
GCSE

Computer Science

Scenario 3 – Traditional Application

Landscape Gardening

Controlled Assessment Exemplar Solution

4512

Version: 1.0

Exemplar solution

Landscape Gardening

AQA GCSE Computer Science

Controlled Assessment

Design of Solution

Explanation of the Problem

I need to create a solution to the following problem:

A gardening company creates gardens using the following materials:

- Lawn
- Concrete Patio
- Wooden Decking
- Ponds (rectangular)
- Water Features
- Garden Lights

Each of these materials has an associated cost (for lawn, concrete, decking and ponds this is a cost per square metre, for water features and lights it is the cost per unit). Each one also has a set time to install (again, the first four is timed per square metre and the last two have a set time per unit). Labour is charged at a cost per hour.

The company wants to use these costs to work out quotes for clients and also to save these quotes for later use. They have a specific way they would like the quotes to appear.

They want the costs of the materials to be stored in an external file to allow the user to change them when needed.

The first task includes working out subtotals for the quote and then the final cost. Parts 1-4 are very similar (input required is the length and the width, a constant is the cost per square metre and the output is the total cost), and parts 5-6 are similar (input is number of units, constant is cost per unit and output is the total cost). The total labour cost is the number of hours required (calculated by the number of square metres for each of 1-4 multiplied by the time in minutes and the number of units for 5-6 multiplied by the time in minutes). The lengths and widths are positive real numbers and the number of units is a positive integer (all can be 0).

The second task requires the system to save quotes, this will obviously require some sort of database. The database has two tables: the client and the details of the quote (because a client can have more than one quote but a quote belongs to only one client). The client will need a primary key to uniquely identify them, as will the quote.

The third task asks for the data on the cost of raw materials to be stored in an external file. This could be a database but I assume that I should use a text file (this could be so that someone could change the data without having access to the quotes). This will require a format such as CSV for the file and the program will have to read this information every time it is started. There are 6 different raw materials given earlier.

The fourth task is about preparing monthly reports on the total amount and cost of all the raw materials used. The user would type in a number (the example given is both "June" or 6 for June)

and they would see all of the material used in June. It isn't clear if this is for every June or just for the most recent one so I will make it just for the month that has passed most recently (i.e. if it is May now (2012) and month 5 was searched for then the results from May 2011 would be shown). This will involve the quotes in the database stored with two dates: one for when the quote was given (this will help with search in task 2) and one for when the work is to be done (this is for the monthly material report). The first date should be required but the second date should only be put in if it is known (if it isn't then it should be null), so jobs with no second date set should not be included in the material monthly report. There is a set format for the monthly report data to be presented.

Overview Plan

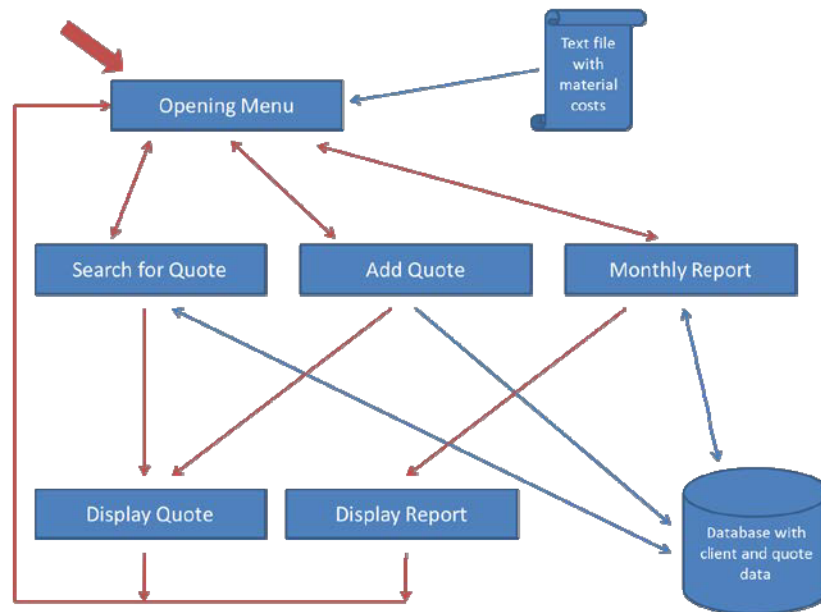
The **database** needs to have the following information:

client		
clientID	Integer (primary key)	Required
clientName	Text (up to 100 characters)	Required

quote		
quoteID	Integer (primary key)	Required
clientID	Integer (foreign key)	Required
dateOfQuote	Date	Required
dateOfJob	Date	Not Required
lawnLength	Float	Required
lawnWidth	Float	Required
lawnCostPerM	Float	Required
patioLength	Float	Required
patioWidth	Float	Required
patioCostPerM	Float	Required
deckingLength	Float	Required
deckingWidth	Float	Required
deckingCostPerM	Float	Required
pondLength	Float	Required
pondWidth	Float	Required
pondCostPerM	Float	Required
featureNumber	Integer	Required
featuresCostEa	Float	Required
lightingNumber	Integer	Required
lightingCostEa	Float	Required

This means that if the cost of the material changes then the cost of the quote will not change too.

The **system** will work together like this:



Where the red arrows show how you can get between the various parts of the program and the blue arrows showing how the data flows.

The **opening menu** needs to have four choices:

1. Search for a quote
2. Add quote
3. Monthly report
4. Exit

The brief doesn't say how the quotes are to be searched so for my system I am going to allow searching by client name and searching by date of quotation as well as the quote reference number (clientName, dateOfQuote and quoteID in the quote table). Because the cost of the raw material can change I am going to recalculate the cost of the work every time using the price of raw materials stored in the database. I could use the database to store all of the information that has been calculated but I want to leave it like this so the quote can be easily changed (eg the length of the lawn can be changed but the costs and area won't need to be).

The **search for quote** requires the user to enter a quote number.

The information returned will be a combination of the working cost and labour cost tables seen in the specimen.

The **add quote menu** will need the following options:

1. Existing client
2. Add new client

If add new client is selected then the user should be prompted to enter their name and will be returned to this menu, if existing client is selected then the user will be presented with a list of all the clients in alphabetical order and will need to choose the correct number, e.g.:

1. Adams, 000223

2. Babledy, 000144
3. Carnock, 001003
4. ...

This isn't suitable if the program will be used for hundreds of clients but it will work on a small scale.

Once the user has selected their client they will need to enter the following information, prompted one after the other (if nothing is entered then the user will be prompted before a zero will be entered). The information to be entered will be:

1. Lawn Length
2. Lawn Width
3. Patio Length
4. Patio Width
5. Decking Length
6. Decking Width
7. Pond Length
8. Pond Width
9. Number of Water Features
10. Number of Lights
11. Date of job (can be left blank)

Entries 1-8 will be validated as positive floats and entries 9 and 10 will be validated as positive integers.

As soon as the quote has been entered (i.e. the 10th item is finished) the quote will be processed and will display on the console. The opening menu will be asked again.

The **monthly report menu** will list the 12 months of the year before the current month (the brief said that you can enter either the name of the month or the number of the month). When the user chooses the number 1-12 (validated as only one of those numbers) then the report for that month is generated.

Proposed Solution

The overall solution will be as follows:

```
# check to see if the database is created (first time only)
# SQL will check if tables already exist
RUNFIRSTTIME()

# get the materials cost from the external file
# and store them in a dictionary
costs ← GETCOSTSFROMFILE("materials.txt")
labourCostPerHour ← 16.49 # a constant

# loop until the user chooses EXIT
WHILE true
    DISPLAYMENU()           # display the main menu
    menuChoice ← INPUT      # user input will be validated (1-4)
```

```

        CASE menuChoice OF
            1: SEARCHQUOTE()
            2: NEWQUOTE()
            3: MONTHREPORT()
            4: EXIT()
        ENDCASE
    ENDWHILE

PROCEDURE DISPLAYMENU()
    # displays the four choices shown earlier
ENDPROCEDURE

PROCEDURE SEARCHQUOTE()
    OUTPUT "Enter quote number"
    quoteNumber ← INPUT # user input will be prompted and validated
    quoteData ← GETQUOTEFROMNUMBER(quoteNumber) # quoteData is a dictionary
    DISPLAYQUOTE(quoteData) # procedure to display to console
ENDPROCEDURE

# search the database and get the quote details from the quote number
# results are stored in a dictionary datastructure
FUNCTION GETQUOTEFROMNUMBER(quoteNumber)
    OPENDATABASE(db, user, pass) # open the database connection
    details ← QUERYDB(query) # SQL query (see later)
    IF details ≠ NULL
    THEN
        RETURN details
    ELSE
        RETURN none # return 'none' if search failed
    ENDFUNCTION

# process and print the quote information
PROCEDURE DISPLAYQUOTE(quoteData)
    totalMaterialCost ← 0
    OUTPUT "Material Costs"

    # this is for lawn ...
    OUTPUT "Lawn Costs"
    OUTPUT "length:"
    OUTPUT quoteData['lawnLength']
    OUTPUT "width:"
    OUTPUT quoteData['lawnWidth']
    # calculate total area
    # save it in a dictionary for use later (same for patio, pond and decking)
    runningTotals['lawnArea'] ← quoteData['lawnLength'] * quoteData['lawnWidth']
    OUTPUT "total area:"
    OUTPUT runningTotals['lawnArea']
    OUTPUT "cost per square metre:"
    OUTPUT costs['lawnCostPerMetre']
    OUTPUT "total cost:"
    # calculate total cost using area and the costs dictionary
    cost ← runningTotals['lawnArea'] * costs['lawnCostPerMetre']
    OUTPUT cost
    totalMaterialCost ← totalMaterialCost + cost
    # ... patio, decking and pond is identical except for titles

    # this is for the water features ...
    OUTPUT "Water Features"
    OUTPUT "number:"
    OUTPUT quoteData['featureNumber']

