

I confirm that this assignment is my own work.

Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.

Programming assignment

Part 1

Bora Sozer  
P438457

Contents

[Weather Tracker Program 2](#_Toc160398141)

[Introduction 2](#_Toc160398142)

[Section 1 :Algorithm 3](#_Toc160398143)

[Section 2: Technical Overview 4](#_Toc160398144)

[Section 3: Python Code with comments 5](#_Toc160398145)

[Section 4: Testing 7](#_Toc160398146)

[Testing for development: 7](#_Toc160398147)

[Testing for evaluation: 8](#_Toc160398148)

[Section 5: Evaluation and Summary 10](#_Toc160398149)

[Section 6: References 10](#_Toc160398150)

[Event Booking App 11](#_Toc160398151)

[Introduction 11](#_Toc160398152)

[Section 1: Algorithm 12](#_Toc160398153)

[Main 12](#_Toc160398154)

[Functions 13](#_Toc160398155)

[Section 2: Technical Overview 16](#_Toc160398156)

[Section 3: Python Code with comments 19](#_Toc160398157)

[Example CSV File 19](#_Toc160398158)

[Code 19](#_Toc160398159)

[Section 4: Testing 26](#_Toc160398160)

[Testing for development: 26](#_Toc160398161)

[Testing for evaluation: 27](#_Toc160398162)

[Section 5: Evaluation and Summary 29](#_Toc160398163)

[Section 6: References 29](#_Toc160398164)

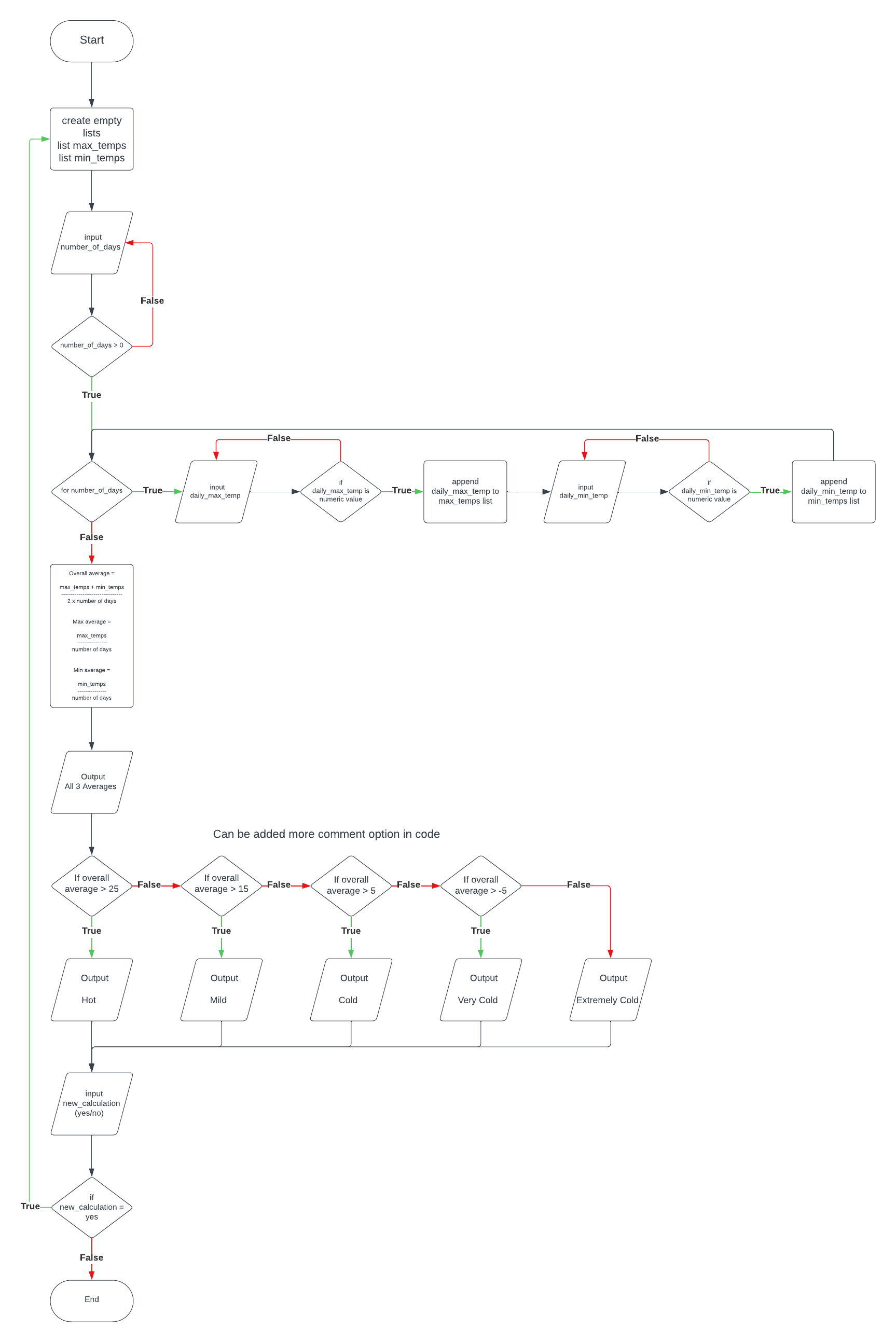
# Weather Tracker Program

## **C:\Users\hakan\Desktop\cold-and-hot-icon-temperature-illustration-sign-thermometer-symbol-heat-logo-vector.jpg**Introduction

The main purpose of this Python programme is to take the daily maximum and minimum temperature values for each day for a predetermined period of time from the user, calculate the average and comment according to the average.

