```
'''
      ___  ______  ___   _____________    _____________     ______
     /  /_______/  /  /  /_______/  /  /  /________/    /  /_/ /
    /  /       /  /  /  /       /  /  /  /            /  /  / /
   /  /       /  /  /  /_______/  /  /  /            /  /   / /
  /  /       /  /  / // // // /  /  /  /_________    /  /____/ / BILLING
 /  /       /  /  /  /-------/  /  /  /_________/   /  /_____/ / FRAMEWORK
/  /       /  /  /  /       /  /  /  /            /  /       / /
/  /_______/  /  /  /______ /  /  /  /            /  /       / /
/__/_______/__/  /__/_______/__/  /__/            /__/       /_/

    www.github.com/deltaonealpha/
    Program repo: https://deltaonealpha.github.io/DBFA/
'''

from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
pdfmetrics.registerFont(TTFont('MiLanProVF',
r'C:\Users\balaj\OneDrive\Documents\GitHub\DBFA\master\MiLanProVF.ttf'))
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.units import cm
from reportlab.lib.enums import TA_JUSTIFY
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Image
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from tabulate import tabulate

print("FHJ")
import sys, os
class HiddenPrints:
            def __enter__(self):
                self._original_stdout = sys.stdout
                sys.stdout = open(os.devnull, 'w')
            def __exit__(self, exc_type, exc_val, exc_tb):
                sys.stdout.close()
                sys.stdout = self._original_stdout
                print()

def telegram_bot_sendtext(bot_message):
    import requests
    with HiddenPrints():
        bot_token = '1215404401:AAEvVBwzogEhOvBaW5iSpHRbz3Tnc7fCZis'
        bot_chatID = '680917769'
        send_text = 'https://api.telegram.org/bot' + bot_token + '/sendMessage?
chat_id=' + bot_chatID + '&parse_mode=Markdown&text=' + bot_message
        response = requests.get(send_text)
        return response.json()


import getpass, time, pathlib, os, sqlite3
import os #library used to open the notepad application to display the sales records
if os.path.exists(r'userblock.zconf'):
    print("Decrypting authenication blobs...")
    print(" ")
    p = Path('userblock.zconf')
    p.rename(p.with_suffix('.txt'))
if os.path.exists(r'userblock.txt'):
```

```
    userblock = open(r"userblock.txt","r") #Opening / creating (if it doesn't exist already) the .txt record file
    valfn = 1
else:
    valfn = 0
if os.path.exists(r'userblock.txt'):
    userblock.close()
    os.remove(r'userblock.txt')
elif os.path.exists(r'userblock.zconf'):
    userblock.close()
    os.remove(r'userblock.zconf')

if os.path.exists(r'DBFA.zconf'):
    pass
else:
    print("Getting the database online.......")
    time.sleep(0.5)
    con = sqlite3.connect(r'DBFA.db')
    print("Rebuilding database..")
    c = con.cursor()
    #c.execute("DROP TABLE cust;")
    c.execute("""CREATE TABLE IF NOT EXISTS cust
        (custt INTEGER PRIMARY KEY,
        custname TEXT,
        email TEXT);""")


ssh = sqlite3.connect(r'DBFA_handler.db')
ssh7 = ssh.cursor()
#c.execute("DROP TABLE cust;")
ssh7.execute("""CREATE TABLE IF NOT EXISTS sshandler
    (prodid INTEGER,
    ssstock INTEGER);""")
ssh = sqlite3.connect('DBFA_handler.db')
ssh7 = ssh.cursor()
if os.path.exists(r'DBFA_handler.db'):
    pass
else:
    ssh7.execute("""CREATE TABLE IF NOT EXISTS sshandler
        (prodid INTEGER,
        ssstock INTEGER);""")
ssh.close()


# FOR CUSTOMER RECORDS
def massmaintainer(inxstock):  #defining a function to input data into the SQL database's table
    try:
        idList = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40]
        ssxconn = sqlite3.connect(r"DBFA_handler.db")
        ssxsql = 'DELETE FROM sshandler'