```

```

OUTPUT "cost per feature:"
OUTPUT costs['featureCost']
cost ← costs['featureCost'] * quoteData['number']
OUTPUT cost
totalMaterialCost ← totalMaterialCost + cost
# ... lighting is identical except for titles

OUTPUT "total working costs:"
OUTPUT totalMaterialCost

# next work out the labour costs
totalLabourMinutes ← 0
OUTPUT "Labour Costs"

# this is for the lawn...
OUTPUT "Minutes per metre square:"
OUTPUT costs['lawnMinutes']
OUTPUT "total area"
OUTPUT runningTotals['lawnArea']
OUTPUT "total minutes:"
minutes ← costs['lawnMinutes'] * runningTotals['lawnArea']
OUTPUT minutes
totalLabourMinutes ← totalLabourMinutes + minutes
# ...patio, decking and pond is identical except for titles

# this is for the water features...
OUTPUT "Minutes per water feature:"
OUTPUT costs['featuresMinutes']
OUTPUT "number purchased:"
OUTPUT quoteData['featureNumber']
OUTPUT "total minutes:"
minutes ← costs['featuresMinutes'] * quoteData['featureNumber']
OUTPUT minutes
totalLabourMinutes ← totalLabourMinutes + minutes
# ...lighting is identical except for labels

OUTPUT "Total Work (minutes):"
OUTPUT totalLabourMinutes
OUTPUT "Total Work (hours):"
hours ← totalLabourMinutes / 60
OUTPUT hours
OUTPUT "Cost of Labour (per hour)"
OUTPUT labourCostPerHour
OUTPUT "Total Labour Cost"
totalLabourCost ← totalLabourMinutes * labourCostPerHour
OUTPUT totalLabourCost

# finally output the grand totals
OUTPUT "Total working costs:"
OUTPUT totalMaterialCost
OUTPUT "Total labour costs:"
OUTPUT totalLabourCost
OUTPUT "Total to Pay by Customer:"
OUTPUT totalMaterialCost + totalLabourCost
ENDPROCEDURE

PROCEDURE NEWQUOTE()
    OPENDATABASE(db, user, pass)

```



```

# displays the two choices shown earlier
DISPLAYNEWQUOTEMENU()
menuChoice ← INPUT          # user input will be validated (1-2)
IF menuChoice = 1           # user chooses existing client
THEN
    clientNumber ← DISPLAYCLIENTS() # display the whole list
ELSE
    # user adds new client
    OUTPUT "add new client name"
    clientName ← INPUT
    clientNumber ← QUERYDB(query) # SQL query to save new client
ENDIF

# now get all of the information about the garden plan
OUTPUT "Enter lawn length:"
# save this as a dictionary that can be used by the display quote proc
quoteData['lawnLength'] ← INPUT # validate as ≤ 0
OUTPUT "Enter lawn width:"
quoteData['lawnWidth'] ← INPUT # validate as ≤ 0
# ...this follows in the same way for all the remaining choices

# ask when the job will be done but can be blank
OUTPUT "Enter when job to be done (can leave blank)"
jobDone ← INPUT

# save all the information to the database
IF jobDone
THEN
    QUERYDB(query) # SQL query including jobDone
ELSE
    QUERYDB(query) # same but without jobDone
ENDIF

# display the quote information
DISPLAYQUOTE(quoteData)
ENDPROCEDURE

PROCEDURE DISPLAYCLIENTS()
# get all the clients names and clientIDs
# save the information in a two dimensional array
OPENDATABASE(db, user, pass)
clients ← QUERYDB(query) # SQL query to get all clientIDs and clientNames
FOR i ← 1 TO LEN(clients)
    OUTPUT i
    OUTPUT clients[i][2] # the client name
    OUTPUT clients[i][1] # the client ID
ENDFOR
OUTPUT "choose a number"
choice ← INPUT # validate this as 1 - LEN(clients)
RETURN clients[choice][1]
ENDPROCEDURE

PROCEDURE MONTHREPORT()
currentMonth ← GETCURRENTMONTH() # use a built in function for this
# display all the months like 1. January, 2. February,...
# DISPLAYMONTHS
monthChosen ← INPUT
# work out if it will be last year's month or this year's month
IF monthChosen ≥ currentMonth
THEN

```

```

        year = CURRENTYEAR() - 1 # use a built in function
ELSE
        year = CURRENTYEAR()
ENDIF

# get the monthly totals
OPENDATABASE(db, user, pass)
totalQuotes ← QUERYDB(query) # query with of all the quotes in month, year
# keep running totals on all materials
totalLawn ← 0
totalPatio ← 0
totalDecking ← 0
totalPond ← 0
totalFeatures ← 0
totalLighting ← 0
# iterate over the query results
FOR i ← 1 TO LEN(totalQuotes)
    totalLawn ← totalLawn +
(totalQuotes[i]['lawnWidth']*totalQuotes[i]['lawnLength'])
    # ...same for patio, decking and ponds
    totalFeatures ← totalFeatures + totalQuotes[i]['features']
    # ...same for lighting
ENDFOR

# keep the total cost
totalCost ← 0

# display the results
OUTPUT "Lawn total metres:"
OUTPUT totalLawn
OUTPUT "Lawn total monthly value:"
cost ← totalLawn * costs['lawn']
OUTPUT cost
totalCost ← totalCost + cost
# ...exactly the same for the others apart from the labels

# display total costs
OUTPUT "Total Monthly Value:"
OUTPUT totalCost
ENDPROCEDURE

```

The external text file with the material costs will be called **materials.txt** and will be a CSV file with headers of **material** and the **price per m²**, the initial file will look like this:

```

lawn,15.50
patio,20.99
decking,15.75
pond,20.00
feature,150.00
lights,5.00

```

The SQL queries are:

```

SELECT * FROM quote WHERE quote.quoteID = ?

SELECT * FROM quote WHERE quote.quoteDate = ?

```

```

SELECT * FROM quote WHERE quote.clientID = ?

INSERT INTO client(client.name) VALUES ?

INSERT INTO quote(quote..., client.clientID) VALUES (?...,?)

SELECT client.clientID, client.clientName FROM client

SELECT quote.lawnLength, quote.lawnWidth, quote.patioLength... FROM quote
WHERE quote.dateOfQuote ≥ "01/month/year" AND quote.dateOfQuote <
"01/month+1/year"

```

Solution Development

Needs of the User

This is the program being used to do a variety of things (it is not complete but this can be seen in the testing section), the black text is the computer output and the highlighted red text is the user input. This shows the menus for the user and the calculations as well as showing data that is saved to and read from the database:

```
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
2
1. Add quote for existing client
2. Add quote for new client
2
Add new client name:
Bob Yeazley
Entering a quote for client 1338309698
Enter lawn length6
Enter lawn width4
Enter patio length0
Enter patio width0
Enter decking length0
Enter decking width0
Enter pond length3
Enter pond width2.5
Enter number of water features2
Enter number of lighting features5
Quote reference number: 1338309719
Material Costs

Lawn Costs
Length: 6.0
Width: 4.0
Total Area: 24.0
Cost per square metre: 15.5
Total Cost: 372.0
Patio Costs
Length: 0.0
Width: 0.0
Total Area: 0.0
Cost per square metre: 20.99
Total Cost: 0.0
Decking Costs
Length: 0.0
Width: 0.0
Total Area: 0.0
Cost per square metre: 15.75
Total Cost: 0.0
Pond Costs
Length: 3.0
Width: 2.5
Total Area: 7.5
Cost per square metre: 25.0
Total Cost: 187.5
```

Water Feature Costs
Number of water features: 2
Cost per feature: 150.0
Total Cost: 300.0
Lighting Costs
Number of lights: 5
Cost per light: 5.0
Total Cost: 25.0

Total Working Costs: 884.5

Labour Costs

Lawn labour costs
Minutes per square metre: 20.0
Total area: 24.0
Total minutes: 480.0
Patio labour costs
Minutes per square metre: 20.0
Total area: 0.0
Total minutes: 0.0
Decking labour costs
Minutes per square metre: 30.0
Total area: 0.0
Total minutes: 0.0
Pond labour costs
Minutes per square metre: 45.0
Total area: 7.5
Total minutes: 337.5
Water feature labour costs
Minutes per water feature: 60.0
Number Purchased 2
Total minutes: 120.0
Garden lighting labour costs
Minutes per water feature: 10.0
Number Purchased 5
Total minutes: 50.0

Total Work (minutes): 987.5
Total Work (hours): 16.4583333333
Cost of Labour (per hour): 16.49
Total Labour Cost 271.397916667

Total Working Costs: 884.5
Total Labour Costs: 271.397916667
Total to Pay by Customer: 1155.89791667

1. Search for a quote
2. Add quote
3. Monthly report
4. Exit

1

Enter the quote number 13323
Invalid quote number
Quote Number: 1338309515
Client Name: Andrew Zachary

