In [ ]:
```
import pandas as pd
from datetime import datetime, timedelta, date
import numpy as np
from openpyxl import load_workbook
from openpyxl.workbook import Workbook
from openpyxl.utils.dataframe import dataframe_to_rows
import uuid
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

#File path information(loading excel file)
file_path = "Youre\\file\\path"
def load_from_excel(file_path):
  try:
    df = pd.read_excel(file_path)
    return(df)
  except FileNotFoundError:
    print("File does not exist.")
  except PermissionError:
    print("This file is currently open close and try again.")
df = load_from_excel(file_path)
```

In [ ]:
```
#Function to generate transaction id number
def generate_transaction_id():
    long = str(uuid.uuid4()) #the default format is too long so just halved it
    short = long[:8]
    return(short)
generate_transaction_id()
```

In [ ]:
```
def append_df_to_excel(df, file_path, sheet_name='Sheet1'):
    try:
        wb = load_workbook(file_path)  # Load the existing workbook
        ws = wb[sheet_name] if sheet_name in wb.sheetnames else wb.create_sheet(sheet_name)  # Get the right she
    except FileNotFoundError:
        print("Workbook does not exist. Creating a new one.")
        wb = Workbook()
        ws = wb.active
        ws.title = sheet_name
        for c_idx, header in enumerate(df.columns, start=1):
            ws.cell(row=1, column=c_idx, value=header)

  #I was having a duplicate issue so this checks to make sure that there arent any duplicates
    existing_ids = [row[0].value for row in ws.iter_rows(min_row=2, max_col=1, max_row=ws.max_row)]

    # Append only new rows that do not have a transactionID already in the sheet
    for r_idx, row in enumerate(dataframe_to_rows(df, index=False, header=False), start=ws.max_row+1):
        transaction_id = row[1]  # Assumes transactionID is the first column in the DataFrame
        if transaction_id not in existing_ids:
            for c_idx, value in enumerate(row, start=1):
                ws.cell(row=r_idx, column=c_idx, value=value)

    wb.save(file_path)
    print("Data successfully appended to {} in '{}' sheet.".format(file_path, sheet_name))
```

In [ ]:
```
#Im going to stack the application into one single formula this will serve as the head.
def introduction():
    while True: #Start of while loop
        try: #try to convert input into an int
            intro = int(input("Hi. Type 1 to quit. If you'd like to track your package information please login
            if intro in [1, 2, 3]: #if intro is in the list move on to the following function
                return intro
            else:
                print("Please enter a valid option: 1, 2, or 3.") #if conversion fails go back to the top promp
        except ValueError:
            print("Please enter a valid number")
```

In [ ]:
```
def existing_appointment():
    global df #none of the functions take arguments so like in Java or C i have to define the df as a global va
    #it anywhere as I please

    while True:
      username = input("Type Q to quit or Enter username: ").capitalize() #capitalize the username for uniformi
      if username == "Q": #checking to exit loop
          print("Ending program...")
          break

      #User validation
      if username in df['username'].values:  #checks if entered username is in the list
          current_user = df[df['username'] == username]
          password = int(input("Enter password: "))
```

```
        if password == current_user['password'].iloc[0]:  #if password matches to the first(theoretically only
            print("Login Successful") #print Login Successful and their order information
            print(current_user[['name', 'product', 'selling_price', 'order_date', 'delivery_date']])
            print("To make any changes, please contact the sales department at barbershopbook@gmail.com. ")
            break
        else:
            print("Incorrect password.")
    else:
        print("Username not found.")
```

