UNIT 3

REGULAR EXPRESSIONS AND MULTITHREADED PROGRAMMING

SIA GROUP

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Define regular expressions

Answer :

Model Paper-I, Q1(e)

The regular expressions provided an infrastructure to perform certain functions such as extraction, advance text pattern matching and search-and-replace. Regular expressions are in the form of strings including special symbols and characters for representing pattern repetition or several characters in order to match the strings with pattern.

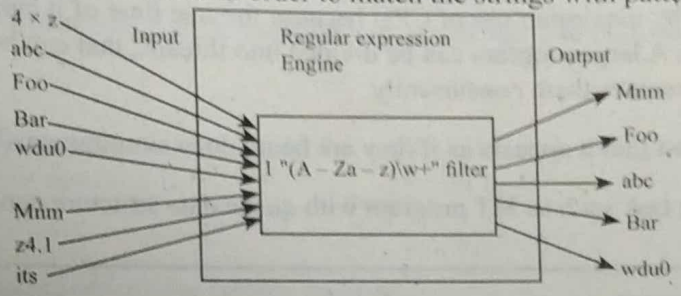

Figure: Filtering the Regular Expressions with Valid Identifiers

Python provides a standard library remodule to support regular expressions. They can contain text with upper case or lower case alphabets and numeric digits including or excluding the specialized alphabets.

Q2. Listout various special symbols used in python.

Answer :

Model Paper-II, Q1(e)

The most commonly used special symbols in python are as follows,

Special symbol	Description	Example
literal	It matches the literal string value literal	foo
.	It matches any character	x.x
$	It matches the end of string	/bin/*sh$
re1/re2	It matches the regular expression re1 or re2	foo/bar
*	It matches 0 or more occurrences of the previous regular expression	[A-Za-zo-9] *
^	It matches the beginning of string	^string
+	It matches 1 or more occurrences of previous regular expression	[a – z] +\.com
{N}	It matches N occurrences of previous regular expression	[0 – 9] {2}
?	It matches 0 or 1 occurrences of previous regular expression.	xyz?
(...)	It matches the enclosed regular expression and saves it as subgroup.	([0 – 9] {2}) ?, x(yz/a) pqr
{M, N}	It matches from M to N occurrences of previous regular expression.	[0 – 9] {2, 6}
[...]	It matches any single character from the character class	[aeiou]
[^...]	It does not match any of the character from character class	[^aeiou], [^A – Za – zo – 9_]
[..x – y..]	It matches a single character from x to y	[0 – 9], [A – Za – z]
(*|+|?|{})?	It does not apply "non-greedy" versions of the specified symbols	.*?[a – z]

Q3. List out various special characters in python.

Answer :

The most commonly used special characters in Python are as follows,

Special Character	Description	Example
\d	It matches a decimal digit like [0 – 9]	data \d+.txt
\w	It matches alphanumeric character like [A-Za-zo-9]	[A-Za-z]\w+
\b	It matches as word boundary	\b the \b
\c	It matches any special character C precisely	\.,\\,*
\s	It matches white space character as [\n\t\r\v\f]	of\sthe
\A(\z)	It matches the beginning of string.	\ADear
\nn	It matches the saved subgroup nn	price:\20

Q4. What is multithreaded programming?

Answer : Model Paper-III, Q1(s)

Multithreaded programming is a concept in which a program is divided into two or more subprograms that can be executed at a time. The part of program or subprogram is called a thread and each thread executes separately. It is a powerful approach that enables to write efficient programs by maximum use of CPU because the idle time of it can be reduced to minimum. It allows to perform multiple things at a time. A large program can be divided into threads, that can be executed parallely. The processor switches between the threads and executes them concurrently.

It executes every thread so fast that it appears as if they are being done simultaneously.

For example, a programming task such as MT program with queue data structure can be accomplished with few threads that have certain functions.

Q5. Define threads.

Answer :

Thread can be defined as a set of executable instructions that are executed independently. A program can be divided into multiple subprograms and each subprogram is called a thread. Every individual thread is executed separately, thereby decreasing the execution time of a program.

Q6. Define processes.

Answer : Model Paper-I, Q1(f)

Process is the fundamental concept of operating systems structure. A program under execution is referred to as a process. It can also be defined as an active entity that can be assigned to a processor for execution. A process is a dynamic object that resides in main memory. A process includes the current values of the program counter and processor's registers. Each process possesses its own virtual CPU. A process contains the following two elements,

(i) Program code and

(ii) A set of data.

