McDonald’s Management System

Project Documentation

Madhu Sudhanan 7/7/17 12 - B

مــــدرستـنــا الـثــانــويـــة، الـورقـاء
OUR OWN HIGH SCHOOL, AL WARQA'A

GEMS
EDUCATION

Certificate

Roll No. ....

Certified that Master...

of grade.................................... *Section* ....*has carried*

out practical work in Computer Lab prescribed by the Central Board

of Secondary Education, New Delhi during the academic year 2016-201.

....

Teacher-in-charge

Date: ....

....

External Examiner *Internal Examiner*

ACKNOWLEDGEMENTS

During the course of completion of the project, I am highly indebted to the following people for their valuable support and suggestions, which enabled me to complete and present the project in its final form:

The Computer Science Department of my school and especially our Computer Teacher Mrs. Anjali Manoj for the support and guidance and valuable suggestion for improvisations that she has provided without which our efforts would not have been fruitful.

I would also like to thank my parents for the constant help motivation they have given us.

Last but certainly not the least, all the members of my group for their continuous support and valuable contribution. These compiled efforts are always open to criticisms and suggestions and all endeavors have been made to minimize discrepancies.

INDEX

INTRODUCTION TO PYTHON

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.[25]

Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Features of Python:

Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a mix of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution.

Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. Python can also be embedded in existing applications that need a programmable interface. This design of a small core language with a large standard library and an easily extensible interpreter was intended by Van Rossum from the start because of his frustrations with ABC, which espoused the opposite mindset.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. As Alex Martelli put it: "To describe something as clever is *not* considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython that would offer a marginal increase in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or try using PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is making it fun to use. This is reflected in the origin of the name, which comes from Monty Python, and in an occasionally playful approach to tutorials and reference materials, such as using examples that refer to spam and eggs instead of the standard foo and bar.

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Typing in Python:

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass `type` (itself an instance of itself), allowing metaprogramming and reflection.

Before version 3.0, Python had two kinds of classes: *old-style* and *new-style*. The syntax of both styles is the same, the difference being whether the class `object` is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long term plan is to support gradual typing and as of Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named *mypy* supports compile-time type checking.

Who uses Python?

A lot of large companies and projects use the Python programming language:

- Zope Application Server
- Google
- YouTube
- Bit Torrent
- NASA
- Industrial Light & Magic

Python is also used as an embedded scripting language in programs like OpenOffice, Gimp and Blender.

OBJECTIVE OF THE PROGRAM

The McDonald’s Management system is a GUI software that performs various functions required for an organized management of the McDonald’

This system will be used by the cashiers of a McDonald’s store (e.g.: McDonald’s Drive Thru) for various purposes such as Billing the products requested by the customers, Adding and Deleting new items or Stock, Creating new item menus, Editing the price, stock, code etc. of the product.

OVERVIEW OF THE PROGRAM

The software is split in 3 main Frames (areas) and a menu bar:

1. **BILLING FRAME**
2. **BILL ENTRIES FRAME**
3. **STOCK MANAGEMENT FRAME**
4. **MENU BAR**

Note: Before getting into the main screen of the software, the cashier will require to enter the login password which will be set by the higher authorities of the company.

1. BILLING FRAME:

The billing frame is used for the billing of the products requested by the customer. This can be done by entering the product code, the quantity requested and then clicking ‘OK’ button. The requested product can also be cancelled by selecting the product from the display panel (panel that shows the product name, unit price, quantity requested and net amount) and clicking ‘Cancel’ button.

2. BILL ENTRIES FRAME:

This is where the total bill amount is shown. The cashier can enter the discount for the product (if any) and the amount paid by the customer. Then pressing ‘Enter’ will display the balance amount that should be returned to the customer. Then pressing ‘Next customer’ will clear all the entries.

3. STOCK MANAGEMENT FRAME:

This meant for the company’s store management. Here the cashier or the person in charge for stock deliveries etc. can manage the stock by adding new items to menu, adding stock, deleting items, deleting stock and many more.

4. MENU BAR:

Here there are many options such as creating a new items menu, viewing items, editing different parameters etc.

TECHNICAL DOCUMENTATION

ARCHITECTURE:
This program was developed using Python version 2.7

SYSTEM REQUIREMENTS:

1. System must have Windows XP or higher versions installed.
2. System must have Python 2.7.[x] installed.
    - Download Python 2.7

FUNCTIONS IN THE PROGRAM AND THEIR USE

1. FOR LOGIN SCREEN:

Function Name	Use
login_tk()	Main function for creating the login screen GUI Interface.
login123()	Used to verify the login code entered by the Cashier.

2. FOR BILLING FRAME:

Function Name	Use
mcD()	Main function for creating the GUI interface.
OK_button()	Used to perform all the billing calculations and updating the stock of products purchased.
cancel_button()	Used to cancel any order given by the customer.

Instances:

- **Products:** Displays the products ordered.
- **Price:** Displays the unit price of each ordered product.
- **Qty:** Displays the quantity of the product ordered.
- **Net amt:** Displays the total amount for that product.
- **Code:** The cashier should enter the code for the product ordered.
- **Qty:** The cashier should enter the quantity ordered by the customer.

3. FOR BILL ENTRIES FRAME:

Function Name	**Use**
mcD()	Main function for creating the GUI interface.
balance_amt()	Used to calculate the balance amount and display it.
next_customer1()	Used to clear all the entries keep them ready for The next customer.

Instances:

- **Paid amt:** The cashier should enter the amount paid by the customer.

4. FOR STOCK MANAGEMENT FRAME:

Function Name	**Use**
mcD()	Main function for creating the GUI interface.
a_i_button()	Used to add new items to the item menu.
a_i_warning()	Used to give a warning (asking yes or no) if the entered code for a new item already exists in the current item menu.
a_i_w_yes_click()	If user click yes (from above function), then the old item will be replaced by the new item.
n_d_s_buttons()	Used to add or delete stock
d_i_button()	Used to delete items from the menu.

Instances:

- **Code:** The user should enter the desired code for the product.
- **Product:** The user should enter the name of the product.
- **Price:** The user should enter the price of the product.
- **Stock:** The user should enter the current stock available for the product.

5. FOR MENU BAR FRAME:

Function Name	**Use**
mcD()	Main function for creating the GUI interface.
n_i_m_warning()	Used to warn the user if he is trying to create a new item menu.
warning_yes_click()	If user clicks yes (from above function), then all current items and their info. in the menu will be deleted and new items can be added to the menu.
n_i_m_ok_button()	Used to add the new items entered to the menu.
view_items()	Used to view all the items and their info. in the current item menu.
ext_login_screen()	Used to exit the main screen and go to the login Screen.
exit_warning()	Used to confirm (by asking yes or no) the user if he/she wishes to exit the application when exit command is clicked.
destroy()	Used to close the application.
c_spc_ok_button()	Used to change stock, unit price and code of a product
change_stock()	Used to create a change stock GUI interface and control passes to the above function when the user click OK button.
change_price()	Used to create a change unit price GUI interface an control passes to the above function when the user click OK button.
change_code()	Used to create a change code GUI interface and control passes to the above function when the user click OK button.
about()	Used to display the credits of this program.
documentation()	Used to open this documentation for the user's reference.

6. OTHER FUNTIONS:

Function Name	Use
main()	Used for program initialization and check resources required for software to run without any file errors.

USER DOCUMENTATION

Note: After logging in, once you reach the main screen, maximize it to get the optimum screen size.

To bill items: -

- Enter the product code under the field named 'Code'.
- Enter the quantity requested by the customer under the field named 'Qty'.
- Click 'Ok'.
- Continue this process as many times as wanted.
- Then in the adjacent column, enter the amount paid by the customer the field named 'Paid amt.'.
- Enter the discount in percent (you don't have to enter '%' symbol) under the field named 'Discount' if any.
- Then click 'Enter'.
- Once transaction is over, click 'Next Customer' to clear all the fields for the next customer.

To cancel an order: -

- Select the product that you want to cancel
- Then click on the 'Cancel button'.

To add items to menu: -

NOTE: 'Add items' option is located at the extreme top-right corner of the screen.

- Enter a desired product code under the field named 'Code'.
- Enter the name of the product you want to add under the field named 'Product'.
- Enter the price of the product under the field named 'Price'.
- Enter the current stock of the product under the field named 'Stock'.

- Then click on the ‘Add Item’ button to add the items and its info. to the menu

To add stock to an existing item: -

- ‘Add Stock’ option is just under the ‘Add Items’ option.
- Enter the product code for which you want to add stock under the field named ‘Code’.
- Enter the new stock delivered under the field named ‘Stock’.
- Then click ‘Add Stock’ button to add the stock.
- NOTE: This will add the entered stock to the current stock. If you want to edit the stock as a whole, view ‘Edit Stock’.

