Q2: (Tutorial) Email (1/3)

We would like to write three different classes (`Server`, `Client`, and `Email`) to simulate a system for sending and receiving email. Fill in the definitions below to finish the implementation!

Important: We suggest that you approach this problem by first filling out the `Email` class, then the `register_client` method of `Server`, the `Client` class, and lastly the `send` method of the `Server` class.

```
class Email:
    """Every email object has 3 instance attributes: the
    message, the sender name, and the recipient name.
    """
    def __init__(self, msg, sender_name, recipient_name):
        "*** YOUR CODE HERE ***"
```

Q2: (Tutorial) Email (2/3)

We would like to write three different classes (`Server`, `Client`, and `Email`) to simulate a system for sending and receiving email. Fill in the definitions below to finish the implementation!

Important: We suggest that you approach this problem by first filling out the `Email` class, then the `register_client` method of `Server`, the `Client` class, and lastly the `send` method of the `Server` class.

```
class Server:
    """Each Server has an instance attribute clients, which is a dictionary
    that associates client names with client objects."""

    def __init__(self):
        self.clients = {}

    def send(self, email):
        """Take an email and put it in the inbox of the client
        it is addressed to."""




    def register_client(self, client, client_name):
        """Takes a client object and client_name and adds them
        to the clients instance attribute."""
```

Q2: (Tutorial) Email (3/3)

We would like to write three different classes (`Server`, `Client`, and `Email`) to simulate a system for sending and receiving email. Fill in the definitions below to finish the implementation!

Important: We suggest that you approach this problem by first filling out the `Email` class, then the `register_client` method of `Server`, the `Client` class, and lastly the `send` method of the `Server` class.

```python
class Client:
    def __init__(self, server, name):
        self.inbox = []
       "*** YOUR CODE HERE ***"




    def compose(self, msg, recipient_name):
        """Send an email with the given message
        msg to the given recipient client.
        """




    def receive(self, email):
        """Take an email and add it to the
        inbox of this client.
        """
```

```
"""Every Client has instance attributes
name (which is used for addressing emails
to the client), server (which is used to
send emails out to other clients), and
inbox (a list of all emails the client
has received).

>>> s = Server()
>>> a = Client(s, 'Alice')
>>> b = Client(s, 'Bob')
>>> a.compose('Hello, World!', 'Bob')
>>> b.inbox[0].msg
'Hello, World!'
>>> a.compose('CS 61A Rocks!', 'Bob')
>>> len(b.inbox)
2
>>> b.inbox[1].msg
'CS 61A Rocks!'
"""
```

Q4: (Tutorial) NoisyCat

More cats! Fill in this implemention of a class called `NoisyCat`, which is just like a normal `Cat`. However, `NoisyCat` talks a lot -- twice as much as a regular `Cat`! If you'd like to test your code, feel free to copy over your solution to the `Cat` class above.

```
class ____________________________ # Fill me in!
    """A Cat that repeats things twice."""
    def __init__(self, name, owner, lives=9):
        # Is this method necessary? Why or why not?
        "*** YOUR CODE HERE ***"






    def talk(self):
        """Talks twice as much as a regular cat.

        >>> NoisyCat('Magic', 'James').talk()
        Magic says meow!
        Magic says meow!
        """
        "*** YOUR CODE HERE ***"
```

Q7: (Tutorial) Merge

Write a generator function `merge` that takes in two infinite generators `a` and `b` that are in increasing order without duplicates and returns a generator that has all the elements of both generators, in increasing order, without duplicates.

```
def merge(a, b):
    """
    >>> def sequence(start, step):
    ...     while True:
    ...         yield start
    ...         start += step
    >>> a = sequence(2, 3) # 2, 5, 8, 11, 14, ...
    >>> b = sequence(3, 2) # 3, 5, 7, 9, 11, 13, 15, ...
    >>> result = merge(a, b) # 2, 3, 5, 7, 8, 9, 11, 13, 14, 15
    >>> [next(result) for _ in range(10)]
    [2, 3, 5, 7, 8, 9, 11, 13, 14, 15]
    """
    "*** YOUR CODE HERE ***"
```