It is user friendly and also prevents the programme from crashing in case of an incorrect input. Each part is explained as a comment in the code so that the programme can be improved in the future.

## Section 1 :Algorithm



## Section 2: Technical Overview

|  |  |  |
| --- | --- | --- |
| **Variables** | **Type** | **Description** |
| days | integer | number of days to be calculated |
| max\_temp | float | daily maximum temperature value |
| min\_temp | float | daily minimum temperature value |
| average\_max | float | average of maximum temperature values (max\_temps / days) |
| average\_min | float | average of minimum temperature values (min\_temps / days) |
| average\_overall | float | average of maximum and minimum temperature values ((average\_max + average\_min) / 2) |
| max\_temps | list | list which stores all daily max\_temp values |
| min\_temps | list | list which stores all daily min\_temp values |

There is no defined function.(def())

|  |  |
| --- | --- |
| **Function** | **Description** |
| If-else | Used if-else function to check whether the number of days selected is greater than 0 and in order to be able to make comments according to the calculated average |
| While | Used while function to get input again after errors, to repeat the code when the user chooses to perform a new tracking after the tracking |
| For | Used for function to receive input for the selected number of days |
| Append | Used list.append(value) function to add value to the list.  max\_temps.append(max\_temp)  low\_temps.append(low\_temp) |
| Int() | A function used to convert a value to an integer. |
| Float() | A function used to convert a value to decimal. |
| Sum() | A function used to calculate the sum of the elements. |
| Print() | Used to print output to the screen. |
| Exit() | It is a function used to terminate the programme. |

## Section 3: Python Code with comments

while True:

#Stored the values in two lists because objective says "the program should calculate the average for both the highs and the lows over the set period"

#the lists are in a while loop at the beginning to reset the list on each new calculation

max\_temps = []

min\_temps = []

#The purpose of this loop is to learn number of days from user

while True:

#Used try-except function to prevent a programme error when a value other than a number is entered.

try:

days = int(input("\nNumber of days:"))

#Wanted the number to be greater than 0 because we need at least 1 day to calculate it

if days <= 0:

print("Please enter a valid number of days (greater than 0).")

else:

print("")

break

except ValueError:

print("Not a valid number.")

#Input and store temperatures from user

for day in range(days):

#while loop to request data entry again for the same day if the min value entered is greater than max

while True:

#get maximum temperature

while True:

#Showed user-friendly error messages and allow re-entry if inputs are invalid

try:

#Used a float function in case the temperatures might be fractional

max\_temp = float(input(f"{day+1}.day maximum temperature:"))

#Limited the entry by taking the lowest and highest air temperature measured in the world because wanted to make an program that could be used anywhere in the world

if -90 < max\_temp < 60:

break

else:

print("Invalid temperature value. Please enter a temperature between -90 and 60 degrees Celsius.")

continue

except ValueError:

print("Please enter a valid value.")

#get minimum temperature

while True:

try:

min\_temp = float(input(f"{day+1}.day minimum temperature:"))

if -90 < min\_temp < 60:

break

else:

print("Invalid temperature value. Please enter a temperature between -90 and 60 degrees Celsius.")

continue

except ValueError:

print("Please enter a valid value.")

#Ensuring that minimum temperature value not higher than same day maximum temperature value

if min\_temp <= max\_temp:

max\_temps.append(max\_temp)

min\_temps.append(min\_temp)

break

else:

print("Minimum temperature can't be higher than maximum temperature. Please rewrite temperatures for day", day + 1, "\n")

#Calculate and print average

average\_max = sum(max\_temps) / days

average\_min = sum(min\_temps) / days

average\_overall = (average\_max + average\_min) / 2

#print fraction point limited using ":.1f" to avoid showing long fractional values

print(f"\nAverage max temperature:{average\_max:.1f}")

print(f"Average overall temperature:{average\_overall:.1f}")

print(f"Average min temperature:{average\_min:.1f}\n")

#Comment based on the average temperature.

#I tried to interpret the average both for myself and for the UK.

if average\_overall > 35:

print("It's extremely hot on average.")

elif average\_overall > 30:

print("It's quite hot on average.")

elif average\_overall > 25:

print("It's warm on average.")

elif average\_overall > 20:

print("It's moderately warm on average.")

elif average\_overall > 15:

print("It's mild on average.")

elif average\_overall > 10:

print("It's cool on average.")

elif average\_overall > 5:

print("It's cold on average.")

elif average\_overall > 0:

print("It's very cold on average.")

elif average\_overall > -5:

print("It's extremely cold on average.")

elif average\_overall > -10:

print("It's dangerously cold on average.")

else:

print("It's deadly cold on average.")