Quote Number: 1338309719
Client Name: Bob Yeazley

Enter the quote number 1338309515

Material Costs

Lawn Costs

Length: 10.0

Width: 8.0

Total Area: 80.0

Cost per square metre: 15.5

Total Cost: 1240.0

Patio Costs

Length: 5.0

Width: 5.0

Total Area: 25.0

Cost per square metre: 20.99

Total Cost: 524.75

Decking Costs

Length: 0.0

Width: 0.0

Total Area: 0.0

Cost per square metre: 15.75

Total Cost: 0.0

Pond Costs

Length: 3.0

Width: 4.0

Total Area: 12.0

Cost per square metre: 25.0

Total Cost: 300.0

Water Feature Costs

Number of water features: 3

Cost per feature: 150.0

Total Cost: 450.0

Lighting Costs

Number of lights: 10

Cost per light: 5.0

Total Cost: 50.0

Total Working Costs: 2564.75

Labour Costs

Lawn labour costs

Minutes per square metre: 20.0

Total area: 80.0

Total minutes: 1600.0

Patio labour costs

Minutes per square metre: 20.0

Total area: 25.0

Total minutes: 500.0

Decking labour costs

Minutes per square metre: 30.0

Total area: 0.0

Total minutes: 0.0

Pond labour costs

Minutes per square metre: 45.0

Total area: 12.0

Total minutes: 540.0

Water feature labour costs

Minutes per water feature: 60.0

Number Purchased 3

Total minutes: 180.0

Garden lighting labour costs

Minutes per water feature: 10.0

Number Purchased 10
Total minutes: 100.0

Total Work (minutes): 2920.0
Total Work (hours): 48.6666666667
Cost of Labour (per hour): 16.49
Total Labour Cost 802.513333333

Total Working Costs: 2564.75
Total Labour Costs: 802.513333333
Total to Pay by Customer: 3367.26333333

1. Search for a quote
2. Add quote
3. Monthly report
4. Exit

3

11: Nov
10: Oct
12: Dec
1: Jan
3: Mar
2: Feb
5: May
4: Apr
7: Jul
6: Jun
9: Sep
8: Aug

Enter the number of a month (1-12) 5

Lawn total metres: 104.0
Lawn total monthly value: 1612.0
Patio total metres: 25.0
Patio total monthly value: 524.75
Decking total metres: 0.0
Decking total monthly value: 0.0
Pond total metres: 19.5
Pond total monthly value: 487.5
Water Feature total: 5.0
Water Feature total monthly value: 750.0
Lighting total: 15.0
Lighting total monthly value: 75.0
Total Monthly Value: 3449.25

1. Search for a quote
2. Add quote
3. Monthly report
4. Exit

4

system closing

Annotated Code

This is the complete code for my project, it is annotated with comments.

```
import sys
import csv
import sqlite3
import datetime
```

```

import time

# constants used in the program
LABOURPERHOUR = 16.49
DB = 'gardening.db'
MATERIALFILE = 'materials.txt'

# structure of the database
def runFirstTime():
    # use try-except to catch any database errors
    try:
        conn = sqlite3.connect(DB)
        c = conn.cursor()
        # create the client table with its two attributes
        # client is created first because clientID is a foreign key in the
quote table
        query = '''CREATE TABLE IF NOT EXISTS client (
                        clientID int, clientName text, primary
key(clientID))'''
        c.execute(query)
        conn.commit()
        # create the quote table with 24 attributes
        query = '''CREATE TABLE IF NOT EXISTS quote (
                                quoteID int,
                                clientID int,
                                dayOfQuote int,
                                monthOfQuote int,
                                yearOfQuote int,
                                dayOfJob int,
                                monthOfJob int,
                                yearOfJob int,
                                lawnLength real,
                                lawnWidth real,
                                lawnCostPerM real,
                                patioLength real,
                                patioWidth real,
                                patioCostPerM real,
                                deckingLength real,
                                deckingWidth real,
                                deckingCostPerM real,
                                pondLength real,
                                pondWidth real,
                                pondCostPerM real,
                                featureNumber int,
                                featuresCostEa real,
                                lightingNumber int,
                                lightingCostEa real,
                                primary key (quoteID),
                                foreign key (clientID) references
client(clientID))'''
        c.execute(query)
        conn.commit()
        c.close()
    except:
        pass

# read in the costs from the external file
def getCostsFromFile(filename):
    # save the results in a dictionary
    costs = {}
    try:

```



```

        # use the built in CSV reader
        fileReader = csv.reader(open(MATERIALFILE, 'rb'))
        # loop over the rows
        for row in fileReader:
            # the text becomes the key in the dictionary and the float is
the value
            costs[row[0]] = float(row[1])
        return costs
    except:
        print "make sure materials.txt exists and is in the correct format"
        sys.exit()

# procedure to display the four options
def displayMenu():
    print "1. Search for a quote"
    print "2. Add quote"
    print "3. Monthly report"
    print "4. Exit"

# if the search for quote option is chosen
def searchQuote():
    # get the quote number
    quoteNumber = getValidInt("Enter the quote number", 0)
    # check to see that menuChoice is one of the three allowed choices
    # result is a list of the quotes attributes
    result = getQuoteFromNumber(quoteNumber)
    # check a result has been returned
    if not result:
        print "Invalid quote number"
        showAllQuotes()
        quoteNumber = getValidInt("Enter the quote number", 0)
        result = getQuoteFromNumber(quoteNumber)
    # convert result into a dictionary with named keys
    quoteData = {}
    quoteData['lawnLength'] = result[0]
    quoteData['lawnWidth'] = result[1]
    quoteData['patioLength'] = result[3]
    quoteData['patioWidth'] = result[4]
    quoteData['deckingLength'] = result[6]
    quoteData['deckingWidth'] = result[7]
    quoteData['pondLength'] = result[9]
    quoteData['pondWidth'] = result[10]
    quoteData['featureNumber'] = result[12]
    quoteData['lightingNumber'] = result[14]
    # display the quote
    displayQuote(quoteData)

# get the quote from the quote number
def getQuoteFromNumber(quoteNumber):
    try:
        conn = sqlite3.connect(DB)
        c = conn.cursor()
        # SQL query to find all of the details based on the quote number
(ID)
        c.execute('''SELECT lawnLength, lawnWidth,
                        lawnCostPerM, patioLength, patioWidth, patioCostPerM,
                        deckingLength, deckingWidth, deckingCostPerM,
                        pondLength,
                        pondWidth, pondCostPerM, featureNumber, featuresCostEa,
                        lightingNumber, lightingCostEa FROM quote WHERE
quoteID=?''', (quoteNumber,))

```

```

        # one result is fetched and returned
        result = c.fetchone()
        return result
    except:
        # this is displayed if a database error has occured
        print "could not get result from database"

# procedure to display all the quote numbers with the clients
def showAllQuotes():
    try:
        conn = sqlite3.connect(DB)
        c = conn.cursor()
        # SQL query to find all of the quotes
        c.execute(''SELECT quote.quoteID, client.clientName FROM quote,
client
                WHERE quote.clientID = client.clientID'')
        # the results are fetched and returned
        result = c.fetchall()
        for quote in result:
            print "Quote Number:",
            print quote[0]
            print "Client Name:",
            print quote[1]
            print ""
    except:
        # this is displayed if a database error has occured
        print "could not get result from database"

# procedure that calculates all of the costs for a quote and displays it
def displayQuote(quoteData):
    # runningTotals keeps the costs of the materials in a dictionary
    runningTotals = {}
    # keeps a running total
    totalMaterialCost = 0.0

    print "Material Costs"
    print ""

    # displays relevant information about this material
    print "Lawn Costs"
    print "Length:", quoteData['lawnLength']
    print "Width:", quoteData['lawnWidth']
    # sets 'lawnArea' to be the length x width
    runningTotals['lawnArea'] = quoteData['lawnLength'] *
quoteData['lawnWidth']
    print "Total Area:", runningTotals['lawnArea']
    print "Cost per square metre:", costs['lawnCostPerMetre']
    print "Total Cost:",
    # sets cost to be the area x the cost per metre
    cost = runningTotals['lawnArea'] * costs['lawnCostPerMetre']
    print cost
    # cost is added to the total
    totalMaterialCost += cost

    # same as lawn costs
    print "Patio Costs"
    print "Length:", quoteData['patioLength']
    print "Width:", quoteData['patioWidth']
    runningTotals['patioArea'] = quoteData['patioLength'] *
quoteData['patioWidth']
    print "Total Area:", runningTotals['patioArea']

```

```

print "Cost per square metre:", costs['patioCostPerMetre']
print "Total Cost:",
cost = runningTotals['patioArea'] * costs['patioCostPerMetre']
print cost
totalMaterialCost += cost

# same as lawn costs
print "Decking Costs"
print "Length:", quoteData['deckingLength']
print "Width:", quoteData['deckingWidth']
runningTotals['deckingArea'] = quoteData['deckingLength'] *
quoteData['deckingWidth']
print "Total Area:", runningTotals['deckingArea']
print "Cost per square metre:", costs['deckingCostPerMetre']
print "Total Cost:",
cost = runningTotals['deckingArea'] * costs['deckingCostPerMetre']
print cost
totalMaterialCost += cost

# same as lawn costs
print "Pond Costs"
print "Length:", quoteData['pondLength'] print
"Width:", quoteData['pondWidth']
runningTotals['pondArea'] = quoteData['pondLength'] *
quoteData['pondWidth']
print "Total Area:", runningTotals['pondArea']
print "Cost per square metre:", costs['pondCostPerMetre']
print "Total Cost:",
cost = runningTotals['pondArea'] * costs['pondCostPerMetre']
print cost
totalMaterialCost += cost

# same as lawn costs except area is not worked out as just the number
is needed
print "Water Feature Costs"
print "Number of water features:", quoteData['featureNumber']
print "Cost per feature:", costs['featureCost']
cost = costs['featureCost'] * quoteData['featureNumber']
print "Total Cost:", cost
totalMaterialCost += cost

# same as water feature
print "Lighting Costs"
print "Number of lights:", quoteData['lightingNumber']
print "Cost per light:", costs['lightingCost']
cost = costs['lightingCost'] * quoteData['lightingNumber']
print "Total Cost:", cost
totalMaterialCost += cost

print ""
print "Total Working Costs:", totalMaterialCost
print ""

# total is kept for the number of minutes
totalLabourMinutes = 0.0
print "Labour Costs"
print ""

print "Lawn labour costs"
print "Minutes per square metre:", costs['lawnMinutes']
print "Total area:", runningTotals['lawnArea']

