

Computing GCSE Coursework

Thomas Bass
Candidate 4869
Centre 52423
OCR A453 Programming Project

Word Count 1678

Made with L^AT_EX
2016-2017

Summary

| | |
|-----------------------------|-----------|
| 1 Objectives | 3 |
| 1.1 Task 1 | 3 |
| 1.2 Task 2 | 3 |
| 1.3 Task 3 | 3 |
| 2 Test Plan | 4 |
| 2.1 Task 1 | 4 |
| 2.2 Task 2 | 4 |
| 2.3 Task 3 | 5 |
| 3 Pseudocode | 6 |
| 3.1 Task 1 | 6 |
| 3.2 Task 2 | 7 |
| 3.3 Task 3 | 7 |
| 4 Data Structure | 8 |
| 4.1 Task 1 | 8 |
| 4.2 Task 2 | 8 |
| 4.3 Task 3 | 8 |
| 5 Development | 9 |
| 5.1 Task 1 | 9 |
| 5.2 Task 2 | 12 |
| 5.3 Task 3 | 14 |
| 6 Testing | 15 |
| 6.1 Task 1 | 15 |
| 6.2 Task 2 | 16 |
| 6.3 Task 3 | 18 |
| 7 Final Program Code | 19 |
| 7.1 Task 1 | 19 |
| 7.2 Task 2 | 20 |
| 7.3 Task 3 | 21 |

1 Objectives

1.1 Task 1

1. Take an input and verify that it is 8 or 7 numerical digits
2. Calculate the 8th check digit:
 - a Multiply the first 7 numbers alternately by 3,1
 - b Total these results
 - c Subtract this sum from its nearest highest multiple of 10
3. Compare this to the given 8th number, or complete the 7-digit number

1.2 Task 2

1. Take an input and validate that it is 8 numerical digits
2. Connect to a SQL database and run a query
3. Collect and display the results
4. Update the database with the customer's order
5. Print a receipt
6. Cope with SQL errors

1.3 Task 3

1. Scan a database and find stock to order
2. Create a receipt of the order
3. Update the database with the updated stock level
4. Cope with any SQL errors

2 Test Plan

2.1 Task 1

1. Input strings of incorrect length. If rejected, it passes.

Test data: 12345

Test data: 1234567890

2. Input strings of letters. If rejected, it passes.

Test data: abcdefg

3. Get program to run a valid input. Print out totals at each stage, and check them manually. If they are the same, it passes.

Test data: 13245627

- i. Manually add the totals of a valid input. If they are the same, it passes.
 - ii. Manually round the total to the highest 10. If it is the same, it passes.
 - iii. Manually collect the distance rounded. If it is the same, it passes.
4. Run the program with a GTIN number taken from a product. If it correctly calculated and verified, it passes.

2.2 Task 2

1. Input strings of incorrect length. If rejected, it passes.

Test data: 12345.

Test data: 1234567890.

2. Input strings of letters. If rejected, it passes.

Test data: abcdefg.

3. Input a valid string to search. If product found, it passes.

Test data: 11440529.

4. Manually check that the program has displayed the correct stock level and information. If it does, it passes.

Stock info: 50 in stock for #11440529 (red paint 100ml).

5. Order a quantity of the product. If the program updates the stock levels, it passes.

Test data: order 5 x QTY of #11440529 (red paint 100ml).

6. Complete a full order. If the program displays a receipt with the correct values, it passes.

Test data: order 5 x QTY of #11440529 (red paint 100ml) AND

6 x QTY of #11509493 (blue paint 100ml).

Expected result: 5 x #11509493 = £9.95 AND 6x #11509493 = £11.94, total: £21.90 .

7. Provide the program with invalid values (such as ordering negative values). If the program rejects these, it passes.

Test data: -3 x QTY #11509493.

Expected result: error and re-input.

Test data 0 x QTY #11509493.

Expected result: error and re-input.

Test data: 100 x QTY #11509493.

Expected result: not enough stock, re-input.

2.3 Task 3

1. Edit database for reduced stock. If reduced stock is identified, it passes.

Test data: Edit a stock level to -10 of stock level.

Print receipt.

2. Edit database for reduced stock. If a human-readable receipt is produced, it passes.

Test data: Edit two stock levels to -10 of stock level.

Print receipt.

3. Complete order for reduced stock. If the database updated, it passes.

Continue from test area 2, and update database.

Check database manually.

4. If the program handles SQL errors, it passes.

Attempt to update from more than stock level.