        ssxcur = ssxconn.cursor()
        ssxcur.execute(ssxsql)
        ssxconn.commit()
    except sqlite3.Error as error:
        print("Failed to flush multiple records from sqlite table", error)

    ssh = sqlite3.connect(r'DBFA_handler.db')
```

```
    ssh7 = ssh.cursor()
    namiex = ["TV 4K OLED 50", "TV FHD OLED 50", "8K QLED 80", "Redmi K20 PRO",
"Redmi K20", "Redmi Note 9 PRO", "POCOPHONE F", "Mi MIX ALPHA", "Wireless
Headphones", "Noise-Cancelling Wireless Headphones", "Essentials Headphones", "Gaming
Headphones", "Truly-Wireless Eadphones", "Neckband-Style Wireless Earphones",
"Essentials Earphones", "Gaming Earphones", "30W Bluetooth Speakers", "20W Bluetooth
Speakers", "9""Essentials Bluetooth Speaker", "BOSE QC35", "Essentials Home Theatre",
"Wired Speaker - 5.", "Essentials Wired Speaker - STEREO", "Tactical Power Bank
30000mah", "5""Essentials Power Bank 0000mah", "Essentials Mouse", "Logitech RGB
Gaming Mouse with Traction & Weight Adjustment", "Tactical Essentials Keyboard",
"Mechanical Cherry MX (Red) RGB Gaming Keyboard", "Polowski Tactical Flashlight",
"OneFiber Wi-Fi Router AX7", "Mijia Mesh Wi-Fi Router", "lapcare 0W Laptop Adapter",
"lapcare 60W Laptop Adapter", "Spigen Phone Case(s)", "Essentials Phone Charger
150W", "HyperPower Type-C Gallium-Nitride Charger 100W", "ASUS Zephyrus G4 Gaming
Laptop", "L XPS 5 Content Creator's Laptop", "Hewlett-Packard Essential's Student's
Laptop (Chromebook)"]
    for crrt in namiex:
        gg = (namiex.index(crrt)) + 1
        str = "insert into sshandler(prodid, ssstock) values({}, {})"
        # io = (gg)
        ssh7.execute(str.format(gg, inxstock))
        ssh.commit()
    ssh7.close()
    time.sleep(1)
    print("DBFA QuickVend service - Stock universally enforced to", inxstock)
    time.sleep(1)


# FOR CUSTOMER RECORDS
def ssxupdatescript(inxssincremental, prodid):
    ssh = sqlite3.connect('DBFA_handler.db')
    ssh7 = ssh.cursor()
    updatetr = """UPDATE sshandler SET ssstock = ssstock + ? WHERE prodid = ?"""
    xrindicator = (inxssincremental, prodid)
    ssh7.execute(updatetr, xrindicator)
    ssh.commit()
    ssh7.close()
    time.sleep(1)
    print("DBFA QuickVend Service - Stock added for", prodid, "as", inxssincremental)


def ssxsuperfetch():
    ssh = sqlite3.connect('DBFA_handler.db')
    ssh7 = ssh.cursor()
    print("Connecting to QuickVend Service... ~~~")  #SQL connection prompt
    print("Store Stock:: ")
    time.sleep(1.5)
    #Re-writing to refresh connection
    ssh7 = ssh.cursor()
    ssh7.execute("SELECT DISTINCT prodid, ssstock FROM sshandler")
    rows = ssh7.fetchall()
    for row in rows:
        print(row)
        #print(" ")

def ssxstockmaintainer(prodid):
    ssh = sqlite3.connect('DBFA_handler.db')
    ssh7 = ssh.cursor()
    updatetrtt = """UPDATE sshandler SET ssstock = ssstock - 1 WHERE prodid = ?"""
    xrindicatortt = (prodid,)
```

```
    ssh7.execute(updatetrtt, xrindicatortt)
    ssh.commit()
    ssh7.close()
    time.sleep(1)
    toaster.show_toast("DBFA QuickVend Service - Background Sync", duration = 0.3427)

def ssxstockmaster(prodid):
    global ssxvarscheck
    ssxvarscheck = 0
    ssh = sqlite3.connect('DBFA_handler.db')
    ssh.row_factory = lambda cursor, row: row[0]
    ssh7 = ssh.cursor()
    csrr = ("SELECT ssstock FROM sshandler WHERE prodid = (?);")
    csrrxt = (prodid,)
    ssh7.execute(csrr, csrrxt)
    rows = ssh7.fetchall()
    # print(rows) #debug point
    values = ','.join(str(v) for v in rows)
    print("Current product stock: ", values)
    ssxdsccheck = "1 2 3 4"
    if values in ssxdsccheck:
        print("[Stock running out] Currently in stock: ", values, "pieces. Restock
ASAP...")
