2023 C2 TPE solution (with marker’s comments)

Task 1

Solution (part 1)

SARAH.py (server/sarah)	Comments

`import socket`
`mysocket = socket.socket()`
`mysocket.bind(('127.0.0.1',3036))`
`mysocket.listen()`
`user, addr = mysocket.accept()`
	Students are discouraged from changing the code provided.

`nuggets = input('How many nuggets? ') # input total`
`user.sendall(nuggets.encode() + b'\n') # sends total`
	a few students forgot to send the number or sent a full message.

`while True:`
`    data = b''`
`    while b'\n' not in data:`
`        data += user.recv(1024)`
`    msg = data.decode().strip()`	

`    if msg == 'end': # not digit, 'end' received`
`        print('You win.')`
`        break`
	the question asked for ‘end’ to be sent

`    elif msg.isdigit():`
`        nuggets = int(msg)`
`        print(f'There are {nuggets} nuggets left.')`
`        num = int(input('How many nuggets? '))`
`        while (num > 3) or (num < 1) or (num > nuggets):`
`            print(f'There are {nuggets} nuggets left.')`
`            num = int(input('How many to take? '))`
`        nuggets -= num # remove from total`
	when checking for validity, a while-loop should be used.
	students are discouraged from using a mix of `and` and `or` statements

`        if nuggets > 0:`
`            print(f'{num} nuggets taken, {nuggets} left.')`
`            user.sendall(str(nuggets).encode() + b'\n')`
`        else:`
`            print('You lose.')`
`            user.sendall('end'.encode() + b'\n')`
`            break`
	some students sent messages different from the instructions in the question

`user.close()`
`mysocket.close()`	

Solution (part 2)

CHLOE.py (client/chloe)	**Comments**

`import socket`
`mysocket = socket.socket()`
`mysocket.connect(('127.0.0.1',3036))`
	students should be using the port provided in the file SARAH.py

`while True:`
`    data = b''`
`    while b'\n' not in data:`
`        data += mysocket.recv(1024)`
`    msg = data.decode().strip()`
	students are encouraged to use any format/style that is provided

`    if msg.isdigit():`
`        nuggets = int(msg)`
`        num = int(input('How many nuggets? '))`
`        while (num > 3) or (num < 1) or (num > nuggets):`
`            print(f'There are {nuggets} nuggets left.')`
`            num = int(input('How many to take? '))`
`        nuggets -= num`
	no additional marks were given here as most of this part is similar to SARAH.py
	
	only need to change the socket name when `.sendall()` and `.recv()`

`        if nuggets > 0:`
`            print(f'{num} nuggets taken, {nuggets} left.')`
`            mysocket.sendall(str(nuggets).encode() + b'\n')`
`        else:`
`            print('You lose.')`
`            mysocket.sendall('end'.encode() + b'\n')`
`            break`	

`    elif msg == 'end': # not digit, 'end' received`
`        print('You win.')`
`        break`	

`mysocket.close()`	

Task 2 Appropriate messages and white space to present the output in a readable manner.

```
# Task 2.1
# connect to server
import pymongo, json
client = pymongo.MongoClient('127.0.0.1', 27017)
client.drop_database('Store') # drop database for testing

# create database Store with collection Order - Method 1
db = client['Store']
coll = client['Store']['Order']
# # create database Store with collection Order - Method 2
# db = client.get_database('Store')
# coll = db.get_collection('Order')

# insert the order documents
file = open('DRINK.JSON', 'r')
data = json.load(file)
file.close()
coll.insert_many(data)

# display all the order documents
docs = coll.find({},{"_id":0})
for doc in docs:
    print(doc)
```

Most students completed this task. Note that we may need the code to drop the database for testing purposes.

There are many ways to create a database and insert data. If we forget the syntax to read the data file, we can copy the data into the codes.

