```python
# yackety_ui.py
#
# ICS 32 Winter 2014
# Code Example
#
# This module implements a simple, console-based Yackety client, which allows
# the user to interact with the Yackety service without being aware of all of
# the underlying details.  It's sort of akin to a smartphone application that
# interacts with Twitter.
#
# Modules are modules in Python, and our modules are on a level field with
# the ones in the Python Standard Library.  We import our modules the same
# way and we call the functions the same way (by qualifying them with the
# name of the module).  The only tricky part is setting them up so that
# Python can find them, the simplest solution to which is to put all of
# the modules comprising a program into the same directory.

import yackety_protocol



# There are better solutions than embedding these kinds of details in the
# code of a program, but this will do for now.  You will need to change
# this string so that it indicates the machine where the Yackety server
# is running.
YACKETY_HOST = 'REPLACE THIS WITH THE HOST WHERE YACKETY IS RUNNING!'
YACKETY_PORT = 6543




# Note how this function reads a lot like English, since most of what it
# does is to call other functions that have clear names.  This is a
# technique you'll want to use in your programs.

def _run_user_interface() -> None:
    '''
    Runs the console-mode user interface from start to finish.
    '''
    _show_welcome_banner()
    username = _ask_for_username()

    connection = yackety_protocol.connect(YACKETY_HOST, YACKETY_PORT)

    try:
        if yackety_protocol.login(connection, username):
            print('Welcome!')
        else:
            print('Login failed')

        # Notice how _handle_command returns False only when there are
        # no more commands to be processed.  That gives us the ability
        # to get out of this loop.
        while _handle_command(connection):
            pass

    finally:
        # No matter what, let's make sure we close the Yackety connection
        # when we're done with it.
        yackety_protocol.close(connection)
```

```
def _handle_command(connection: yackety_protocol.YacketyConnection) -> bool:
    '''
    Handles a single command from the user, by asking the user what command
    they'd like to execute and then handling it.  Returns True if additional
    commands should be processed after this one, False otherwise.
    '''
    command = input('[S]end, [L]ast, or [G]oodbye? ').strip().upper()

    if command == 'S':
        _handle_send_command(connection)
        return True
    elif command == 'L':
        _handle_last_command(connection)
        return True
    elif command == 'G':
        _handle_goodbye_command(connection)
        return False
    else:
        print('Invalid command; try again')
        return True



def _handle_send_command(connection: yackety_protocol.YacketyConnection) -> None:
    '''
    Handles a Send command by asking the user what message they'd like to
    send, then sending it to the server.
    '''
    message_to_send = input('Message to Send: ').strip()

    if len(message_to_send) == 0:
        print('Empty messages are not allowed')
    else:
        if yackety_protocol.send(connection, message_to_send):
            print('Succeeded')
        else:
            print('Failed')



def _handle_last_command(connection: yackety_protocol.YacketyConnection) -> None:
    '''
    Handles a Last command by asking the user how many messages they'd like to
    see, then asking the user to send back those messages.  The number of
    messages must be a positive number.
    '''
    try:
        how_many_messages = int(input('How many messages would you like to see?
'))

    except ValueError:
        # This code will be reached if the user enters a non-number when asked
        # how many messages they'd like to see.
        print('Invalid number of messages; not a number')
        return

    if how_many_messages < 1:
        print('Invalid number of messages; must be positive')
    else:
        messages = yackety_protocol.last(connection, how_many_messages)
        # This chunk of code, in a nutshell, is why the YacketyMessage
        # namedtuple from the yackety_protocol module is so useful.
        # By getting back objects that have a "username" and a "text"
        # field, printing the messages is much more natural than it would
        # be if, for example, we had to parse the username and separate
        # it from the text here.

        print('{} message(s) found'.format(len(messages)))

        for message in messages:
            print(message.username)
            print('    ' + message.text)
            print()



def _handle_goodbye_command(connection: yackety_protocol.YacketyConnection) ->
None:
    '''
    Handles a Goodbye command by exchanging GOODBYE messages with the server.
    '''
    print('Goodbye!')
    yackety_protocol.goodbye(connection)



def _show_welcome_banner() -> None:
    '''
    Shows the welcome banner
    '''
    print('Welcome to Yackety!')
    print()
    print('Please login with your username.')
    print('Remember that usernames must begin with an @ symbol')
    print()


def _ask_for_username() -> str:
    '''
    Asks the user to enter a username and returns it as a string.  Continues
    asking repeatedly until the user enters a username that begins with an
    @ symbol, as Yackety requires.
    '''
    while True:
        username = input('Login: ')

        if username.startswith('@') and len(username) > 1:
            return username
        else:
            print('That username does not start with an @ symbol; please try
again')



if __name__ == '__main__':
    _run_user_interface()
```