#Learn from the user whether to continue the calculation or exit the app

new\_calculation = input("\n1-New calculation\n2-Quit\nType number:")

while not new\_calculation in ["1","2"]:

new\_calculation = input("\nInvalid choice. Write 1 or 2:")

if new\_calculation == "2":

exit()

## Section 4: Testing

### Testing for development:

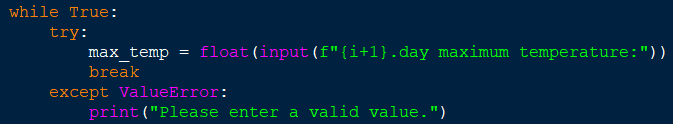
**Problem 1:**

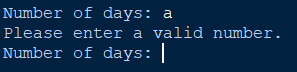
In order to collect the values entered as temperature, it must be converted to integer. Python int() or float()function was giving an error when an input other than a number was made.(Figure 1)

C:\Users\hakan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_8.png

**Figure 1.**

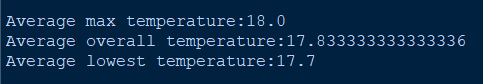
I solved this problem as Figure 2-3 with the try-except function that I searched and found from *“pythonbasics.org/try-except/”*.



**Figure 2.**

**Figure 3.**

**Problem 2:**

****Since I also share the average result with the user, if the result was long fractional, the output was not user-friendly and unnecessary. (Figure 4)

**Figure 4.**

I solved this by displaying a digit after the comma using “.1f “

C:\Users\hakan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_11.png “.1f “is a format specifier that formats floating-point numbers with one decimal place. This means it displays only one decimal place after the decimal point. (Figure 5-6) **Figure 5.**

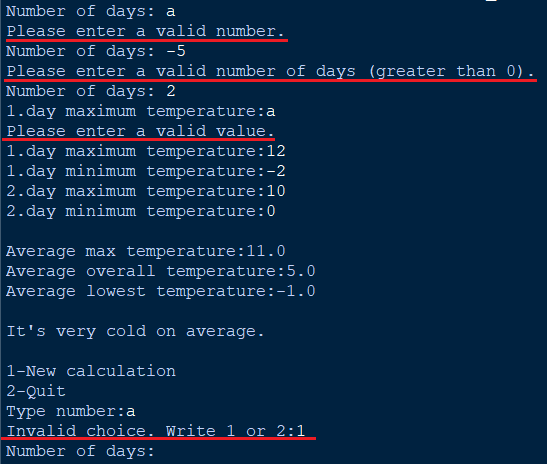


**Figure 6.**

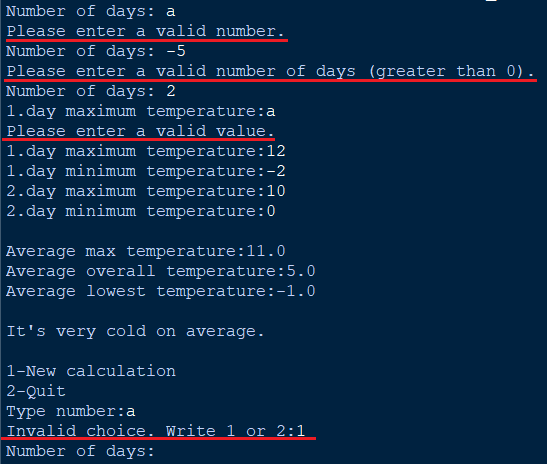
### Testing for evaluation:

Program was tested for every possibilities. I have tried with different inputs to make the programme give an error, but thanks to the functions I use, the programme can solve the problem without giving an error.

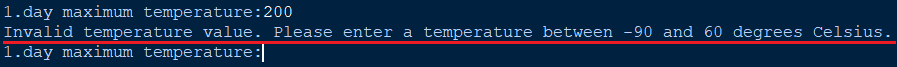
|  |  |  |
| --- | --- | --- |
| **Purpose of test** | **How the test was carried out** | **Outcome from that test** |
| **Validate user input for number of days**  Ensure the program does not accept negative or non-numeric values. | Negative number and non-numeric values tried as input from user | Program warned the user that they had entered invalid and asked them to type again.  **(Figure 7.)** |
| **Validate user input for temperature values**  Ensuring that only numeric input is accepted and that temperatures are within reasonable ranges. | Non-numeric values and unreasonable temperature input tried | Program warned the user that they had entered invalid or unreasonable values and asked them to try again.  **(Figure 8-9.)** |
| **Validate user input for temperature values**  Ensuring that minimum temperature value not higher than same day maximum temperature value. | Tried to enter minimum temperature value which is higher than same day maximum temperature value | Program warned the user about an error related to the values and asked them to rewrite them.  **(Figure 10.)** |
| **Validate user input for new calculation choice**  Ensuring that only options is accepted. | Entered a choice that is not in the options | Program warned the user that they had entered invalid choice and asked them to type again.  **(Figure 11.)** |
| **Validate that the programme works properly and calculates**  Ensure that outputs are user-friendly | Correct data input was made | Correct calculation, logical interpretation and output  **(Figure 12.)** |

