Fixing Tkinter

Background

This task is intended as practice for the **WJEC 2025 Computer Science** curriculum (specifically the new examination style for **Unit 2**).

This is a version of the PyShop activity written using Tkinter. The aim of the task is to understand the program and fix all errors present within, while also improving the learners' ability to provide good commenting and documentation.

In this program we have tried to include every programming technique (*good, bad and worse*) that the WJEC included in the **Sample Assessment Materials**. While not all of these techniques include errors that must be fixed, it is hoped that having to grasp the intended function of the code will aid learners in understanding the variety of Python programming styles.

Materials

The materials for this task - which includes both the completed program for reference by the educator and broken program for learners to fix, as well as a presentation to aid with content delivery - can be found at the following link**:**

Fixing Tkinter

Task 1 - Fixing Login

In the first task, learners must look at the **'gui_shop_login.py'** file and fix the **6 errors** present within. This is intended as an introductory task, as not to overwhelm by providing them with 4 broken modules of a large program.

However, to run and begin testing this file, learners must write a **main module** to call it. This is *essential Python knowledge*, as the majority of programs will be **decoupled** into modules; where a main module calls others and runs the program!

Main Module

Create a new Python file called 'gui_shop_main.py', saved in the same directory as the other modules, and include the following code:

```
from gui_shop_login import create_login

create_login()
```

Fixing Tkinter

Login Error #1

The stored username and password variables were left blank, both should have values to ensure only an authenticated user can access the program:

```
stored_username = "A.Username"
stored_password = "goodPassw0rd"
```

Login Error #2

The login check ensured that **either** the username **OR** password matched the stored variables. A login system should ensure that **both** match:

```
if ((username_entry.get() == stored_username) and
    (password_entry.get()== stored_password)):
```

Login Error #3

The window title was nonsense, a well-designed program should have informative names for each window:

```
login_root.title("Shop Program")
```

Fixing Tkinter

Login Error #4

Any Tkinter elements must be placed within a Frame using a **geometry manager**, in this case, we're using **pack**.

```
title_label = Label(login_frame,
                    text = "Welcome to Shop Login!")
title_label.pack()
```

Login Error #5

In the GUI the password field appeared to the left of the password label. This seems like a mistake and should be rectified so that the label is on the left.

```
password_label = Label(password_frame,
                       text = "password”)
password_label.pack(side = LEFT)
password_entry = Entry(password_frame)
password_entry.pack(side = RIGHT)
```

Fixing Tkinter

Login Error #6

When the Submit Button is pressed, it should run the **process_login** function (that means the **pointless_method** can be deleted).

```
submit_button = Button(login_frame,
                       text = "submit",
                       bg = "White", fg = "Red",
                       command = process_login)
```

Note that this function is provided without brackets (i.e. it is not calling the function), meaning it is only a **reference** to the function (i.e. a variable containing the function call). This is why process_login is **nested** within create_login, so that it is within scope to make a reference.

Try replacing the `command = pointless_method` with the `command = print("This method is pointless.")` within, and see what happens (it is immediately called upon running the program and the button no longer does anything)!

Fixing Tkinter

Task 2 - Commenting

The second task is to ensure that the code is well commented. The learners must add comments where needed in the 'gui_shop_login' module - though this task should continue throughout.

Commenting is an essential programming skill, and is included as example questions in the new curriculum.

Best commenting practice:

- **Don't comment every line of code** - instead space-out sections of code and comment those chunks that do a particular thing.
- **Comments shouldn't explain what the code is doing** - that's what the code is for, they should only explain the purpose. This can be the tricky one to understand, so this example might help:
    - **Bad Comment**

```
# Prints out 'Login Successful' then on the next line 'Hi' along
# with the username entered by the user
print("Login Successful!\nHi ", username)
```

    - **Good Comment**

```
# Welcome message for user
print("Login Successful!\nHi ", username)
```

- **Functions should be commented with """docs"""** - these should explain the functions purpose and outline any arguments (written as **Args:**) and/or **Return:** statements present in the function.