3 Pseudocode

3.1 Task 1

START

User INPUT choice for calculate or verify

IF Calculate:

Number Length is 7

INPUT GTIN

CALL Verify function

ELSE IF Verify:

Number Length is 8

INPUT GTIN

CALL verify Function

ENDIF

ENDIF

Verify Function:

IF GTIN length = Length AND is all numeric:

For a loop of 7 by step of 2:

total = total+(GTIN [counter]*3)

IF counter =6:

Round total UP to nearest multiple of 10

result = roundedNumber - total

If Length = 7:

Print result

ELSE

If GTIN at position Length = result:

Print GTIN is a valid number

ELSE:

Print GTIN is an invalid number

ENDIF

ELSE

Multiply GTIN at position of counter+1 by 1 and add to total

ENDIF

ELSE

Print error and return to GTIN input

ENDIF

END

3.2 Task 2

```
START
User INPUT GTIN number
IF GTIN is numerical AND GTIN = 8 characters:
    Search Database for GTIN number
    IF result found:
        User INPUT quantity to order
        IF quantity > 0 AND quantity <= stock available
            PRINT receipt with total cost (cost per item * quantity ordered)
            Update Database with new stock (stock available a quantity ordered)
            User INPUT choice to order more items
            IF choice = yes:
                Add to order list and return to GTIN INPUT
            ELSE
                PRINT final receipt (order list) and end program
            ENDIF
        ELSE
            Return to Quantity INPUT
        ENDIF
    ELSE
        Return to GTIN INPUT
    ENDIF
ELSE
    Return to GTIN INPUT
ENDIF
END
```

3.3 Task 3

```
START
Connect to SQL Database
Search database:
    IF stock level < target stock level:
        RETURN results
PRINT results
APPEND results to order list
IF order complete:
    UPDATE database:
        stock level = target stock level
ELSE
    END
ENDIF
END
```

4 Data Structure

4.1 Task 1

| Variable Name | Variable Description |
|-----------------|---|
| ask | The choice whether the user wants to verify or calculate |
| gtin | The GTIN number used |
| length | The length the GTIN should be |
| total | The running total of all the multiplications |
| checkdig | The 8th digit in a verification |
| rounded | total Rounded up to the nearest 10 |
| result | The 8th digit as the program calculates it |
| again | The choice of whether the user wants to run the program again |

4.2 Task 2

| Variable Name | Variable Description | Value |
|--|--|---|
| con and cur | Connections to SQL database | N/A |
| var | User Input GTIN number | User Defined |
| results | Fetchall results from SQL query | N/A (list) |
| product | Equal to results , reformatted | Equal to results |
| sizeName and sizeNameRaw | Variables used to format the name of the product | Name of product selected |
| QtyToOrder | User Input quantity ordered | User Defined |
| NewStockAvab | Variable used to update the SQL database with the new stock levels | Stock Available minus QtyToOrder |
| costOfOrder | Total cost of order | Price of product* QtyToOrder |
| currentOrderAddRaw and currentOrderAdd and currentOrder | Variables used to format and append the order to the entire list of order (to print receipt) | N/A |

4.3 Task 3

| Variable Name | Variable Description | Value |
|--|--|---|
| con and cur | Connections to SQL database | N/A |
| sql1 | SQL command for finding reduced stock levels | SELECT* FROM Inventory WHERE StockAvab != Target Stock |
| results | Fetchall results from SQL query | N/A, List |
| toOrder | Value of how much stock needs to be ordered for each product | targetStock - stockAvab |
| sizeName and sizeNameRaw | Used to correctly format the product name | N/A |
| stockOrderAddRaw and stockOrderAdd and stockOrder | Used to format the list of the stock update order | N/A |
| orderList | Final order list | N/A |

5 Development

5.1 Task 1

For all my development, the code was written in Python's IDLE, an integrated development environment which comes packaged with Python installs.

Objective 1: Take an input and verify that it is 8 or 7 numerical digits

start() Function

```
print('GCSE Controlled Assesment A453\nThomas Bass 4869\nTask 1')
def start():
    ask = input('Press [c] to calculate the 8th digit from 7 \nPress [v] to verify an 8 digit GTIN Number \n')
    if ask == 'c' or == 'C':
        length == 7
    elif ask == 'v' or == 'V':
        length == 8
    else:
        print('Error: Please enter either \'c\' or \'v\' ')
        start()
    check(length)
```

This function takes a user input of 'ask' to decide if the user wants to either calculate the 8th GTIN digit from 7 digits (choice c), or verify the 8th GTIN digit from 8 digits (choice v). If the choice is c, the program creates the variables `gtin` and `length`, and sets `length` to 7. It then calls the `check()` function carrying `length` with it.

If the choice is v, the program creates the variables `gtin` and `length`, and sets `length` to 8. It then calls the `check()` function carrying `length` with it.