To delete items from menu: -

- ‘Delete Items’ option is just under the ‘Add Stock’ option.
- Enter the code of the product of which you want to delete under the field named ‘Code’.
- Then click ‘Delete Item’ button to delete the item from the menu.

To delete stock of an existing item: -

- ‘Delete Stock’ option is just under the ‘Delete Items’ option.
- Enter the product code for which you want to delete stock under the field name ‘Code’.
- Enter the stock number you want to delete under the field named ‘Stock’.
- Then click ‘Delete Stock’ to delete the stock from the menu.
- NOTE: This will not change the current stock of the product to the entered but will remove the entered amount from the current stock. To change stock as a whole, view ‘Edit Stock’.

To create a new item menu: -

NOTE: This can be used if you want to completely delete the current item menu and make a new one.

- Click on the ‘File’ menu.
- Then click on ‘Create new item menu’.
- Once clicked, you will get a confirmation for the action, click ‘yes’ if you want to continue else click ‘no’.
- Then a window that contains the fields will open.
- Enter the desired information to make a new item menu.

To view current item menu: -

- Click on ‘File’ menu.
- Then click on ‘View all items’ option.

To exit the application or exit to login screen: -

- Click on ‘File’ menu.
- Then click on ‘Exit to login screen’ to exit to login screen or click ‘Exit’ to exit application.

To change current product code to new product code: -

- Click on ‘Edit’ menu.
- Then click on ‘Change code’ option.
- A small window will appear.
- Enter the product code for which you want to change to new code under the field named ‘Old Code’.
- Enter the new product code under the field named ‘New Code’.
- Then click ‘OK’.

To Edit stock as a whole: -

- Click on ‘Edit’ menu.
- Then click on ‘Change Stock’ option.
- A small window will appear.
- Enter the code for which you want to change stock under the field named ‘Code’.

- Enter the stock to which you want to change under the field named 'Stock'.
- Then click 'OK'.

<u>To change the price of a product: -</u>

- Click on 'Edit' menu.
- Then click on 'Change price' option.
- A small window will appear.
- Enter the Product's code for which you want to change price under the field named 'Code'.
- Enter the price to which you want to change under the field named 'Price'.
- Then click 'OK'.