Make sure we test Task 2.2 and 2.3 based on the correct database. Some students inserted data multiple times and presented duplicate output in all the tasks.

```
# Task 2.2
# orders without upsizing
docs = coll.find({"Upsize":{"$exists":False}}, {"_id":0})
print("Orders without upsizing:")
for doc in docs:
    print(doc)
print()

# orders without bubble topping
docs = coll.find({"Toppings":{"$nin":["Bubble"]}}, {"_id":0})
# docs = coll.find({"Toppings":{"$ne":"Bubble"}}, {"_id":0})
print("Orders without bubble topping:")
for doc in docs:
    print(doc)
print()

# orders with healthier choices (sugar level <= 50%)
docs = coll.find({"Sugar":{"$lte":50}}, {"_id":0})
print("Orders with healthier choices (sugar level up to 50%):")
for doc in docs:
    print(doc)
print()
```

Query without bubble:

There can be no toppings at all. Topping can be a string but not bubble, or a list but does not contain bubble. Note that `nin` and `ne` can solve all cases.

```
# total number of orders by Jason
orders = coll.find({"Member":"Jason"}).count()
# orders = coll.count({"Member":"Jason"})
print("Total number of orders by Jason:", orders)
print()

# average price per order
docs = coll.find()
orders = 0
total = 0
for doc in docs:
    orders += 1
    total += doc['Price']
print("Average price per order:", round(total/orders, 1))
print()
```

Some students are not aware of `count`.

Full price of 1 d.p. shall be included in the calculation and the average shall be at least the same accuracy level (1 d.p.).

```
Orders without upsizing:
{'Date': 'May 20', 'Time': '1430', 'Tea': 'Jasmine Green Tea', 'Sugar': 50,
'Price': 5.5, 'Toppings': 'Bubble', 'Member': 'Alice'}
{'Date': 'May 20', 'Time': '2100', 'Tea': 'Caramel Milk Tea', 'Sugar': 25,
'Price': 5.3, 'Toppings': 'Bubble', 'Member': 'Jason'}
{'Date': 'May 21', 'Time': '1120', 'Tea': 'Mango Milk Tea', 'Sugar': 75, 'Price':
5.8, 'Toppings': 'Bubble', 'Member': 'Alice', 'Ice': 'Less'}
{'Date': 'May 21', 'Time': '1510', 'Tea': 'Honey Green Tea', 'Sugar': 50,
'Price': 6.5, 'Toppings': ['Bubble', 'Ai Yu'], 'Member': 'Alice'}
{'Date': 'May 21', 'Time': '1820', 'Tea': 'Yakult Green Tea', 'Sugar': 25,
'Price': 4.5, 'Ice': 'No', 'Toppings': ['Bubble', 'Konjac']}

Orders without bubble topping:
{'Date': 'May 21', 'Time': '0920', 'Tea': 'Green Milk Tea', 'Sugar': 75, 'Price':
4.8, 'Upsize': 'True', 'Member': 'Jason'}
{'Date': 'May 21', 'Time': '1350', 'Tea': 'Honey Oolong Tea', 'Sugar': 25,
'Price': 5.3, 'Ice': 'Less', 'Upsize': 'True'}
{'Date': 'May 21', 'Time': '1403', 'Tea': 'Lemon Tea', 'Sugar': 120, 'Price':
7.5, 'Upsize': 'True', 'Toppings': ['Ai Yu', 'Coffee Jelly']}
{'Date': 'May 22', 'Time': '1330', 'Tea': 'Hazelnut Milk Tea', 'Sugar': 75,
'Price': 6.3, 'Upsize': 'True', 'Toppings': ['Coffee Jelly', 'Taro Q'], 'Member':
'Jason'}
{'Date': 'May 22', 'Time': '1502', 'Tea': 'Lychee Milk Tea', 'Sugar': 0, 'Price':
5.1, 'Ice': 'Less', 'Upsize': 'True'}

Orders with healthier choices (sugar level up to 50%):
{'Date': 'May 20', 'Time': '1430', 'Tea': 'Jasmine Green Tea', 'Sugar': 50,
'Price': 5.5, 'Toppings': 'Bubble', 'Member': 'Alice'}
{'Date': 'May 20', 'Time': '2100', 'Tea': 'Caramel Milk Tea', 'Sugar': 25,
'Price': 5.3, 'Toppings': 'Bubble', 'Member': 'Jason'}
{'Date': 'May 21', 'Time': '1350', 'Tea': 'Honey Oolong Tea', 'Sugar': 25,
'Price': 5.3, 'Ice': 'Less', 'Upsize': 'True'}
{'Date': 'May 21', 'Time': '1510', 'Tea': 'Honey Green Tea', 'Sugar': 50,
'Price': 6.5, 'Toppings': ['Bubble', 'Ai Yu'], 'Member': 'Alice'}
```

```
{'Date': 'May 21', 'Time': '1820', 'Tea': 'Yakult Green Tea', 'Sugar': 25,
'Price': 4.5, 'Ice': 'No', 'Toppings': ['Bubble', 'Konjac']}
{'Date': 'May 22', 'Time': '1502', 'Tea': 'Lychee Milk Tea', 'Sugar': 0, 'Price':
5.1, 'Ice': 'Less', 'Upsize': 'True'}

Total number of orders by Jason: 3

Average price per order: 5.9
```

```
# Task 2.3
# Exclude ice for all the orders from Alice on May 21
query = {"Member":"Alice", "Date":"May 21"}
update = {"$set":{"Ice":"No"}}
coll.update_many(query, update)

# Remove the field of upsizing for all orders
coll.update_many({},{"$unset":{"Upsize":0}})


# Remove all the orders by Jason.
coll.delete_many({"Member":"Jason"})

# display all the order documents
print("After changes made:")
docs = coll.find({},{"_id":0})
for doc in docs:
    print(doc)
client.close()
```