```

```

print "Total minutes:",
# minutes is the total time for the lawn (area x time per metre)
minutes = costs['lawnMinutes'] * runningTotals['lawnArea']
print minutes
# cost is added to labour total
totalLabourMinutes += minutes

# same as lawn minutes
print "Patio labour costs"
print "Minutes per square metre:", costs['patioMinutes']
print "Total area:", runningTotals['patioArea']
print "Total minutes:",
minutes = costs['patioMinutes'] * runningTotals['patioArea']
print minutes
totalLabourMinutes += minutes

# same as lawn minutes
print "Decking labour costs"
print "Minutes per square metre:", costs['deckingMinutes']
print "Total area:", runningTotals['deckingArea']
print "Total minutes:",
minutes = costs['deckingMinutes'] * runningTotals['deckingArea']
print minutes
totalLabourMinutes += minutes

# same as lawn minutes
print "Pond labour costs"
print "Minutes per square metre:", costs['pondMinutes']
print "Total area:", runningTotals['pondArea']
print "Total minutes:",
minutes = costs['pondMinutes'] * runningTotals['pondArea']
print minutes
totalLabourMinutes += minutes

# same as lawn minutes (except number not area)
print "Water feature labour costs"
print "Minutes per water feature:", costs['featureMinutes']
print "Number Purchased", quoteData['featureNumber']
print "Total minutes:",
minutes = costs['featureMinutes'] * quoteData['featureNumber']
print minutes
totalLabourMinutes += minutes

# same as lawn minutes (except number not area)
print "Garden lighting labour costs"
print "Minutes per water feature:", costs['lightingMinutes']
print "Number Purchased", quoteData['lightingNumber']
print "Total minutes:",
minutes = costs['lightingMinutes'] * quoteData['lightingNumber']
print minutes
totalLabourMinutes += minutes

print ""
print "Total Work (minutes):", totalLabourMinutes
# minutes converted to hours
hours = totalLabourMinutes / 60.0
print "Total Work (hours):", hours
print "Cost of Labour (per hour):", LABOURPERHOUR
print "Total Labour Cost",
# total labour cost is calculated
totalLabourCost = hours * LABOURPERHOUR

```

```

print totalLabourCost

# all the totals and the grand total are displayed
print ""
print "Total Working Costs:", totalMaterialCost
print "Total Labour Costs:", totalLabourCost
grandTotal = totalMaterialCost + totalLabourCost
print "Total to Pay by Customer:", grandTotal
print ""

# procedure if a new quote is added to the system
def newQuote():
    # choice is displayed
    print "1. Add quote for existing client"
    print "2. Add quote for new client"
    menuChoice = raw_input()
    # input is validated (either 1 or 2)
    while menuChoice not in ['1', '2']:
        print "Enter either 1 or 2"
        menuChoice = raw_input()
    # the clientNumber is got by either method
    if menuChoice == '1':
        clientNumber = displayClients()
    else:
        clientNumber = addClient()
    # check to see if the user choose a client
    if not clientNumber:
        print "No client was selected"
        return
    print "Entering a quote for client " + str(clientNumber)
    # enter all the quote data into a dictionary
    quoteData = {}
    # each entry is validated either as float or an int
    # with the prompt to the user and the minimum value
    quoteData['lawnLength'] = getValidFloat("Enter lawn length", 0)
    quoteData['lawnWidth'] = getValidFloat("Enter lawn width", 0)
    quoteData['patioLength'] = getValidFloat("Enter patio length", 0)
    quoteData['patioWidth'] = getValidFloat("Enter patio width", 0)
    quoteData['deckingLength'] = getValidFloat("Enter decking length", 0)
    quoteData['deckingWidth'] = getValidFloat("Enter decking width", 0)
    quoteData['pondLength'] = getValidFloat("Enter pond length", 0)
    quoteData['pondWidth'] = getValidFloat("Enter pond width", 0)
    quoteData['featureNumber'] = getValidInt("Enter number of water
features", 0)
    quoteData['lightingNumber'] = getValidInt("Enter number of lighting
features", 0)
    # this information is added to the database
    try:
        conn = sqlite3.connect(DB)
        c = conn.cursor()
        # todays date is got using this built in function
        today = datetime.date.today()
        # the day, month and year is found out from the date
        quoteDay = today.day
        quoteMonth = today.month
        quoteYear = today.year
        # the primary key is a time in milliseconds (will be unique)
        quoteID = int(time.time())
        # this is a tuple of all the data to be added
        entries = (quoteID, clientNumber, quoteDay, quoteMonth, quoteYear,
quoteData['lawnLength'], quoteData['lawnWidth'], costs['lawnCostPerMetre'],

```

```

        quoteData['patioLength'], quoteData['patioWidth'],
costs['patioCostPerMetre'],
        quoteData['deckingLength'], quoteData['deckingWidth'],
costs['deckingCostPerMetre'],
        quoteData['pondLength'], quoteData['pondWidth'],
costs['pondCostPerMetre'],
        quoteData['featureNumber'], costs['featureCost'],
quoteData['lightingNumber'], costs['lightingCost'])
    # this is the query to insert all of the data
    c.execute('''INSERT INTO quote(quoteID, clientID, dayOfQuote,
monthOfQuote, yearOfQuote, lawnLength, lawnWidth,
        lawnCostPerM, patioLength, patioWidth,
patioCostPerM, deckingLength,
        deckingWidth, deckingCostPerM, pondLength,
pondWidth, pondCostPerM,
        featureNumber, featuresCostEa, lightingNumber,
lightingCostEa) VALUES(
        ?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)''',
entries)
    conn.commit()
    c.close()
except:
    # an error message in case the data couldn't be saved
    print "quote could not be saved to database"
    print "Quote reference number:",
    print quoteID
    displayQuote(quoteData)

# validates user input based on type (float) and a minimum value
def getValidFloat(prompt, minValue):
    valid = False
    answer = raw_input(prompt)
    while not valid:
        try:
            # this will cause an error if it is the wrong type and
            # will be False if it is greater than the min value
            if float(answer) >= minValue:
                valid = True
        except:
            answer = raw_input("Enter a valid number")
    # returns the answer in the correct type
    return float(answer)

# same as getValidFloat only for integer
def getValidInt(prompt, minValue):
    valid = False
    answer = raw_input(prompt)
    while not valid:
        try:
            if int(answer) >= minValue:
                valid = True
        except:
            answer = raw_input("Enter a valid number")
    return int(answer)

# function to add a new client, returns the clientID
def addClient():
    print "Add new client name:"
    # get the client name from user input
    clientName = raw_input()
    # check they have entered something

```

```

while not clientName:
    clientName = raw_input("Add a name")
try:
    conn = sqlite3.connect(DB)
    c = conn.cursor()
    # get the current time in milliseconds to be the primary key (will
be unique)
    clientNumber = int(time.time())
    # SQL to add a new client
    c.execute('''INSERT INTO client VALUES(?,?)''', (clientNumber,
clientName))
    conn.commit()
    c.close()
    # return the primary key
    return clientNumber
except:
    # error message if database error
    print "could not save new client to database"
    return

# function for the user to choose a client (shows all clients)
def displayClients():
    try:
        conn = sqlite3.connect(DB)
        c = conn.cursor()
        # SQL to get all the clients
        c.execute('''SELECT clientID, clientName FROM client''')
        result = c.fetchall()
        c.close()
        clientNumbers = []
        # loops over all clients and stores their primary key in a list
        for client in result:
            print "Client Number: " + str(client[0]) + " Name: " +
client[1]
            # primary key is converted to a string before stored
            clientNumbers.append(str(client[0]))
            clientNumber = raw_input("Choose a client number:")
            # the user input is checked to be in the list of primary keys, if
not it asks again
            while clientNumber not in clientNumbers:
                print "You didn't enter a correct client number"
                clientNumber = raw_input("Choose a client number:")
            # the primary key is returned
            return clientNumber
    except:
        print "couldn't access information from the database"

# procedure to display the monthly report information
def monthReport():
    # gets the current month from the built in function
    currentMonth = int(datetime.date.today().month)
    # calls a function to get the chosen month
    monthChosen = displayMonths()
    # makes sure the month is either the current one of earlier (not a
month in the future)
    if monthChosen > currentMonth:
        whatYear = int(datetime.date.today().year) - 1
    else:
        whatYear = int(datetime.date.today().year)
    try:
        conn = sqlite3.connect(DB)

