Quiz 11 **Name: _____**

1. q1.py is a buggy implementation of the generate_initials question from ICE9.

[Difficulty: **] Implement the function called **generate_initials().** It has a parameter, names (type: list): A list of names (type: str) of length 1 or greater. The function then returns a list of tuples in the format of (initial, name).

To generate the initials, do the following:

a. take the first character of each word in the name.
b. If the name consists of 1 word, take the first 2 characters of the word.
c. If the name consists of more than 3 words, take only the first character of the first 3 words.
d. If the initial has already been used, then append a running number behind it (2,3, ..) for the 2nd, 3rd, .. occurrences.

Example 1: If the function is invoked like this:

```
name_list = ['Alan TAN Ah Beng', 'Apple  LIM', 'Ann Lee']
print(generate_initials(name_list))
```

the statement generates the following output:

```
[('ATA', 'Alan TAN Ah Beng'), ('AL', 'Apple  LIM'), ('AL2', 'Ann Lee')]
```

A `test.py` is provided below:

```
print('Test 1')
result = generate_initials(['Lily TAN', 'Ilse Corina Ee'])
print("Expected:[('LT', 'Lily TAN'), ('ICE', 'Ilse Corina Ee')]")
print('Actual  :' + str(result))
print()

print('Test 2')
result = generate_initials(['Lily TAN', 'Ilse Corina Ee'])
print("Expected:<class 'list'>")
print('Actual  :' + str(type(result)))
print()
```

```
from q1 import generate_initials

print('Test 3')
result = generate_initials(['Lily TAN', 'Ilse Corina Ee'])
print("Expected:<class 'tuple'>")
print('Actual  :' + str(type(result[0])))
print()

print('Test 4')
result = generate_initials(['Alvin LIN Jun Jie', 'Kelvin'])
print("Expected:[('ALJ', 'Alvin LIN Jun Jie'), ('KE', 'Kelvin')]")
print('Actual  :' + str(result))
print()


print('Test 5')
result = generate_initials(['Amy LIM', 'Lucy TAN', 'Lily Tang',
                            'Lesley TOH', 'Lucase TEO'])
print("Expected:[('AL', 'Amy LIM'), ('LT', 'Lucy TAN'),
                 ('LT2', 'Lily Tang'), ('LT3', 'Lesley TOH'),
                 ('LT4', 'Lucase TEO')]")
print('Actual  :' + str(result))
print()

print('Test 6')
result = generate_initials(['Kelvin', 'Kenny', 'Kenneth', 'Kevin',
                            'TAN Beng Suan', 'TAN Beh Sock'])
print("Expected:[('KE', 'Kelvin'), ('KE2', 'Kenny'),
                 ('KE3', 'Kenneth'), ('KE4', 'Kevin'),
                 ('TBS', 'TAN Beng Suan'), ('TBS2', 'TAN Beh Sock')]")
print('Actual  :' + str(result))
print()
```

Identify and correct **ALL** execution and logic errors (i.e., errors that cause the program to behave incorrectly when executed). An error has been identified for you on page 3.

```
# q1.py
def    # mistake 1: should be def instead of define
define generate_initials_for_name(name):
    '''
    this function generates the initials for a person:
    a) if the name consists of 1 word, take the first 2 characters
       of the word.
    b) If the name consists of more than 3 words, take only the
       first character of the first 3 words.

    Parameter:
        name (type:str): The person's name. E.g. 'Awesome TAN Ah Beng'

    Returns:
        The initials for this person.
        E.g. 1: if the name is 'Awesome TAN Ah Beng',
                this function returns 'ATA'
        E.g. 2: if the name is 'Awesome',
                this function returns 'AW'
    '''

    words = name.split()



    if len(words) == 1:

                        2
        return name[:3]


    initials = ''



    for i in range(min(len(words), 3)):

                            [i][0]
        initials += words[0][i]



    return initials.upper()
```

```
def generate_initials(name_list):
    # this is used to store all the initials
    all_initials = []

    result = []  # this is used to store the tuples (initial, name)

    for name in name_list:
        initials = generate_initials_for_name(name)
        all_initials.extend(initials)

        num_occurrences = all_initials.count(initials)
        if num_occurrences  > 1:
            initials += str(num_occurrences - 1)
        result.extend((initials, name))

    return result
```

