1. **Calculate the sum of squares of numbers from 1 to 100 using four threads. Divide the range equally among the threads, and each thread calculates the sum of squares for its range. Finally, combine the results to get the total sum of squares.**

```
import threading
total_sum = 0
lock = threading.Lock()

def calc_square_sum(start, end):
    global total_sum
    local_sum = sum(i * i for i in range(start, end + 1))
    with lock:
        total_sum += local_sum

threads = []
ranges = [(1, 25), (26, 50), (51, 75), (76, 100)]

for r in ranges:
    t = threading.Thread(target=calc_square_sum, args=r)
    threads.append(t)
    t.start()

for t in threads:
    t.join()
print("Total sum of squares from 1 to 100:", total_sum)
```

2. **Create two threads, one printing even numbers and the other printing odd numbersfrom 1 to 10. Ensure proper synchronization to alternate between even and odd numbers.**

```
import threading
condition = threading.Condition()
turn = "even"

def print_even():
    global turn
    for i in range(2, 11, 2):
        with condition:
            while turn != "even":
                condition.wait()
            print("Even:", i)
            turn = "odd"
            condition.notify()
```

```
def print_odd():
    global turn
    for i in range(1, 10, 2):
        with condition:
            while turn != "odd":
                condition.wait()
            print("Odd:", i)
            turn = "even"
            condition.notify()

t1 = threading.Thread(target=print_even)
t2 = threading.Thread(target=print_odd)

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

3. **Implement two threads to print lowercase and uppercase alphabets concurrently from 'a' to 'z' and 'A' to 'Z'.**

```
import threading
import time

def print_lowercase():
    for c in range(ord('a'), ord('z') + 1):
        print("Lower:", chr(c))
        time.sleep(0.05)

def print_uppercase():
    for c in range(ord('A'), ord('Z') + 1):
        print("Upper:", chr(c))
        time.sleep(0.05)

t1 = threading.Thread(target=print_lowercase)
t2 = threading.Thread(target=print_uppercase)

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

4. **Implement a producer-consumer problem with a limited buffer of size 5. Create two producer threads and two consumer threads. Producers produce items, and consumers consume them. Ensure proper synchronization to avoid buffer overflows or underflows.**

```
import threading
import time
import random

buffer = []
BUFFER_SIZE = 5
lock = threading.Lock()
empty = threading.Semaphore(BUFFER_SIZE)
full = threading.Semaphore(0)

def producer(name):
    for _ in range(5):
        item = random.randint(1, 100)
        empty.acquire()
        with lock:
            buffer.append(item)
            print(f"{name} produced: {item}")
        full.release()
        time.sleep(random.random())

def consumer(name):
    for _ in range(5):
        full.acquire()
        with lock:
            item = buffer.pop(0)
            print(f"{name} consumed: {item}")
        empty.release()
        time.sleep(random.random())

p1 = threading.Thread(target=producer, args=("Producer1",))
p2 = threading.Thread(target=producer, args=("Producer2",))
c1 = threading.Thread(target=consumer, args=("Consumer1",))
c2 = threading.Thread(target=consumer, args=("Consumer2",))
```

```
p1.start()
p2.start()
c1.start()
c2.start()

p1.join()
p2.join()
c1.join()
c2.join()
```