        ssxvarscheck = 1
    elif values == "0":
        ssxvarscheck = 2
    elif values != "0":
        ssxvarscheck = 1
    time.sleep(1)
    toaster.show_toast("DBFA QuickVend Service - Background Sync", duration = 0.3427)

#Values stored in two dictionaries
data = {"1":40000, "2":55000, "3":67000, "4":25000, "5":21000, "6":14000, "7":13000,
"8":220000, "9":4500, "10":17000, "11":1200, "12":3700, "13":4500, "14":2200,
"15":700, "16":2750, "17":6499, "18":1499, "19":799, "20":27000, "21":6750,
"22":2100, "23":1199, "24":3210, "25":989, "26":750, "27":1700, "28":600, "29":2175,
"30":890, "31":2100, "32":7158, "33":597, "34":347, "35":500, "36":300, "37":1097,
"38":80000, "39":87900, "40":23790}
namie = {"1":"TV 4K OLED 50", "2":"TV FHD OLED 50", "3":"8K QLED 80", "4":"Redmi K20
PRO", "5":"Redmi K20", "6":"Redmi Note 8 PRO", "7":"POCOPHONE F1", "8":"Mi MIX
ALPHA", "9":"Wireless Headphones", "10":"Noise-Cancelling Wireless Headphones",
"11":"Essentials Headphones", "12":"Gaming Headphones", "13":"Truly-Wireless
Eadphones", "14":"Neckband-Style Wireless Earphones", "15":"Essentials Earphones",
"16":"Gaming Earphones", "17":"30W Bluetooth Speakers", "18":"10W Bluetooth
Speakers", "19":"Essentials Bluetooth Speaker", "20":"ULTRA Home Theatre",
"21":"Essentials Home Theatre", "22":"  Wired Speaker - 5.1", "23":"  Essentials
Wired Speaker - STEREO", "24":"Tactical Power Bank 30000mah", "25":"Essentials Power
Bank 10000mah", "26":"Essentials Mouse", "27":"Logitech RGB Gaming Mouse with
Traction & Weight Adjustment", "28":"Tactical Essentials Keyboard", "29":"Mechanical
Cherry MX (Red) RGB Gaming Keyboard", "30":"Polowski Tactical Flashlight",
"31":"OneFiber Wi-Fi Router AX17", "32":"Mijia Mesh Wi-Fi Router", "33":"lapcare 120W
Laptop Adapter", "34":"lapcare 60W Laptop Adapter", "35":"Spigen Phone Case(s)",
"36":"Essentials Phone Charger 10W", "37":"HyperPower Type-C Gallium-Nitride Charger
100W", "38":"ASUS Zephyrus G14 Gaming Laptop", "39":"L XPS 15 Content Creator's
Laptop", "40":"Hewlett-Packard Essential's Student's Laptop (Chromebook)"}
namiex = ["TV 4K OLED 50", "TV FHD OLED 50", "8K QLED 80", "Redmi K20 PRO", "Redmi
K20", "Redmi Note 9 PRO", "POCOPHONE F", "Mi MIX ALPHA", "Wireless Headphones",
"Noise-Cancelling Wireless Headphones", "Essentials Headphones", "Gaming Headphones",
"Truly-Wireless Eadphones", "Neckband-Style Wireless Earphones", "Essentials
Earphones", "Gaming Earphones", "30W Bluetooth Speakers", "20W Bluetooth Speakers",
```

```
"9""Essentials Bluetooth Speaker", "BOSE QC35", "Essentials Home Theatre", "Wired Speaker - 5.", "Essentials Wired Speaker - STEREO", "Tactical Power Bank 30000mah", "5""Essentials Power Bank 0000mah", "Essentials Mouse", "Logitech RGB Gaming Mouse with Traction & Weight Adjustment", "Tactical Essentials Keyboard", "Mechanical Cherry MX (Red) RGB Gaming Keyboard", "Polowski Tactical Flashlight", "OneFiber Wi-Fi Router AX7", "Mijia Mesh Wi-Fi Router", "lapcare 0W Laptop Adapter", "lapcare 60W Laptop Adapter", "Spigen Phone Case(s)", "Essentials Phone Charger 150W", "HyperPower Type-C Gallium-Nitride Charger 100W", "ASUS Zephyrus G4 Gaming Laptop", "L XPS 5 Content Creator's Laptop", "Hewlett-Packard Essential's Student's Laptop (Chromebook)"]
datax = [40000, 55000, 67000, 25000, 21000, 14000, 3000, 220000, 4500, 17000, 1200, 3700, 4500, 2200, 700, 2750, 6499, 1499, 799, 27000, 6750, 2100, 1199, 3210, 989, 750, 1700, 600, 2175, 890, 2100, 7158, 597, 347, 500, 300, 1097, 80000, 87900, 23790]
dataxr = []
for i in datax:
    i = "₹" + '%d' % i
    dataxr.append(i)
tablx = zip(namiex, dataxr)
titlex = ["Product:", "Pricing:"]

'''
def floodscreen():
    import cv2
    image = cv2.imread("imagepx.png")
    cv2.imshow("Initializing... ", image)
    cv2.waitKey(3000)
    cv2.destroyAllWindows()
'''

print("DBFA Billing Framework: Version 2.227 (alpha) ")
print(" Licensed under the GNU PUBLIC LICENSE")
print("<DBFA>  Copyright (C) 2020 Pranav Balaji and Sushant Gupta")
print(" ")
print("Visit: www.github.com/deltaonealpha/deltaBillingFramework for complete licensing terms. ")