It show warning from application when try to input invalid values

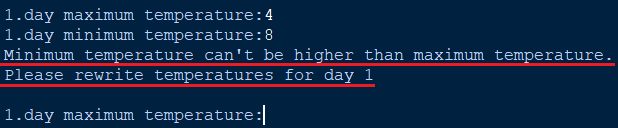
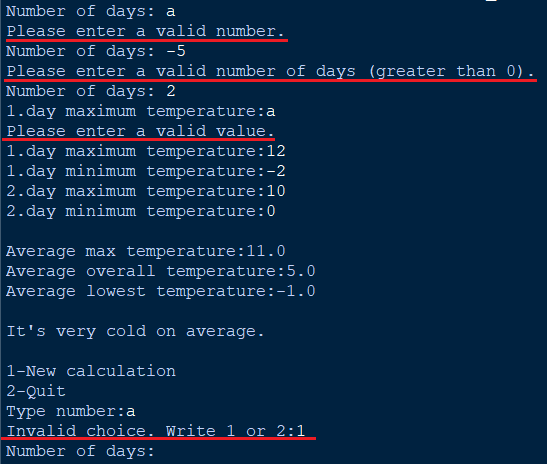
**Figure 7.**



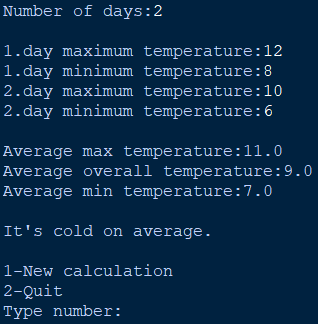
**Figure 8.**



**Figure 9.**

 **Figure 10.**

**Figure 11.**



**Figure 12.**

## Section 5: Evaluation and Summary

Programme fulfils all the desired conditions and successfully fulfils the objective. It receives input data from the user in a simple way. It validate user inputs, ensuring they are numeric values. If not, it output user-friendly warning message and wants re-input. Stores all values and calculates the overall average temperature. Makes comments to the user according to the overall average temperature. Outputs the average of the daily maximum, minimum, overall degrees. If there are too many fractions, it rounds and leaves a single fraction.

The programme is in such a way that **everyone can understand** and use it. Input and output parts are **user friendly**. All vulnerabilities that would cause the programme to crash have been closed. Also checked in tests and passed successfully.

If the programme is developed in the future, it can make user-based comments at an advanced level. For example, when the application is opened, a question can be asked to measure the user's temperature perception and comments can be made accordingly. It checks that the number of days is greater than 0 and temperature is within reasonable values. I also stored the temperature values in a list instead of adding them to a single variable one after the other, because if the programme is developed in the future, it may be desired to use the temperature values day by day. For example, to find the hottest and coldest day.

## Section 6: References

1. high – low temperature picture on introduction:

https://www.vecteezy.com/vector-art/22962714-cold-and-hot-icon-vector-temperature-illustration-sign-thermometer-symbol-heat-logo

1. LucidChart for Weather Tracking App:

<https://lucid.app/lucidchart/96a4a116-01ea-475d-b4dd-1b95abbce24a/edit?invitationId=inv_d328f64b-71a6-4913-84bc-ffb52b48d285>

1. Used to understand try-except error solving:

*Try and except in Python* (no date) *Try and Except in Python - Python Tutorial*. Available at: <https://pythonbasics.org/try-except/> (Accessed: 28 February 2024).

# Event Booking App

## **C:\Users\hakan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Adsız tasarım (4).png**Introduction

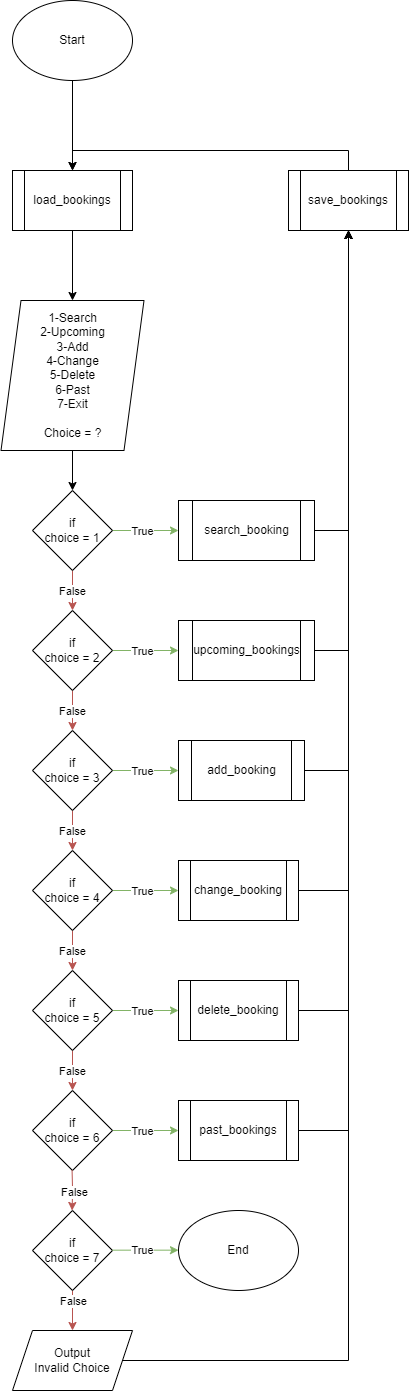
The purpose of this program is to work as an event registration program.