Q7. What is GIL.

Answer : Model Paper-III, Q1(f)

Python virtual machine is responsible for controlling the executing of python code. Only one thread of control executes in the main loop among multiple threads. Access to this python virtual machine is controlled by Global Interpreter Lock (GIL). It allows only one thread to execute at a time in CPU. Working of Python virtual machine in MT environment is as follows,

1. Initially set the GIL
2. Switch the thread to run

3. Execute either (i) or (ii)
   (i) For fixed number of bytecode instructions.
   (ii) If thread generates control voluntarily.
4. Return the thread back to sleep.
5. Unlock GIL
6. Repeat the complete process.

Q8. Write about thread module.

Answer :

Model Paper-II, Q1(f)

The tHRead module is responsible to spawn the threads. It provides a synchronization data structure called lock object. Examples of lock object are mutual exclusion lock, simple lock, primitive lock, mutex and binary semaphore. Some of the commonly used thread functions and lock type lock object methods.

tHRead Module Functions

1. **allocate_lock()**
   This method is used to allocate lock type lock object.
2. **start_new_thread (function, args, kwargs = none)**
   It is used to spawn the new thread and execute function with given args and optional kwargs.
3. **exit()**
   It makes the thread to exit.

Lock Object Methods

1. **Locked()**
   It returns true if lock is acquired other wise false is returned.
2. **acquire (wait = none)**
   It tries to acquire lock object.
3. **release()**
   It releases the lock.

PART-B
ESSAY QUESTIONS WITH SOLUTIONS

3.1 REGULAR EXPRESSIONS

3.1.1 Introduction

Q9. Give a brief introduction on regular expression. Model Paper-I, Q6(a)

Answer :

Regular Expressions

The regular expression provided an infrastructure to perform certain functions such as extraction, advance text pattern matching and search-and-replace. Regular expressions are in the form of strings including special symbols and characters for representing pattern repetition or several characters in order to match the strings with pattern.

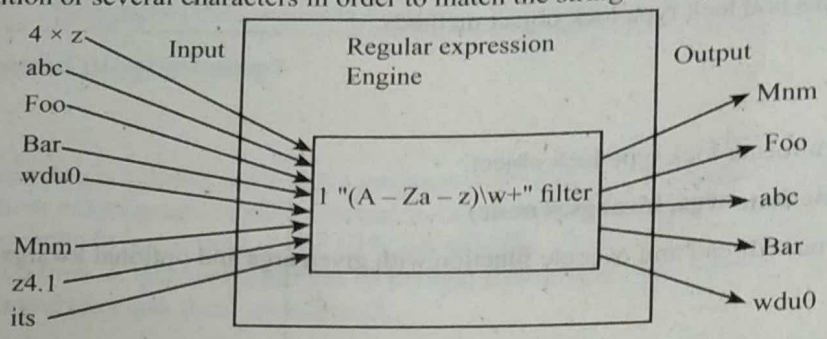

Figure: Filtering the Regular Expressions with Valid Identifiers

Python provides a standard library remodule to support regular expressions. They can contain text with upper case or lower case alphabets and numeric digits including or excluding the specialized alphabets.

3.1.2 Special Symbols and Characters

Q10. List out various special symbols and characters.

Answer : Model Paper-II, Q6(a)

Special Symbols

The most commonly used special symbols in python are as follows,

Special symbol	Description	Example
literal	It matches the literal string value literal	foo
•	It matches any character	x.x
$	It matches the end of string	/bin/*sh$
re1/re2	It matches the regular expression re1 or re2	foo/bar
*	It matches 0 or more occurrences of the previous regular expression	[A-Za-zo-9] *
^	It matches the beginning of string	^string
+	It matches 1 or more occurrences of previous regular expression	[a – z] +\.com
{N}	It matches N occurrences of previous regular expression	[0 – 9] {2}
?	It matches 0 or 1 occurrences of previous regular expression.	xyz?
(...)	It matches the enclosed regular expression and saves it as subgroup.	([0 – 9] {2}) ?, x(yz/a) pqr
{M, N}	It matches from M to N occurrences of previous regular expression.	[0 – 9] {2, 6}
[...]	It matches any single character from the character class	[aeiou]
[^...]	It does not match any of the character from character class	[^aeiou], [^A – Za – zo – 9_]
[..x – y..]	It matches a single character from x to y	[0 – 9], [A – Za – z]
(*|+|?|{})?	It does not apply "non-greedy" versions of the specified symbols	•*?[a – z]