```

```

c = conn.cursor()
# SQL to find all the quotes in the month
c.execute('''SELECT lawnLength, lawnWidth, patioLength, patioWidth,
              deckingLength, deckingWidth, pondLength, pondWidth,
              featureNumber, lightingNumber FROM quote
              WHERE quote.monthOfQuote=? AND quote.yearOfQuote=?''',
(monthChosen, whatYear))
    result = c.fetchall()
    c.close()
except:
    print "Couldn't access information from the database"

# total square metres/number of the different materials
totalLawn = 0.0
totalPatio = 0.0
totalDecking = 0.0
totalPond = 0.0
totalFeatures = 0.0
totalLighting = 0.0
# loop over the different quotes
for quote in result:
    # add the totals to the running totals
    totalLawn += quote[0] * quote[1]
    totalPatio += quote[2] * quote[3]
    totalDecking += quote[4] * quote[5]
    totalPond += quote[6] * quote[7]
    totalFeatures += quote[8]
    totalLighting += quote[9]

# find out the total cost
totalCost = 0.0

# use the total square metres of the material and the cost
# to find the total cost (same for all water feature and lighting
# are number and not square metre
print "Lawn total metres:",
print totalLawn
cost = totalLawn * costs['lawnCostPerMetre']
print "Lawn total monthly value:",
print cost
totalCost += cost

print "Patio total metres:",
print totalPatio
cost = totalPatio * costs['patioCostPerMetre']
print "Patio total monthly value:",
print cost
totalCost += cost

print "Decking total metres:",
print totalDecking
cost = totalDecking * costs['deckingCostPerMetre']
print "Decking total monthly value:",
print cost
totalCost += cost

print "Pond total metres:",
print totalPond
cost = totalPond * costs['pondCostPerMetre']
print "Pond total monthly value:",
print cost

```



```

totalCost += cost

print "Water Feature total:",
print totalFeatures
cost = totalFeatures * costs['featureCost']
print "Water Feature total monthly value:",
print cost
totalCost += cost

print "Lighting total:",
print totalLighting
cost = totalLighting * costs['lightingCost']
print "Lighting total monthly value:",
print cost
totalCost += cost

# display the total cost
print "Total Monthly Value:",
print totalCost

# function for user to choose the correct month
def displayMonths():
    # dictionary linking month number to month name
    months = {'1': 'Jan', '2': 'Feb', '3': 'Mar', '4': 'Apr', '5': 'May',
'6': 'Jun',
'7': 'Jul', '8': 'Aug', '9': 'Sep', '10': 'Oct', '11': 'Nov',
'12': 'Dec'}
    # print out all the numbers and names
    for num, name in months.iteritems():
        print num + ": " + name
    monthChosen = raw_input("Enter the number of a month (1-12)")
    valid = False
    while not valid:
        # make sure the user has chosen one of the correct numbers
        if monthChosen in months.keys():
            valid = True
        else:
            monthChosen = raw_input("Make sure you enter a number (1-12)")
    # return the number (int) of the month chosen
    return int(monthChosen)

# the 'main' part - where the program starts
if __name__ == '__main__':
    # check the database exists and if not create the tables
    runFirstTime()

    # get all the costs from the external file
    costs = getCostsFromFile(MATERIALFILE)

    # carry on going round (the user exits by choosing 4)
    while True:
        # display the main menu
        displayMenu()
        menuChoice = raw_input()
        # validate the user input as being 1-4
        while menuChoice not in ['1', '2', '3', '4']:
            displayMenu()
            menuChoice = raw_input("Enter a number 1-4")
        # choose the correct procedure based on the user input
        if menuChoice == '1':
            searchQuote()

```

```
elif menuChoice == '2':  
    newQuote()  
elif menuChoice == '3':  
    monthReport()  
else:  
    # if they choose 4 then exit  
    print "system closing"  
    sys.exit()
```

Programming Techniques Used

Different Programming Techniques

Using libraries and built in functions

I've used built in functions but also had to use these libraries. Sys is for the system exit function, csv is to read a CSV file, sqlite3 means I can use an sqlite database and datetime and date are needed for when I need to find out the current date.

```
import sys
import csv
import sqlite3
import datetime
import time
```

Use of constants

My program has three constants at the start:

```
LABOURPERHOUR = 16.49
DB = 'gardening.db'
MATERIALFILE = 'materials.txt'
```

I have kept them in capitals to make it obvious they are constants. They can be updated at the top of the program and it will change them everywhere.

Use of relational database (create tables, add data, select data)

I am using sqlite for my database. This has the advantage that I don't have to configure it with passwords and make connections as it is just a file in the same folder. The file is called gardening.db. I use two related tables: client and quote. Both of these tables are implemented in the same way as the design.

I use the create table if not exists command at the start of my program so it doesn't create an error if the database already exists.

This is an example query searching for quotes by the quoteID:

```
c.execute('''SELECT lawnLength, lawnWidth, lawnCostPerM, patioLength,
patioWidth, patioCostPerM, deckingLength, deckingWidth, deckingCostPerM,
pondLength, pondWidth, pondCostPerM, featureNumber, featuresCostEa,
lightingNumber, lightingCostEa FROM quote WHERE quoteID=?''',
(quoteNumber,))
```

After the query is made the result is put into a list. If more than one result is returned then the results are in a two dimensional list. This is an example of where I have updated the database (with new clientNumbers and clientNames):

```
c.execute('''INSERT INTO client VALUES(?,?)''', (clientNumber, clientName))
```

Error catching

Databases can make errors and errors can happen in my code, particularly if the user input is not correct. I have used try-except statements to catch any errors, such as:

```
try:
    if float(answer) >= minValue:
        valid = True
except:
    answer = raw_input("Enter a valid number")
```

In this example if answer is not able to convert to type float then an error is called (and the user will be prompted again).

Reading data from an external file

All the material costs are stored in a file called `materials.txt`, this is read into the program every time it starts. To do this I used the CSV library which makes it an easier task:

```
def getCostsFromFile(filename):
    costs = {}
    try:
        fileReader = csv.reader(open(MATERIALFILE, 'rb'))

        for row in fileReader:
            costs[row[0]] = float(row[1])
        return costs
    except:
        print "make sure materials.txt exists and is in the correct format"
        sys.exit()
```

Every line is read in and a dictionary called costs is set with the key being the material cost and the value being the amount. If the file cannot be read it is probably because it is not in the correct format so an error message is displayed saying this.

Use of functions and procedures

I have used 14 functions and procedures in my code. This is an example of a function that takes two parameters and returns a float:

```
def getValidFloat(prompt, minValue):
    valid = False
    answer = raw_input(prompt)
    while not valid:
        try:
            if float(answer) >= minValue:
                valid = True
        except:
            answer = raw_input("Enter a valid number")
    return float(answer)
```

Validation of user input

The example above shows how user input is validated both by making sure it is in a suitable type (float) and that it is above a minimum value. All of my user input is validated and if the user enters an unacceptable value they are prompted again and again until they do.

Lists and dictionaries

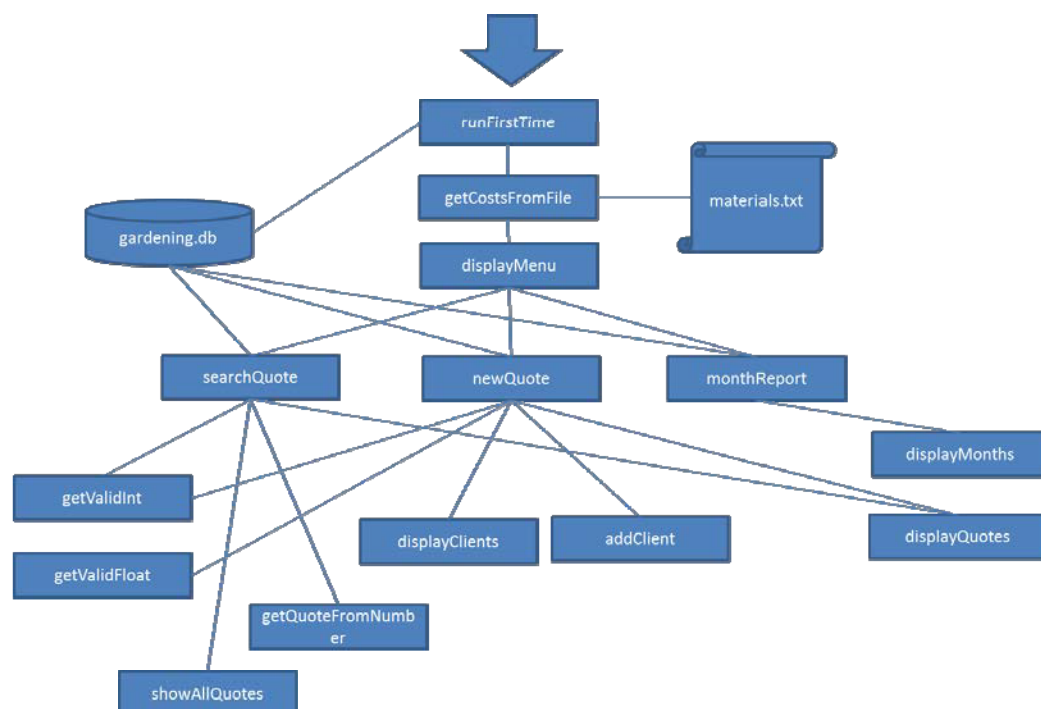
I have used both of these data structures in my program. `Costs` is an example of a dictionary (see Reading Data from an External File) where values can be found by entering their key. Lists are like Python's arrays, every time the database queries return a value they are in lists and so their values are accessed using an index, like this:

```
totalLawn += quote[0] * quote[1]
totalPatio += quote[2] * quote[3]
totalDecking += quote[4] * quote[5]
totalPond += quote[6] * quote[7]
```

Parts Working Together

Different Functions and Procedures

This diagram shows how my functions and procedures work together along with the database and materials file to make the whole system work:



The file `materials.txt` is only used once at the start of the program (after the database is checked to see if it is created). After that the `displayMenu` function is called, this is where the program returns after every user action until the user chooses to exit the program. The three main sub-menus of `displayMenu` are `searchQuote`, `newQuote` and `monthReport`. `searchQuote` first of all calls the `showAllQuotes` function that queries the database and displays all of the quote numbers and client names, the user chooses a quote number and this is passed as a parameter to the `getQuoteFromNumber` function that returns all of the details to `searchQuote`, finally the

displayQuotes function prints all of the information to the screen. newQuote gives the user the option to choose either an existing client (using the displayClients function) or to add a new client (using the addClient function). The rest of the newQuote prompts the user to enter all of the gardening information before saving this to the database and displaying it using the displayQuotes function. Both searchQuote and newQuote use the getValidInt and getValidFloat function to check if the user has entered data in the valid type and range. Finally the monthReport function calls the displayMonths to allow the user to choose a correct month, this works differently from the other two sub-menus and so doesn't share any functions with them.

