PYTHON PROGRAMMING

Presented By
POTU NARAYANA
Research Scholar
Osmania University
Dept. of CSE
B.Tech, M.Tech, M.A, PGDHR, MIAENG, MIS,(Ph.D)
Email : pnarayana@osmania.ac.in
Contact: +91 9704868721

REGULAR EXPRESSION

- A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.
- The module **re** provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.
- We would cover two important functions, which would be used to handle regular expressions. But a small thing first: There are various characters, which would have special meaning when they are used in regular expression. To avoid any confusion while dealing with regular expressions, we would use Raw Strings as **r'expression'**.

Multi Threaded Programming

- A **process** is an instance of a computer program that is being executed. Any process has 3 basic components:

  An executable program.

  The associated data needed by the program (variables, work space, buffers, etc.)

  The execution context of the program (State of process)
- A **thread** is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System).
- In simple words, a **thread** is a sequence of such instructions within a program that can be executed independently of other code. For simplicity, you can assume that a thread is simply a subset of a process!
- A thread contains all this information in a **Thread Control Block (TCB)**:
- **Thread Identifier:** Unique id (TID) is assigned to every new thread
- **Stack pointer:** Points to thread's stack in the process. Stack contains the local variables under thread's scope.
- **Program counter:** a register which stores the address of the instruction currently being executed by thread.
- **Thread state:** can be running, ready, waiting, start or done.
- **Thread's register set:** registers assigned to thread for computations.
- **Parent process Pointer:** A pointer to the Process control block (PCB) of the process that the thread lives on.

Multithreading

- Multiple threads can exist within one process where:
- Each thread contains its own **register set** and **local variables (stored in stack)**.
- All thread of a process share **global variables (stored in heap)** and the **program code**
- **Multithreading** is defined as the ability of a processor to execute multiple threads concurrently.
- In Python, the **threading** module provides a very simple and intuitive API for spawning multiple threads in a program.

```
# Python program to illustrate the concept
# of threading
# importing the threading module
import threading

def print_cube(num):
        """
        function to print cube of given num
        """
        print("Cube: {}".format(num * num * num))

def print_square(num):
        """
        function to print square of given num
        """
        print("Square: {}".format(num * num))

if __name__ == "__main__":
        # creating thread
        t1 = threading.Thread(target=print_square, args=(10,))
        t2 = threading.Thread(target=print_cube, args=(10,))

        # starting thread 1
        t1.start()
        # starting thread 2
        t2.start()

        # wait until thread 1 is completely executed
        t1.join()
        # wait until thread 2 is completely executed
        t2.join()

        # both threads completely executed
        print("Done!")
```

Global Interpreter Lock

- Python Global Interpreter Lock (GIL) is a type of process lock which is used by python whenever it deals with processes. Generally, Python only uses only one thread to execute the set of written statements. This means that in python only one thread will be executed at a time. The performance of the single-threaded process and the multi-threaded process will be the same in python and this is because of GIL in python. We can not achieve multithreading in python because we have global interpreter lock which restricts the threads and works as a single thread.
- **What problem did the GIL solve for Python :**
- Python has something that no other language has that is a reference counter. With the help of the reference counter, we can count the total number of references that are made internally in python to assign a value to a data object. Due to this counter, we can count the references and when this count reaches to zero the variable or data object will be released automatically.

Impact on multi-threaded Python programs :

When a user writes Python programs or any computer programs then there's a difference between those that are CPU-bound in their performance and those that are I/O-bound. CPU push the program to its limits by performing many operations simultaneously whereas I/O program had to spend time waiting for Input/Output.

Code 1: CPU bound program that perform simple countdown

```
# Python program showing
# CPU bound program
import time
from threading import Thread
COUNT = 50000000
def countdown(n):
        while n>0:
                n -= 1
start = time.time()
countdown(COUNT)
end = time.time()
print('Time taken in seconds -', end - start)
```

Output:

Time taken in seconds - 2.5236213207244873

Two threads running parallel

```
# Python program showing
# two threads running parallel
import time
from threading import Thread
COUNT = 50000000
def countdown(n):
        while n>0:
                n -= 1
t1 = Thread(target = countdown, args =(COUNT//2, ))
t2 = Thread(target = countdown, args =(COUNT//2, ))
start = time.time()
t1.start()
t2.start()
t1.join()
t2.join()
end = time.time()
print('Time taken in seconds -', end - start)
```

Outpit

Time taken in seconds - 2.183610439300537

- # The thread Module
- Let’s take a look at what the thread module has to offer. In addition to being able to spawn threads, the thread module also provides a basic synchronization data structure called a *lock object* (a.k.a. *primitive lock*, *simple lock*, *mutual exclusion lock*, *mutex*, and *binary semaphore*). As we mentioned earlier, such synchronization primitives go hand in hand with thread management.
- **thread *Module Functions***
- start_new_thread(*function, args, kwargs*=None) : Spawns a new thread and executes function with the given args and optional *kwargs*
- allocate_lock() : Allocates LockType lock object
- exit() : Instructs a thread to exit
- **LockType Lock *Object Methods***
- acquire(*wait*=None) : Attempts to acquire lock object
- locked() : Returns True if lock acquired, False otherwise
- release() : Releases lock

Threading Module

- We will now introduce the higher-level tHReading module, which gives you not only a THRead class but also a wide variety of synchronization mechanisms to use to your heart's content

threading Module Objects

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 or of some data value EventGeneral version of condition variables whereby any number of threads are waiting for some event to occur and all will awaken when the event happens
• 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 period	Similar to Thread except that it waits for an allotted of time before running

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
- **Thread Object Methods**

Method	***Description***
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; blocks unless timeout (in seconds)* is given
getName()	Return name of thread
setName*(name)*	*Set name of thread*
isAlive()	Boolean flag indicating whether thread is still running
isDaemon()	Return daemon flag of thread
setDaemon*(daemonic)*	*Set the daemon flag of thread as per the Boolean daemonic (must be called* before thread start()ed)

Other Threading Module Functions

- In addition to the various synchronization and threading objects, the Threading module also has some supporting functions.
- **threading Module Functions**

Function	***Description***
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*

Related Modules

- The table below lists some of the modules you may use when programming multithreaded applications.
- **Threading-Related Standard Library Modules**

Module	***Description***
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

References

- Core Python Programming , Wesley J.chun, Second Edition, Pearson.
- Python Programming Using Problem Solving Approach, Reema Thareja.
- Core Python Programming, Dr. R.Nageswara Rao
- Core Python Material by Durga Software Solutions