The first revision produced syntax errors on line 4 and 6. This is because the variable has to be re-stated after an `or` condition. After this was corrected, the IDLE gave the following error:

```
Traceback (most recent call last):
  File "/Users/ThomasBass/GitHub/GSCE-Coursework-Python-GTIN/Task 1/FINAL/Task 1.py", line 44, in <module>
    start()
  File "/Users/ThomasBass/GitHub/GSCE-Coursework-Python-GTIN/Task 1/FINAL/Task 1.py", line 7, in start
    length == 7
NameError: name 'length' is not defined
```

This error was produced as the incorrect operators were used in lines 5 and 7. The IDLE interpreted the code to compare `length` to the number 7 (line 5) or 8 (line 7). These were then corrected to `length = 7` and `length = 8` respectively. The program then gave the following output:

```
RESTART: /Users/ThomasBass/GitHub/GSCE-Coursework-Python-GTIN/Task 1/FINAL/Task 1.py
GCSE Controlled Assesment A453
Thomas Bass 4869
Task 1
Press [c] to calculate the 8th digit from 7
Press [v] to verify an 8 digit GTIN Number
c
```

From the following code:

```
def start():    ## Main process
    ask = input('Press [c] to calculate the 8th GTIN Number from 7 numbers. \nPress [v] to verify an 8 digit GTIN
    if ask == 'c' or ask == 'C':
        gtin = 0
        length = 7
        check(length)
    elif ask == 'v' or ask == 'V':
        gtin = 0
        length = 8
        check(length)
    else: print('Error: Please enter either \'c\' or \'v\' ')
    start()
```

This shows that the `start()` function is working correctly

check() Function

```
def check(length):  
    print('Enter the', length, 'digit GTIN number')  
    gtin = input(': ')  
    if len(gtin) == length:  
        total = 0  
    else:  
        print('Error: Only', length, 'numbers are allowed. Try again ')
```

This function prints a statement asking the user for the `length` length GTIN. It then takes the user input of `gtin`.

If the length of `gtin` is equal to `length` and `gtin.isnumeric` function is `True` (the variable is numerical) then it creates the variable `total`. Else, it prints an error message, and returns to the `check()` function.

The first revision gave a syntax error on line 4, as there was a missing colon. This was added, and the second revision ran, and rejected incorrect lengths, but allowed correct lengths including letters. The `math` Python library provides the `isnumeric` function that gives boolean `False` if the variable carried contains characters other than numerical. This was used as `if gtin.isnumeric() == True`. The program then gave the following output:

```
Enter the 7 digit GTIN number  
: abcdef  
Error: Only 7 numbers are allowed. Try again
```

The program worked and rejected invalid strings, but if the user had entered an invalid input, the program would reject it, but terminate the program. The code was then added after line 7 to call `check(length)` after an invalid input. `length` is carried so that the program does not have to re-start. The program then ran the following output:

```
Enter the 7 digit GTIN number  
: abcdef  
Error: Only 7 numbers are allowed. Try again  
Enter the 7 digit GTIN number  
: |
```

With the following code:

```
def check(length):  
    print('Enter the', length, 'digit GTIN number')  
    gtin = input(': ')  
    if len(gtin) == length and gtin.isnumeric() == True:  
        total = 0  
    else:  
        print('Error: Only', length, 'numbers are allowed. Try again ')  
        check(length)
```

This shows that the program is correctly calling `check(length)`, and is working correctly.

Objective 2: Calculate the 8th check digit

Multiply the first 7 numbers alternately by 3,1

```
for counter in range(0, 7, 2):  
    total = total + ((int(gtin[counter]))*3)  
    total = total + ((int(gtin[counter+1]))*1)
```

This snippet starts a loop for the value of `counter`, which goes from 0 to 7, stepping by 2 each time. The counter could have gone 0 to 3 stepping by 1, but this would make it more complicated. The program then adds the following to `total` : integer of: `gtin` at position of `counter` multiplied by 3. It then adds to following to `total` : integer of `gtin` at position of `counter+1` multiplied by 1.

Total these results

The program simply adds all of these calculations to `total`

Subtract this sum from its nearest highest multiple of 10

```
if counter == 6:  
    checkdig = int(gtin[length-1])  
    rounded = (int(math.ceil(total / 10.0)) * 10)  
    result = (rounded - total)
```

This snippet checks to see if the loop is on its final iteration (`counter = 6`). It then sets `checkdig` to the penultimate digit of `gtin`. It then creates the variable `rounded` and sets it to to the nearest highest multiple of 10 of `total`. It then creates the variable `result` and sets it to the result of `rounded - total`.