There are options to add, modify and delete bookings. There are also features that allow you to see past or future bookings. It is also possible to search for events by event date or name.

It stores the information as CSV file. While all this is being done, the programme does not give an error at any stage

## Section 1: Algorithm

### Main

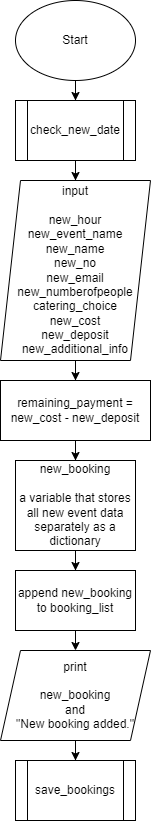
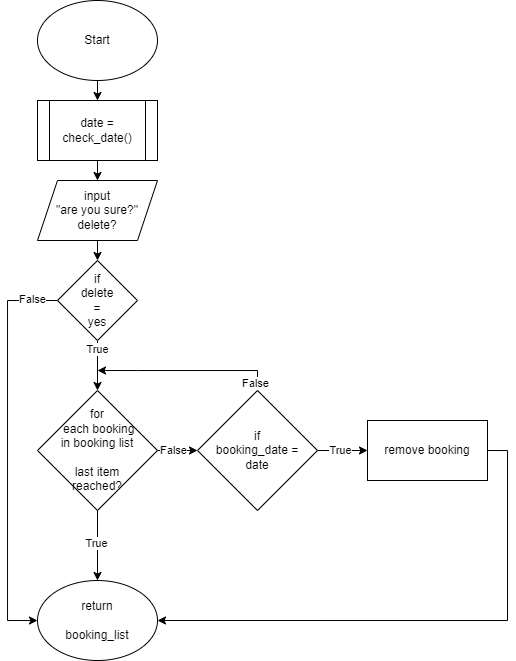


### Functions

Flowcharts show the general logic and do not contain every detail. Because I made the flowchart to be of general help while preparing the application. Then I improved the application by adding functions that were not in the flowchart to the application.

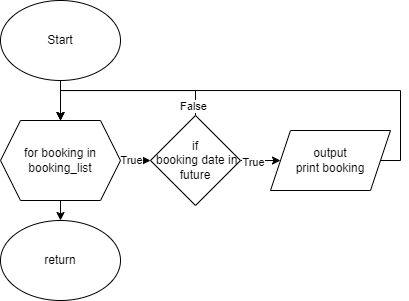
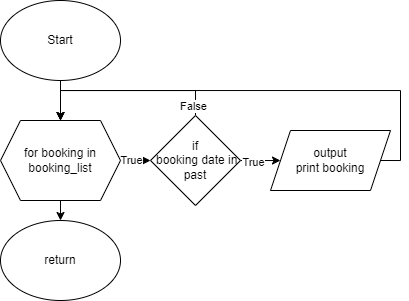
add\_booking

delete\_booking



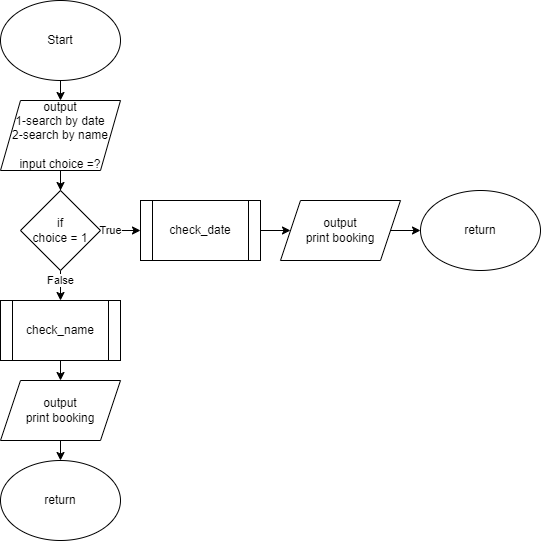
past\_bookings

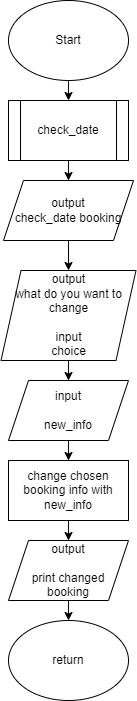
upcoming\_bookings



search\_booking

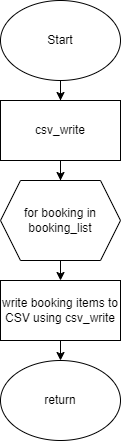
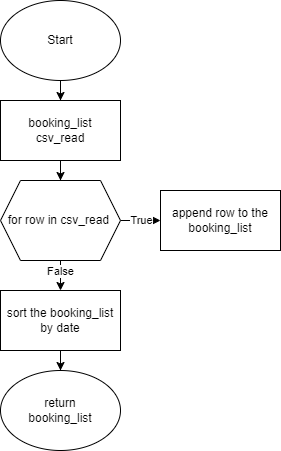
change\_booking