In [ ]:
```
def newly_created_purchase():
    global df # Using the global dataframe to append the new purchase

    while True:
        new_name = input("Enter your name: ").capitalize()  # Ensure name starts with a capital letter for unifo
        # Ensure the name is made up of only letters or spaces
        if all(char.isalpha() or char.isspace() for char in new_name):
            break
        else:
            print("The name should only contain letters. Please try again.")

    new_username = input("Enter your new username: ").capitalize()  # Again, for uniformity

    # New password criteria checking loop
    while True:
        new_password = input("Enter your new password (minimum 10 characters and at least one digit): ")
        if len(new_password) >= 10 and any(char.isdigit() for char in new_password):
            print("Password accepted")
            break
        else:
            print("Password must be at least 10 characters long and contain at least one digit. Please try agai

    product_map = {"XL rug": 150, 'M rug': 100, 'S rug': 75}
    products = list(product_map.keys())
    print("Which rug are you interested in?")
    for number, product in enumerate(products, start=1):
        print(f"{number}. {product}: ${product_map[product]}")

    while True:
        try:
            product_choice = int(input("Please select a product by entering the number: "))
            if 1 <= product_choice <= len(products):
                selected_product = products[product_choice - 1]
                break
            else:
                print("Invalid selection. Please choose a number from the list.")
        except ValueError:
            print("Please enter a number.")

    product_price = product_map[selected_product]

    try:
        new_shipping_method = int(input("We only offer 2-day shipping(1) and standard shipping 5 days(2): "))
        if new_shipping_method == 1:
            new_shipping_method = "Air"
        elif new_shipping_method == 2:
            new_shipping_method = "Road"
    except ValueError:
        print("Please enter a valid option for shipping method.")
        return  # Exit the function early if there's a ValueError

    new_location = input("Which state are you currently located in? ")

    while True:
        new_email = input("What email can we reach you at, to skip press (1)? ")
        if new_email == "1":
            new_email = np.nan  # Use np.nan for missing values
            break
        elif "@" not in new_email or "." not in new_email:
            print("Email not valid. Please enter a valid email address.")
        else:
            break

    new_number = input("What number can we reach you at, to skip press (1)? ")
    if new_number == "1":
        new_number = np.nan  # Use np.nan for missing values

    new_transaction_id = generate_transaction_id()
    new_purchase = pd.DataFrame({
        "userID": [df['userID'].max() + 1],
        'transactionID': [new_transaction_id],
        'name': [new_name],
        'username': [new_username],
        'password': [new_password],
        'email': [new_email],
        'number': [new_number],
        'order_date': [datetime.today().strftime('%D/%M/%Y')],
        'product': [selected_product],
        'selling_price': [product_price],
        'shipping_method': [new_shipping_method],
        'location': [new_location],
    })

    df = pd.concat([df, new_purchase], ignore_index=True)
    append_df_to_excel(df, file_path, sheet_name='Sheet1')
```

In [ ]:
```
def application():
  while True:
    intro = introduction()
    if intro == 1:
      break
    elif intro == 2:
      existing_appointment()
      break
    elif intro == 3:
      newly_created_purchase()
      break
    else:
      print("Not a valid choice")
      continue
```

In [ ]:
```
application()
```

In [ ]:
```
today = date.today()
df['order_date'] = pd.to_datetime(df['order_date']).dt.date
on_email_list = df[(df['email'].notna()) & (df['order_date'] == today)]

def send_emails_for_confirmation(on_email_list):
    sender_email = "Company@gmail.com" #Company email address
    sender_password = "CompanyPassword@example.com" #Email address app key
    smtp_server = "smtp.gmail.com" #Hosting site
    port = 587

    for index, order in on_email_list.iterrows():
        receiver_email = order['email']
        name = order['name']
        expected_delivery_date = order['delivery_date'].strftime('%m/%d/%Y')
        transaction_id = order['transactionID']


        message = MIMEMultipart("alternative")
        message["Subject"] = "Order Confirmation" #Subject
        message["From"] = sender_email #Who will be receiving the email \ Both defined earlier in this block
        message["To"] = receiver_email #Who will be sending the email   /


  #Body of email
        text = "Hi {},\nThis message is to inform you we received your order.".format(name)
        html = f"""\
            <html>
              <body>
                <p>Hi {name},<br>
                Attached is your transaction number, which can be used in reference to your current order <b>{t
                we'll send an email once we've shipped it as well as the day its expected to arrive <b>{expecte
                Thank you again for your business if you need to make any changes you can reach us directly here
                via this email address. Thanks.
                </p>
              </body>
            </html>
            """
        part1 = MIMEText(text, "plain")
        part2 = MIMEText(html, "html")
        message.attach(part1)
        message.attach(part2)

        try:
            with smtplib.SMTP(smtp_server, port) as server:
                server.starttls()  # Secure the connection
                server.login(sender_email, sender_password)
                server.sendmail(sender_email, receiver_email, message.as_string())
                print("Email sent to ID {}: {}".format(transaction_id, receiver_email ))
        except Exception as e:
            print("Error sending email to {}: {}".format(receiver_email, e))
```