Coded Efficiently

Use of relational database

I could have saved my data as an external text file (and actually did that for the material costs but that was because the brief said they may need to be updated and I thought that would be the easiest way without having to go into the program, really this could have been included in the relational database too but it might mean I would have to be careful with security and access to the database), but I chose a relational database instead. This is because the data is kept efficiently because I don't need to repeat the client information for every quote as I can just use the clientID as a foreign key in the quotes table. Also this is efficient because the database is very quick at finding and search for data, particularly if compared to having to read in data from an external file and then write my own searches using this data. Using SQL queries instead of just writing my own also means they are more likely to work. I have used SQLite which I don't think is the fastest database available and probably isn't as quick as something like MySQL but for this project it doesn't make a big difference because it won't ever hold a huge amount of data and it also has the advantage that it doesn't need usernames and passwords to access it because it is just a file in the same directory as my Python program. My queries all just make simple matching queries and so they should all run OK even with lots of data in the tables.

I chose not to save the calculated quote data in the database even though this would make the monthly reports function run quicker (and also slightly speed up the showQuotes function, although not so that the user would notice). The reason for this is because I also save the information for how much the materials cost – this means that the quote could use the old data for the quote if the costs have been updated but would the current data for the monthly reports. My program doesn't actually do this although the database is set up to do it this way and it could be done if needed.

Functions and Procedures

All of my functions run quite quickly as I don't have many nested loops that could make my program run more slowly.

Although using functions does not make my program run any faster it does mean that I have had to use less code because I can reuse functions whenever they are needed (the getValidFloat is a good example of this because it is used throughout, also the displayQuote is a long procedure that is called in more than one place). This means easier to read code and also easier to spot mistakes.

Data Structures

I have mainly used four different data structures in my code: lists (a bit like arrays in other languages), dictionaries, a CSV file and a relational database.

This is an example of using a list to check if user input is not a member:

```
while menuChoice not in ['1', '2', '3', '4']:
```

Lists hold data in order so they are used by the built in functions for the different parts of a row in a CSV file like this:

```
fileReader = csv.reader(open(MATERIALFILE, 'rb'))
for row in fileReader:
    costs[row[0]] = float(row[1])
```

Lists are also easy to use in for loops in Python. Lists are also used by the Python SQLite function that gets the result of a query (all of the parts of SELECT are the different elements in the list in the order that they were written). When more than one result is returned then Python uses lists of lists and so I use a for loop to go through one list at a time like this:

```
result = c.fetchall()
for quote in result:
    print "Quote Number:",
    print quote[0]
```

I use lists quite a lot when I want to check if a choice is not in a list of values like this:

```
while menuChoice not in ['1', '2']:
    print "Enter either 1 or 2"
```

I use lists when I want to hold data in an order and dictionaries when I need to access values based on a key. For instance the months of the year are stored as a dictionary because I need to number of the month to query the database:

```
months = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun',
          7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec' }
```

This means that if the user enters 3 I can use months[3] to find the value 'Mar'. I could have used a list for this but because Python lists start at 0 and not 1 I didn't want to complicate things by always having to add 1 to the index to get the right month.

I've explained my reasons for using a relational database in the previous section. My database is almost the same as it was in my design although it uses the types int, real and text instead of int, float, text and date. I started to use date but found it very difficult to deal with and so I changed it to three different integers for the day, month and year of the quote (this also makes it easier to check what month the quote was in). The primary keys for both the client and quote tables are the integer time (in milliseconds I think) that they were entered into the database, this means that no two will have the same because it is impossible to enter all of the data that quickly.

Robust Solution

User validation

All user input in my code is validated. The `getValidInt` and `getValidFloat` functions make sure numbers are entered in the correct format and with at least a minimum value. If entries are not valid then the program prompts the user and loops (normally with a while loop) until a valid entry is made.

Error Catching

As I mentioned earlier my program uses try-except statements that 'catch' when an error is detected. This way my program doesn't crash but an error message is printed to the screen and the user is taken back to the previous menu.

Example

This is the `getValidFloat` function:

```
def getValidFloat(prompt, minValue):
    valid = False
    answer = raw_input(prompt)
    while not valid:
        try:
            if float(answer) >= minValue:
                valid = True
        except:
            answer = raw_input("Enter a valid number")
    return float(answer)
```

In the third line the user is prompted to enter a number (the prompt is one of the parameters). The user could do four things here:

1. Enter a number in the correct format (float) and above or equal to the minimum value (minValue).
2. Enter a number in the correct format but below the minimum value.
3. Enter something in an incorrect format (in which case the minimum value doesn't matter).
4. Not enter anything.

The Boolean value `valid` is set to `False` and the while loop will continue until the user enters a valid entry (choice 1). If they did choice 2 then the Boolean conditional `float(answer) >= minValue` would be `False` and so `valid` would not be set to `True`, if they did choice 3 or 4 then `float(answer)` would cause an error because it wouldn't be able to work and would mean the try-except statement would print an answer. I have just realised that choice 2 would make the loop go on forever so I have changed the function to say the following:

```
def getValidFloat(prompt, minValue):
    valid = False
    answer = raw_input(prompt)
    while not valid:
        try:
            if float(answer) >= minValue:
                valid = True
            else:
                answer = raw_input("Make sure the number is at least " +
str(minValue))
        except:
            answer = raw_input("Enter a valid number")
```



```
return float(answer)
```

Now if the user enters a number below the minimum value the error message "Make sure the number is at least minValue" and then prompts them to enter another one.

Most of the user input is numbers but the user also has to enter the client name, to make sure they don't leave this blank I have used this code:

```
clientName = raw_input()
while not clientName:
    clientName = raw_input("Add a name")
```

If they enter nothing then Python thinks that clientName is like a Boolean value False and so a while loop will continue until they enter something.

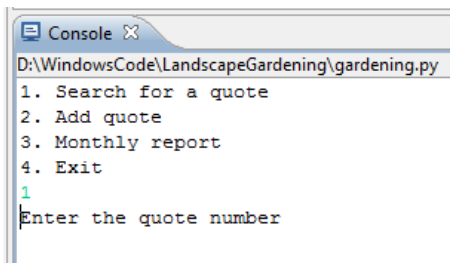
Testing & Evaluation

Test Plan & Evidence of Tests

Number	Description	Test	Input Data/User Action	Expected Outcome	Actual Result	Action Required
1.1	Opening menu works	Menu choice 1 works	1	Quote number is prompted	As expected	No
1.2		Menu choice 2 works	2	The two client sub menus are displayed	As expected	No
1.3		Menu choice 3 works	3	Months of the year are displayed (number then month)	All displayed although not in the obvious order	Re-order months by numbers (1 before 11)
1.4		Menu choice 4 works	4	Closing message displayed and program exits	As expected	No
2.1	Search by quote number works	Accepts correct quote number				
2.2		Rejects incorrect quote number				
2.3		Rejects non-numeric data				
5.1	Adding quote works	Adding a quote for a new client	Client Name: Charlie North, lawn length 10, lawn width 8, patio length 0, patio width 0, decking length 5, decking width 8, pond length 2, pond width 4, water features 1, garden lights 0	New client Charlie North is stored in the database and the total working costs are £2220.00, total labour costs are £884.96 and the total to	As expected although all currency is displayed as a long float number	Round currency to two decimal places

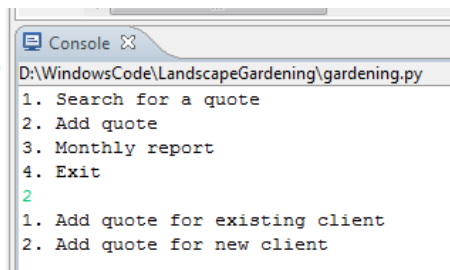
				pay is £3104.96 all of which is stored in the database		
5.2		Adding a quote for an existing client	Client Name: Charlie North, lawn length 10, lawn width 8, patio length 0, patio width 0, decking length 5, decking width 8, pond length 2, pond width 4, water features 1, garden lights 0	A list of clients is displayed and it accepts the choice of Charlie North and the total working costs are £2220.00, total labour costs are £884.96 and the total to pay is £3104.96 all of which is stored in the database	As expected although all currency is displayed as a long float number	Round currency to two decimal places
6.1	Monthly report works	Two quotes for May are selected	Month chosen is May (5)	The two quotes in the database both have labour of £2220.00 so the total should be £4440.00	As expected	No
7.1	Changing external file	Double all costs	New materials.txt is shown in the image (all costs are doubled)	Restart the program and new quote is entered with the same figures as before. The costs have doubled so the total material cost should be £4440.00	As expected	No