append

append

2. [**5 marks]** Implement the following function `buy_fruits`. This function takes in 2 parameters:
   a. `inventory` (type: `dict`): Represents the stock available in the shop's inventory. Each key-value pair is in the form `{name:quantity}`, i.e., the key is the product's name and the value is the quantity available.
   b. `order` (type: `dict`): Represents items in a customer's order. Each item is a key-value pair is in the form `{name:quantity}`, i.e., the key is the product's name and the value is the quantity that the customer wishes to purchase.

   For each order's item, this function will check the inventory to see if the item is available, and of sufficient quantity. If all items are available and of sufficient quantities, proceed to process the order (i.e. reduce the quantity from the inventory) and return `True`. Otherwise, return `False`.

```
from q2 import buy_fruits

print("Test 1")
inventory = {'apple':3, 'orange':4, 'pear':5}
result = buy_fruits(inventory, {'apple':4})
print("Expected:False")
print(f"Actual  :{result}")
print("Expected inventory:{'apple': 3, 'orange': 4, 'pear': 5}")
print(f"Actual inventory  :{inventory}")
print()

print("Test 2")
inventory = {'apple':3, 'orange':4, 'pear':5}
result = buy_fruits(inventory, {'apple':4, 'orange':2})
print("Expected:False")
print(f"Actual  :{result}")
print("Expected inventory:{'apple': 3, 'orange': 4, 'pear': 5}")
print(f"Actual inventory  :{inventory}")
print()

print("Test 3")
inventory = {'apple':3, 'orange':4, 'pear':5}
result = buy_fruits(inventory, {'apple':2, 'orange':4, 'papaya':1})
print("Expected:False")
print(f"Actual  :{result}")
print("Expected inventory:{'apple': 3, 'orange': 4, 'pear': 5}")
print(f"Actual inventory  :{inventory}")
print()
```

```
print("Test 4")
inventory = {'apple':3, 'orange':4, 'pear':5}
result = buy_fruits(inventory, {'apple':2, 'orange':4})
print("Expected:False")
print(f"Actual  :{result}")
print("Expected inventory:{'apple': 1, 'orange': 0, 'pear': 5}")
print(f"Actual inventory  :{inventory}")
print()

print("Test 5")
inventory = {'apple':3, 'orange':4, 'pear':5}
result = buy_fruits(inventory, {'apple':2, 'orange':4})
print("Expected:<class 'bool'>")
print(f"Actual  :{type(result)}")
print()
```

It will generate the following output:

```
Test 1
Expected:False
Actual  :False
Expected inventory:{'apple': 3, 'orange': 4, 'pear': 5}
Actual inventory  :{'apple': 3, 'orange': 4, 'pear': 5}

Test 2
Expected:False
Actual  :False
Expected inventory:{'apple': 3, 'orange': 4, 'pear': 5}
Actual inventory  :{'apple': 3, 'orange': 4, 'pear': 5}

Test 3
Expected:False
Actual  :False
Expected inventory:{'apple': 3, 'orange': 4, 'pear': 5}
Actual inventory  :{'apple': 3, 'orange': 4, 'pear': 5}

Test 4
Expected:False
Actual  :True
Expected inventory:{'apple': 1, 'orange': 0, 'pear': 5}
Actual inventory  :{'apple': 1, 'orange': 0, 'pear': 5}

Test 5
Expected:<class 'bool'>
Actual  :<class 'bool'>
```

```
# Answer
def buy_fruits(inventory, order):
    is_sufficient = True
    for key, value in order.items():
        if key in inventory:
            if value > inventory[key]:
                is_sufficient = False
                break

        else:
            is_sufficient = False
            break

    if is_sufficient:
        for key, value in order.items():
            inventory[key] -= value

        return True
    else:
        return False
```