Special Characters

The most commonly used special characters in Python are as follows,

Special Character	Description	Example
\d	It matches a decimal digit like [0 – 9]	data \d+.txt
\w	It matches alphanumeric character like [A-Za-zo-9_]	[A-Za-z_]\w+
\b	It matches as word boundary	\b the \b
\c	It matches any special character C precisely	\.,\\,*
\s	It matches white space character as [ln\t\r\v\f]	of\sthe
\A(\z)	It matches the beginning of string.	\ADear
\nn	It matches the saved subgroup nn	price:\20

Example

Matching Single Character

The dot or period(.) symbol is used to match the single character excluding the NEWLINE. It includes letter, number, whitespace, symbol, printable and non printable characters.

RE Pattern	Strings Matched
..	A pair of characters
x.o	A character between x and o such as x20, x$o etc
.end	It is,a character before string end

Matching more than one RE Pattern with alternation (1)

A pipe symbol (1) i.e., a vertical bar represents alternation operation. It is used to select one among various regular expressions separated by pipe.

RE Pattern	Strings Matched
me/home	me, home
got/get/gst	got,get,gst
x5c5\v5po	x5c5, v5po

Creating Character Class

Some times user might need to match desired set of characters. This is possible by creating character class using [ ]. The regular expression will match the characters enclosed in the brackets.

RE Pattern	Strings Matched
g [aeiou] t	gat, get, gef, got, gut
[ry] [23] [ag] [05]	A string of 4 characters: The first one is "r" or "y" then "2" or "3", followed by "a" or "g" and at last "o" or "5"

3.1.3 RES and Python

Q11. Explain in detail about RES and python.

Answer : Model Paper-III, Q6(a)

Python supports regular expressions through remodule. The core functions and methods of re module are as follows,

Remodule Function

Function	Description
Compile (pattern, flags = 0)	It compiles RE pattern with any optional flags and returns a regex object

Remodule Functions and Regex Object Methods

Function	Description
match (pattern, string, flags = 0)	It matches RE Pattern to string with optional flags, and returns match object upon success.
split(pattern, string, max = 0)	It splits the string into list with respect to RE pattern delimeter and returns set of successful matches.
search (pattern, string, flags = 0)	It searches the first occurrence of RE pattern in string with flags and returns match object upon success
sub (pattern, repl, string, max = 0)	It replaces all occurrences of RE pattern in string with repl and substitutes the occurrences if max is provided.
finditer (pattern, string [, flags])	It searches for occurrences of pattern and returns iterator
findall (pattern, string [, flags])	It matches for the occurrences of pattern in string and returns set of matches

Match Object Methods

group ()	It returns the matching subgroups in tuple
group (num = 0)	It returns the complete match

Example

Matching more than one string (1) and single character (.)

```
Python 2.7.17 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 2.7.17 (v2.7.17:c2f86d86e6, Oct 19 2019, 21:01:17) [MSC v.1500 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import re
>>> gt='got|get|git'
>>> x=re.match(gt,'got')
>>> if x is not None:x.group()

'got'
>>> x=re.search(gt,'He got it')
>>> if x is not None:x.group()

'got'
>>> p='.end'
>>> q=re.match(p,'send')
>>> if q is not None:q.group()

'send'
>>> q=re.search('.end','The bend')
>>> if q is not None:q.group()

'bend'
>>>
```

3.2 MULTITHREADED PROGRAMMING

3.2.1 Introduction

Q12. Write in short about multi threaded programming.

Answer :

Model Paper-I, Q6(b)

Multithreaded Programming

Multithreaded programming is a concept in which a program is divided into two or more subprograms that can be executed at a time. The part of program or subprogram is called as thread and each thread executes separately. It is a powerful approach that enables to write efficient programs by maximum use of CPU because the idle time of it can be reduced to minimum. It allows to perform multiple things at a time. A large program can be divided into threads, that can be executed parallely. The processor switches between the threads and executes them concurrently.

It executes every thread so fast that it appears as if they are being done simultaneously.

For example, a programming task such as MT program with queue data structure can be accomplished with few threads that have certain functions.