SOURCE CODE

```
from Tkinter import *
import tkMessageBox, Tkinter, pickle, winsound, os

#items list order -> ['Product name', Stock, Unit price]

class Login:
    def __init__(self, dict1):
        global login
        self.login=Tkinter.Tk()
        self.login.configure(bg='yellow')
        self.login.resizable(0,0)

        #=============== Creating Lables ===============#
        self.label=Label(self.login, text='       ', bg='yellow').pack(anchor='center')
        self.mc_label=Label(self.login, text='           WELCOME TO McDonald\'s
', bg='yellow',  font=('Lucida Calligraphy', 20)).pack(anchor='center')
        self.code_label=Label(self.login, font=('Calibri', 15, 'bold','underline'),
text='Code:', bg='yellow').pack(anchor='center')
        #=============== Creating Lables ===============#

        #=== Creating Entry text boxe ===#
        global code_entry
        self.code_entry=Entry(self.login, font=('Calibri', 15, 'bold'), show='*',
bd=7, bg='dark green', fg='white', relief='ridge')
        self.code_entry.pack(anchor='center')
        #=== Creating Entry text boxes ===#

        #================= Creating Button ==================#
        self.login_button=Button(self.login, text='Login', bd=7, font=('Calibri',
15, 'bold'), bg='white', fg='red',
                    activebackground='red', activeforeground='white',
command=lambda: self.login123(dict1,self.code_entry))
        self.login_button.pack(anchor='center', pady=10)
        #================= Creating Button ==================#
        self.login.mainloop()

    def login123(self, dict1,code_entry):
        '''try:
            winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
        except RuntimeError:
            pass'''
        Pass=code_entry.get()
        if Pass=='1':
            mcd=Mcd(dict1, self.login)
        else:
            tkMessageBox.showerror('McDonalds', 'Invalid code')





class Mcd:
    def __init__(self,dict1, login):
        self.destroy(login)
        global mcD
        self.mcD=Tkinter.Tk()
        self.mcD.configure(bg='yellow')
        self.mcD.geometry('1366x768+0+0')

        #================================Creating
Frames================================#
        self.name=Frame(self.mcD, bg='red', width='1280', height='100', bd=15,
relief='groove')
        self.name.pack(side=TOP, fill=X)
        self.name_label=Label(self.name, bg='red', fg='yellow', font=('Lucida
Calligraphy', 30, 'bold'), text='McDonald\'s Management System',
bd=10).pack()
        self.billing=Frame(self.mcD, bg='yellow',width='100', height='368',
bd=15, relief='groove')
```

```
        self.billing.pack(side=LEFT, anchor='n')
        self.entries=Frame(self.mcD, bg='yellow', width='100', height='200',
bd=15, relief='groove')
        self.entries.pack(side=LEFT, anchor='s')
        self.recipt=Frame(self.mcD, bg='yellow', width='350', height='700',
bd=15, relief='groove')
        self.recipt.pack(side=RIGHT, anchor='ne')

        #==================================Creating
Frames================================#

        #==========================self.Recipt Frame
Widgets================================#
        self.disable_entries=StringVar()
        self.disable_entries.set('XXXXXXXXXXXXXXXXXXXXXXXX')
        #------------------------Add item-----------------------------------------#
        self.new_menu_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 15, 'bold'), padx=117, text='Add Items to Menu').grid(row=0,
columnspan=4)
        self.new_code_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Code').grid(row=1, column=0)
        self.new_product_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Product').grid(row=1, column=1)
        self.new_Price_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Price').grid(row=1, column=2)
        self.new_stock_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Stock').grid(row=1, column=3)
        self.new_code_entry=Entry(self.recipt, bg='dark green', fg='white',
font=('Calibri', 11, 'bold'), relief='ridge', bd=7, width=10)
        self.new_code_entry.grid(row=2, column=0)
        self.new_product_entry=Entry(self.recipt, bg='dark green', fg='white',
font=('Calibri', 11, 'bold'), relief='ridge', bd=7)
        self.new_product_entry.grid(row=2, column=1)
        self.new_price_entry=Entry(self.recipt, bg='dark green', fg='white',
font=('Calibri', 11, 'bold'), relief='ridge', bd=7, width=7)
        self.new_price_entry.grid(row=2, column=2)
        self.new_stock_entry=Entry(self.recipt, bg='dark green', fg='white',
font=('Calibri', 11, 'bold'), relief='ridge', bd=7, width=5)
        self.new_stock_entry.grid(row=2, column=3)
        self.new_button=Button(self.recipt, bg='red', fg='yellow',
activebackground='yellow', activeforeground='red', pady=6,
padx=35,text='Add Item',
                        command=lambda: self.a_i_button(self.new_stock_entry,
self.new_code_entry, self.new_product_entry, self.new_price_entry))
        self.new_button.grid(row=4, columnspan=4)

        #-----------------------Add Stock-----------------------------------------#
        self.new_add_sep=Label(self.recipt, bg='yellow', fg='black', font=('Ariel',
10, 'bold'),

text='============================================').grid(row=5,
columnspan=4)
        self.add_item_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 15, 'bold'), text='Add Stock').grid(row=6, columnspan=4)
        self.add_item_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'),  text='Code').grid(row=7, column=0)
        self.add_product_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Product').grid(row=7, column=1)
        self.add_Price_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Price').grid(row=7, column=2)
        self.add_stock_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Stock').grid(row=7, column=3)
        self.add_code_entry=Entry(self.recipt, bg='dark green', fg='white',
font=('Calibri', 11, 'bold'), relief='ridge', bd=7, width=10)
        self.add_code_entry.grid(row=8, column=0)
        self.add_product_entry=Entry(self.recipt, font=('Calibri', 11, 'bold'),
relief='ridge', textvariable=self.disable_entries, state=DISABLED, bd=7)
        self.add_product_entry.grid(row=8, column=1)
        self.add_price_entry=Entry(self.recipt, font=('Calibri', 11, 'bold'),
relief='ridge', textvariable=self.disable_entries, state=DISABLED, bd=7,
width=7)
        self.add_price_entry.grid(row=8, column=2)
        self.add_stock_entry=Entry(self.recipt, bg='dark green', fg='white',
font=('Calibri', 11, 'bold'), relief='ridge', bd=7, width=5)
        self.add_stock_entry.grid(row=8, column=3)
        self.add_button=Button(self.recipt, bg='red', fg='yellow',
activebackground='yellow', activeforeground='red', pady=6, padx=35, text='
Add Stock',
                         command=lambda: self.n_d_s_buttons(0,
self.add_code_entry, self.add_stock_entry)).grid(row=9, columnspan=4)

        #---------------------Delete Item------------------------------------#
        self.add_del_sep=Label(self.recipt, bg='yellow', fg='black', font=('Ariel',
10, 'bold'),

text='=============================================').grid(row=10,
columnspan=4)
        self.del_item_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 15, 'bold'), text='Delete Items from Menu').grid(row=11,
columnspan=4)
        self.del_item_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Code').grid(row=12, column=0)
        self.del_product_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Product').grid(row=12, column=1)
        self.del_Price_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Price').grid(row=12, column=2)
        self.del_stock_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Stock').grid(row=12, column=3)
        self.del_code_entry=Entry(self.recipt, bg='dark green', fg='white',
relief='ridge', font=('Calibri', 11, 'bold'), bd=7, width=10)
        self.del_code_entry.grid(row=13, column=0)
        self.del_product_entry=Entry(self.recipt, relief='ridge', font=('Calibri',
11, 'bold'), textvariable=self.disable_entries, state=DISABLED, bd=7)
        self.del_product_entry.grid(row=13, column=1)
        self.del_price_entry=Entry(self.recipt, relief='ridge', font=('Calibri', 11,
'bold'), textvariable=self.disable_entries, state=DISABLED, bd=7, width=7)
        self.del_price_entry.grid(row=13, column=2)
        self.del_stock_entry=Entry(self.recipt, relief='ridge', font=('Calibri', 11,
'bold'), textvariable=self.disable_entries, state=DISABLED, bd=7, width=5)
        self.del_stock_entry.grid(row=13, column=3)
        self.del_button=Button(self.recipt, bg='red', fg='yellow',
activebackground='yellow', activeforeground='red', pady=6, padx=35,
text='Delete Item',
                          command=lambda:
self.d_i_button(self.del_code_entry)).grid(row=14, columnspan=4)

        #--------------------Delete Stock-----------------------------------#
        self.add_del_sep=Label(self.recipt, bg='yellow', fg='black', font=('Ariel',
10, 'bold'),

text='============================================').grid(row=15,
columnspan=4)
        self.sdel_item_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 16, 'bold'), text='Delete Stock').grid(row=16, columnspan=4)
        self.sdel_item_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Code').grid(row=17, column=0)
        self.sdel_product_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Product').grid(row=17, column=1)
        self.sdel_Price_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Price').grid(row=17, column=2)
        self.sdel_stock_label=Label(self.recipt, bg='yellow', fg='red',
font=('Calibri', 10, 'bold'), text='Stock').grid(row=17, column=3)
        self.sdel_code_entry=Entry(self.recipt, bg='dark green', fg='white',
relief='ridge', font=('Calibri', 11, 'bold'), bd=7, width=10)
        self.sdel_code_entry.grid(row=18, column=0)
        self.sdel_product_entry=Entry(self.recipt, relief='ridge', font=('Calibri',
11, 'bold'), textvariable=self.disable_entries, state=DISABLED, bd=7)
        self.sdel_product_entry.grid(row=18, column=1)
        self.sdel_price_entry=Entry(self.recipt, relief='ridge', font=('Calibri', 11,
'bold'), textvariable=self.disable_entries, state=DISABLED, bd=7, width=7)
        self.sdel_price_entry.grid(row=18, column=2)
        self.sdel_stock_entry=Entry(self.recipt, bg='dark green', fg='white',
relief='ridge', font=('Calibri', 11, 'bold'), bd=7, width=5)
```

```
        self.sdel_stock_entry.grid(row=18, column=3)
        self.sdel_button=Button(self.recipt, bg='red', fg='yellow', 
activebackground='yellow', activeforeground='red', pady=6, padx=35, 
text='Delete Stock',
                         command=lambda: self.n_d_s_buttons(1, 
self.sdel_code_entry, self.sdel_stock_entry)).grid(row=19, columnspan=4)
        self.sep1=Label(self.recipt, bg='yellow').grid(row=20, column=0)
        self.sep2=Label(self.recipt, bg='yellow').grid(row=21, column=0)
        #========================END-self.Recipt Frame Widgets-
END==============================#

        '''bill_recipt=Text(self.recipt, height=36, width=39)
        bill_recipt.grid(row=0, column=0)
        bill_recipt.insert(INSERT)'''
        #================================Creating Menu 
Bars==============================#
        self.menubar=Menu(self.mcD)
        self.filemenu=Menu(self.mcD, tearoff=1)
        self.filemenu.add_command(label='Create new item list', font=('Calibri', 
15, 'bold', 'italic'), command=lambda: self.n_i_m_warning(dict1))
        self.filemenu.add_command(label='View all items', font=('Calibri', 15, 
'bold', 'italic'), command=self.view_items)
        self.filemenu.add_separator()
        self.filemenu.add_command(label='Exit to login screen', font=('Calibri', 
15, 'bold', 'italic'), command=lambda:self.ext_login_screen(dict1))
        self.filemenu.add_separator()
        self.filemenu.add_command(label='Exit', font=('Calibri', 15, 'bold', 
'italic'), command=lambda:self.exit_warning(self.mcD))
        self.menubar.add_cascade(label='Menu', menu=self.filemenu)
        self.editmenu=Menu(self.mcD, tearoff=1)
        self.menubar.add_cascade(label='Edit', menu=self.editmenu)
        self.editmenu.add_command(label='Change item stock', font=('Calibri', 
15, 'bold', 'italic'), command=self.change_stock)
        self.editmenu.add_command(label='Change item price', font=('Calibri', 
15, 'bold', 'italic'), command=self.change_price)
```

```
        self.editmenu.add_command(label='Change item code', font=('Calibri',
15, 'bold', 'italic'), command=self.change_code)
        self.aboutmenu=Menu(self.mcD, tearoff=1)
        self.aboutmenu.add_command(label='About...', font=('Calibri', 15,
'bold', 'italic'), command=self.about)
        self.aboutmenu.add_command(label='How to use this application...',
font=('Calibri', 15, 'bold', 'italic'), command=self.documentation)
        self.menubar.add_cascade(label='Help', menu=self.aboutmenu)
        self.mcD.config(menu=self.menubar)
        #============================END-Creating Menu Bars-
END============================#

        #==================================Billing Frame
Widgets==================================#
        self.product_label=Label(self.billing, fg='red', bg='yellow',
text='Products', font=('Ariel', 20, 'bold')).grid(row=0, column=0)
        self.Price_label=Label(self.billing, fg='red', bg='yellow', text='Price',
font=('Ariel', 20, 'bold')).grid(row=0, column=1)
        self.Qty_label=Label(self.billing, fg='red', bg='yellow', text='Qty',
font=('Ariel', 20, 'bold')).grid(row=0, column=2)
        self.net_price=Label(self.billing, fg='red', bg='yellow', text='Net amt',
font=('Ariel', 20, 'bold')).grid(row=0, column=3)
        self.code_label=Label(self.billing, height=3, fg='red', bg='yellow',
font=('Calibri', 15, 'bold'), text='Code:').grid(row=3, columnspan=1,
sticky='W')
        self.product_code=Entry(self.billing, width=11, bd=7, relief='ridge',
bg='dark green', fg='white', font=('Helvetica', 17, 'bold'))
        self.product_code.grid(row=3, column=0)
        self.ok_button=Button(self.billing, text='OK', relief='raise',  padx=18,
pady=3, font=('Calibri', 12, 'bold'), bd=5, fg='yellow', bg='red',
                    activeforeground='red', activebackground='yellow',
highlightcolor='white', command=lambda: self.OK_button(self.Qty_entry,
self.net_price_listbox,
                              self.price_listbox, self.balance_amt_entry,
self.bill_amt_entry, self.paid_amt_entry, self.items, self.product_code,
self.qty_listbox))
```

```
        self.ok_button.grid(row=4, columnspan=4)
        self.cancel_button1=Button(self.billing, text='Cancel', relief='raise',
padx=18, pady=3, font=('Calibri', 12, 'bold'), bd=5, fg='yellow', bg='red',
                    activeforeground='red', activebackground='yellow',
highlightcolor='white', command=lambda:
self.cancel_button(self.balance_amt_entry,
                                                self.paid_amt_entry, self.bill_amt_entry,
self.items, self.price_listbox, self.qty_listbox, self.net_price_listbox))
        self.cancel_button1.grid(row=4, columnspan=3, sticky=E)
        self.Qty_label=Label(self.billing, fg='red', bg='yellow', text='Qty:',
font=('Calibri', 15, 'bold')).grid(row=3, column=1, sticky='W')
        self.Qty_entry=Entry(self.billing, relief='ridge', bd=7, width=4, bg='dark
green', fg='white', font=('Helvetica', 17, 'bold'))
        self.Qty_entry.grid(row=3, columnspan=2, sticky=E)
        self.items=Listbox(self.billing, fg='red', relief='ridge', font=('Calibri', 22,
'bold'), selectmode=SINGLE, height=10, width=17, bd=10, bg='light yellow')
        self.price_listbox=Listbox(self.billing, fg='red', relief='ridge',
font=('Calibri', 22, 'bold'), selectmode=SINGLE, height=10, width=6, bd=10,
                        bg='light yellow')
        self.items.grid(row=2, column=0, sticky='N')
        self.qty_listbox=Listbox(self.billing, fg='red', relief='ridge', font=('Calibri',
22, 'bold'), height=10, width=4, bd=10, bg='light yellow')
        self.qty_listbox.grid(row=2, column=2)
        self.price_listbox.grid(row=2, column=1, sticky='N')
        self.net_price_listbox=Listbox(self.billing, fg='red', relief='ridge',
font=('Calibri', 22, 'bold'), height=10, width=6, bd=10, bg='light yellow')
        self.net_price_listbox.grid(row=2, column=3, sticky='N')
        #============================END-self.Billing Frame Widgets-
END============================#

        #================================Entries Frame
Widgets================================#
        self.dicount_label=Label(self.entries, bg='yellow', fg='red', font=('Ariel',
16, 'bold'), text='Special\nDiscount').grid(row=1, column=0, sticky=W)
        self.bil_entry_label=Label(self.entries, height=3, bg='yellow',
font=('Ariel', 16, 'bold'), text='Bill amt.').grid(row=2, column=0, sticky=W)
```

```
        self.initial_bill_entry=StringVar()
        self.initial_bill_entry.set('0')
        self.initial_discount_entry=StringVar()
        self.initial_discount_entry.set('0')
        '''self.recipt_no=StringVar()
        self.recipt_no.set('0')
        self.recipt_no_entry=Entry(self.entries, relief='ridge', bd=7, 
textvariable=self.recipt_no, width=6, font=('Helvetica', 22, 'bold'), bg='dark 
green', fg='white')
        self.recipt_no_entry.grid(row=0, column=1, sticky=W)'''
        self.discount_entry=Entry(self.entries, relief='ridge', bd=7, 
textvariable=self.initial_discount_entry, width=6, font=('Helvetica', 22, 
'bold'),
                              bg='dark green', fg='white')
        self.discount_entry.grid(row=1, column=1, sticky=W)
        self.percent_label=Label(self.entries, bg='yellow', 
fg='red',font=('Helvetica', 22, 'bold'), text='%').grid(row=1, column=2, 
sticky=W)
        self.bill_amt_entry=Entry(self.entries, relief='ridge', bd=7, 
textvariable=self.initial_bill_entry, width=6, font=('Helvetica', 22, 'bold'),
                              bg='dark green', fg='white')
        self.bill_amt_entry.grid(row=2, column=1, sticky=W)
        self.paid_entry_label=Label(self.entries, height=3, bg='yellow', 
font=('Ariel', 16, 'bold'), text='Paid amt.').grid(row=3, column=0, sticky=W)
        self.paid_amt_entry=Entry(self.entries, relief='ridge', bd=7, width=6, 
font=('Helvetica', 22, 'bold'), bg='dark green', fg='white')
        self.paid_amt_entry.grid(row=3, column=1, sticky=E)
        self.balance_entry_label=Label(self.entries, height=3, bg='yellow', 
font=('Ariel', 16, 'bold'), text='Balance').grid(row=4, column=0, sticky=W)
        self.balance_amt_entry=Entry(self.entries, relief='ridge', bd=7, width=6, 
font=('Helvetica', 22, 'bold'), bg='dark green', fg='white')
        self.balance_amt_entry.grid(row=4, column=1, sticky=E)
        self.pay_amt_button=Button(self.entries, pady=4, bg='red', 
fg='yellow',text='  Enter  ', font=('Calibri', 15, 'bold'),
                        activebackground='dark green', activeforeground='white',
command=lambda: self.balance_amt(self.discount_entry,
self.balance_amt_entry,

self.bill_amt_entry, self.paid_amt_entry))
        self.pay_amt_button.grid(row=3, column=2)
        self.next_customer=Button(self.entries, padx=2, fg='yellow', bg='red',
activebackground='dark green', activeforeground='white',
text='Next\nCustomer',
                        font=('Calibri', 12, 'bold'),
                        command=lambda: self.next_customer1(self.discount_entry,
self.bill_amt_entry, self.paid_amt_entry, self.balance_amt_entry, self.items,
self.price_listbox,
                                                self.net_price_listbox, self.qty_listbox))
        self.next_customer.grid(row=4, column=2)
        #============================END-self.Entries Frame Widgets-
END============================#


        self.mcD.title('McDonalds Billing')
        self.mcD.mainloop()


    def ext_login_screen(self, dict1):
        self.destroy(self.mcD)
        log=Login(dict1)

    def destroy(self, window):
        window.destroy()



    def documentation(self):
        try:
            os.startfile('Project documentation.pdf')
        except WindowsError:
            tkMessageBox.showerror('File Error!', 'File not found!')