time.sleep(1.3)
command = "cls"
os.system(command)

def mainmenu(): #defining a function for the main menu
    from colorama import init, Fore, Back, Style #color-settings for the partner/sponsor adverts
    init(convert = True)
    print(Fore.RED) #red-line to indicate program start
    print("-------------------------------------------")
    print(Fore.WHITE)
    print('A word from our partner: ' + Fore.BLACK + Back.CYAN + 'HOTEL? Trivago!') #Text over here #Custom advert
    print(Style.RESET_ALL)
    print("-------DBFA standardised billing framework-------")
    print("Input: ")
    print("'1' to GENERATE INVOICE")
    print("'2' to REGISTER CUSTOMER,")
    print("'3' to VIEW REGISTERED CUSTOMERS,")
    print("'4' to VIEW CUSTOMER PURCHASE RECORDS")
    print("'5' to VIEW GENERATED INVOICES,")
    print("'6' to REVIEW STORE LISTING,")
    print("'7' to REVIEW LICENSING INFORMATION,")
    print("'8' to VIEW OR UPDATE STOCK,")
    print("and '9' to exit the framework.")
```

```
240     print("~ input TPM code to enter testing mode ~")
241     print("-------------------------------------------")
242     print()
243     print()
244
245
246 #void-setup phase
247 from datetime import datetime  #for reporting the billing time and date
248 now = datetime.now()
249 dt_string = now.strftime("%d/%m/%Y %H:%M:%S")  #datetime object containing current
    date and time
250 logger = open(r"registry.txt", "a+")  #Opening / creating (if it doesn't exist
    already) the .txt record file
251 logger.write("----------------------------------------- \n")
252 logger.write("DBFA Billing Framework by Pranav Balaji and Sushant Gupta\n")
253 logger.write("Licensed under the GNU PUBLIC LICENSE\n")
254 logger.write('ed')
255 logger.write("\n")
256 logger.write("Automated Store Registry:\n")
257 import time  #to provide delays to make the system run seamlessly
258
259 xon = sqlite3.connect(r'DBFA_CUSTCC.db')
260 xbr7 = xon.cursor()
261 #c.execute("DROP TABLE cust;")
262 xbr7.execute("""CREATE TABLE IF NOT EXISTS custcc
263     (custt INTEGER PRIMARY KEY,
264     custname VARCHAR(500),
265     purchasecount INTEGER,
266     ptotalx INTEGER);""")
267 xon = sqlite3.connect('DBFA_CUSTCC.db')
268 xbr7 = xon.cursor()
269 if os.path.exists(r'DBFA_CUSTCC.db'):
270     pass
271 else:
272     xbr7.execute("""CREATE TABLE IF NOT EXISTS custcc
273         (custt INTEGER PRIMARY KEY,
274         custname VARCHAR(500),
275         purchasecount INTEGER,
276         ptotalx INTEGER);""")
277 xon.close()
278
279 conn = sqlite3.connect('DBFA.db')
280 if os.path.exists(r'DBFA.db'):
281     pass
282 else:
283     conn.execute('''CREATE TABLE COMPANY
284          (ID INT PRIMARY KEY     NOT NULL,
285          NAME           TEXT    NOT NULL,
286          AGE            INT     NOT NULL,
287          ADDRESS        CHAR(50),
288          SALARY         REAL);''')
289 def inserter(custt, custname, email):  #defining a function to input data into the
    SQL database's table
290     global conn
291     str = "insert into cust(custt, custname, email) values('%s', '%s', '%s')"
292     io = (custt, custname, email)
293     conn.execute(str % io)
294     conn.commit()
295     print("Customer", custname, "registered in store directory")
296
```

```
297 # FOR CUSTOMER RECORDS
298 def custcc(custt, purchasecount, ptotalx):  #defining a function to input data into
    the SQL database's table
299     global xon
300     xon = sqlite3.connect(r'DBFA_CUSTCC.db')
301     xbr7 = xon.cursor()
302     str = "insert into custcc(custt, purchasecount, ptotalx) values('%s', '%s',
    '%s')"
303     io = (custt, purchasecount, ptotalx)
304     xbr7.execute(str % io)
305     xon.commit()
306     xbr7.close()
307     print("FJHG")
308
309
310 # FOR CUSTOMER RECORDS
311 def updatescript(custt, pincrement):
312     try:
313         xon = sqlite3.connect('DBFA_CUSTCC.db')
314         xbr7 = xon.cursor()
315         # hidden prints here ig
316         updatexr = """UPDATE custcc SET purchasecount = purchasecount + 1 WHERE custt
    = ?"""