1. **User Request Thread:** It reads client input from I/O channel. The program creates a thread for each client whose requests are sent through the queue.
2. **Request Processor:** It retrieves the request from queue, processes them and generates the output.
3. **Reply Thread:** It extracts output that is to be sent to the user and forwards to local file system/database or sends if back either.

Use of multiple threads to perform such task can reduce complexity and enables clean and efficient implementation.

3.2.2 Threads and Processes

Q13. Discuss in brief about threads and processes.

Answer :

Thread

Model Paper-II, Q6(b)

A thread is defined as software entities that can share memory and execute concurrently. It is a basic unit of CPU utilization. It is also called as a light-weight process. Multiple threads can be created by a particular process. Each thread shares code, data segments and open files of that process with other threads. However, each of them has their separate register-set values and stacks. Consider an example of a pagemaker or any word processing application. It has two important threads executing in it, one to read the keystrokes from the keyboard and other a spelling and grammar checking thread running in background. The following figure shows, a process having two threads in it.

1. Creating a thread is ten times faster than creating a process i.e., it takes less time to create a new thread within an already existing process rather than creating it in a new process.
2. Thread termination is faster than the process termination.
3. Switching between the threads of a single process is faster as no memory mapping has to be set up and the memory and address translation caches need not be invalidated.
4. Threads are more efficient than processes as they provide feature of shared memory thereby requiring any system calls for inter-thread communication. Therefore, threads are more appropriate for tightly coupled parallel activities and similar data structures are used.
5. The processes of thread creation and destruction are cheaper as no allocation and deallocation of new address spaces or other process resources are required.

Process

Process is the fundamental concept of operating systems structure. A program under execution is referred to as a process. It can also be defined as an active entity that can be assigned to a processor for execution. A process is a dynamic object that resides in main memory. A process includes the current values of the program counter and processor's registers. Each process possesses its own virtual CPU. A process contains the following two elements,

(i) Program code and

(ii) A set of data.

Process Attributes

Each process in an operating system is characterized by a set of attributes as follows,

1. **Identifier**

   Each process is associated with a unique identifier which makes it distinguishable from all other processes.

2. **Process State**

   The state of a process describes whether a process is ready, running, blocked or suspended.

3. **Priority**

   Each process has some priority relative to other processes within the system.

4. **Program Counter**

   The address of the instruction to be executed next is stored in the program counter.

5. **Memory Pointer**

   It refers to the pointers associated with the program code and data along with the memory blocks shared with other processes.

6. **Context Data**

   This data is usually stored in the processor registers, while the process is executing.

7. **Input/Output Status Information**

   It contains all the information related to I/O processing, I/O requests, I/O devices etc.

8. **AccountingInformation**

   It contains the information about the processor time, clock time used, time limits, account numbers, etc

3.2.3 Python, Threads and the Global Interpreter Lock

Q14. Write short notes on global interpreter lock. How do threads exit?

Model Paper-III, Q8(b)

Answer :

Global Interpreter Lock (GIL)

Python virtual machine is responsible for controlling the executing of python code. Only one thread of control executes in the main loop among multiple threads. Access to this python virtual machine is controlled by Global Interpreter Lock (GIL). It allows only one thread to execute at a time in CPU. Working of python virtual machine in MT environment is as follows,

1. Initially set the GIL
2. Switch the thread to run
3. Execute either (i) or (ii)
   (i) For fixed number of bytecode instructions.
   (ii) If thread generates control voluntarily.
4. Return the thread back to sleep.
5. Unlock GIL
6. Repeat the complete process.

The GIL gets locked and no longer works if any external code such as built-in-functions of C/C++ are called. But GIL can be unlocked by certain extension programmers. For example, in I/O based routines, the GIL is released before and I/O call is made so that other threads can continue while I/O is performed.

Exiting Threads

A thread can exit after it completes the execution of the function for which it is created. It can exit by calling tHRead.exit or sys.exit () functions. It can even raise a system exit exception. The thread module is mostly not used because when the main thread exits, all the remaining threads die without any cleanup. Whereas the threading module makes the complete process to be alive until all the threads exit.

Q15. How thread are accessed from python? Explain how life is without threads.

Answer :

Model Paper-II, Q7

Accessing Threading from Python

The concept of multithreaded programming is supported by Python based on the operating system on which it is run. The Unix based platforms such as Linux, Solaris, Macos X, BSD and win32 support it. Python makes use of pthreads or POSIX-compliant threads. The threads are activated by default in python while installation.