1.1: Quote number prompted



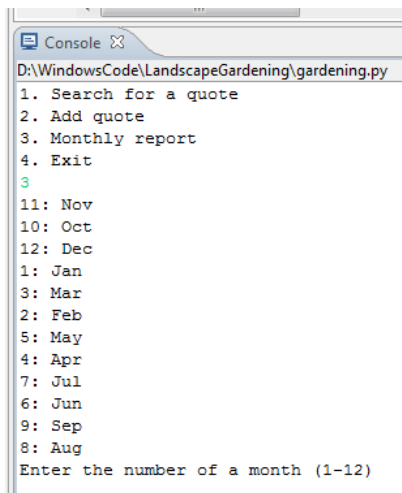
```
Console X
D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
1
Enter the quote number
```

1.2: The two quote submenu options are displayed:



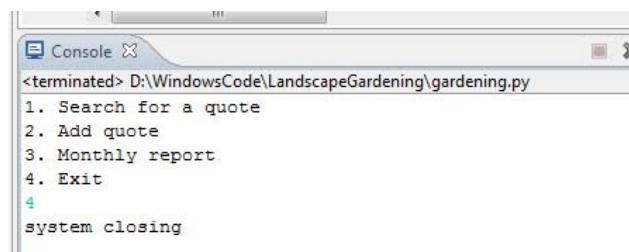
```
Console X
D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
2
1. Add quote for existing client
2. Add quote for new client
```

1.3: Months displayed



```
Console X
D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
3
11: Nov
10: Oct
12: Dec
1: Jan
3: Mar
2: Feb
5: May
4: Apr
7: Jul
6: Jun
9: Sep
8: Aug
Enter the number of a month (1-12)
```

1.4: System closes (grey square means program not running)



```
Console X
<terminated> D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
4
system closing
```

5.1: Adding a new client and a new quote

```

D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
2
1. Add quote for existing client
2. Add quote for new client
2
Add new client name:
Charlie North
Entering a quote for client 1338317308
Enter lawn length10
Enter lawn width8
Enter patio length0
Enter patio width0
Enter decking length5
Enter decking width8
Enter pond length2
Enter pond width4
Enter number of water features1
Enter number of lighting features0
Quote reference number: 1338317339
Material Costs

Lawn Costs
Length: 10.0
Width: 8.0
Total Area: 80.0
Cost per square metre: 15.5
Total Cost: 1240.0
Patio Costs
Length: 0.0
Width: 0.0
Total Area: 0.0
Cost per square metre: 20.99
Total Cost: 0.0
Decking Costs
Length: 5.0
Width: 8.0
Total Area: 40.0
Cost per square metre: 15.75
Total Cost: 630.0
Pond Costs
Length: 2.0
Width: 4.0
Total Area: 8.0
Cost per square metre: 25.0
Total Cost: 200.0
Water Feature Costs
Number of water features: 1
Cost per feature: 150.0
Total Cost: 150.0
Lighting Costs
Number of lights: 0
Cost per light: 5.0
Total Cost: 0.0

Total Working Costs: 2220.0

```

```

Labour Costs

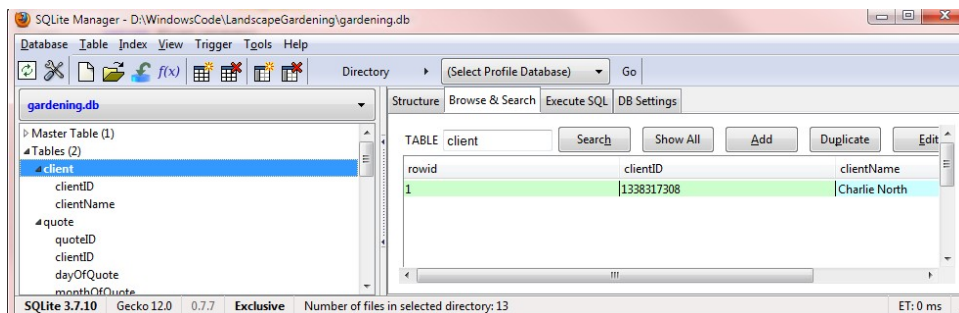
Lawn labour costs
Minutes per square metre: 20.0
Total area: 80.0
Total minutes: 1600.0
Patio labour costs
Minutes per square metre: 20.0
Total area: 0.0
Total minutes: 0.0
Decking labour costs
Minutes per square metre: 30.0
Total area: 40.0
Total minutes: 1200.0
Pond labour costs
Minutes per square metre: 45.0
Total area: 8.0
Total minutes: 360.0
Water feature labour costs
Minutes per water feature: 60.0
Number Purchased 1
Total minutes: 60.0
Garden lighting labour costs
Minutes per water feature: 10.0
Number Purchased 0
Total minutes: 0.0

Total Work (minutes): 3220.0
Total Work (hours): 53.666666667
Cost of Labour (per hour): 16.49
Total Labour Cost 884.963333333

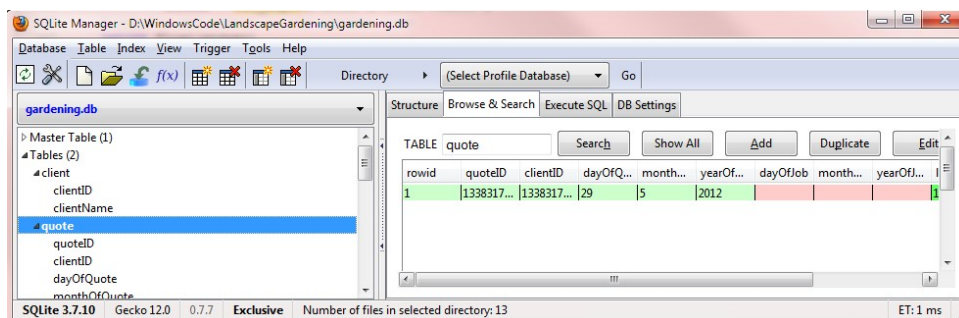
Total Working Costs: 2220.0
Total Labour Costs: 884.96333333
Total to Pay by Customer: 3104.96333333

```

5.1: New client added to database:



5.1: New quote added to database:



5.2: Adding a quote for an existing client:

```
Console X
D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
2
1. Add quote for existing client
2. Add quote for new client
1
Client Number: 1338317308 Name: Charlie North
Choose a client number:1338317308
Entering a quote for client 1338317308
Enter lawn length10
Enter lawn width8
Enter patio length0
Enter patio width0
Enter decking length5
Enter decking width8
Enter pond length2
Enter pond width4
Enter number of water features1
Enter number of lighting features0
Quote reference number: 1338318718
Material Costs

Lawn Costs
Length: 10.0
Width: 8.0
Total Area: 80.0
Cost per square metre: 15.5
Total Cost: 1240.0
Patio Costs
Length: 0.0
Width: 0.0
Total Area: 0.0
Cost per square metre: 20.99
Total Cost: 0.0
Decking Costs
Length: 5.0
Width: 8.0
Total Area: 40.0
Cost per square metre: 15.75
Total Cost: 630.0
Pond Costs
Length: 2.0
Width: 4.0
Total Area: 8.0
Cost per square metre: 25.0
Total Cost: 200.0
Water Feature Costs
Number of water features: 1
Cost per feature: 150.0
Total Cost: 150.0
Lighting Costs
Number of lights: 0
Cost per light: 5.0
Total Cost: 0.0

Total Working Costs: 2220.0
```

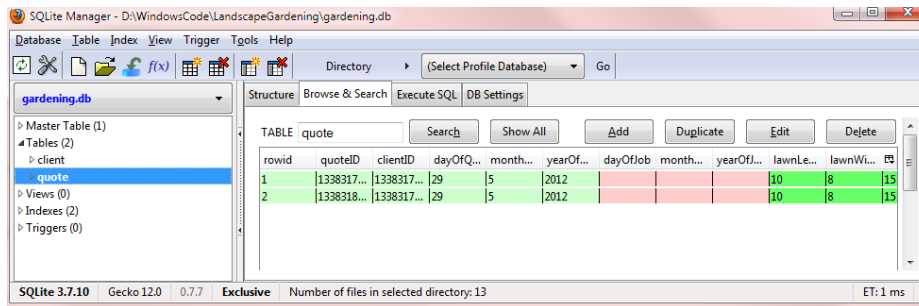
Labour Costs

```
Lawn labour costs
Minutes per square metre: 20.0
Total area: 80.0
Total minutes: 1600.0
Patio labour costs
Minutes per square metre: 20.0
Total area: 0.0
Total minutes: 0.0
Decking labour costs
Minutes per square metre: 30.0
Total area: 40.0
Total minutes: 1200.0
Pond labour costs
Minutes per square metre: 45.0
Total area: 8.0
Total minutes: 360.0
Water feature labour costs
Minutes per water feature: 60.0
Number Purchased 1
Total minutes: 60.0
Garden lighting labour costs
Minutes per water feature: 10.0
Number Purchased 0
Total minutes: 0.0

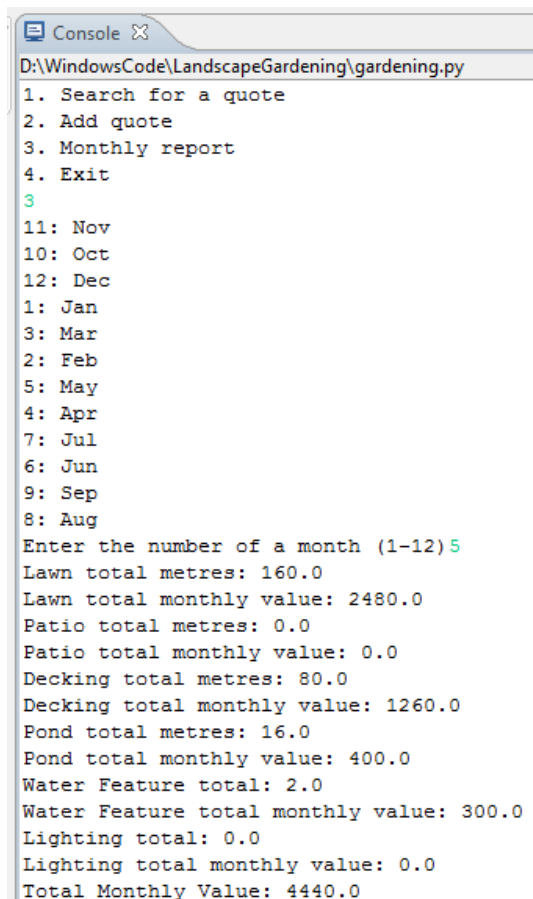
Total Work (minutes): 3220.0
Total Work (hours): 53.6666666667
Cost of Labour (per hour): 16.49
Total Labour Cost 884.963333333

Total Working Costs: 2220.0
Total Labour Costs: 884.963333333
Total to Pay by Customer: 3104.96333333
```