In [ ]:
```
today = date.today()
```

```python
df['shipping_date'] = pd.to_datetime(df['shipping_date']).dt.date
ship_date_email_list = df[(df['email'].notna()) & (df['shipping_date'] == today)]

def send_emails_for_shipping(ship_date_email_list):
    sender_email = "company@gmail.com" #Company email address
    sender_password = "CompanyPassword@example.com" #Email address app key
    smtp_server = "smtp.gmail.com" #Hosting site
    port = 587

    for index, order in ship_date_email_list.iterrows():
        receiver_email = order['email']
        name = order['Name']
        shipped_date = order['shipping_date'].strftime('%m/%d/%Y')
        transaction_id = order['transactionID']


        message = MIMEMultipart("alternative")
        message["Subject"] = "Order Confirmation" #Subject
        message["From"] = sender_email #Who will be receiving the email \ Both defined earlier in this block
        message["To"] = receiver_email #Who will be sending the email   /


  #Body of email
        text = "Hi {},\nThis message is to inform your order will be shipped today.".format(name)
        html = f"""\
            <html>
              <body>
                <p>Hi {name},<br>
                In regards to your order <b>{transaction_id}</b> it left our facilities today <b>{shipped_date}
                Thank you again for your business. You can reach us directly here <b>(356) 123-4567<b> or
                via this email address. We'll send another email to inform you when its out for delivery Thanks
                </p>
              </body>
            </html>
            """
        part1 = MIMEText(text, "plain")
        part2 = MIMEText(html, "html")
        message.attach(part1)
        message.attach(part2)

        try:
            with smtplib.SMTP(smtp_server, port) as server:
                server.starttls()  # Secure the connection
                server.login(sender_email, sender_password)
                server.sendmail(sender_email, receiver_email, message.as_string())
                print("Email sent to ID {}: {}".format(transaction_id, receiver_email ))
        except Exception as e:
            print("Error sending email to {}: {}".format(receiver_email, e))
```

In [ ]:

```python
df['delivery_date'] = pd.to_datetime(df['delivery_date']).dt.date
delivery_list = df[(df['email'].notna()) & (df['delivery_date'] == today)]

def send_emails_for_delivery(delivery_list):
    sender_email = "company@gmail.com" #Company email address
    sender_password = "CompanyPassword@example.com" #Email address app key
    smtp_server = "smtp.gmail.com" #Hosting site
    port = 587

    for index, order in delivery_list.iterrows():
        receiver_email = order['email']
        name = order['Name']
        delivery_date = order['delivery_date'].strftime('%m/%d/%Y')
        transaction_id = order['transactionID']


        message = MIMEMultipart("alternative")
        message["Subject"] = "Order Confirmation" #Subject
        message["From"] = sender_email #Who will be receiving the email \ Both defined earlier in this block
        message["To"] = receiver_email #Who will be sending the email   /


  #Body of email
        text = "Hi {},\nThis message is to inform you we received your order.".format(name)
        html = f"""\
            <html>
              <body>
                <p>Hi {name},<br>
                Hi this message is in reference to your current order <b>{transaction_id}</b>
                it is currently out for delivery and should be delivered on this date <b>{delivery_date}<b>.
                Thank you again for your business have any questions changes you can reach us directly here <b>
                via this email address. Thanks.
                </p>
              </body>
            </html>
            """
        part1 = MIMEText(text, "plain")
        part2 = MIMEText(html, "html")
        message.attach(part1)
        message.attach(part2)

        try:
            with smtplib.SMTP(smtp_server, port) as server:
                server.starttls()  # Secure the connection
                server.login(sender_email, sender_password)
                server.sendmail(sender_email, receiver_email, message.as_string())
                print("Email sent to ID {}: {}".format(transaction_id, receiver_email ))
        except Exception as e:
            print("Error sending email to {}: {}".format(receiver_email, e))
```

In [ ]:

```
def send_emails(df):
    df['order_date'] = pd.to_datetime(df['order_date']).dt.date
    df['shipping_date'] = pd.to_datetime(df['shipping_date']).dt.date
    df['delivery_date'] = pd.to_datetime(df['delivery_date']).dt.date
    today = datetime.date.today()

    on_email_list = df[(df['email'].notna()) & (df['order_date'] == today)]
    if not on_email_list.empty:
        send_emails_for_confirmation(on_email_list)

    ship_date_email_list = df[(df['email'].notna()) & (df['shipping_date'] == today)]
    if not ship_date_email_list.empty:
        send_emails_for_shipping(ship_date_email_list)

    delivery_list = df[(df['email'].notna()) & (df['delivery_date'] == today)]
    if not delivery_list.empty:
        send_emails_for_delivery(delivery_list)
```

In [ ]:

```
send_emails(df)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js