Orders without Ice field means normal ice, must also be updated to ‘No Ice’.

Many students are not familiar with the use of `set` and `unset`.

```
After changes made:
{'Date': 'May 20', 'Time': '1040', 'Tea': 'Golden Oolong Tea', 'Sugar': 100,
'Price': 6.2, 'Toppings': ['Bubble', 'Konjac']}
{'Date': 'May 20', 'Time': '1430', 'Tea': 'Jasmine Green Tea', 'Sugar': 50,
'Price': 5.5, 'Toppings': 'Bubble', 'Member': 'Alice'}
{'Date': 'May 20', 'Time': '2030', 'Tea': 'Assam Black Tea', 'Sugar': 100,
'Price': 7.8, 'Toppings': ['Bubble', 'Konja', 'Taro Q', 'Ai Yu']}
{'Date': 'May 21', 'Time': '1120', 'Tea': 'Mango Milk Tea', 'Sugar': 75, 'Price':
5.8, 'Toppings': 'Bubble', 'Member': 'Alice', 'Ice': 'No'}
{'Date': 'May 21', 'Time': '1350', 'Tea': 'Honey Oolong Tea', 'Sugar': 25,
'Price': 5.3, 'Ice': 'Less'}
{'Date': 'May 21', 'Time': '1403', 'Tea': 'Lemon Tea', 'Sugar': 120, 'Price':
7.5, 'Toppings': ['Ai Yu', 'Coffee Jelly']}
{'Date': 'May 21', 'Time': '1510', 'Tea': 'Honey Green Tea', 'Sugar': 50,
'Price': 6.5, 'Toppings': ['Bubble', 'Ai Yu'], 'Member': 'Alice', 'Ice': 'No'}
{'Date': 'May 21', 'Time': '1820', 'Tea': 'Yakult Green Tea', 'Sugar': 25,
'Price': 4.5, 'Ice': 'No', 'Toppings': ['Bubble', 'Konjac']}
{'Date': 'May 22', 'Time': '1420', 'Tea': 'Peach Green Tea', 'Sugar': 100,
'Price': 5.8, 'Toppings': 'Bubble', 'Member': 'Alice'}
{'Date': 'May 22', 'Time': '1502', 'Tea': 'Lychee Milk Tea', 'Sugar': 0, 'Price':
5.1, 'Ice': 'Less'}
```