317         updatexxr = """UPDATE custcc SET ptotalx = ptotalx + ? WHERE custt = ?"""
318         indicator = (custt)
319         xrindicator = (pincrement, custt)
320         xbr7.execute(updatexr, indicator)
321         xbr7.execute(updatexxr, xrindicator)
322         xon.commit()
323         xbr7.close()
324     except sqlite3.Error as error:
325         pass
326
327
328 #floodscreen()
329 from win10toast import ToastNotifier
330 toaster = ToastNotifier()
331 toaster.show_toast("DFBA Framework Runtime Broker","Please read operational and
    licensing documentation prior to use.", duration = 2)
332 print("Heyy there!",  'ed')
333 time.sleep(1.34)
334 if valfn == 0:
335     logger.write("Oauth bypass - registering for security")
336     time.sleep(1)
337     print("-------DBFA standardised billing framework-------")
338     print("We highly value the security of our code, and our customers.")
339     toaster.show_toast("DFBA Framework Runtime Broker","Unauthenicated login
    detected!")
340     print("It has been detected that you have bypassed the login process.")
341     time.sleep(1)
342     print("Please obtain a genuine version of this program and provide proper
    authenication.")
343     time.sleep(1)
344     print("The program shall now exit. Error code:013")
345     time.sleep(2)
346     print("----------------------------------------")
347     time.sleep(5)
348     exit()
349
350 while(1): #while (always) true
```

```
    mainmenu() #mainmenu
    writer = ""
    telethon = ""
    time.sleep(0.3)  #for a seamless experience
    decfac = int(input("Select option: "))
    #Bill Mode
    if decfac == 1:
        print()
        print("Invoicing: ")
        print()
        custt = input("Enter customer ID (enter if unregistered): ")
        logger.write("-----------------  ") #writing to log file
        logger.write("Cust. ID: \n")
        logger.write(custt)
        logger.write("  \n")
        logger.write("Date and time: \n") #including the date and time of billing (as taken from the system)
        logger.write(dt_string)
        logger.write(" \n")
        abcd1 = 1
        time.sleep(0.3) #for a seamless experience
        telethon = "DBFA Billing System" + "\n" + dt_string + "\n" + "Customer: " + custt + "\n"
        writer = writer + "DBFA Billing Framework" + "\n" + "One-stop solution for all your billing needs!" + "\n" + "\n" + "Billing time: " + dt_string + "\n" + "Customer ID: " + custt + "\n" + "-----------------------------" + "\n" + "\n"
        numfac = int(input("Number of purchased items: "))
        time.sleep(0.34) #for a seamless experience
        afac = 0
        billiemaster = 0 #variable for totalling the price
        while(afac!=numfac):
            item = input("Enter purchased product code: ")
            time.sleep(0.3) #for a seamless experience
            if item in data:
                ssxstockmaster(item)
                if ssxvarscheck == 1:
                    billiemaster+=data[item]
                    print("Product purchased: ", namie[item], " costing: ", data[item])
                    print("---")
                    priceprod = "₹" + '%d' % data[item]
                    logger.write("Appending product to order: \n")  #writing to file
                    logger.write(namie[item])
                    ssxstockmaintainer(item)
                    logger.write(" \n")
                    writer = writer + "Purchased: " + "\n" + namie[item] + "\n" + priceprod + "\n"
                else:
                    print("This product is currently not in stock... The inconvenience is regretted...")
                    continue

            else:
                print("Invalid entry! Retry: ")
                print("---")
            afac+=1

        #tax = int(input("Enter the net tax %: "))  #comment and uncomment tkinter lines to use GUI-based input
        print("18% standard GST - Incoicing!")
```

```
403         time.sleep(0.4)  #for a seamless experience
404         #discount =
int(simpledialog.askstring(title="deltaSTOREMANAGER",prompt="Enter the discount
percentage: "))
405         discount = int(input("Enter discount % (if any): ")) #comment and uncomment
tkinter lines to use GUI-based input
406         print(discount, "% net discount - Invoicing!")