Availability of threads can be checked for interpreter by the below statement.

```
>>> import thread
```

Here, the thread module is tried to be imported from the interactive interpreter.

If threads are available, errors will be generated. Otherwise module import fails.

```
>>> import thread
    Traceback (innermost last):
      File "<stdin>", line 1, in ?
    Importerror: No module named thread
```

In case of above errors, the python interpreter must be recompiled to access the threads. To do this, the configure script must be invoked with "_ _ with _thread" option.

Life without Threads

Consider and example of time.sleep () to depict the working of threads. This function accepts floating point as argument and then "sleeps" for specific number of seconds by halting the executing temporarily from time import sleep, ctime

```
t5.py - C:\Users\SIA\AppData\Local\Programs\Python\Python37\t5.py (3.7.4)
File Edit Format Run Options Window Help
from time import sleep, ctime
def loopA():
    print('Begin loop A at:',ctime())
    sleep(2)
    print('loop A done at:',ctime())
def loopB():
    print('Begin loop B at:',ctime())
    sleep(4)
    print('loop B done at:',ctime())
def main():
    print('Begins at:', ctime())
    loopA()
    loopB()
    print('all DONE at:', ctime())

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

OUTPUT

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:/Users/SIA/AppData/Local/Programs/Python/Python37/t5.py ======
Begins at: Tue Oct 22 11:43:05 2019
Begin loop A at: Tue Oct 22 11:43:05 2019
loop A done at: Tue Oct 22 11:43:07 2019
Begin loop B at: Tue Oct 22 11:43:07 2019
loop B done at: Tue Oct 22 11:43:11 2019
all DONE at: Tue Oct 22 11:43:11 2019
>>>
```

In the above example two loops are used and they are run sequentially. Here one loop must complete for the other loop to start. When one thread sleeps when the other is running or vice versa. Since both of them are two separate functions, they can be run parallely to cut down the overall running time.

3.2.4 Thread Module, Threading Module, Related Modules

Q16. Explain about tHRead module.

Model Paper-I, Q

Answer :

tHRead Module

The tHRead module is responsible to spawn the threads. It provides a synchronization data structure called lock objec Example of lock object are mutual exclusion lock, simple lock, primitive lock, mutex and binary semaphore. Some of t commonly used thread functions and lock type lock object methods.

tHRead Module Functions

1. **allocate_lock ()**
   This method is used to allocate lock type lock object.
2. **start_new_thread (function, args, kwargs = none)**
   It is used to spawn the new thread and execute function with given args and optional kwargs.
3. **exti ()**
   It makes the thread to exit.

Lock Object Methods

1. **Locked ()**
   It returns true if lock is acquired other wise false is returned.
2. **acquire (wait = none)**
   It tries to acquire lock object.
3. **release ()**
   It releases the lock.

Example

```
3a.py - C:\Python27\3a.py (2.7.17)
File Edit Format Run Options Window Help
import thread
from time import sleep,ctime
def loopA():
    print'start the loop A at:', ctime()
    sleep(5)
    print'loop A ends at:', ctime()
def loopB():
    print'start the loop B at:', ctime()
    sleep(3)
    print'loop B ends at:', ctime()
def main():
    print'Beginning at:', ctime()
    thread.start_new_thread(loopA, ())
    thread.start_new_thread(loopB, ())
    sleep(4)
    print'completed at:', ctime()
if __name__ == '__main__':
    main()
```

OUTPUT

```
Python 2.7.17 Shell
File Edit Shell Debug Options Window Help
Python 2.7.17 (v2.7.17:c2f86d86e6, Oct 19 2019, 21:01:17) [MSC v.1500 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Python27/3a.py ==========
Beginning at: Tue Oct 22 15:56:12 2019
start the loop A at:start the loop B at: Tue Oct 22 15:56:12 2019
loop B ends at: Tue Oct 22 15:56:15 2019
completed at: Tue Oct 22 15:56:16 2019
>>>
```

Q17. Write about tHReading module.