Task 3

```
# Task 3

class WordRank:
    def __init__(self, word, rank):
        self.left = None
        self.right = None
        self.data = (word, rank) # 2-tuple data


class Tree:
    def __init__(self):
        self.root = None

    def store(self, word, rank):
        new = WordRank(word, rank) # create node for storage
        inserted = False
        if self.root == None: # store in empty tree
            self.root = new
            inserted = True
        ptr = self.root
        while not inserted:
            if ptr.data[1] > new.data[1]: # compare rank
                if ptr.left == None: # store left if None
                    ptr.left = new
                    inserted = True
                else:
                    ptr = ptr.left # move left
            else:
                if ptr.right == None:
                    ptr.right = new
                    inserted = True
                else:
                    ptr = ptr.right


    def inorder(self, subtree):
        if subtree != None: # terminating condition
            self.inorder(subtree.left)
            print(f"{subtree.data[1]}:{subtree.data[0]}",end=", ")
            self.inorder(subtree.right)


    def preorder(self, subtree):
        if subtree != None: # terminating condition
            print(subtree.data[0], end=" ")
            self.preorder(subtree.left)
            self.preorder(subtree.right)
```

data is a tuple - most student store the data as list instead of a tuple

Student needs to know what self represent. - the current object operated on

```
# main program – Method 1
# open file using csv.reader()
import csv
csvfile = open('words.csv', 'r')
csvdata = csv.reader(csvfile)

newBST = Tree() # create empty Tree
for row in csvdata: # iterate the csv file
    if row[1].isdigit(): # resolve header [word, OEC rank]
        newBST.store(row[0], int(row[1])) # store rank as integer
csvfile.close()

newBST.inorder(newBST.root)
print('\n')
newBST.preorder(newBST.root)
```

rank needs to be converted to int Otherwise you will be using string comparison instead of int comparison

```
# main program – Method 2
wordsfile = open('words.csv', 'r')
text = wordsfile.read()
wordsfile.close()

BST = Tree() # create empty Tree
# resolve header [word, OEC rank]
data = text.strip().split('\n')[1:]
for item in data: # iternate the csv file
    word, rank = item.strip().split(',')
    BST.store(word, int(rank)) # store rank as integer

BST.inorder(BST.root)
print('\n')
BST.preorder(BST.root)
```

```
1: the, 2: be, 3: to, 4: of, 5: and, 6: a, 7: in, 8: that, 9: have, 10: I, 11: it, 12:
for, 13: not, 14: on, 15: with, 16: he, 17: as, 18: you, 19: do, 20: at, 21: this, 22:
but, 23: his, 24: by, 25: from, 26: they, 27: we, 28: say, 29: her, 30: she, 31: or, 32:
an, 33: will, 34: my, 35: one, 36: all, 37: would, 38: there, 39: their, 40: what, 41: so,
42: up, 43: out, 44: if, 45: about, 46: who, 47: get, 48: which, 49: go, 50: me, 51: when,
52: make, 53: can, 54: like, 55: time, 56: no, 57: just, 58: him, 59: know, 60: take, 61:
people, 62: into, 63: year, 64: your, 65: good, 66: some, 67: could, 68: them, 69: see,
70: other, 71: than, 72: then, 73: now, 74: look, 75: only, 76: come, 77: its, 78: over,
79: think, 80: also, 81: back, 82: after, 83: use, 84: two, 85: how, 86: our, 87: work,
88: first, 89: well, 90: way, 91: even, 92: new, 93: want, 94: because, 95: any, 96:
these, 97: give, 98: day, 99: most, 100: us,

a and be the of to about all an as for have in that I it he not on with at do you but this
by his from her say they we or she my will one if out so their there would what up after
also can get who go which make me when come could good him just like no time into know
people take year your some look now other see them than then only its over think back any
because even first how two use our work way well new want day give these most us
```

Task 4

Comments in every task is a habit not just to score, but more importantly to help us design our codes in an organized and well-paced manner. General problem solving questions allow us to design creative solutions so it is even more important to explain our design properly.

