MULTI-THREADING IN PYTHON

Multithreading is the ability of a program or an operating system to enable more than one task at a time or without requiring multiple copies of the program running on the computer.

Multithreading can also handle various requests from the same user.

multithreading is a way of achieving multitasking.

In multithreading, the concept of **threads** is used.

Threads are the smallest units of execution within a program.

Advantages of threads:

- Improved performance
- Responsiveness
- **Simplified programming**

COMMON FUNCTIONS OF THREADING

```
# List of Functions are there in Threading

#  threading.Thread() is used to create threads
# start() is used to start the thread
# join() is used to wait for the thread to complete
# isAlive() is used to check if the thread is alive
# enumerate() is used to get all the threads
# current_thread() is used to get the current thread
```

Example of threading

```
import threading

def fucn1():
```

```
    for i in range(50):
        print("i------", i)


def fucn2(num):
    for j in range(num):
        print("j------", j)

# creating first thread
t1 = threading.Thread(target=fucn1)
# creating second thread
t2 = threading.Thread(target=fucn2, args=(50,))

# starting first thread
t1.start()
# starting second thread
t2.start()

# wait for first thread to complete
t1.join()
# wait for second thread to complete
t2.join()

print("done")
```

is_alive() function

```
if t1.is_alive():
    print("t1 is alive")
if t2.is_alive():
    print("t2 is alive")

# wait for first thread to complete
t1.join()
# wait for second thread to complete
t2.join()

if not t1.is_alive():
    print("t1 is not alive")
if not t2.is_alive():
    print("t2 is not alive")
```

```
print("done")
```

enumerate() function

```
# getting active threads
active_threads = threading.enumerate()
for i in active_threads:
    print(i)
```

Current_thread() function

```
import threading

def print_thread_name():
    current_thread = threading.current_thread()
    print("Current thread name:", current_thread.name)

# Create a thread object
thread = threading.Thread(target=print_thread_name)

# Start the thread
thread.start()
```

Event Create

Using this event option we can control the thread execution.
here in this below example, we are setting the second thread to execute first and then the first thread will execute.

```
import threading
import time

def fucn1(event):
    event.wait()  # Wait for the event to be set
    for i in range(5):
```

```
        time.sleep(1)
        print("i------", i)


def fucn2(event, num):
    for j in range(num):
        time.sleep(1)
        print("j------", j)
    event.set()  # Set the event to allow the first thread to proceed

# Create an event object
event = threading.Event()

# Create the first thread
t1 = threading.Thread(target=fucn1, args=(event,))
# Create the second thread
t2 = threading.Thread(target=fucn2, args=(event, 5))

# Start the second thread first
t2.start()
# Start the first thread
t1.start()

# Wait for the first thread to complete
t1.join()
# Wait for the second thread to complete
t2.join()

print("done")
```