    def exit_warning(self, mcD):
        choice=tkMessageBox.askquestion('Exiting Application', 'Are you sure
you want to exit the application')
        if choice=='yes':
            self.destroy(self.mcD)
        else:
            pass




    def about(self):
        tkMessageBox.showinfo('McDonald\'s - Credits', 'Madhu - Group
Leader\nAbel - self.Billing function\nJudewin - Add items function\nAlbert -\
    Delete items function')




    def warning_yes_click(self, dict1):
        global n_i_m
        n_i_m=Toplevel()        #n_i_m ------> new_item_menu
        n_i_m.resizable(0,0)
        n_i_m.configure(bg='yellow')
        n_i_m.title('Create new items menu')
        n_i_m.wm_attributes('-topmost', 1)
        n_i_m.wm_attributes('-toolwindow', 1)
        prod_code2=Label(n_i_m, font=('Calibri', 15, 'bold'), bg='yellow',
fg='red', text='Code').grid(row=0, column=0)
        product=Label(n_i_m, font=('Calibri', 15, 'bold'), bg='yellow', fg='red',
text='Product').grid(row=0, column=1)
        price=Label(n_i_m, font=('Calibri', 15, 'bold'), bg='yellow', fg='red',
text='Price').grid(row=0, column=2)
```

```
        stock2_label=Label(n_i_m, font=('Calibri', 15, 'bold'), bg='yellow',
fg='red', text='Stock').grid(row=0, column=3)
        global code2_entry
        code2_entry=Entry(n_i_m, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', bd=7, width=5, relief='ridge')
        code2_entry.grid(row=1, column=0)
        global product2_entry
        product2_entry=Entry(n_i_m, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', bd=7, width=10, relief='ridge')
        product2_entry.grid(row=1, column=1)
        global price2_entry
        price2_entry=Entry(n_i_m, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', bd=7, width=5, relief='ridge')
        price2_entry.grid(row=1, column=2)
        global stock2_entry
        stock2_entry=Entry(n_i_m, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', bd=7, width=5, relief='ridge')
        stock2_entry.grid(row=1, column=3)
        ok_button3=Button(n_i_m, font=('Calibri', 15, 'bold'),
activebackground='yellow', activeforeground='red', bg='red', fg='yellow',
text='OK', padx=10, command=lambda:
self.n_i_m_ok_button(dict1)).grid(row=2, columnspan=3)
        try:
            with open('Stock Burgers.dat', 'rb') as file:
                dict1=pickle.load(file)
                dict1={}
                file.close()
        except EOFError:
            dict1=dict1



    def n_i_m_ok_button(self, dict1):
        '''try:
            winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
        except RuntimeError:
```

```
        pass'''
      with open('Stock Burgers.dat', 'rb') as file:
        dict3=pickle.load(file)
        key=dict3.keys()
        file.close()
      try:
        code=int(code2_entry.get())
        prod=(product2_entry.get())
        prod=prod.title()
        if code in key:
          choice=tkMessageBox.askquestion('WARNING', 'A product for this
code already exists.\nIf you click yes, the product will be replaced\
  with this product.\nYou cannot undo this process.\n\nAre you sure you
want to continue?')
          if choice=='yes':
            try:
              price=float(price2_entry.get())
              try:
                stock=int(stock2_entry.get())
                dict2={code:[prod, stock, price]}
                dict1.update(dict2)
                with open('Stock Burgers.dat', 'wb') as file:
                  pickle.dump(dict1, file)
                  file.close()
                  tkMessageBox.showinfo('McDonalds', 'Items added to
menu!')
              except ValueError:
                tkMessageBox.showerror('McDonalds', 'Invalid stock entry')
            except ValueError:
              tkMessageBox.showerror('McDonalds', 'Invalid price entry')
          else: pass
        else:
          try:
            price=int(price2_entry.get())
            try:
              stock=int(stock2_entry.get())
```

```
                  dict2={code:[prod, stock, price]}
                  dict1.update(dict2)
                  with open('Stock Burgers.dat', 'wb') as file:
                      pickle.dump(dict1, file)
                      file.close()
                      tkMessageBox.showinfo('McDonalds', 'Items added to
menu!')
              except ValueError:
                  tkMessageBox.showerror('McDonalds', 'Invalid stock entry')
          except ValueError:
              tkMessageBox.showerror('McDonalds', 'Invalid price entry')
      except ValueError:
          tkMessageBox.showerror('McDonalds', 'Invalid code entry')



  def next_customer1(self, discount_entry, bill_amt_entry, paid_amt_entry,
balance_amt_entry, items, price_listbox, net_price_listbox, qty_listbox):
      '''try:
          winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
      except RuntimeError:
          pass'''
      bill_amt_entry.delete(0, END)
      discount_entry.delete(0, END)
      bill_amt_entry.insert(0, '0')
      discount_entry.insert(0, '0')
      paid_amt_entry.delete(0, END)
      balance_amt_entry.delete(0, END)
      items.delete(0, END)
      price_listbox.delete(0, END)
      net_price_listbox.delete(0, END)
      qty_listbox.delete(0, END)



  def n_i_m_warning(self, dict1):
      n_i_m_w=tkMessageBox.askquestion('CRITICAL WARNING', 'You are
attempting to create a new item menu.\nCreating a new item menu will
delete the current \
  item menu.\nYou cannot undo this process.\n\nAre you sure you want to
continue?')
      if n_i_m_w=='yes':
          dict1.clear()
          self.warning_yes_click(dict1)
      else:
          pass



  def view_items(self):
      v_i=Toplevel()     #v_i ----> view_items
      v_i.title('View current items menu')
      v_i.resizable(0,0)
      v_i.wm_attributes('-topmost', 1)
      product_label2=Label(v_i, text='Product').grid(row=0, column=1)
      stock_label2=Label(v_i, text='Stock').grid(row=0, column=2)
      code4_label=Label(v_i, text='Code').grid(row=0, column=0)
      scrollbar=Scrollbar(v_i)
      scrollbar.grid(rowspan=2, column=4)
      item_listbox=Listbox(v_i, bg='yellow', fg='red', font=('Calibri', 25, 'bold'),
height=15, width=32, yscrollcommand=scrollbar.set)
      item_listbox.grid(row=1, column=1)
      stock_listbox=Listbox(v_i, bg='yellow', fg='red', font=('Calibri', 25,
'bold'), height=15, width=5, yscrollcommand=scrollbar.set)
      stock_listbox.grid(row=1, column=2)
      code4_listbox=Listbox(v_i, bg='yellow', fg='red', font=('Calibri', 25,
'bold'), height=15, width=5, yscrollcommand=scrollbar.set)
      code4_listbox.grid(row=1, column=0)
      price_label=Label(v_i, text='Price').grid(row=0, column=3)
      price_listbox=Listbox(v_i, bg='yellow', fg='red', font=('Calibri', 25, 'bold'),
height=15, width=7, yscrollcommand=scrollbar.set)
      price_listbox.grid(row=1, column=3)
      scrollbar.config(command=stock_listbox.yview)
      scrollbar.config(command=code4_listbox.yview)
      scrollbar.config(command=price_listbox.yview)
      scrollbar.config(command=item_listbox.yview)
      try:
          with open('Stock Burgers.dat', 'rb') as file:
              dict1=pickle.load(file)
              file.close()
              key=dict1.keys()
              key.sort()
              for i in key:
                  item1=dict1[i]     #it gets the item in the list eg:['burger', 60, 15]
                  item_listbox.insert(i, item1[0])  #inserts item eg: 'burger'
                  stock_listbox.insert(i, item1[1])  #inserts stock eg: '60'
                  price_listbox.insert(i, '$ '+str(item1[2]))  #inserts price eg: '15'
              for j in key:
                  code4_listbox.insert(i, j)
      except EOFError:
          pass



  def d_i_button(self, del_code_entry):
      '''try:
          winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
      except RuntimeError:
          pass'''
      try:
          with open('Stock Burgers.dat', 'rb') as file:
              dict1=pickle.load(file)
              file.close()
              try:
                  code3=int(del_code_entry.get())
                  try:
                      del dict1[code3]
                      with open('Stock Burgers.dat', 'wb') as file:
```

```
                    pickle.dump(dict1, file)
                    file.close()
                  tkMessageBox.showinfo('McDonalds', 'Item successfully
deleted from menu.')
                  del_code_entry.delete(0, END)
                except KeyError:
                  tkMessageBox.showinfo('Cannot Delete!', 'Product for this code
doesn\'t exist')
              except ValueError:
                tkMessageBox.showerror('McDonalds', 'Invalid entry')
          except EOFError:
            tkMessageBox.showinfo('McDonalds', 'There are no items in the
menu to delete!')



    def n_d_s_buttons(self, x, code_entry, stock_entry):     #
new_delete_stock_buttons
        '''try:
            winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
        except RuntimeError:
            pass'''
        with open('Stock Burgers.dat', 'rb') as file:
            dict1=pickle.load(file)
            file.close()
            try:
              code1=int(code_entry.get())
              qty=int(stock_entry.get())
              try:
                dict_list=dict1[code1]
                val=dict_list[1]
                if x==0:                         # if x==0, which means control is
comming from add_button
                  val=val+qty                      # hence qty should be added
                  dict_list[1]=val
                  dict1[code1]=dict_list
```

```
                    with open('Stock Burgers.dat', 'wb') as file:
                        dict1=pickle.dump(dict1, file)
                        file.close()
                        tkMessageBox.showinfo('mcDonalds', 'Item stock successfully
added.')
                        code_entry.delete(0, END)
                        stock_entry.delete(0, END)
                elif x==1:                              # if x==1, which means control is
comming from del_button
                    if qty>val:
                        tkMessageBox.showinfo('McDonalds', 'Quantity enterd is
more than current stock')
                    else:
                        val=val-qty                      #  hence qty should be subtracted
                        dict_list[1]=val
                        dict1[code1]=dict_list
                        with open('Stock Burgers.dat', 'wb') as file:
                            dict1=pickle.dump(dict1, file)
                            file.close()
                            tkMessageBox.showinfo('mcDonalds', 'Item stock
successfully deleted.')
                            code_entry.delete(0, END)
                            stock_entry.delete(0, END)
            except KeyError:
                tkMessageBox.showerror('McDonalds', 'Product for this code
doesn\'t exist :-(')
        except ValueError:
            tkMessageBox.showerror('mcDonalds', 'Invalid Entry')



    def a_i_button(self, new_stock_entry, new_code_entry,
new_product_entry, new_price_entry):
        '''try:
            winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
        except RuntimeError:
          pass'''
        with open('Stock Burgers.dat', 'r+') as file:
          dict1=pickle.load(file)
          key=dict1.keys()
          file.close()
          try:
            global code
            code=int(new_code_entry.get())
            if code in key:
              self.a_i_warning(new_stock_entry, new_code_entry,
new_product_entry, new_price_entry)
            else:
              prod_entry=new_product_entry.get().title()
              price2=float(new_price_entry.get())
              qty=int(new_stock_entry.get())
              item_list=[prod_entry, qty, price2]
              dict1[code]=item_list
              with open('Stock Burgers.dat', 'wb') as file:
                pickle.dump(dict1, file)
                file.close()
              tkMessageBox.showinfo('mcDonalds', 'Item successfully added to
menu.')
              new_product_entry.delete(0, END)
              new_price_entry.delete(0,END)
              new_stock_entry.delete(0, END)
              new_code_entry.delete(0,END)
          except ValueError:
            tkMessageBox.showerror('mcDonalds', 'Invalid entry')



    def a_i_warning(self, new_stock_entry, new_code_entry,
new_product_entry, new_price_entry):           # a_i_w -------->
Add_item_warning
```

```
        a_i_w=tkMessageBox.askquestion('CRITICAL WARNING', 'A product for
this code already exists!\nIf you click yes, product will be replaced for this
code.\
    \nYou cannot undo this process\n\nAre you sure you want to continue?')
        if a_i_w=='yes':
            self.a_i_w_yes_click(new_stock_entry, new_code_entry,
new_product_entry, new_price_entry)
        else:
            pass



    def a_i_w_yes_click(self, new_stock_entry, new_code_entry,
new_product_entry, new_price_entry):
        with open('Stock Burgers.dat', 'rb') as file:
            dict1=pickle.load(file)
            key=dict1.keys()
            file.close()
            prod_entry=new_product_entry.get().title()
            price2=float(new_price_entry.get())
            qty=int(new_stock_entry.get())
            item_list=[prod_entry, qty, price2]
            dict1[code]=item_list
            new_product_entry.delete(0, END)
            new_price_entry.delete(0,END)
            new_stock_entry.delete(0, END)
            new_code_entry.delete(0, END)
            with open('Stock Burgers.dat', 'wb') as file:
                pickle.dump(dict1, file)
                file.close()
            tkMessageBox.showinfo('mcDonalds', 'Item successfully updated')



    def c_spc_ok_button(self, x, code_entry, c_s_p_entry):
        '''try:
            winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
        except RuntimeError:
            pass'''
        with open('Stock Burgers.dat', 'rb') as file:
            dict1=pickle.load(file)
            file.close()
            key=dict1.keys()
        try:
            code=int(code_entry.get())

            item_list=dict1[code]
            if x==0:                    # if x==0, then the control is comming from
change_stock()
                try:                    # hence stock should be changed
                    entry=int(c_s_p_entry.get())
                    item_list[1]=entry
                    dict1[code]=item_list
                    tkMessageBox.showinfo('McDonald\'s', 'Stock Updated')
                except ValueError:
                    tkMessageBox.showinfo('McDonald\'s', 'Invalid Stock Entry!')
            elif x==1:                  # if x==1, then the control is comming from
change_price()
                try:                    # hence price should be changed
                    entry=float(c_s_p_entry.get())
                    item_list[2]=entry
                    dict1[code]=item_list
                    tkMessageBox.showinfo('McDonald\'s', 'Price Updated')
                except ValueError:
                    tkMessageBox.showerror('McDonald\'s', 'Invalid Price Entry!')
            elif x==2:                  # if x==2, then the control is comming from
change_code()
                try:                    # hence the code should be changed
                    entry=int(c_s_p_entry.get())
                    if entry in key:
                        confirm=tkMessageBox.askquestion('Warning!', 'Items for this
code already exists\nIf you click yes, items for this code\
     will be replaced\nDo you want to continue?')
                        if confirm=='yes':
                            old_item_list=dict1[entry]
                            dict1[code],dict1[entry]=old_item_list,item_list
                            tkMessageBox.showinfo('McDonald\'s', 'Code Updated')
                        else:
                            pass
                    else:
                        dict1[entry]=item_list
                        del dict1[code]
                        tkMessageBox.showinfo('McDonald\'s', 'Code Updated')
                except ValueError:
                    tkMessageBox.showerror('McDdonald\'s', 'Invalid "New code"
entry!')


            with open('Stock Burgers.dat', 'wb') as file:
                pickle.dump(dict1, file)
                file.close()
        except ValueError:
            tkMessageBox.showerror('McDonald\'s', 'Invalid Code Entry')
        except KeyError:
            tkMessageBox.showerror('Invalid code!', 'Product for this code
doesn\'t exist')



  def change_stock(self):
        c_s=Toplevel(bg='yellow')        #c_s -----> change_stock
        c_s.resizable(0,0)
        c_s.wm_attributes('-topmost', 1)
        product_code1=Label(c_s, font=('Calibri', 15, 'bold'), bg='yellow',
fg='red', text='Product code').grid(row=0, column=0)
        qty_label1=Label(c_s, font=('Calibri', 15, 'bold'), bg='yellow', fg='red',
text='Stock').grid(row=0, column=1)
        global product_code1_entry
```

```
        product_code1_entry=Entry(c_s, font=('Calibri', 15, 'bold'), bg='dark
green', fg='white', bd=7, width=10)
        product_code1_entry.grid(row=1, column=0)
        global stock_entry
        stock_entry=Entry(c_s, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', bd=7, width=5)
        stock_entry.grid(row=1, column=1)
        global c_s_ok_button
        c_s_ok_button=Button(c_s, font=('Calibri', 15, 'bold'),
activebackground='yellow', activeforeground='red', bg='red', fg='yellow',
                        text='   OK   ', command=lambda: self.c_spc_ok_button(0,
product_code1_entry, stock_entry))
        c_s_ok_button.grid(row=2, columnspan=2)



    def change_price(self):
        c_p=Toplevel(bg='yellow')         #c_p -----> change_price
        c_p.resizable(0,0)
        c_p.wm_attributes('-topmost', 1)
        product_code1=Label(c_p, font=('Calibri', 15, 'bold'), bg='yellow',
fg='red', text='Product code').grid(row=0, column=0)
        qty_label1=Label(c_p, font=('Calibri', 15, 'bold'), bg='yellow', fg='red',
text='Price').grid(row=0, column=1)
        global product_code1_entry
        product_code1_entry=Entry(c_p, font=('Calibri', 15, 'bold'), bg='dark
green', fg='white', bd=7, width=10)
        product_code1_entry.grid(row=1, column=0)
        global price_entry
        price_entry=Entry(c_p, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', bd=7, width=5)
        price_entry.grid(row=1, column=1)
        global c_p_ok_button
        c_p_ok_button=Button(c_p, font=('Calibri', 15, 'bold'),
activebackground='yellow', activeforeground='red', bg='red', fg='yellow',
                         text='   OK   ', command=lambda: self.c_spc_ok_button(1,
product_code1_entry, price_entry))
        c_p_ok_button.grid(row=2, columnspan=2)



    def change_code(self):
        c_c=Toplevel(bg='yellow')         #c_c -----> change_code
        c_c.resizable(0,0)
        c_c.wm_attributes('-topmost', 1)
        old_code=Label(c_c, font=('Calibri', 15, 'bold'), bg='yellow', fg='red',
text='Old code').grid(row=0, column=0)
        new_code=Label(c_c, font=('Calibri', 15, 'bold'), bg='yellow', fg='red',
text='New code').grid(row=0, column=1)
        old_code_entry=Entry(c_c, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', relief='ridge', bd=7, width=10)
        old_code_entry.grid(row=1, column=0)
        new_code_entry=Entry(c_c, font=('Calibri', 15, 'bold'), bg='dark green',
fg='white', relief='ridge', bd=7, width=10)
        new_code_entry.grid(row=1, column=1)
        c_c_ok_button=Button(c_c, font=('Calibri', 15, 'bold'),
activebackground='yellow', activeforeground='red', bg='red', fg='yellow',
                         text='   OK   ', command=lambda: self.c_spc_ok_button(2,
old_code_entry, new_code_entry))
        c_c_ok_button.grid(row=2, columnspan=2)



    def cancel_button(self, balance_amt_entry, paid_amt_entry,
bill_amt_entry, items, price_listbox, qty_listbox, net_price_listbox):
        try:
            selection=items.curselection()
            selection_items=items.get(0, END)[int(selection[0])]
            selection_qty=qty_listbox.get(int(selection[0]))

selection_net_amt=float(net_price_listbox.get(int(selection[0])).lstrip(' $'))
```