407         time.sleep(0.2) #for a seamless experience
408         print("Invoicing... DBFA")
409         time.sleep(0.4) #for a seamless experience
410         tota = (((18/100)*billiemaster)+billiemaster)
411         total = tota-((discount/100)*tota)
412         discountx = '%d' % discount
413         telethon = telethon + "\n" + "Tax amount: 18%" + "\n"  + "Discount: " +
discountx + "%" + "\n" + "\n"
414         writer = writer + "\n" + "\n" + "----------------------------" + "\n" + "Tax
amount: 18%"  + "\n"  + discountx + "\n"  + "\n"
415
416         rupeesymbol = "\u20B9".encode("utf-8")
417         print("Invoice ID: ", abcd1, "; Total: ", total)
418         toaster.show_toast("DFBA Framework Runtime Broker:      Total billed for-
",str(total), duration = 1)
419         logger.write("Total amount billed for: \n") #writing to file
420         #regin.write("NET TOTAL: \n") #writing to file
421         telethon = telethon + "NET TOTAL: \n" + "\u20B9" + str(total) + "\n"
422         writer = writer + "NET TOTAL: \n" + str(total) + "\n"
423         logger.write(str(total))
424         logger.write("\n")
425         #regin.write(str(total))
426         #regin.write("\n")
427         updatescript(custt, total)
428         abcd1+=1
429         afac+=1
430         now = datetime.now()
431         dt_string = now.strftime("%d/%m/%Y %H:%M:%S")  #datetime object containing
current date and time
432         daterey = (dt_string.replace("/","")).replace(":", "")
433         namer = 'invoice'+ daterey+'.pdf'
434         can = SimpleDocTemplate(namer, pagesize=A4,
435                                 rightMargin=2*cm,leftMargin=2*cm,
436                                 topMargin=2*cm,bottomMargin=2*cm)
437         #can.setFont("MiLanProVF", 24)
438         can.build([Paragraph(writer.replace("\n", "<br />"), getSampleStyleSheet()
['Normal']),])
439
440         import shutil
441         source = namer
442         destination =
r'C:\Users\balaj\OneDrive\Documents\GitHub\DBFA\master\Generated_invoices'
443         dest = shutil.move(source, destination)
444         time.sleep(1.5) #for a seamless experience
445
446         import os, sys
447         #regin.close()
448         with HiddenPrints():
449             try:
450                 sender = telegram_bot_sendtext(telethon)
451                 print(sender)
452             except Exception:
453                 pass
```

```
454
455         print()
456     #Register Customer
457     elif decfac == 2:
458         print("Loading server connection....")  #SQL connection prompt
459         time.sleep(0.4)  #for a seamless experience
460         #conn.execute("select * from cust")
461         #takes values from the SQL database
462         cursor = conn.cursor()
463         cursor.execute("select * from cust")
464         results = cursor.fetchall()
465         idd = len(results)+1
466         print("Registering customer with ID: ", idd)
467         custname = input("Name: ")
468         email = input("Customer's E-mail ID: ")
469         inserter(idd, custname, email) #argumental function to insert values into the
    SQL database
470         nullvalue = 0
471         custcc(idd, nullvalue, nullvalue)
472         print(" ")
473         logger.write("---------------------------------------- \n")
474         logger.write("  \n")
475         logger.write("Date and time: ") #including the date and time of billing (as
    taken from the system)
476         logger.write(dt_string)
477         logger.write(" \n")
478         logger.write("New customer registered: ")
479         x = " custname: " + custname + " custemail: " + email + "\n"
480         logger.write(x)
481         logger.write("---------------------------------------- \n")
482         print("Customer ID", idd, "registered in directory.")
483         print("----------------------------------------")
484         print(" ")
485         print(" ")
486         #conn.close()
487         time.sleep(1) #for a seamless experience
488     #VIEW ALL CUSTOMERS
489     elif decfac == 3:
490         print()
491         print("Loading server connection....")  #SQL connection prompt
492         time.sleep(0.7) #for a seamless experience
493         print("The registered customers are: ")
494         #Re-writing to refresh connection
495         cur = conn.cursor()
496         cur.execute("SELECT * FROM cust")
497         rows = cur.fetchall()
498         for row in rows:
499             print(row)
500             print(" ")
501         toaster.show_toast("DFBA Superfetch Service", "Superfetch: Database
    acessed!", duration = 2)
502         #takes values from the SQL database
503         '''
504         while row is not None:
505             print(row)
506             #row = conn.fetchone()
507         '''
508         logger.write("---------------------------------------- \n")
509         logger.write("  \n")
```

```
510         logger.write("Date and time: ") #including the date and time of billing (as
    taken from the system)
511         logger.write(dt_string)
512         logger.write(" \n")
513         logger.write("Customer registry accessed! \n")
514         logger.write("-------------------------------------- \n")
515         conn.close()
516         conn.close()
517         print()
518         print()
519         time.sleep(2) #delay for easy-table viewing
520 
521     #View Customer Purchase Records
522     elif decfac == 4:
523         xon = sqlite3.connect(r'DBFA_CUSTCC.db')
524         xbr7 = xon.cursor()
525         xbr7.execute("SELECT * FROM custcc")
526         rows = xbr7.fetchall()
527         for row in rows:
528             print(row)
529             print(" ")
530         xbr7.close()
531         toaster.show_toast("DFBA Superfetch Service", "Superfetch: Database
    acessed!", duration = 0.5)
532 
533     #View Generated Bills
534     elif decfac == 5:
535         #password verification as sales record is not to be shown to all;
536         print("Password echo shall be supressed for security.")