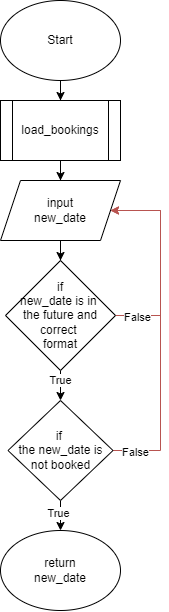
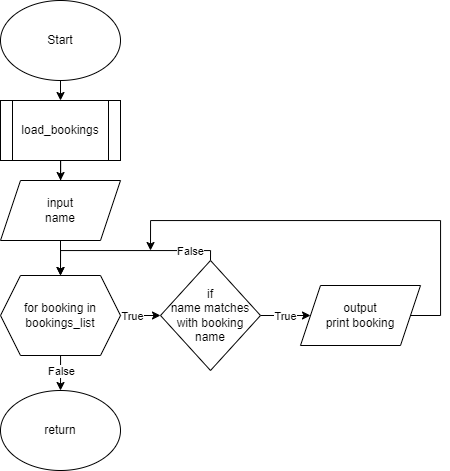
save\_bookings

load\_bookings

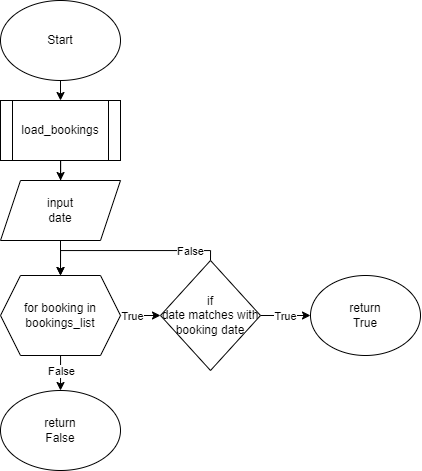


check\_name

check\_new\_date



check\_date



## Section 2: Technical Overview

The purpose of most variables used is to store the input received from the user before it is saved. Some of them store information from functions or enable functions to run. Data structure (booking\_list) provides the connection between csv and the application.

You can see all variables, data structures and functions in detail below.

|  |  |  |
| --- | --- | --- |
| **Variables** | **Type** | **Description** |
| filename | string | it contains CSV filename  filename = "bookings.csv" |
| csv\_read | \_csv.reader | variable which contains “.reader()” function  csv\_read=csv.reader(booking\_csv) |
| rows | list | a variable representing each row in the CSV file. |
| new\_date\_str | string | stores user input for booking date before save to the list |
| new\_date | string | variable which stores “.strptime()” function version of new\_date\_str |
| new\_hour | string | stores user input for booking hour before save to the list |
| new\_event\_name | string | stores user input for event name before save to the list |
| new\_name | string | stores user input for contact name before save to the list |
| new\_no | string | stores user input for booking date before save to the list |
| new\_email | string | stores user input for contact email before save to the list |
| new\_numberofpeople | string | stores user input for number of people before save to the list |
| new\_catering | boolean | first stores user input for catering choice after stores boolean value for catering choice before save to the list |
| new\_cost | string | stores user input for event cost before save to the list |
| new\_deposit | string | stores user input for event deposit before save to the list |
| new\_remaining\_payment | string | stores user input for remaining payment before save to the list |
| new\_additional\_info | string | stores user input for additional info before save to the list |
| choice | string | a variable used for the user to select from the menu. |
| date\_str | string | stores user input for booking date before checking possible date matches |
| date | datetime.datetime | variable which stores “.strptime()” function version of date\_str |
| searched\_name | string | stores user input for booking name before checking possible name matches |
| change | string | a variable containing the number of the item the user wants to change. |
| info\_choice | string | a variable that specifies how the additional information is to be processed. |
| delete | string | a variable used to confirm the deletion. |
| event\_date | string | a variable used to find upcoming or past bookings by comparing event dates with today's date to print upcoming and past bookings |
| booking\_list | list | list which stores all bookings as dictionary |
| bookin\_list items | dictionary | booking\_list items is dictionaries which stores event info |