Objective 3: Compare this to the given 8th number, or complete the 7-digit number

```
if length == 7:  
    print('Final Check Digit = ', result)  
    print('Whole GTIN-8 Number = ', gtin,result)  
    park()  
else:  
    if checkdig == result: print(gtin, 'is a Valid Number')  
    else: print(gtin, 'is an Invalid Number')  
    park()
```

In this section of code, if `length` equals 7, the program prints the final check digit (`result`), and also prints the whole calculated GTIN (`gtin` and `result`). It then calls the `park()` function. Else, if `checkdig` is equal to `result`, it prints that `gtin` is a valid number. Else, it prints that `gtin` is an invalid number.

park() Function

The `park()` function is used after a verification or calculation has been made, and allows the user to carry out more verifications or calculations without restarting the program.

```
def park():  
    again = input('Do you want to calculate or verify another number? \n[n] No [y] Yes: ')  
    if again == 'n' or again == 'N':  
        sys.exit()  
    elif again == 'y' or again == 'Y':  
        start()
```

This program asks the user if they want to restart the program. If yet, it calls `start()` function, and if no it ends the program. The `sys` Python library provides the `.exit()` function to terminate the program. In IDLE it simply ends it, but when compiled into command line it will close the window.

5.2 Task 2

Objective 1: Take an input and validate that it is 8 numerical digits

```
def verify(con, cur, currentOrder):  
    var = input('Enter GTIN for the product you wish to purchase:\n> ')  
    if len(var) == 8 and var.isnumeric() == True:  
        findStock(con, cur, currentOrder, var)  
    else:  
        print('Enter a 8 digit number')  
        verify(con, cur, currentOrder)
```

This function takes the user input `var`. If it is 8 digits long and `var.isnumeric` function is True (the variable is numerical), it calls `findStock` function, carrying `con` and `cur` (variables for SQL database connection), `currentOrder`, and `var`. Else, it prints an error message and returns to `verify()` function.

Objective 2: Connect to a SQL database and run a query

```
con = lite.connect('dbuse.db')  
cur = con.cursor()
```

This snippet uses the `sqlite3` library to connect to the database file `dbuse.db`.

```
def findStock(con, cur, currentOrder, var):  
    cur.execute('SELECT * FROM Inventory WHERE GTIN = ?', (var,))  
    results = cur.fetchall()      ## collects results from SQL  
    con.commit()
```

The `findStock()` function executes the SQL query to find the record in `Inventory` where GTIN is equal to `var`. It then sets the variable `results` to the results of this query.

Objective 3: Collect and display the results

```
for product in results:  
    if product[2] == "":  
        sizeName = "  
    elif product[2] == 'Small' or product[2] == 'Medium' or product[2] == 'Large':  
        sizeName = product[2]  
    else:  
        sizeNameRaw = product[2], 'ml'  
        sizeName = "".join(sizeNameRaw)  
    print(' Name: ', sizeName, product[1], '\n Price: ', product[3], '\n Stock Available: ', product[4])
```

This section of code formats the product name: If the product has no size, it displays the name. If it has a small/medium/large quantity, it appends that to the end. If it has a ml volume, it appends that to the front. It then prints the name of the product as product name, volume, price, stock available.

Objective 4: Update the database with the customer's order

```
def enterOrder(sizeName, product, var, results, currentOrder, cur, con):
    QtyToOrder = input('-----\nEnter Quantity to order:\n>')
    if QtyToOrder.isnumeric() == False:
        print('Enter a valid Number')
        enterOrder(sizeName, product, var, results, currentOrder, cur, con)
    elif int(QtyToOrder) > int(product[4]):
        print('Error: Not enough stock. Please order', product[4], 'or less')
        enterOrder(sizeName, product, var, results, currentOrder, con, cur)
    elif int(QtyToOrder) < 1:
        print('You can't order less than 1. Try again')
        enterOrder(sizeName, product, var, results, currentOrder, con, cur)
```

This section of `enterOrder()` function ensures that the customer can only order more than 1, and less than or equal the number of stock available.

```
else:
    print('Adding to order...')
    NewStockAvab = 0
    costOfOrder = float(product[3])*int(QtyToOrder)
    currentOrderAddRaw = str(QtyToOrder), ' x ', str(sizeName), ' ', str(product[1]), ' (GTIN: ', str(product[0]), ') @ £', str(product[3]), ' = £', str(costOfOrder)
    currentOrderAdd = ''.join(currentOrderAddRaw)
    print('Added to order!')
    print('Updating Stock Levels...')
    QtyInt = int(QtyToOrder)
    NewStockAvab = str((int(product[4])) - QtyInt)
    sql = ("UPDATE INVENTORY SET STOCKAVAB = "+NewStockAvab+" WHERE GTIN LIKE '"+product[0]+"")
```

This section creates a verbose receipt, and makes an SQL query to update the database with the new stock levels

Objective 5: Print a receipt

```
currentOrder.append(currentOrderAdd)
again = input('Order another item? [Y/N]:\n>')
if again == 'Y' or again == 'y':
    verify(con, cur, currentOrder)
if again == 'N' or again == 'n':
    print('Order finished!')
    print('Receipt:')
    for order in currentOrder:
        print(order)
```

Objective 6: Cope with SQL errors

```
except:
    print('Error: Inventory Update failed to commit.')
    currentOrder.append(currentOrderAdd+' [CANCELLED]')
    print("\n\nPLEASE ENTER \'I00\' TO ESCAPE ERROR\n\n")
    enterOrder(sizeName, product, var, results, currentOrder, cur, con)
```

If the user enters an invalid number, the `sqlite3` library may get confused, and may enter an error loop. To break this loop, the user has to force the library to send another invalid request to the database.