Answer : Model Paper-III, Q7

tHReading Module

The tHReading module consists of THRead class and a set of synchronization mechanisms. Various objects of tHReading module are as follows,

Object	Description
Thread	It indicates a single thread of execution
Timer	It is similar to thread but it waits for the specified time before running
Semaphore	It offers a structure like "waiting area" for the threads that are waiting on lock
Lock	It is a primitive lock object.
RLock	It is re-entrant lock object that allows a single thread to acquire an already held lock
setDaemon (daemonic)	It sets the daemon flag of thread according to the boolean daemonic

Threads can be created in three ways, they are as follows,

By passing thread instance in function.

By passing tHRead instance in callable class instance.

By deriving the tHRead and creating a subclass instance.

In this method a THRead instance is created by passing it in the function. This function will be executed when thread begins its execution.

Example:

t1.py - C:\Users\SIA\AppData\Local\Programs\Python\Python37\t1.py (3.7.4)

File Edit Format Run Options Window Help

```
import threading
from time import sleep, ctime

loops = [4,1]

def loop(nloop, nsec):
    print('start loop', nloop, 'at:', ctime())
    sleep(nsec)
    print('loop', nloop, 'done at:', ctime())

def main():
    print('started at:', ctime())
    threads = []
    nloops = range(len(loops))

    for a in nloops:
        t1 = threading.Thread(target=loop,args=(a, loops[a]))
        threads.append(t1)

    for a in nloops: # start threads
        threads[a].start()

    for a in nloops: # wait for all
        threads[a].join() # threads to finish
    print('Completed at', ctime())

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

OUTPUT

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\SIA\AppData\Local\Programs\Python\Python37\t1.py =====
started at: Thu Oct 17 14:30:31 2019
start loopstart loop  01  at: Thu Oct 17 14:30:31 2019
at: Thu Oct 17 14:30:31 2019
loop 1 done at: Thu Oct 17 14:30:32 2019
loop 0 done at: Thu Oct 17 14:30:35 2019
Completed at Thu Oct 17 14:30:35 2019
>>>
```

2. **By Passing THRead Instance to Callable Class**

In this approach, a thread instance is passed to the callable class. It is flexible and object oriented type approach.

Bounded semaphore	It is similar to semaphore but it does not exceed the initial value.
Condition	It makes a thread to wait until another thread satisfies a condition

Thread Class

The thread class is a primary object in threading module. Various functions of this class are as follows,

Method	Description
start ()	It begins the thread
getName ()	It returns the thread name
run ()	It defines the thread functionality
setName (name)	It sets the name for thread
isAlive ()	It returns a boolean flag that indicates whether the thread is running or not
join (timeout = none)	It suspends the thread until a timeout is issued.

Example:

t2.py - C:\Users\SIA\AppData\Local\Programs\Python\Python37\t2.py (3.7.4)

File Edit Format Run Options Window Help

```
import threading
from time import sleep, ctime

loops = [4,2]

class ThreadFunc(object):

    def __init__(self, fl, pl, name=''):
        self.name = name
        self.fun1 = fl
        self.para1 = pl

    def __call__(self):
        self.fun1(*self.para1)
def loop(nloop, nsec):
    print('start loop', nloop, 'at:', ctime())
    sleep(nsec)
    print('loop', nloop, 'done at:', ctime())

def main():
    print('started at:', ctime())
    threads = []
    nloops = range(len(loops))

    for a in nloops: # create all threads
        t1 = threading.Thread(target=ThreadFunc(loop,(a, loops[a]),loop.__name__
        threads.append(t1)

    for a in nloops: # start all threads
        threads[a].start()

    for a in nloops: # wait for completion
        threads[a].join()

    print('Completed at:', ctime())

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

Ln: 1 Col: 0

OUTPUT

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\SIA\AppData\Local\Programs\Python\Python37\t2.py =====
started at: Thu Oct 17 14:33:01 2019
start loop start loop0  1at:  at:Thu Oct 17 14:33:01 2019
Thu Oct 17 14:33:01 2019
loop 1 done at: Thu Oct 17 14:33:03 2019
loop 0 done at: Thu Oct 17 14:33:05 2019
Completed at: Thu Oct 17 14:33:05 2019
>>>
```

WARNING:

3. **By deriving the THRead and creating a subclass instance**

In this approach the THRead () is derived and a subclass instance is created.