|  |  |
| --- | --- |
| **Defined Functions** | **Description** |
| load\_bookings(filename | Loads the data from the CSV file as a list.  Each booking is represented as a dictionary and these dictionaries are aggregated into a list named booking\_list. Finally, the bookings are sorted by date and this sorted list is returned. |
| save\_bookings(filename, booking\_list) | Saves the bookings in the list to a CSV file. |
| check\_new\_date(booking\_list) | Checks the date for a new booking or change date for existing booking.  If the date is valid or not booked, it returns the new date. |
| add\_booking(booking\_list) | Adds a new booking.  Gets the required information from the user and creates a new booking dictionary with this information.  Adds the created booking dictionary to the booking\_list and saves it with calling save\_bookings function. |
| check\_date(booking\_list) | This function allows to find existing booking by date in search, change, delete booking sections. |
| check\_name(booking\_list) | This function allows to find existing booking by name in search booking section. |
| change\_booking(booking\_list) | Change booking details.  Prompts the user to select the date on which they want to change the booking information and then select which information to change. |
| main() | Executes the main programme loop.  It offers options to the user and calls the relevant function according to the user's choice.  Contains the code which helps functions to work and which no function is required for options. |

|  |  |
| --- | --- |
| **Functions** | **Description** |
| datetime.**strptime**(new\_date\_str, "%d-%m-%Y") | It converts the entered date string in its own format to check the input |
| **lambda** x: datetime.strptime(x["Date"], "%d-%m-%Y") | The lambda function converts the value in the "Date" key to a date using the datetime.strptime() function |
| **sorted**(booking\_list, key=lambda x: datetime.strptime(x["Date"], "%d-%m-%Y")) | Sorted() function sorts the booking list in date order according to the values in the "Date" key |
| open() | Function used to open files |
| **csv.reader()** | Function used to read a CSV. It reads the file line by line and returns each line as a list. |
| **csv.writer()** | It is a function used to write to a CSV file. |
| **datetime.strptime()** | A function to returns datetime object representing the current date and time |
| **str.replace()** | A function to convert one character in a string to another character.  Ensures that there are no errors when using “.” or “/” or “-“ in date separation. |
| **str.isdigit()** | Checks whether the input is a number before integer and float functions |
| **str.lower()** | It makes all the letters of the string lowercase.  I used it to avoid uppercase and lowercase problems in specific user inputs, such as Yes, NO. |
| **input()** | It is a function used to get data from the user. |
| **exit()** | It is a function used to terminate the programme. |
| **print()** | Used to print output to the screen. |
| **len()** | Used to get the length of an object |

## Section 3: Python Code with comments

### Example CSV File

Date,Hour,Event Name,Contact Name,Contact No,Contact Email,Number of people,Catering,Cost,Deposit,Remaining Payment,Additional Info

02-01-2024,Morning,seminar,Alex,-,alex123@gmail.com,75,False,1000,300,700,-

30-03-2024,19.00,birthday party,bora,05317044491,borasozer@hotmail.com,10,False,100,10,90,free tequila shots for everyone

20-06-2024,15.00,graduation,bella,01273 641050,-,350,True,7800,1000,6800,dj

21-09-2024,15.00,birthday party,burak sozer,05318645010,burakksozer@gmail.com,10,False,400,100,300,make potato cake

07-10-2024,7pm,nadeen wedding,bora,21243,bora,5000,True,23929321212,10,23929321202,66 layer cake and free vape for everyone and special crown for the queen

### Code

import csv

from datetime import datetime

#the filename used can be changed with a single change to the coode

filename = "bookings.csv"

#exports the data contained in csv to the list

def load\_bookings(filename):

booking\_list = []

try:

with open(filename, 'r', newline = "") as booking\_csv:

csv\_read = csv.reader(booking\_csv)

#skips first line because there are headings on the first line

next(csv\_read)

#as long as the line in the csv is not missing, it saves each line in the list by assigning a key to each of the values in it and making a dictionary

for rows in csv\_read:

if len(rows) == 12:

booking\_list.append({"Date":rows[0],"Hour":rows[1],"Event Name":rows[2],"Contact Name":rows[3],"Contact No":rows[4],"Contact Email":rows[5],"Number of people":rows[6],"Catering":rows[7],"Cost":rows[8],"Deposit":rows[9],"Remaining Payment":rows[10],"Additional Info":rows[11]})

else:

print("There is missing info on this event:", rows, "\nPlease delete and re-add event.")

except FileNotFoundError:

#if csv file is not found, opens one from scratch

open(filename, "w")

with open(filename, "w", newline="") as booking\_csv:

csv\_write = csv.writer(booking\_csv)

csv\_write.writerow(["Date", "Hour", "Event Name", "Contact Name", "Contact No", "Contact Email","Number of people","Catering","Cost","Deposit","Remaining Payment", "Additional Info"])

#sorting events in the list by date

return sorted(booking\_list, key=lambda x: datetime.strptime(x["Date"], "%d-%m-%Y"))

#exports the data contained in the list to csv

def save\_bookings(filename, booking\_list):

with open(filename, "w", newline="") as booking\_csv:

csv\_write = csv.writer(booking\_csv)

csv\_write.writerow(["Date", "Hour", "Event Name", "Contact Name", "Contact No", "Contact Email","Number of people","Catering","Cost","Deposit","Remaining Payment", "Additional Info"])

for booking in booking\_list:

csv\_write.writerow([booking["Date"], booking["Hour"], booking["Event Name"], booking["Contact Name"], booking["Contact No"], booking["Contact Email"], booking["Number of people"], booking["Catering"], booking["Cost"], booking["Deposit"], booking["Remaining Payment"], booking["Additional Info"]])