Fixing Tkinter

Task 3 - Main Module Overhaul

We need a way to **switch** between the login window and the shop face via the log-in/log-out buttons.

This is quite tricky to do when we're using Python **procedurally**, as we can run into errors by 'infinitely' switching windows (this is a RecursionError, as recursive functions are only permitted to go so many layers deep - Tkinter is **intended** to be used in an Object Oriented structure which removes many of the issues we come across!)

So, we'll need a **creative solution** to avoid the error! We need to **destroy** the login window **before switching** to the shop (so that no recursion occurs) - but if we do this within the login window the shop will never be called (as the window has been destroyed before calling the login)!

To get around this, we'll have to implement everything within 'gui_shop_main', by defining functions that destroy the current window and call the new window. This will require **passing** references to these functions into each window, as we previously saw in *Login Error #6*. Additionally, we will have to modify 'gui_shop_login' to accept and make use of this reference.

Fixing Tkinter

Main Module Overhaul

```
from gui_shop_login import create_login
from gui_shop_shopface import create_shopface

def switch_to_shop(login_root):
    login_root.destroy()
    create_shopface(switch_to_login)

def switch_to_login(shopface_root):
    shopface_root.destroy()
    create_login(switch_to_shop)

create_login(switch_to_shop)
```

Update Login Module

```
def create_login(switch_to_shop):

~~~

if ((username_entry.get() == stored_username) and
    (password_entry.get() == stored_password)):

    switch_to_shop(login_root)
```

Fixing Tkinter

Task 4 - Fixing the Shop Face

Now that the login window works as intended, learners can run the **'gui_shop_main.py'** file to test the program (they must have all files, including the **'gui_shop_shopface.py'**, **'gui_shop_filehandling.py'** and **'items.csv'** in the same directory).

This should now allow the user to log in and access the shop face, though this window will appear very broken. We will now have to fix the issues present in both **'gui_shop_shopface.py'** and **'gui_shop_filehandling.py'**.

This task can be provided as one large challenge, but learners may find it beneficial to solve the errors present one method at a time. This is a fairly large Python module, so it may feel overwhelming without guidance, as there are a total of **13 errors** to find and fix across 10 methods.

Fixing Tkinter

create_shopface - Shop Face Error #1, 2

Two errors exist within the create_shopface method (**'gui_shop_shopface.py'**).

The first of these requires the variables stock and total_sales to be declared as **global**. These variables are displayed within the GUI, so making them global and accessible by all methods is essential to having them update within the GUI.

The second error is that the stock variable is never set. This must be set by calling the load_stock method from **'gui_shop_filehandling.py'**.

```
def create_shopface(switch_to_login):

    # stock and total_sales defined as global...
    global stock
    global total_sales

    # shopface_frame is defined as global...
    global shopface_frame

    # Initialize stock dictionary by loading...
    stock = load_stock()
```

Fixing Tkinter

get_stock_file - FileHandling Error #1

To successfully load the stock into the shopface window, **2 errors** must be fixed within **'gui_shop_filehandling.py'**. The first of these simply requires the correct file name to be provided (the path is generated dynamically and *should* work on all devices).

```
def get_stock_file(mode):

    directory = os.path.dirname(os.path.abspath(
                                        __file__))

    stock_path = os.path.join(directory,
                                      "items.csv")
```

get_stock_file - FileHandling Error #2

The second requires that the price is loaded from the csv as a float - this is because it is a decimal value and cannot be typecast to an integer.

```
def load_stock():

  ~~~

  price = float(item[2].strip())
```

create_menu_bar - ShopFace Error #3

The third error can be found within the create_menu_bar method (**'gui_shop_shopface.py'**).

