CET - 02

- customized exceptions :-

→ Also known as user - defined exceptions, it allows programmers to define their own type of exceptions.

→ In python, users can define custom exceptions by creating a new class.

→ This new exception class should be derieved from the built in exception class.

→ whenever we are developing a large python program, it is a good practice to place all user - defined exceptions that could be raise in program.

→ User - defined exception class can implement everything a normal class can do, but we generally make them simple and consise.

eg:
```
class underage(Exception):
        Pass
def checkage():
    try:
        age = 17
        if age < 18:
            raise underage
    except underge:
        Print("Age less than 18")
checkage()
```

- Thread :-

→ A thread is a seqvence of such instructions within a program that can be executed independently of other program.

→ ~~Threads share the memory space and resources.~~

→ Thread states are running, ready, waiting, start or done.

→ For computations, registers are assigned to thread.

→ Uniqve id is assigned to every new thread created.

→ A register which stores the address of the instruction currently being executed by thread.

- Multithreading :-

→ It is a well-known technique in which multiple threads in a process share data space with the main thread, which makes info sharing and communication within threads easy.

→ Threads are lighter than processes.

→ The purpose of multithreading is to run multiple tasks at same time.

→ Multithreading is very useful for saving time and improving performance.

→ Multithreading can be used only when the dependency between individual threads does not exist.

→ Applications :-
- web servers
- computer games
- text editors
- web browsers

- How to create threads in Python?

→ There are two built-in modules to create and mange threads :-

1) The thread module
2) The threading module

1) Thread module :-

→ To use this module you have to import "_thread" module.

→ To create thread we have to use start_new_thread(), it takes two arguements, first :- function name and second :- arguements of function.

→ NOTE :- "_thread" lacks support for advanced features compared to "threading" module.

eg :- import time
" _thread
def thread_test(name, wait):
time.sleep(wait)
print("Running", name)

_thread.start_new_thread(thread_test, ("one", 2)
" " " " , ("two", 4)

2] Threading module

→ Threading module provides rich features and better support for threads than -thread module.

→ It also provides better object-oriented approach.

→ Threading module contains functions, classes, objects and exceptions to manage multithreading.

→ some basic functions are:-
activeCount() , currentThread() , enumerate() , isAlive().

→ Some basic Thread class methods are:-
- start() :- starts the activity of thread.
- ~~run()~~
- Join() :- It stops other executions until the ~~used~~ thread on which join() called gets terminated.

→ The 'threading.thred' class is used to create new threads. it is created by creating object of thread.

syntax

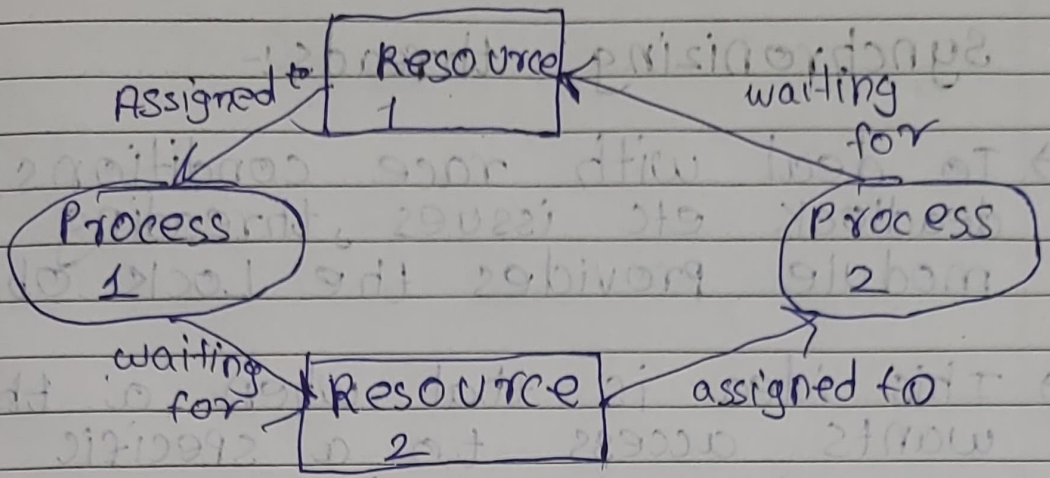
```
t1= threading.Thread(target=func)
```

eg:-
```
import time
    threading

def sq(n, wait)
    a = n * n
    time.sleep(wait)
    print(a)

t1 = threading.Thread(target=sq(2,3))
t2 =  "        "      "    sq(4,5))

t1.start()
t2.start()

t1.join()
t2.join()
```

- Deadlock :-

→ It is a situation where set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

→ For eg, Process 1 is holding resource1 and waiting for resource 2 which is acquired by process 2 & process 2 is waiting for resource 1.



→ Deadlocks are the most feared issue that developers face when writing multithreaded applications in Python.

- Race condition :-

→ A race condition is an unwanted state of a program which occurs when a system performs two or more operations.

→ ~~threads can overlap, and value~~

→ A race condition occurs when two threads try to access a shared variable.

→ Then both threads try to ~~change~~ modify the shared resource and they race to see which thread modifies the the thread last.

- Synchronizing threads :-

→ To deal with race conditions, deadlock etc issues, threading module provides the Lock object.

→ This idea is that when a thread wants access to a specific resource, it acquires a lock for that resource. Once a thread locks a resource, no other thread can access it until lock is released.

→ A lock can be in one of 2 states locked or unlocked.

- lock() :- If lock-state is unlocked, calling the acquire() will change it to locked state.

- release() :- It is used to change the state to unlocked.