex:

```
import re
# multiline string
string = 'abc 12\
de 23 \n f45 6'
# matches all whitespace characters
pattern = '\s+'
replace = ''
new_string = re.sub(r'\s+', replace,
string, 1)
print(new_string)
# Output:
# abc12de 23
# f45 6
```

7. re.subn()

- ❖ The re.subn() is similar to re.sub().
- ❖ It returns a tuple of 2 items containing the new string and the number of substitutions made.

Ex- :

```
# Program to remove all whitespaces
import re
# multiline string
string = 'abc 12\
de 23 \n f45 6'
# matches all whitespace characters
pattern = '\s+'
# empty string
replace = ''
```


Edit with WPS Office

```
new_string = re.subn(pattern, replace, string)
print(new_string)
# Output: ('abc12de23f456', 4)
```

3.3.1Match object methods:-

- A Match Object is an object containing information about the search and the result.

Note: If there is no match, the value None will be returned, instead of the Match Object.

- Different match objects methods are match.group(), match.groups(),match.start(), match.end() and match.span() etc...

1. **match.group() :-** The group() method returns the part of the string where there is a match.

   Ex:-

```
import re
string = '39801 356, 2102 1111'
# Three digit number followed by space followed by two digit
number
pattern = '(\d{3}) (\d{2})'
# match variable contains a Match object.
match = re.search(pattern, string)
if match:
  print(match.group())
else:
  print("pattern not found")
# Output: 801 35
```

- Here, match variable contains a match object.
- Our pattern (\d{3}) (\d{2}) has two subgroups (\d{3}) and (\d{2}).
- You can get the part of the string of these parenthesized subgroups. Here's how:

  >>> match.group(1)

  '801'

  >>> match.group(2)

  '35'

Edit with WPS Office

```
>>> match.group(1, 2)
('801', '35')
>>> match.groups()
('801', '35')
```

- match.start(), match.end() and match.span()
- The start() function returns the index of the start of the matched substring. Similarly, end() returns the end index of the matched substring.
- >>> match.start() O/P:- 2
- >>> match.end() O/P:- 8
- The span() function returns a tuple containing start and end index of the matched part.
- >>> match.span() O/P:- (2, 8)

3.4 MULTITHREADED PROGRAMMING

3.4.1 Introduction:-

- Multithreading allows you to break down an application into multiple sub-tasks and run these tasks simultaneously.
- Such programming tasks can be organized or partitioned into multiple streams of
  execution where each has a specific task to accomplish.
- Depending on the application, these subtasks may calculate intermediate results that could be merged into a final piece of output.
- MT programming is ideal for programming tasks that are asynchronous in nature, require multiple concurrent activities, and where the processing of each activity may be nondeterministic, i.e., random and unpredictable.
- If you use multithreading properly, your application speed, performance will be improved.
- Using an MT program with a shared data structure such as a Queue this programming task can be organized with a few threads that have specific functions to perform:

  1. UserRequestThread
  2. RequestProcessor

Edit with WPS Office

3.ReplyThread

1. UserRequestThread: - Responsible for reading client input from an I/O channel. A number of threads would be created by the program, one for each current client, with requests being entered into the queue.

2. RequestProcessor: - A thread that is responsible for retrieving requests from the queue and processing them, providing output for yet a third thread.

3. ReplyThread: - Responsible for taking output destined for the user and either sending it back, if in a networked application, or writing data to the local file system or database.

Note:- Organizing this programming task with multiple threads will provide the following benefits

1. Reduces the complexity of the program.
2. Enables clean implementation.
3. Efficient and well organized execution.

3.4.2. Threads & Processes

- A process is a program in execution.
- Each process has its own address space, memory, a data stack and other auxiliary data to keep track of execution.
- Threads are light-weight processes and they do not require much memory overhead.
- They are cheaper than processes, they all execute within the same process and share the same context.
- A thread has a beginning, an execution sequence, and a conclusion.
- It has an instruction pointer that keeps track of where within its context it is currently running.
- It can be pre-empted (interrupted)
- Each thread has its own job to do; you merely have to design each type of thread to do one thing and do it well.
- The logic in each thread is typically less complex because it has a specific job to do. For example, the UserRequestThread simply reads input from a user and places the data into a queue for further processing by another thread, etc.