Example:

```
t3.py - C:\Users\SIA\AppData\Local\Programs\Python\Python37\t3.py (3.7.4)
File Edit Format Run Options Window Help
import threading
from time import sleep, ctime
loops = (4, 2)

class MyThread(threading.Thread):
    def __init__(self, funi, parai, name=''):
        threading.Thread.__init__(self)
        self.name = name
        self.funi = funi
        self.parai = parai

    def run(self):
        self.funi(*self.parai)

def loop(nloop, nsec):
    print('start loop', nloop,'at: ', ctime())
    sleep(nsec)
    print('loop', nloop, 'done at:', ctime())
def main():
    print('started at:', ctime())
    threads = []
    nloops = range(len(loops))

    for a in nloops:
        t1 = MyThread(loop, (a, loops[a]),loop.__name__)
        threads.append(t1)

    for a in nloops:
        threads[a].start()

    for a in nloops:
        threads[a].join()
    print('Completed at:', ctime())

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

OUTPUT

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\SIA\AppData\Local\Programs\Python\Python37\t3.py =====
started at: Thu Oct 17 14:38:45 2019
start loopstart loop 1 at:   Thu Oct 17 14:38:45 2019
 0 at:   Thu Oct 17 14:38:45 2019
loop 1 done at: Thu Oct 17 14:38:47 2019
loop 0 done at: Thu Oct 17 14:38:49 2019
Completed at: Thu Oct 17 14:38:49 2019
>>>
```

Q18. Explain producer consumer problem and queue module.

Answer :

The producer is responsible for creating goods and storing in a data structure called queue. The time in between producing the goods as well as consuming these goods is said to be non deterministic. The queue module will provide an inter thread communication mechanism that enables the threads to share data among them.

Example:

```
from random import randint
from time import sleep
from Queue import Queue
from threading import Thread
def writeQ(queue):
    print('producing object for Q...',
        queue.put('xxx', 1))
    print("size now", queue.qsize())
def readQ(queue):
```

```
            val = queue.get(1)
            print('consumed object from Q... size now', \
                queue.qsize())
        def writer(queue, loops):
            for i in range(loops):
                writeQ(queue)
                sleep(randint(1, 3))
        def reader(queue, loops):
            for i in range(loops):
                readQ(queue)
                sleep(randint(2, 5))
        funcs = [writer, reader]
        nfuncs = range(len(funcs))
        def main():
            nloops = randint(2, 5)
            q = Queue()
            threads = []
            for i in nfuncs:
                t = MyThread(funcs[i], (q, nloops),
                funcs[i].__name__)
                threads.append(t)
            for i in nfuncs:
                threads[i].start()
            for i in nfuncs:
                threads[i].join()
            print('all DONE')
    if __name__ == '__main__':
        main()
```

Q19. Write about related modules.

Answer :

The modulus that are used while programming multithreaded applications are as follows,

Module	Description
tHRead	It is a basic and low-level thread module
Queue	It is a synchronized FIFO queue that is used for multiple threads.
mutex	It is a module for mutual exclusion objects.
threading	It is a higher-level threading and synchronization objects.
Socket server	It is a module for TCP and UDP managers with some threading control.

For example, the regular expression and the strings matching them are as follows,

RE pattern	String Matched
Python	Python
xyz789	xyz789
Me	Me

```
        val = queue.get(1)
        print('consumed object from Q... size now', \
            queue.qsize())
    def writer(queue, loops):
        for i in range(loops):
            writeQ(queue)
            sleep(randint(1, 3))
    def reader(queue, loops):
        for i in range(loops):
            readQ(queue)
            sleep(randint(2, 5))
    funcs = [writer, reader]
    nfuncs = range(len(funcs))
    def main():
        nloops = randint(2, 5)
        q = Queue()
        threads = []
        for i in nfuncs:
            t = MyThread(funcs[i], (q, nloops),
            funcs[i].__name__)
            threads.append(t)
        for i in nfuncs:
            threads[i].start()
        for i in nfuncs:
            threads[i].join()
        print('all DONE')
    if __name__ == '__main__':
        main()
```

Q19. Write about related modules.

Answer :

The modulus that are used while programming multithreaded applications are as follows,

Module	Description
tHRead	It is a basic and low-level thread module
Queue	It is a synchronized FIFO queue that is used for multiple threads.
mutex	It is a module for mutual exclusion objects.
threading	It is a higher-level threading and synchronization objects.
Socket server	It is a module for TCP and UDP managers with some threading control.

For example, the regular expression and the strings matching them are as follows,

RE pattern	String Matched
Python	Python
xyz789	xyz789
Me	Me