5.3 Task 3

Objective 1: Scan a database and find stock to order

```
def findStock(con, cur, stockOrder):  
    sql1 = 'SELECT * FROM Inventory WHERE StockAvab != TargetStock'  
    cur.execute(sql1)  
    results = cur.fetchall()  
    orderList = []
```

The `findStock()` function sends a SQL query to the database to find any record where the stock level does not equal the target stock level. It then sets the variable `results` to these results.

Objective 2: Create a receipt of the order

```
print('Low stock! Order', toOrder, 'x', product[1], 'more to fill stock')  
if product[2] == '':  
    sizeName = ''  
elif product[2] == 'Small' or product[2] == 'Medium' or product[2] == 'Large':  
    sizeName = product[2]  
else:  
    sizeNameRaw = product[2], 'ml'  
    sizeName = ''.join(sizeNameRaw)  
stockOrderAddRaw = str(toOrder), ' x ', str(sizeName), ' ', str(product[1]), ' (GTIN: ', str(product[0]), ' )'  
stockOrderAdd = ''.join(stockOrderAddRaw)  
stockOrder.append(stockOrderAdd)  
orderList.append(product[0])
```

This section is similar to Task 2 Objective 4, and formats a verbose name of the product, and adds it to the receipt list.

Objective 3: Update the database with the updated stock level

```
def updateStock(con, cur, stockOrder, toOrder, orderList):  
    ##print(orderList)  
    print('Updating Database...')  
    for productUpdate in orderList:  
        sql2 = "UPDATE INVENTORY SET STOCKAVAB = TARGETSTOCK WHERE GTIN LIKE '"+productUpdate+"'"  
        try:  
            cur.execute(sql2)  
            con.commit()  
            print('Complete #', productUpdate)  
        except:  
            print('An exception occurred.')
```

This function connects to the SQL database and updates the new stock levels. If it fails, it returns to the last function.

Objective 4: Cope with any SQL errors

```
except:  
    print('An exception occurred.')
```

There should not be any SQL errors, as the user does not input any information that is used directly in the SQL query. This also protects against SQL injection.

6 Testing

6.1 Task 1

Test 1: Input strings of incorrect length. If rejected, it passes

```
Enter the 8 digit GTIN number
: 12345
Error: Only 8 numbers are allowed. Try again
Enter the 8 digit GTIN number
: |
```

```
Enter the 8 digit GTIN number
: 1234567890
Error: Only 8 numbers are allowed. Try again
Enter the 8 digit GTIN number
: |
```

These show that the length verification is working.

Test 2: Input strings of letters. If rejected, it passes

```
Enter the 8 digit GTIN number
: abcdefg
Error: Only 8 numbers are allowed. Try again
Enter the 8 digit GTIN number
:
```

This shows that the numerical verification is working

Test 3: Get program to run a valid input. Print out totals at each stage, and check them manually. If they are the same, it passes

```
Enter the 8 digit GTIN number
: 13245627
total: 3
total: 6
total: 12
total: 16
total: 31
total: 37
total: 43
rounded: 50
13245627 is a Valid Number
```

$$1 \times 3 = 3$$

$$3 + 3 \times 1 = 6$$

$$6 + 2 \times 3 = 12$$

$$12 + 4 \times 1 = 16$$

$$16 + 5 \times 3 = 31$$

$$31 + 6 \times 1 = 37$$

$$37 + 2 \times 3 = 43$$

$$\text{Highest 10 of } 43 = 50$$

$$50 - 43 = 7$$

$$\text{Checkdig} = 7$$

Test 4: Run the program with a GTIN number taken from a product. If it correctly calculated and verified, it passes



Enter the 8 digit GTIN number
:01412871
01412871 is a Valid Number

This shows that my program works in a real world situation, and can easily verify or calculate real GTIN codes.

6.2 Task 2

Test 1: Input strings of incorrect length. If rejected, it passes

| | |
|--|---|
| Enter GTIN for the product you wish to purchase: > 12345 Enter a 8 digit number Enter GTIN for the product you wish to purchase: > | Enter GTIN for the product you wish to purchase: > 1234567890 Enter a 8 digit number Enter GTIN for the product you wish to purchase: > |
|--|---|

From these, you can see that my length verification is working.