As seen in the login (specifically the **submit_button**), a command must be given to the button. In this case the **switch_to_login** method passed from **'gui_shop_main.py'** is provided, so that this window is destroyed and the login window called upon logging out.

Note the use of the **lambda** function, this allows a function to be passed to the button without immediately being called - in the login we solved this with a reference to a nested function, but here we require a parameter to be passed, so a reference would not suffice. This is a better solution, as nested functions are against convention.

```
def create_menu_bar(switch_to_login,
                                shopface_root):

  ~~~

  log_out_button = Button(
                   menu_frame,
                   text = "Log Out",
                   bg = "White", fg = "Red",
                   command = lambda:
                   switch_to_login(shopface_root))
```

Fixing Tkinter

create_stock_table - ShopFace Error #4, 5

There are **6 errors** to be found within the create_stock_table method (**'gui_shop_shopface.py'**), these appear consecutively and have been grouped into errors #4 & 5. All of these errors are simple mistakes in the value of which row to be displayed on.

```
def create_stock_table():

  ~~~

  id_label = Label(table_frame,
                   text = "Product ID").grid(
                   row = 0, column = 0)

  name_label = Label(table_frame,
                   text = "Name").grid(
                   row = 0, column = 1)

  ~~~

    for i, item in enumerate(stock):
        id_label = Label(table_frame,
                text = item).grid(
                row = i + 1, column = 0)
        name_label = Label(table_frame,
                text = stock[item]["Name"]).grid(
                row = i + 1, column = 1)
```

create_purchase_bar - ShopFace Error #6, 7

There are **2 errors** to be found within the create_purchase_bar method (**'gui_shop_shopface.py'**). The first of these is that the number of decimal places shown for the total_sales in the GUI is incorrect. This is defined with an f-string, so the formatting must be corrected.

The second error is again in the button. This button is even more complex than the last as the lambda function now contains an if statement - it checks whether the process_purchase_attempt method returns True, and if so it runs the process_purchase method (else it does None). This requires the correct parameter is passed to **process_purchase_attempt.**

```
def create_purchase_bar():

    ~~~

    sales_value = Label(purchase_frame,
          text = str(f"{total_sales.get():.2f}"))
    sales_value.grid(row = 1,column = 2)

    # Create a purchase button to make purchases
    purchase_button = Button(purchase_frame,
              text = "Make Purchase",
              bg = "White", fg = "Green",
              command = lambda: process_purchase(
              sales_value, purchase_entry.get())
              if process_purchase_attempt(
              purchase_entry.get()) else None)
```

Fixing Tkinter

update_stock_table - ShopFace Error #8

Only **1 error** exists within update_stock_table (**'gui_shop_shopface.py'**). After the current table is destroyed, another table must be created in its place.

```
def update_stock_table():

    global table_frame
    table_frame.destroy()
    create_stock_table()
```

process_purchase_attempt - ShopFace Error #9

Only **1 error** exists within the process_purchase_attempt method (**'gui_shop_shopface.py'**). If the attempt is unsuccessful due to an invalid input, then a **meaningful** error should be provided to the user..

```
def process_purchase_attempt(user_input):

    ~~~
    except (ValueError, KeyError):
        messagebox.showwarning(
                            "Enter a Valid Key!",
                            "Enter a Valid Key!")
        return False
```

Fixing Tkinter

process_purchase - ShopFace Error #10

Only **1 error** exists within process_purchase method (**'gui_shop_shopface.py'**). Whenever **global variables** are used within a Python method, they must be **redeclared** within that method (this is to inform Python that the global variable is being invoked, not a local variable with the same name)..

```
def process_purchase(sales_value, user_input):

    #...
    global stock
    global total_sales
```

write_stock - FileHandling Error #3

The final error exists back in **'gui_shop_filehandling.py'** within the write_stock method. In order to write any changes to file, the file must be opened in write mode, not read mode.

```
def write_stock(stock):

    stock_file = get_stock_file('w')
```