5.2: New quote added to database:

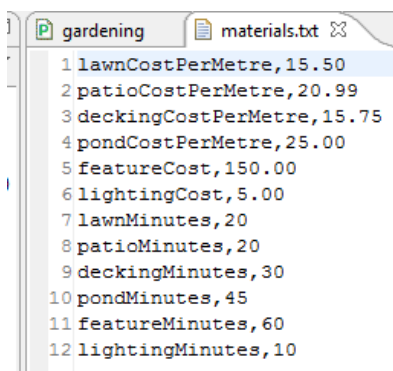


6.1: Monthly total (for May):



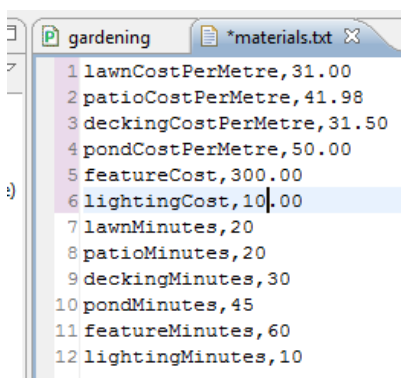
```
Console X
D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
3
11: Nov
10: Oct
12: Dec
1: Jan
3: Mar
2: Feb
5: May
4: Apr
7: Jul
6: Jun
9: Sep
8: Aug
Enter the number of a month (1-12) 5
Lawn total metres: 160.0
Lawn total monthly value: 2480.0
Patio total metres: 0.0
Patio total monthly value: 0.0
Decking total metres: 80.0
Decking total monthly value: 1260.0
Pond total metres: 16.0
Pond total monthly value: 400.0
Water Feature total: 2.0
Water Feature total monthly value: 300.0
Lighting total: 0.0
Lighting total monthly value: 0.0
Total Monthly Value: 4440.0
```

6.1: Change to the external file (original materials.txt):



```
gardening materials.txt X
1 lawnCostPerMetre,15.50
2 patioCostPerMetre,20.99
3 deckingCostPerMetre,15.75
4 pondCostPerMetre,25.00
5 featureCost,150.00
6 lightingCost,5.00
7 lawnMinutes,20
8 patioMinutes,20
9 deckingMinutes,30
10 pondMinutes,45
11 featureMinutes,60
12 lightingMinutes,10
```

6.1: Change to the external file (new materials.txt):



```
gardening *materials.txt X
1 lawnCostPerMetre,31.00
2 patioCostPerMetre,41.98
3 deckingCostPerMetre,31.50
4 pondCostPerMetre,50.00
5 featureCost,300.00
6 lightingCost,10.00
7 lawnMinutes,20
8 patioMinutes,20
9 deckingMinutes,30
10 pondMinutes,45
11 featureMinutes,60
12 lightingMinutes,10
```

6.1: Result of doubling the material cost:

```
Console X
D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
2
1. Add quote for existing client
2. Add quote for new client
1
Client Number: 1338317308 Name: Charlie North
Choose a client number:1338317308
Entering a quote for client 1338317308
Enter lawn length10
Enter lawn width8
Enter patio length0
Enter patio width0
Enter decking length5
Enter decking width8
Enter pond length2
Enter pond width4
Enter number of water features1
Enter number of lighting features0
Quote reference number: 1338319349
Material Costs

Lawn Costs
Length: 10.0
Width: 8.0
Total Area: 80.0
Cost per square metre: 31.0
Total Cost: 2480.0
Patio Costs
Length: 0.0
Width: 0.0
Total Area: 0.0
Cost per square metre: 41.98
Total Cost: 0.0
Decking Costs
Length: 5.0
Width: 8.0
Total Area: 40.0
Cost per square metre: 31.5
Total Cost: 1260.0
Pond Costs
Length: 2.0
Width: 4.0
Total Area: 8.0
Cost per square metre: 50.0
Total Cost: 400.0
Water Feature Costs
Number of water features: 1
Cost per feature: 300.0
Total Cost: 300.0
Lighting Costs
Number of lights: 0
Cost per light: 10.0
Total Cost: 0.0

Total Working Costs: 4440.0
```

Remedial Action


Test 1.3 shows that I should order the months to be displayed in the right order. I have changed the keys of the dictionary to be integers and altered some of the code so this would work:

```

50 # function for user to choose the correct month
51 def displayMonths():
52     # dictionary linking month number to month name
53     months = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun',
54               7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
55     # print out all the numbers and names
56     for num, name in months.items():
57         print(str(num) + ": " + name)
58     monthChosen = raw_input("Enter the number of a month (1-12)")
59     valid = False
60     while not valid:
61         # make sure the user has chosen one of the correct numbers
62         try:
63             if int(monthChosen) in months.keys():
64                 valid = True
65             else:
66                 monthChosen = raw_input("Make sure you enter a number (1-12)")
67         except:
68             monthChosen = raw_input("Make sure you enter a number (1-12)")
69     # return the number (int) of the month chosen
70     return int(monthChosen)
71

```

The months are now displayed like this:



```

D:\WindowsCode\LandscapeGardening\gardening.py
1. Search for a quote
2. Add quote
3. Monthly report
4. Exit
3
1: Jan
2: Feb
3: Mar
4: Apr
5: May
6: Jun
7: Jul
8: Aug
9: Sep
10: Oct
11: Nov
12: Dec
Enter the number of a month (1-12)

```

Tests 5.1 and 5.2 show that I should display currency in two decimal places.

This code shows a floating number to two decimal places:

```
print("£%.2f" % totalMaterialCost)
```

And so I have changed every part of the function displayQuote to display floats like this. The result is this (only the last part shown):

```

Cost per feature: 150.0
Total Cost: 150.00
Lighting Costs
Number of lights: 0
Cost per light: 5.0
Total Cost:
0.00

Total Working Costs: 2220.00

Labour Costs

Lawn labour costs
Minutes per square metre: 20.0
Total area: 80.0
Total minutes: 1600.00
Patio labour costs
Minutes per square metre: 20.0
Total area: 0.0
Total minutes: 0.00
Decking labour costs
Minutes per square metre: 30.0
Total area: 40.0
Total minutes: 1200.00
Pond labour costs
Minutes per square metre: 45.0
Total area: 8.0
Total minutes: 360.00
Water feature labour costs
Minutes per water feature: 60.0
Number Purchased 1
Total minutes: 60.00
Garden lighting labour costs
Minutes per water feature: 10.0
Number Purchased 0
Total minutes: 0.00

Total Work (minutes): 3220.0
Total Work (hours): 53.67
Cost of Labour (per hour): 16.49
Total Labour Cost 884.96

Total Working Costs: 2220.00
Total Labour Costs: 884.96
Total to Pay by Customer: 3104.96

```

Evaluation

My program works completely and the remedial action means it now displays currency and months correctly too. My solution is robust as the user can enter anything and the program will just keep going and ask them again until they get it right. The obvious next step would be to use a GUI but I thought this is too complicated for this project and I wanted to focus on the data and the processing.

My database works and holds information effectively. I have used sqlite which does not require any authorisation (neither does my program) so if this was to be a program used by an actual company then more security would be needed. Also more details on the clients (not just their names) would have to be stored. I use a long integer for the primary key which is a bit annoying because the user

has to enter a long number exactly to get the right answer, this could easily be changed to a smaller number.

My solution allows the user to search for quotes and client names so the user doesn't have to remember all of them. It does this by querying the database. If the user adds a new client or quote then the database is updated. All of the processing is done every time – this is slightly inefficient (although it is not much processing) but it means the data can be kept up to date with the details in the external file.

The external file is a CSV and so can be easily read and understood by another user (without having to understand a relational database). If the data is in an incorrect form then it will not be loaded and the program will exit, this is deliberate because I do not want my program to use bad data.

The user interface is what lets this project down a bit but like I said this would be the first thing to change if I extended this program. The functions and procedures I have used wouldn't make this difficult.