Codio Activity: The Producer-Consumer Mechanism

```
# code source: https://techmonger.github.io/55/producer-consumer-python/

from threading import Thread
from queue import Queue

q = Queue()
final_results = []

def producer():
    for i in range(100):
        q.put(i)


def consumer():
    while True:
        number = q.get()
        result = (number, number**2)
        final_results.append(result)
        q.task_done()


for i in range(5):
    t = Thread(target=consumer)
    t.daemon = True
    t.start()

producer()

q.join()

print (final_results)
```

Run the code

Output:

```
codio@precisepoint-delphinova:~/workspace$ python3 producer-consumer.py
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100), (11, 1
21), (12, 144), (13, 169), (14, 196), (15, 225), (16, 256), (17, 289), (18, 324), (19, 361), (20, 400),
(21, 441), (22, 484), (23, 529), (24, 576), (25, 625), (26, 676), (27, 729), (28, 784), (29, 841), (30,
900), (31, 961), (32, 1024), (33, 1089), (34, 1156), (35, 1225), (36, 1296), (37, 1369), (38, 1444), (39
, 1521), (40, 1600), (41, 1681), (42, 1764), (43, 1849), (44, 1936), (45, 2025), (46, 2116), (47, 2209),
 (48, 2304), (49, 2401), (50, 2500), (51, 2601), (52, 2704), (53, 2809), (54, 2916), (55, 3025), (56, 31
36), (57, 3249), (58, 3364), (59, 3481), (60, 3600), (61, 3721), (62, 3844), (63, 3969), (64, 4096), (65
, 4225), (66, 4356), (67, 4489), (68, 4624), (69, 4761), (70, 4900), (71, 5041), (72, 5184), (73, 5329),
 (74, 5476), (75, 5625), (76, 5776), (77, 5929), (78, 6084), (79, 6241), (80, 6400), (81, 6561), (82, 67
24), (83, 6889), (84, 7056), (85, 7225), (86, 7396), (87, 7569), (88, 7744), (89, 7921), (90, 8100), (91
, 8281), (92, 8464), (93, 8649), (94, 8836), (95, 9025), (96, 9216), (97, 9409), (98, 9604), (99, 9801)]
codio@precisepoint-delphinova:~/workspace$
```

Questions

1. How is the queue data structure used to achieve the purpose of the code?
Like stack, queue is a linear data structure that stores items in First In First Out (FIFO) manner. In this case the queue stores the elements created by producer.

2. What is the purpose of q.put(I)?
This adds an element to the queue

3. What is achieved by q.get()?
This gets the next element from the head of the queue

4. What functionality is provided by q.join()?
q.join () actually means to wait until the queue is empty before performing other operations.

5. Extend this producer-consumer code to make the producer-consumer scenario available in a secure way. What technique(s) would be appropriate to apply?