537         passw = getpass.getpass(prompt='Enter root password to view store activity
    registry: ', stream=None)
538         logger.write("  \n")
539         logger.write("Date and time: ") #including the date and time of billing (as
    taken from the system)
540         logger.write(dt_string)
541         logger.write(" \n")
542         if passw == "root":
543                 time.sleep(1) #for a seamless experience
544                 print("Hold on, moneybags.")
545                 with HiddenPrints():
546                     try:
547                         sender = telegram_bot_sendtext(dt_string + "\n" + "Registry
    files accessed - DBFA SECURITY")
548                         print(sender)
549                     except Exception:
550                         pass
551                 time.sleep(0.4)
552                 print("There ya go:: ")
553                 time.sleep(0.2) #for a seamless experience
554                 logger.write("Log file access attempt - Oauth complete \n")
555                 logger.close() #to change file access modes
556                 logger = open("registry.txt", "r+")
557                 # Uncomment the below lines if the program has to be modified to show
    the records in the shell itself and not externally
558                 # print(logger.read())
559                 # print()
560                 # print("Opening sales log externally now. ")
561                 time.sleep(1.4) #for a seamless experience
562                 os.startfile('registry.txt') #to open the external notepad
    application
```

```
563         else:
564             logger.write("  \n")
565             logger.write("Date and time: ")  #including the date and time of billing
(as taken from the system)
566             logger.write(dt_string)
567             logger.write(" \n")
568             time.sleep(1) #for a seamless experience
569             logger.write("Log file access attempt - Oauth failiure!!! \n")
570             print("Wrong, sneaky-hat. Try again: ")
571             print(" ")
572             print("Password echo shall be supressed for security.")
573             passw = getpass.getpass(prompt='Enter root password to view store
activity registry: ', stream=None)
574             if passw == "root":
575                     time.sleep(1) #for a seamless experience
576                     print("Hold on, moneybags.")
577                     with HiddenPrints():
578                         try:
579                             sender = telegram_bot_sendtext(dt_string + "\n" +
"Registry files accessed - DBFA SECURITY: ATTEMPT 02")
580                             print(sender)
581                         except Exception:
582                             pass
583                     print("There ya go:: ")
584                     time.sleep(0.6) #for a seamless experience
585                     logger.write("  \n")
586                     logger.write("Date and time: \n") #including the date and time of
billing (as taken from the system)
587                     logger.write(dt_string)
588                     logger.write(" \n")
589                     logger.write("Log file access attempt - AUTHORIZATION SUCCESS
\n")
590                     logger.close() #to change file access modes
591                     logger = open("log.txt","r+")
592                     # print(logger.read())
593                     # print()
594                     # print("Opening sales log externally now. ")
595                     time.sleep(1.4) #for a seamless experience
596                     os.startfile('log.txt')
597             else:
598                 with HiddenPrints():
599                     try:
600                         sender = telegram_bot_sendtext(dt_string + "\n" + "[ACCESS
DENIED!!] - Registry file  - DBFA SECURITY [ACCESS DENIED!!]")