Test 2: Input strings of letters. If rejected, it passes

```
> abcdefg
Enter a 8 digit number
Enter GTIN for the product you wish to purchase:
>
```

From this, you can see that my numerical verification is working.

Test 3: Input a valid string to search. If product found, it passes

```
Enter GTIN for the product you wish to purchase:
> 11440529
Product Found!
Name: 100ml Red Paint
Price: 1.99
Stock Available: 50
-----
Enter Quantity to order:
>
```

The program successfully located a valid product

Test 4: Manually check that the program has displayed the correct stock level and information. If it has, it passes.

| | GTIN | ProductName | Size | Price | StockAvab | TargetStock |
|---|----------|-------------|--------|--------|-----------|-------------|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 11440529 | Red Paint | 100 | 1.99 | 50 | 50 |

The program has displayed the correct information for product #11440529

Test 5: Order a quantity of the product. If the program updates the stock levels, it passes.

| | | | | |
|--|--|-----------|--------|----|
| Enter Quantity to order: >5 Adding to order... Added to order! Updating Stock Levels... Updated | <table><tr><td>StockAvab</td></tr><tr><td>Filter</td></tr><tr><td>45</td></tr></table> | StockAvab | Filter | 45 |
| StockAvab | | | | |
| Filter | | | | |
| 45 | | | | |

The program successfully ordered 5 of #11440529 and updated the database

Test 6: Complete a full order. If the program displays a receipt with the correct values, it passes

```
Order finished!
Receipt:
5 x 100ml Red Paint (GTIN: 11440529) @ £1.99 = £9.95
6 x 100ml Blue Paint (GTIN: 11509493) @ £1.99 = £11.94
```

The program successfully ordered the products, printed a receipt, and calculated the correct cost.

Test 7: Provide the program with invalid values (such as ordering negative values). If the program rejects these, it passes.

| | | |
|---|--|--|
| Enter Quantity to order: >-3 Enter a valid Number ----- Enter Quantity to order: > | Enter Quantity to order: >0 You can't order less than 1. Try again ----- Enter Quantity to order: > | Enter Quantity to order: >100 Error: Not enough stock. Please order 44 or less ----- Enter Quantity to order: > |
|---|--|--|

This shows that the program rejects negative and 0 quantities, as well as quantities above the stock level.

6.3 Task 3

Test 1: Edit database for reduced stock. If reduced stock is identified, it passes

| | GTIN | ProductName | Size | Price | StockAvab | TargetStock |
|---|----------|-------------|--------|--------|-----------|-------------|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 11440529 | Red Paint | 100 | 1.99 | 45 | 50 |

Low stock! Order 5 x Red Paint more to fill stock
Added to order form

This shows that the program can correctly identify a reduced stock

Test 2: Edit database for reduced stock. If a human-readable receipt is produced, it passes

| | | | | | | |
|---|----------|------------|-----|------|----|----|
| 1 | 11440529 | Red Paint | 100 | 1.99 | 45 | 50 |
| 2 | 11509493 | Blue Paint | 100 | 1.99 | 45 | 50 |

Low stock! Order 5 x Red Paint more to fill stock
Added to order form
Low stock! Order 5 x Blue Paint more to fill stock
Added to order form

~~~~~Stock Order Form~~~~~  
5 x 100ml Red Paint (GTIN: 11440529)  
5 x 100ml Blue Paint (GTIN: 11509493)  
Is the order complete? [Y/N]  
>

This shows that the program can correctly identify two reduced stock levels, and append them to a verbose receipt

**Test 3: Complete order for reduced stock. If the database updated, it passes.**

Is the order complete? [Y/N]  
>y  
Updating Database...  
Complete # 11440529  
Complete # 11509493

| StockAvab |
|-----------|
| Filter    |
| 50        |
| 50        |

This shows that the program can update the database with the new stock level

**Test 4: If the program handles SQL errors, it passes**

| StockAvab |
|-----------|
| Filter    |
| 50        |
| 50        |

The program does not allow the user to order any stock if all stock levels are up to date

## 7 Final Program Code

### 7.1 Task 1