Edit with WPS Office

- It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.
- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
- If two or more threads access the same piece of data, inconsistent results may arise because of the ordering of data access. This is commonly known as a **race condition**.
- Most thread libraries come with some sort of synchronization primitives that allow the thread manager to control execution and access.
- Threads may not be given equal and fair execution time this is because some functions block until they have completed. If not written specifically to take threads into account, this skews the amount of CPU time in favour of such greedy functions.

3.5 PYTHON, THREADS, AND THE GLOBAL INTERPRETER LOCK

3.5.1 Global Interpreter Lock (GIL)

- Execution of Python code is controlled by the Python Virtual Machine (the interpreter main loop).
- Python was designed in such a way that only one thread of control may be executing in this main loop, similar to how multiple processes in a system share a single CPU.
- Many programs may be in memory, but only one is live on the CPU at any given moment.
- Likewise, although multiple threads may be "running" within the Python interpreter, only one thread is being executed by the interpreter at any given time.
- Access to the Python Virtual Machine is controlled by the global interpreter lock (GIL).
- This lock ensures that exactly one thread is running.
- The Python Virtual Machine executes in the following manner in an MT environment:

1. Set the GIL

Edit with WPS Office

2. Switch in a thread to run
3. Execute either ...
    a. For a specified number of bytecode instructions, or
    b. If the thread voluntarily yields control (can be accomplished time.sleep(0))
4. Put the thread back to sleep (switch out thread)
5. Unlock the GIL, and ...
6. Do it all over again (lather, rinse, repeat)

3.5.2. Exiting Threads

- When a thread completes execution of the function it was created for, it exits.
- Threads may also quit by calling an exit function such as tHRead. exit(), or any of the standard ways of exiting a Python process, i.e., sys.exit() or raising the SystemExit exception.
- When the main thread exits, all other threads die without cleanup.
- The other module, threading, ensures that the whole process stays alive until all "important" child threads have exited.
- Main threads should always be good managers, though, and perform the task of knowing what needs to be executed by individual threads, what data or arguments each of the spawned threads requires, when they complete execution, and what results they provide.
- With this functionality the main threads can collect and combine individual results into a final and meaningful conclusion.

3.5.3 Accessing Threads from Python

- Python supports multithreaded programming, depending on the operating system that it is running on.
- It is supported on most Unix-based platforms, i.e., Linux, Solaris, MacOS X, *BSD, as well as Win32systems.
- To tell whether threads are available for your interpreter, simply attempt to import the

thread module from the interactive interpreter.

```
>>> import thread
```

3.5.4. Life without Threads

Edit with WPS Office

❖ For examples, take the time.sleep() function to know how threads will work.

❖ time.sleep() takes a argument and "sleeps" for the given number of seconds, meaning that execution is temporarily halted for the amount of time specified.

❖ Let us create two "time loops," one that sleeps for 4 seconds and one that sleeps for 2 seconds, loop0() and loop1(), respectively.

❖ If we were to execute loop0() and loop1() sequentially in a one-process or singlethreaded program, the total execution time would be at least 6 seconds.

❖ There may or may not be a 1-second gap between the starting of loop0() and loop1(), and other execution overhead which may cause the overall time to be bumped to 7 seconds.

Example 1. Loops Executed by a Single Thread

❖ Executes two loops consecutively in a single-threaded program.

❖ One loop must complete before the other can begin.

❖ The total elapsed time is the sum of times taken by each loop.

#!/usr/bin/env python (This command is known as a Shebang Line.If you have installed many versions of Python, then #!/usr/bin/env ensures that the interpreter will use the first installed version on your environment's $PATH)

```
from time import sleep, ctime
def loop0():
print 'start loop 0 at:', ctime()
sleep(4)
print 'loop 0 done at:', ctime()
def loop1():
print 'start loop 1 at:', ctime()
sleep(2)
print 'loop 1 done at:', ctime()
def main():
print 'starting at:', ctime()
loop0()
```


Edit with WPS Office

```
        loop1()
        print 'all DONE at:', ctime()

        if __name__ == '__main__':
        main()
```

o/p:-

```
starting at: Sun Aug 13 05:03:34 2006
start loop 0 at: Sun Aug 13 05:03:34 2006
loop 0 done at: Sun Aug 13 05:03:38 2006
start loop 1 at: Sun Aug 13 05:03:38 2006
loop 1 done at: Sun Aug 13 05:03:40 2006
all DONE at: Sun Aug 13 05:03:40 2006
```

Note:- To cut down the overall running time we have to run them in parallel This is achieved by using multi threading concept

3.5.5. Python Threading Modules

- Python provides several modules to support MT programming, including the tHRead, tHReading, and Queue modules.
- The thread and threading modules allow the programmer to create and manage threads.
- The thread module provides basic thread and locking support, while threading provides higher-level, fully featured thread management.
- The Queue module allows the user to create a queue data structure that can be shared across multiple threads.
- We recommend avoiding the tHRead module for many reasons.