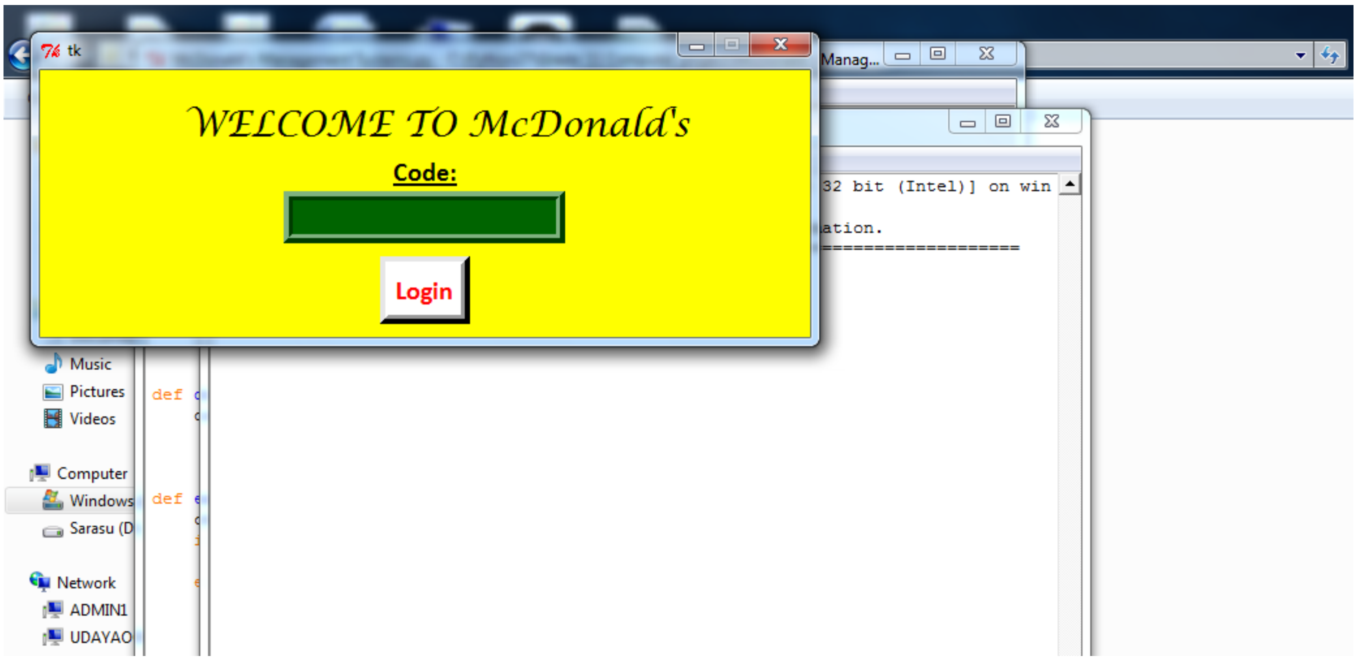
```
        with open('Stock Burgers.dat', 'rb') as file:
          dict1=pickle.load(file)
          file.close()
        keys=dict1.keys()                    #To refer the keys in the following code
        selection_tot_amt=0
        for i in range(len(keys)):
          item_list=dict1[keys[i]]                         #To compare each and every
selected item with the item list
          if selection_items==item_list[0]:
            item_list[1]+=selection_qty                    #Since the customer
cancelled the order, the quantity must be added back to the stock
            dict1[keys[i]]=item_list
          else:
            pass

        net_bill=bill_amt_entry.get()
        net_bill=float(net_bill)
        net_bill=net_bill-float(selection_net_amt)       #Should be subtracted
from the current total amount
        bill_amt_entry.delete(0, END)
        net_bill=round(net_bill, 2)
        bill_amt_entry.insert(0, net_bill)


        items.delete(int(selection[0]))
        price_listbox.delete(int(selection[0]))
        qty_listbox.delete(int(selection[0]))
        net_price_listbox.delete(int(selection[0]))

        with open('Stock Burgers.dat', 'wb')as file:
          pickle.dump(dict1, file)
          file.close()
      except IndexError:
        pass
```

```
 def OK_button(self, Qty_entry, net_price_listbox, price_listbox,
balance_amt_entry, bill_amt_entry, paid_amt_entry, items, product_code,
qty_listbox):
    '''try:
      winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
    except RuntimeError:
      pass'''
    tot_bill_amt=float(bill_amt_entry.get())

    try:
      with open('Stock Burgers.dat', 'rb') as file:
        dict1=pickle.load(file)
        file.close()
      try:
        code=float(product_code.get())
        product_info=dict1[code]
        item=product_info[0]
        stock=product_info[1]
        price=product_info[2]
        try:
          qty_get=int(Qty_entry.get())
          if qty_get>stock:
            tkMessageBox.showinfo('mcDonalds', 'Qty entered is more
than current stock\n\nCurrent stock for "' + str(item) + '" is - ' +
                        str(stock))
          else:
            product_info=dict1[code]
            item=product_info[0]
            stock=product_info[1]
            price=product_info[2]
            new_stock1=stock-qty_get
            dict1[code]=[item, new_stock1, price]
            items.insert(END, item)
            qty_listbox.insert(END, qty_get)
            price='$ '+str(price)
            price_listbox.insert(END, price)
            product_code.delete(0, END)
            item_price=product_info[2]
            net_amt=item_price*qty_get
            tot_bill_amt+=net_amt
            tot_bill_amt=round(tot_bill_amt, 2)
            bill_amt_entry.delete(0,END)
            bill_amt_entry.insert(0, tot_bill_amt)
            net_amt='$ '+str(net_amt)
            net_price_listbox.insert(END, net_amt)
            Qty_entry.delete(0, END)
            with open('Stock Burgers.dat', 'wb') as file:
              pickle.dump(dict1, file)
              file.close()
        except ValueError:
          tkMessageBox.showinfo('McDonalds', 'Invalid Quantity entry')
      except ValueError:
        tkMessageBox.showerror('mcDonalds', 'Invalid Product code')
      except KeyError:
        tkMessageBox.showerror('mcDonalds', 'Product for this code
doesn\'t exist')
    except EOFError:
      tkMessageBox.showinfo('McDonalds', 'No products added')



  def balance_amt(self, discount_entry, balance_amt_entry, bill_amt_entry,
paid_amt_entry):
    '''try:
      winsound.PlaySound('Button Press.wav', winsound.SND_FILENAME)
    except RuntimeError:
      pass'''
    balance_amt_entry.delete(0, END)
    try:
      bill_amt=float(bill_amt_entry.get())
```