```
import math
import sys
print('GCSE Controlled Assessment A453\nThomas Bass 4869\nTask 1')
def start():
    ask = input('Press [c] to calculate the 8th digit from 7\n'
               'Press [v] to verify an 8 digit GTIN Number \n')
    if ask == 'c' or ask == 'C':
        length = 7
    elif ask == 'v' or ask == 'V':
        length = 8
    else:
        print('Error: Please enter either \'c\' or \'v\' ')
        start()
    check(length)
def check(length):
    print('Enter the', length, 'digit GTIN number')
    gtin = input(': ')
    if len(gtin) == length and gtin.isnumeric() == True:
        total = 0
        for counter in range(0, 7, 2):
            total = total + ((int(gtin[counter]))*3)
        if counter == 6:
            checkdig = int(gtin[length-1])
            rounded = (int(math.ceil(total / 10.0)) * 10)
            result = (rounded - total)
            if length == 7:
                print('Final Check Digit = ', result)
                print('Whole GTIN-8 Number = ', gtin,result)
                park()
            else:
                if checkdig == result: print(gtin, 'is a Valid Number')
                else: print(gtin, 'is an Invalid Number')
                park()
        else:
            total = total + ((int(gtin[counter+1]))*1)
    else:
        print('Error: Only', length, 'numbers are allowed. Try again ')
        check(length)
def park():
    again = input('Do you want to calculate or verify another number?')
    '\n[n] No [y] Yes: ')
    if again == 'n' or again == 'N':
        sys.exit()
    elif again == 'y' or again == 'Y':
        start()
start()

## Imports Math and Sys librarys

## Defines 'start'

## Ask the user if they want to verify or calculate
## If user chooses to calculate
## 'length' is 7
## If user chooses to verify
## 'length' is 8
## else
## Error message
## Return to 'start'
## Call 'check' function
## Define 'check' function
## Ask the user to input the GTIN
## Input 'gtin'
## If the length of 'gtin' is equal to 'length' and 'gtin' is numeric
## 'total' is 0
## For 7 iterations, stepping 'counter' by 2
## 'total' is 'total' plus 'gtin' position 'counter' times 3
## If 'counter' is 6
## 'checkdig' is 'gtin' position 'length' -1
## 'rounded' is ceil of 'total' /10.0 * 10 (rounded up to multiple of 10)
## 'result' is 'rounded' - 'total'
## If 'length' is 7
## Print 'result'
## Print 'gtin'+result'
## Call 'park' function
## Else
## If 'checkdig' equals 'result': GTIN is valid
## Else: GTIN is invalid
## Call 'park' function
## Else
## 'total' is 'total' plus 'gtin' position 'counter'
## Else
## Print error message
## Call 'check' function
## Define 'park' function

## Ask the user to go again
## If No:
## Close program
## If Yes:
## Call 'start' function
## Call 'start' function
```

## 7.2 Task 2

```

__authors__ = ['Thomas Bass']
## Candidate Number 4869 | Centre Number 52423
## TASK 2
import sqlite3 as lite
import random
import math

currentOrder = []
con = lite.connect('dbuse.db')
cur = con.cursor()

def verify(con, cur, currentOrder):
    var = input('Enter GTIN for the product you wish to purchase:\n> ')
    if len(var) == 8 and var.isnumeric() == True:
        findStock(con, cur, currentOrder, var)
    else:
        print('Enter a 8 digit number')
        verify(con, cur, currentOrder)

def findStock(con, cur, currentOrder, var):
    cur.execute('SELECT * FROM Inventory WHERE GTIN = ?', (var,))
    results = cur.fetchall()
    con.commit()
    if results == []:
        print('No product found. Please try again')
        verify(con, cur, currentOrder)
    print('Product Found!')
    for product in results:
        if product[2] == '':
            sizeName = ''
        elif product[2] == 'Small' or product[2] == 'Medium' or product[2] == 'Large':
            sizeName = product[2]
        else:
            sizeNameRaw = product[2], 'ml'
            sizeName = "".join(sizeNameRaw)
        print(' Name: ', sizeName, product[1], '\n Price: ', product[3], '\n Stock Available: ', product[4])
        enterOrder(sizeName, product, var, results, currentOrder, cur, con)

def enterOrder(sizeName, product, var, results, currentOrder, cur, con):
    QtyToOrder = input('-----\nEnter Quantity to order:\n> ')
    if QtyToOrder.isnumeric() == False:
        print('Enter a valid Number')
        enterOrder(sizeName, product, var, results, currentOrder, cur, con)
    elif int(QtyToOrder) > int(product[4]):
        print('Error: Not enough stock. Please order', product[4], 'or less')
        enterOrder(sizeName, product, var, results, currentOrder, cur, con)
    elif int(QtyToOrder) < 1:
        print('You can\'t order less than 1. Try again')
        enterOrder(sizeName, product, var, results, currentOrder, cur, con)
    else:
        print('Adding to order...')
        NewStockAvab = 0
        costOfOrder = float(product[3])*int(QtyToOrder)
        currentOrderAddRaw = str(QtyToOrder), ' x ', str(sizeName), ' ', str(product[1]), ' (GTIN: ', str(product[0]), ') @ £', ## currentOrderAddRaw
        str(product[3]), ' = £', str(costOfOrder)
        currentOrderAdd = "".join(currentOrderAddRaw)
        print('Added to order!')
        print('Updating Stock Levels...')
        QtyInt = int(QtyToOrder)
        NewStockAvab = str((int(product[4]) - QtyInt)
        sql = ("UPDATE INVENTORY SET STOCKAVAB = '"+NewStockAvab+"' WHERE GTIN LIKE '"+product[0]+"") ## SQL query
        try:
            cur.execute(sql)
            con.commit()
            print('Updated')
        except:
            print('Error: Inventory Update failed to commit.')
            currentOrder.append(currentOrderAdd+' [CANCELLED]')
            print('\n\nPLEASE ENTER \'100\' TO ESCAPE ERROR\n\n')
            enterOrder(sizeName, product, var, results, currentOrder, cur, con)
            currentOrder.append(currentOrderAdd)
            again = input('Order another item? [Y/N]:\n> ')
            if again == 'Y' or again == 'y':
                verify(con, cur, currentOrder)
            if again == 'N' or again == 'n':
                print('Order finished!')
                print('Receipt:')
                for order in currentOrder:
                    print(order)