  1. Compared to the thread module, the threading module is much more supportive
  2. Lower-level thread module has only one synchronization primitives (Lock conditions which handle race condition) while threading has many.
  3. In thread there is no control when your process exits. i.e When the main thread finishes, all threads will also die, without warning or proper cleanup. But tHReading allows the important child threads to finish first before exiting.
  4. Use of the tHRead module is recommended only for experts desiring lower-level

Edit with WPS Office

thread access.

3.6 tHRead Module

❖ tHRead Module Functions

1. start_new_thread(function, args, kwargs=None):- Spawns a new thread and execute function with the given args and optional kwargs, [kwargs works just like *args , but instead of accepting positional arguments it accepts keyword (or named) arguments. ... Like args ,]
2. kwargs is just a name that can be changed to whatever you want
3. allocate_lock():- Allocates LockType lock object
4. exit():- Instructs a thread to exit

❖ LockType Lock Object Methods

1. acquire(wait=None) :- Attempts to acquire lock object
2. locked():- Returns True if lock acquired, False otherwise
3. release():- Releases lock

Example

```
#!/usr/bin/env python
 import thread
 from time import sleep, ctime
 def loop0():
 print 'start loop 0 at:', ctime()
 sleep(4)
 print 'loop 0 done at:', ctime()
 def loop1():
 print 'start loop 1 at:', ctime()
 sleep(2)
 print 'loop 1 done at:', ctime()
 def main():
 print 'starting at:', ctime()
 thread.start_new_thread(loop0, ())
 thread.start_new_thread(loop1, ())
 sleep(6)
 print 'all DONE at:', ctime()
 if __name__ == '__main__': // it implies that the module is being run standalone by
the user //and we can do corresponding appropriate actions
 main()
```


Edit with WPS Office

O/P:-
starting at: Sun Aug 13 05:04:50 2006
start loop 0 at: Sun Aug 13 05:04:50 2006
start loop 1 at: Sun Aug 13 05:04:50 2006
loop 1 done at: Sun Aug 13 05:04:52 2006
loop 0 done at: Sun Aug 13 05:04:54 2006
all DONE at: Sun Aug 13 05:04:56 2006

Note: - Rather than taking a full 6 or 7 seconds, our script now runs in 4, the length of time of our longest loop, plus any overhead.

3.7. tHReading Module

3.7.1 tHReading Module Objects Description

Thread:- Object that represents a single thread of execution

Lock:- Primitive lock object (same lock object as in the tHRead module)

RLock:- Re-entrant lock object provides ability for a single thread to (re)acquire an already-held lock (recursive locking)

Condition:- Condition variable object causes one thread to wait until a certain "condition"
has been satisfied by another thread, such as changing of state

Semaphore:- Provides a "waiting area"-like structure for threads waiting on a lock

BoundedSemaphore:- Similar to a Semaphore but ensures it never exceeds its initial value

Timer:- Similar to Thread except that it waits for an allotted period of time before running

3.7.1. Thread Class

❖ The THRead class of the threading is your primary executive object.

❖ It has a variety of functions not available to the thread module,.

start():- Begin thread execution

run():- Method defining thread functionality (usually overridden by application writer in a subclass)

join(timeout = None):- Suspend until the started thread terminates

getName():- Return name of thread

setName(name):- Set name of thread

isAlive():- Boolean flag indicating whether thread is still running

Edit with WPS Office

3.7.2Other Threading Module Functions

activeCount() Number of currently active Thread objects

currentThread() Returns the current THRead object

enumerate() Returns list of all currently active Threads

settrace(func) Sets a trace function for all threads

setprofile(func) Sets a profile function for all threads

3.7.3 Threading-Related Standard Library

tHRead :- Basic, lower-level thread module

threading:- Higher level threading and synchronization objects

Queue:- Synchronized FIFO queue for multiple threads

mutex:- Mutual exclusion objects

SocketServer:- TCP and UDP managers with some threading control

Edit with WPS Office