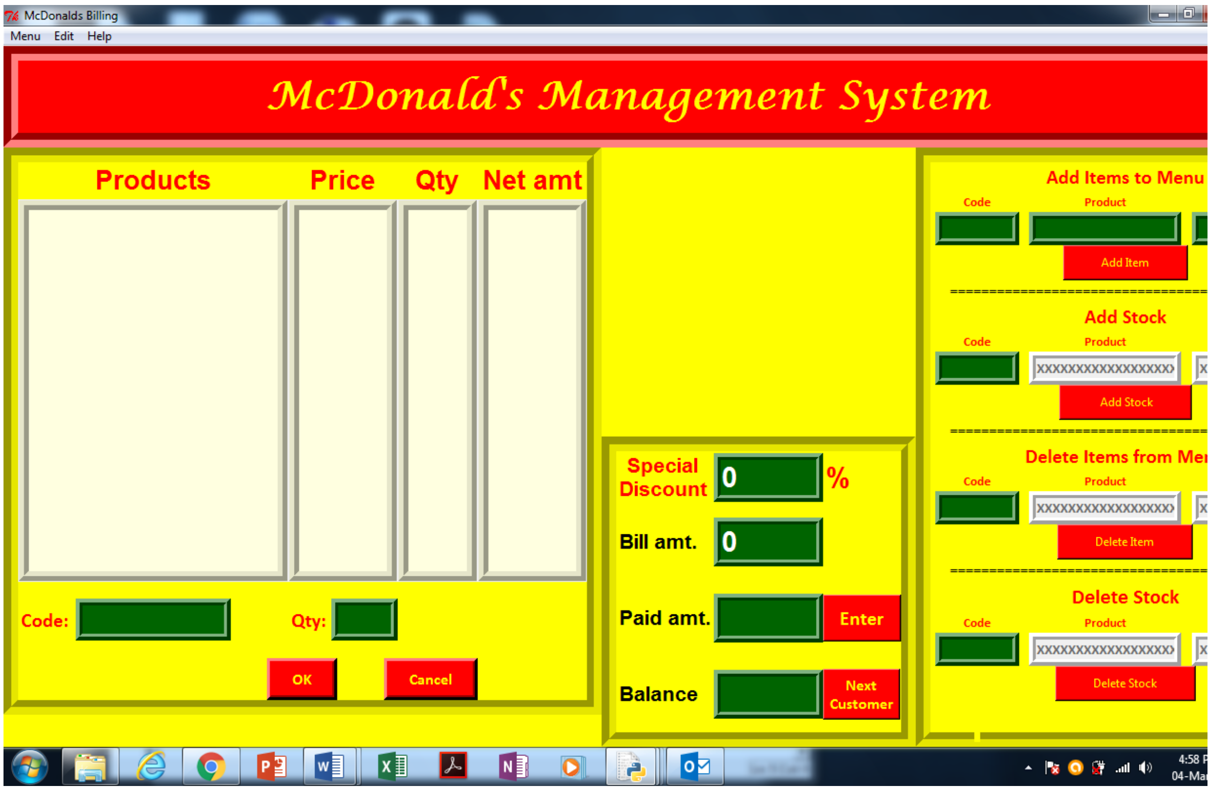
```
        paid_amt=float(paid_amt_entry.get())
        try:
          discount=float(discount_entry.get())
          if discount>100:
            tkMessageBox.showinfo('McDonald\'s', 'Invalid Discount
percent!\n\nShould be less than or equal to 100')
          else:
            final_discount=(discount/100.0)*bill_amt
            tot_bill=bill_amt-final_discount
            balance=paid_amt-tot_bill
            balance=round(balance,2)
            balance_amt_entry.insert(0, balance)
            '''try:
              winsound.PlaySound('Cash Register.wav',
winsound.SND_FILENAME)
            except RuntimeError:
              pass'''
        except ValueError:
          tkMessageBox.showinfo('invalid input', 'Invalid discount entry')
      except ValueError:
        tkMessageBox.showerror('McDonald\'s', 'Invalid Entry')




def main():
  try:
    with open('Stock Burgers.dat', 'rb') as file:
      try:
        dict1=pickle.load(file)
        file.close()
      except EOFError:
        dict1={}
        #mcD(dict1)
  except IOError:
    dict1={}
```