def ask(cur, con, currentOrder):
    askQuery = input('Press [s] to search for a product\'s GTIN number:\n> ')
    if askQuery == 's' or askQuery == 'S':
        verify(con, cur, currentOrder)
    else:
        print('Invalid Choice')
        ask(cur, con, currentOrder)

ask(cur, con, currentOrder)

```

## 7.3 Task 3

```

__authors__ = ['Thomas Bass']
## TASK 3 ##
import sqlite3 as lite
import random
import math
import sys
stockOrder = []
con = lite.connect('dbase.db')
cur = con.cursor()
def findStock(con, cur, stockOrder):
    sql1 = 'SELECT * FROM Inventory WHERE StockAvab != TargetStock'
    cur.execute(sql1)
    results = cur.fetchall()
    orderList = []
    if str(results):
        print('Stock is all up to date!')
        park()
    for product in results:
        toOrder = int(int(product[5]) - int(product[4]))
        print('Low stock! Order', toOrder, 'x', product[1], 'more to fill stock')
        if product[2] == '':
            sizeName = ''
        elif product[2] == 'Small' or product[2] == 'Medium' or product[2] == 'Large':
            sizeName = product[2]
        else:
            sizeNameRaw = product[2], 'ml'
            sizeName = ''.join(sizeNameRaw)
            stockOrderAddRaw = (str(toOrder), 'x', str(sizeName), ' ', str(product[1]), ' (GTIN: ', str(product[0]), ')')
            stockOrderAdd = ''.join(stockOrderAddRaw)
            orderList.append(stockOrderAdd)
            orderList.append(product[0])
        print('      Added to order form')
    print('\n-----Stock Order Form-----')
    for i in stockOrder:
        print(i)
    updateContinue = input('Is the order complete? [Y/N]\n>')
    if updateContinue == 'Y' or 'y':
        updateStock(con, cur, stockOrder, toOrder, orderList)
    else:
        findStock(con, cur, stockOrder)
def updateStock(con, cur, stockOrder, toOrder, orderList):
    print(orderList)
    print('Updating Database...')
    for productUpdate in orderList:
        sql2 = "UPDATE INVENTORY SET STOCKAVAB = TARGETSTOCK WHERE GTIN LIKE '"+productUpdate+"'"
        try:
            cur.execute(sql2)
            con.commit()
            print('Complete #', productUpdate)
        except:
            print('An exception occured.')
            findStock(con, cur, stockOrder)
def park():
    sys.exit()
ask = input('Press [s] to scan inventory stock levels\n> ')
if ask == 's' or ask == 'S':
    findStock(con, cur, stockOrder)
else:
    sys.exit()

```

```

## Imports libraries

## connects to .db
## connects to SQL
## Define findStock
## selects low stock from DBase
## executes
## collects
## Define orderList as array
## If results are a string
## Print stock up to date
## Call park
## For results loop
## math for how many to order
## message
## If product[2] is empty
## SizeName = ''
## Elif
## SizeName = product[2]
## Else
## sizeNameRaw = product[2] + ml
## sizeName = join sizeNameRaw ''
## creates order
## Join stockOrderAdd ''
## Append stockOrderAdd to stockOrder
## Append product[0] orderList
## Print added to order
## Print stock order form
## Print stockOrder
## prints order
## input updateContinue
## If updateContinue = 'y' 'Y'
## Call updateStock
## Else
## Call findStock

## Define updateStock
## Print order list
## Print updating database
## Loop in orderList
## SQL command
## try
## Execute SQL command
## Commit .db-journal
## Print completed number
## Exception
## Print exception error
## Call findStock

## Define stock
## Sys exit

## Ask input
## If ask = 's'

```

```
## Call findStock
```