## 1) Advanced Secret Message Decoder

### Background:

You've stumbled upon a secret message that's crucial for completing your mission. Unfortunately, the message has been encoded using a custom encryption method. Luckily, you've managed to obtain the encoding algorithm, and your task is to reverse-engineer it to decode the message.

### Encoding Algorithm:

The message has been encoded following these steps:

1. Each character in the original message (including letters, spaces, and common special characters such as !, ., ,, ?, ;, :) has been shifted by a certain number of places within a custom character set. The character set includes lowercase letters, space, and the mentioned special characters, in that order. The shift wraps around; for example, after 'z', it shifts to space, then to '!', and so on.
2. After the shift, a sequence of numbers has been appended to each character, indicating the original position of each character in the message, starting from 1. If a character repeats, its subsequent occurrences are numbered sequentially.
3. The encoded message concatenates the shifted characters followed by their respective position numbers.

### Example:

Character set: `"abcdefghijklmnopqrstuvwxyz !.,?;:"`
Original message: `"hello, world!"`
Shift: `3`
Encoded message: `"k1h2o3o4r5:6,7z8r9u10o11g12?13"`

### Your Task:

Write a program that decodes the encoded message back to its original form given the encoded message and the shift amount. Assume the message can include lowercase letters, spaces, and the special characters !, ., ,, ?, ;, :.

*Input:*

- `encoded_msg`: A string representing the encoded message.
- `shift`: An integer representing the shift amount used in the encoding.

*Output:*

- A string representing the original, decoded message.

Function Signature:

```
def decode_advanced_message(encoded_msg: str, shift: int) -> str:
    # Your code here
```

Example Usage:

```
encoded_msg = "k1h2o3o4r5:6,7z8r9u10o11g12?13"
shift = 3
print(decode_advanced_message(encoded_msg, shift))
# Output: "hello, world!"
```

Notes:

- The input will always be valid, and the shift will be a positive integer.
- The character set to consider for shifting is "abcdefghijklmnopqrstuvwxyz !.,?;:".
- Address edge cases, such as a large shift that cycles through the character set multiple times.

Encoding the Example Message:

The encoded message for `"You moved to the next round!"` with a shift of `3` is
“`!1r2x3,4p5r6y7h8g9,10w11r12,13w14k15h16,17q18h19 20w21,22u23r24x25q26g27?28`”

## 2) Implement a Transaction Monitoring System for Anti-Money Laundering (AML)

As part of a bank's efforts to combat financial fraud and adhere to anti-money laundering regulations, you are tasked with developing a Transaction Monitoring System (TMS). The system must analyse bank transactions and flag any account that has transactions exceeding a certain threshold over a specified number of consecutive days.

Scenario and Requirements:

- Every account transaction contains the account ID, transaction amount, and a timestamp.
- An account is flagged as suspicious if within the last `X` consecutive days, `2 * the median of all the transactions that happened` in the account exceeds `threshold`
- If an account does not have X days worth of transactions, it cannot be flagged.
- You need to implement a `TransactionMonitor` class with methods to process incoming transaction data and flag accounts.

```
class TransactionMonitor:
    def __init__(self, threshold, days):
        """
        Initializes the transaction monitor with a transaction threshold
        and the number of consecutive days to monitor.
        """

    def process_transaction(self, account_id, amount, timestamp):
        """
        Processes an incoming transaction and checks for suspicious activity.
        - account_id: A unique identifier for the bank account.
        - amount: The amount of the transaction.
        - timestamp: The time at which the transaction occurred.
        If the alarm critria is met, the account is flagged
        """

    def flag_account(self, account_id):
        """
        Flags the account as suspicious.
        This function should print an alert message with the account ID.
        """
```

Example Usage:

This triggers an alarm because the median of the transactions from 2024-04-21 till 2024-04-23 have a median of 8000, and 2 * 8000 >= 10000

```
tms = TransactionMonitor(threshold=10000, days=3)

# since we defined days=3, we need transaction data for 3 days before we can
trigger alarms
tms.process_transaction('acc123', 5000, '2024-04-21T10:00:00Z')
tms.process_transaction('acc123', 12000, '2024-04-21T22:50:00Z')
tms.process_transaction('acc123', 15500, '2024-04-22T18:00:00Z')
tms.process_transaction('acc123', 2250, '2024-04-22T20:00:00Z')
tms.process_transaction('acc123', 7500, '2024-04-23T16:00:00Z')
tms.process_transaction('acc123', 8500, '2024-04-23T18:00:00Z')
tms.process_transaction('acc123', 500, '2024-04-24T18:00:00Z') # at this point, we
try to process whether we should trigger the alarm.
```

This does not trigger an alarm because the median of the transactions from 2024-04-21 till 2024-04-22 have a median of 3250, and 2 * 3250 < 8000

```
tms = TransactionMonitor(threshold=8000, days=2)

# since we defined days=2, we need transaction data for 2 days before we can
trigger alarms
tms.process_transaction('acc123', 1000, '2024-04-21T10:00:00Z')
```

```
tms.process_transaction('acc123', 2000, '2024-04-21T18:50:00Z')
tms.process_transaction('acc123', 3000, '2024-04-21T20:00:00Z')
tms.process_transaction('acc123', 4000, '2024-04-21T22:00:00Z')
tms.process_transaction('acc123', 4500, '2024-04-22T16:00:00Z')
tms.process_transaction('acc123', 3500, '2024-04-22T18:00:00Z')
tms.process_transaction('acc123', 2000, '2024-04-24T18:00:00Z') # at this point, we try to process whether we should trigger the alarm.
```

## 3) LRU Cache

Design a data structure that follows the constraints of a Least Recently Used (LRU) cache.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialise the LRU cache with positive size `capacity`.
- `int get(int key)` Return the value of the key if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the `capacity` from this operation, evict the least recently used key.

Follow up:
Could you perform `get` and `put` in `O(1)` time complexity?

Example 1:

Input
```
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
```

Output
```
[null, null, null, 1, null, -1, null, -1, 3, 4]
```

Explanation
```
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
```

```
lRUCache.get(1);    // return -1 (not found)
lRUCache.get(3);    // return 3
lRUCache.get(4);    // return 4
```

## 4) Implement a Fixed Window Rate Limiter

Description: As a part of our service's API gateway, we need to ensure that no single client overwhelms our backend services with too many requests. Your task is to implement a basic rate limiter using the Fixed Window algorithm.

Requirements:

1. The rate limiter should support limiting requests on a per-user basis identified by a unique user ID.
2. Each user is allowed to make up to N requests per T seconds.
3. Once a user exceeds N requests in the current window, they should be prevented from making more requests until the next window.
4. The implementation should be as efficient as possible in terms of time and space complexity.

API to Implement:

```
class RateLimiter:
    def __init__(self, N, T):
        """
        Initialize the rate limiter with a limit of N requests every T seconds.
        """

    def allow_request(self, user_id):
        """
        Returns True if the request is allowed for the user_id, False otherwise.
        """

```

Example Usage:

```
limiter = RateLimiter(2, 60)
user_id = "user123"
print(limiter.allow_request(user_id))
print(limiter.allow_request(user_id))
print(limiter.allow_request(user_id)) # doesn't allow this request
```