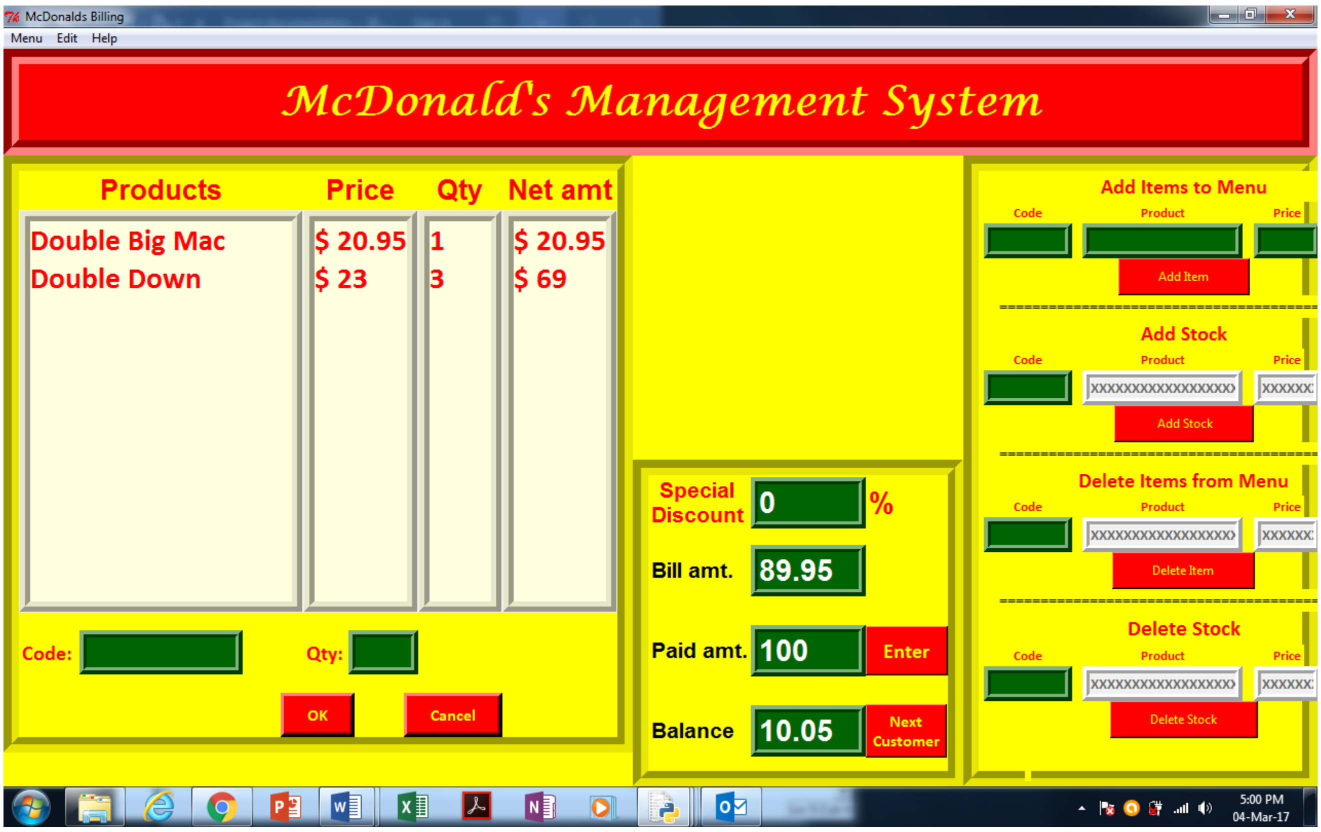
```
        with open('Stock Burgers.dat', 'wb') as file:
            pickle.dump(dict1, file)
            file.close()
            #mcD(dict1)
    finally:
        log=Login(dict1)
main()
```