601                         print(sender)
602                     except Exception:
603                         pass
604                 print("Multiple Unsuccesfull Attempts Detected. Re-run the program to
login now. ")
605                 logger.write("(MULTIPLE ATTEMPTS!): Log file access attempt -
AUTHORIZATION FAILED!!! \n")
606                 time.sleep(1.4) #for a seamless experience
607                 print()
608                 print()
609     #View Listing Option
610     elif decfac == 6:
611         print("Store listing (as per updated records): ")
612         print(tabulate(tablx, headers = titlex, floatfmt = ".4f"))
613 
614     elif decfac == 8:
```

```
        decsfacter = str(input("Enter 'a' to VIEW STOCK, 'b' to ADD STOCK and 'c' to ENFORCE MASS STOCK: "))
        if decsfacter == "a":
            ssxsuperfetch()
        elif decsfacter == "b":
            objid = int(input("Enter the product ID to add stock for: "))
            stockincrement = int(input("Enter the amount of stock to be added: "))
            ssxupdatescript(stockincrement, objid)
            print("Stock updated by", stockincrement, "for product ID:", objid)
        elif decsfacter == "c":
            ggrtrr = int(input("Enter stock to universally enforce: "))
            massmaintainer(ggrtrr)

    #Exit System
    elif decfac == 9:
        if os.path.exists(r'userblock.txt'):
            userblock.close()
            os.remove(r'userblock.txt')
        if os.path.exists(r'userblock.zconf'):
            userblock.close()
            os.remove(r'userblock.zconf')
        toaster.show_toast("DFBA Framework Runtime Broker", "Obsufcating program...", duration = 2)
        def logoprintxrt():
            print("        ___ ______ ___   _____________    _____________ _______")
            time.sleep(0.5)
            print("       /  /_______/  /  /  /_______/  /  /  /________/    /  /_/ /")
            time.sleep(0.5)
            print("      /  /       /  /  /  /       /  /  /  /            /  /  / /")
            time.sleep(0.5)
            print("     /  /       /  /  /  /_______/  /  /  /  CLI        /  /   / /")
            time.sleep(0.5)
            print("    /  /       /  /  / // // // // /  /  /_________    /  /___/ /")
            time.sleep(0.5)
            print("   /  /       /  /  /  /-------/  /  /  /_________/   /  /_____/ /")
            time.sleep(0.5)
            print("  /  /       /  /  /  /       /  /  /  /            /  /      / /")
            time.sleep(0.5)
            print(" /  /_______/  /  /  /______ /  /  /  /            /  /       / /")
            time.sleep(0.5)
            print("/__/_______/__/  /__/_______/__/  /__/            /__/        /__/")
            print(" ")
            print(" ")
        logoprintxrt()
        time.sleep(2)
        break
        exit()
        os.close('securepack.pyw')
    elif decfac == 7:
        print("Fetching latest licensing information.......")
        print(" ")
```

```
        print(" ")
        logoprintxrt()
        time.sleep(1.5)
        print(" ")
        print(" ")
        print("DBFA by Pranav Balaji, 2020")
        print(" ")
        print("_______ Licensing _______")
        print("           GNU PUBLIC LICENSE - TERMS AND CONDITIONS")
        print("    <deltaBillingFramework>  Copyright (C) 2020 Pranav Balaji and Sushant Gupta")
        print("    This program comes with ABSOLUTELY NO WARRANTY; for details type *show w*.")
        print("    This is free software, and you are welcome to redistribute it")
        print("    under certain conditions; type *show c* for details. ")
        toaster.show_toast("DFBA Framework Runtime Broker", "©2020: DBFA by Pranav Balaji and Sushant Gupta", duration = 1.5)
        print(" ")
        print(" ")
        print("Visit: www.github.com/deltaonealpha/deltaBillingFramework for complete licensing terms. ")
        print(" ")
        print(" ")
        aacsbcfac = int(input("Enter '1' to view complete licensing stuff or '2' to return: "))
        if aacsbcfac == 1:
            print(" ")
            print("Please select to open with your prefered text viewer/ edittor.")

 os.startfile(r"C:\Users\balaj\OneDrive\Documents\GitHub\DBFA\master\LICENSE")
            print(" ")
            print(" ")
            print("-------------------------------------------------")

    #CIT
    elif decfac == 10:
        print("CIT INTERNAL TESTING MODE")
        ffxfac = str(input("Enter CIT Testing Mode? (y/n):: "))
        if ffxfac == "y":
            ffrxfac = str(input("Entering CIT may lead to data loss. Confirm entering CIT? (y/n):: "))
            if ffrxfac == "y":
                print("DBFA CIT MODE")
                print("Initialising DELTA dependancies...")
                print(" ")
                print(" ")
                print("CIT INPUTABLES::")
                print("Enter '1' to CLEAR ALL CUSTOMER RECORDS")
                print("Enter '2' to exit CIT")
                citfacin = int(input("Waiting for input / | \ | / | \ |:: "))
                if citfacin == 1:
                    # window.close()
                    os.startfile(r'securepack.py')
                else:
                    continue

            else:
                continue

        elif ffxfac == "n":
```

```
            print("Exiting CIT")
            time.sleep(1)
            continue
        else:
            print("Invalid input. . . . ")
            time.sleep(1)

    else:
        continue

# End of program
# Available on github: www.github.com/deltaonealpha/DBFA
# https://deltaonealpha.github.io/DBFA/
```