Reading the question and following the instructions closely are also very important exam skills. Common mistakes include the data type of the input parameters and the return values, the variable/function names given, the format of the output (no '-', no 'and', no 'zero's).

```
# Task 4.1
def up_to_two_digit(num):
    # convert an integer (1-99) into word form as a string

    if num < 20: # direct match for numbers below 20
        wordlist = ['one','two','three','four','five', \
                    'six','seven','eight','nine','ten',\
                    'eleven','twelve','thirteen','fourteen','fifteen',\
                    'sixteen','seventeen','eighteen','nineteen']
        return wordlist[num - 1]

    else:
        ones = ['one','two','three','four','five', \
                'six','seven','eight','nine']
        tens = ['twenty','thirty','forty','fifty',\
                'sixty','seventy','eighty','ninety']

        if num % 10 == 0: # direct match for multiples of 10
            return tens[num // 10 - 2]
        else: # two components combined
            return tens[num // 10 - 2] + ' ' + ones[num % 10 - 1]

# use function to create text file
f = open('NUMBERS.TXT', 'w')
for i in range(1, 100):
    f.write(up_to_two_digit(i)+'\n')
f.close()
```

Use of a list or dictionary saves our lives to write long if-else statements. We try to use the tools we learnt to avoid too much hard coding.

Note that 20, 30, 40, etc does not have the extra parts of zeros.

Many students did not follow the question to write the words into the text file.

```
# Task 4.2
def up_to_three_digit(num):
    # convert an integer (1-999) into word form as a string

    if num // 100 == 0: # up to two digits
        return up_to_two_digit(num)

    elif num % 100 == 0: # direct match for multiples of 100
        ones = ['one','two','three','four','five',\
                'six','seven','eight','nine']
        return ones[num // 100 - 1] + ' hundred'

    else: # two components combined, using up_to_two_digit function
        ones = ['one','two','three','four','five',\
                'six','seven','eight','nine']
        return ones[num//100-1]+' hundred '+up_to_two_digit(num % 100)

print(up_to_three_digit(345))
print(up_to_three_digit(60))
print(up_to_three_digit(700))
print(up_to_three_digit(890))
```

```
three hundred forty five
sixty
seven hundred
eight hundred ninety
```

```
# Task 4.3
def convert(money):
    # convert a float (1.00 to 999 999 999 999.99) into two forms

    # split the number into dollars and cents
    money_str = str(money)
    dollar, cent = money_str.split('.')

    # break dollar into sections
    sections = []
    extra = len(dollar) % 3
    if extra != 0:
        sections.append(dollar[:extra])
    for i in range(extra, len(dollar), 3):
        sections.append(dollar[i:i+3])

    # create number form with comma separators
    money_comma = ','.join(sections) + '.' + cent

    # create word form with dollars and cents
    money_word = ''
    # match the sections with the words to combine
    words = ['billion', 'million','thousand', '']
    for i in range(- 1, - len(sections) - 1, -1):
        if int(sections[i]) != 0:
            money_word = up_to_three_digit(int(sections[i])) + \
                          ' ' + words[i] + ' ' + money_word
    money_word += 'dollars'
    if int(cent) != 0: # add cents
        money_word += ' '+ up_to_two_digit(int(cent)) + ' cents'

    money_word = money_word.replace('  ', ' ') # optional

    print("Money:", money)
    print("Number Form:", money_comma)
    print("Word Form:", money_word)
    print()

convert(1.00)
convert(1234000567.89)
```

This is the most challenging part that requires good design and meticulous debugging.

```
Money: 1.0
Number Form: 1.0
Word Form: one dollars

Money: 1234000567.89
Number Form: 1,234,000,567.89
Word Form: one billion two hundred thirty four million five hundred sixty seven dollars
eighty nine cents
```