SCREENSHOTS OF THE PROGRAM



```
32 bit (Intel)] on win
ation.
====================
```

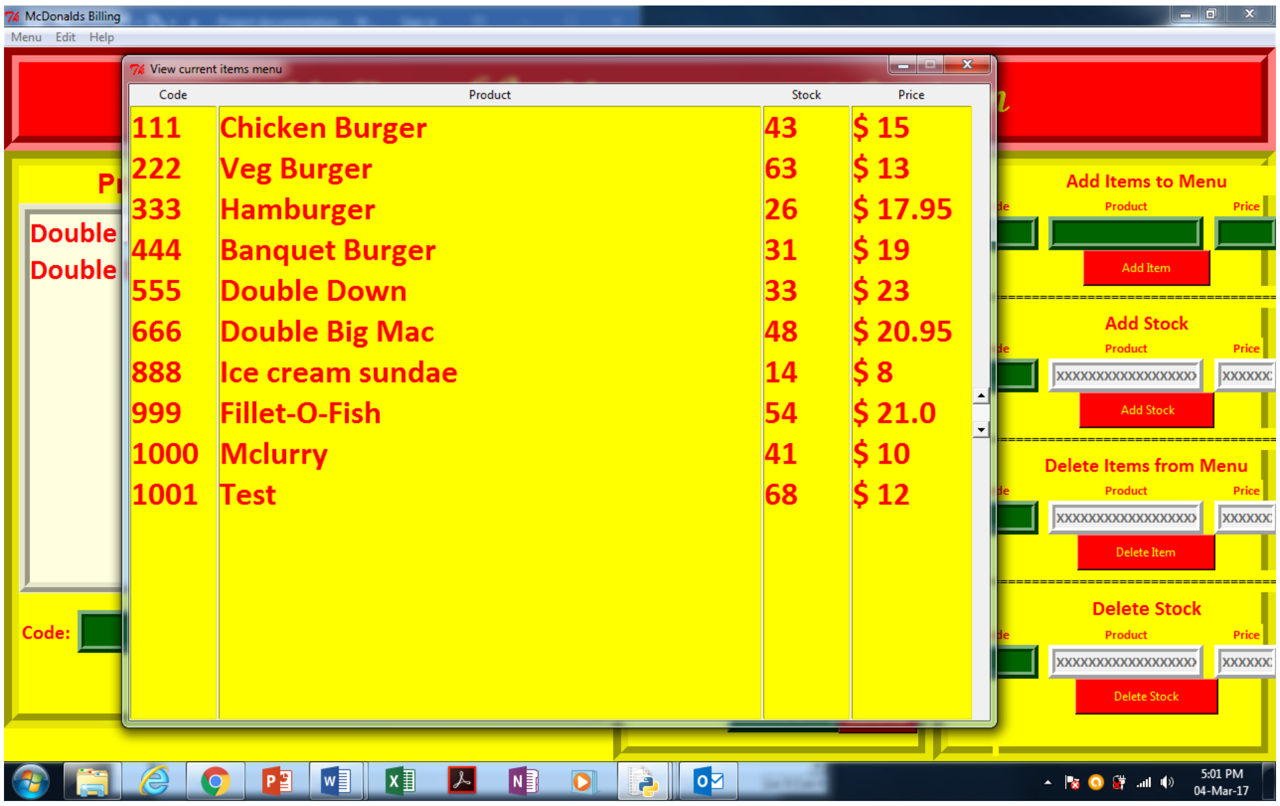




McDonald's Management System

Products	Price	Qty	Net amt
Double Big Mac	$ 20.95	1	$ 20.95
Double Down	$ 23	3	$ 69

Special Discount	0 %
Bill amt.	89.95
Paid amt.	100
Balance	10.05



Code	Product	Stock	Price
111	Chicken Burger	43	$ 15
222	Veg Burger	63	$ 13
333	Hamburger	26	$ 17.95
444	Banquet Burger	31	$ 19
555	Double Down	33	$ 23
666	Double Big Mac	48	$ 20.95
888	Ice cream sundae	14	$ 8
999	Fillet-O-Fish	54	$ 21.0
1000	Mclurry	41	$ 10
1001	Test	68	$ 12

LIMITATIONS OF THE PROJECT

- Does no include complex graphing and data monitoring system such as profit and loss, sales income etc.
- It lacks other necessary features that is required for a billing software.

BIBLIOGRAPHY

- Computer Science Textbook Grade 11 and 12
- https://www.tutorialspoint.com/python/python_gui_programming.htm - For features of python GUI.
- Python built-in documentation – For all python functions.