#take input from user and check date for availability (added this to def because it used both new\_booking and change\_booking)

def check\_new\_date(booking\_list):

load\_bookings(filename)

while True:

while True:

#used try-except and datetime library to avoid date format error

try:

new\_date\_str = input("Please enter the date in day-month-year (dd-mm-yyyy) format or press enter to return menu:")

#if user press enter, it returns to menu

if new\_date\_str == "":

choice = "2"

return None, choice

#changed "." and "/" to "-" to avoid confusion

new\_date\_str = new\_date\_str.replace('/', '-')

new\_date\_str = new\_date\_str.replace('.', '-')

#by converting twice, we prevented double booking on the same date due to the input format

new\_date = datetime.strptime(new\_date\_str, "%d-%m-%Y")

new\_date\_str = new\_date.strftime('%d-%m-%Y')

if new\_date < datetime.now(): #check whether it is in the past history

print("You cannot choose a past date.")

continue

if new\_date > datetime.now().replace(year=datetime.now().year + 1): #check whether the selected date is within the next 1 year

print("You must choose a date within the next year.")

continue

break

except ValueError:

print("Invalid date format.\n")

found = False

for booking in booking\_list:

if booking["Date"] == new\_date\_str:

found = True

print("Date is full."+"\n")

choice = input("1.New date\n2.Menu\nChoice:")

#while loop used to prevent invalid selection

while choice not in ["1", "2"]:

choice = input("Choose one of the options:")

if choice == "1":

break

elif choice == "2":

return new\_date\_str, choice

if found == True:

continue

else:

return new\_date\_str, None

#check date for search, change and delete functions to find an existing event.

def check\_date(booking\_list):

load\_bookings(filename)

while True:

while True:

#used try-except and datetime library to avoid date format error

try:

date\_str = input("Please enter the date in day-month-year (dd-mm-yyyy) format or press enter to return menu:")

#if user press enter, it returns to menu

if date\_str == "":

choice = "2"

return date\_str, choice, None

#changed "." and "/" to "-" to avoid confusion

date\_str = date\_str.replace('/', '-')

date\_str = date\_str.replace('.', '-')

date = datetime.strptime(date\_str, "%d-%m-%Y")

date\_str = date.strftime('%d-%m-%Y')

break

except ValueError:

print("\nInvalid date format.\n")

found = False

for booking in booking\_list:

if booking["Date"] == date\_str:

found = True

return date\_str, None, found

#If the event is not found, whether to perform a new search or return to the menu

if found == False:

print("\nNo event found on", date\_str, "\n")

choice = input("1.New date\n2.Menu\nChoice:")

while choice not in ["1", "2"]:

choice = input("Choose one of the options:")

if choice == "1":

continue

elif choice == "2":

return date\_str, choice, found

#check name for search bookings by event name

def check\_name(booking\_list):

while True:

load\_bookings(filename)

searched\_name = input("Please enter the name of the event or press enter to return menu:")

if searched\_name == "":

return None, None, None

found = False

for booking in booking\_list:

if booking["Event Name"].lower() == searched\_name.lower(): #used .lower function because avoided not being able to find due to case difference

found = True

found\_date = booking["Date"]

print("\n")

#print all events found by exact name

for key, value in booking.items():

print(f"{key}: {value}", end=" | ")

print("")

if found == True:

return searched\_name, None, found

#If the event is not found, whether to perform a new search or return to the menu

elif found == False:

print("\nNo event found called ", searched\_name, "\n")

print("1.Search another name")

print("2.Return menu")

choice = input("Choice:")

while choice not in ["1", "2"]:

choice = input("Choose one of the options:")

if choice == "1":

continue

elif choice == "2":

return searched\_name, choice, found

#search for bookings by date and name

def search\_booking(booking\_list):

#used while loop to check user input for choice

while True:

print("")

print("1-Search by date")

print("2-Search by name")

print("3-Return Menu")

find\_booking = input("Type the number of your choice: ")

#if valid option entered break the loop

if find\_booking in ["1","2","3"]:

break

print("Invalid choice. Please type either 1,2 or 3.")

if find\_booking == "1":

date\_str, choice, found = check\_date(booking\_list) #call check date function

if choice == "2":

return

for booking in booking\_list:

if booking["Date"] == date\_str:

print("")

for key, value in booking.items():

print(f"{key}: {value}", end="\n")

elif find\_booking == "2":

searched\_name, choice, found = check\_name(booking\_list) #call check name function

#prints all upcoming bookings

def upcoming\_bookings(booking\_list):

found = False

for booking in booking\_list:

#check the event date with today's date

event\_date = datetime.strptime(booking["Date"], "%d-%m-%Y")

if event\_date >= datetime.now():

#print if the date hasn't arrived yet

print("\n")

for key, value in booking.items():

print(f"{key}: {value}", end=" | ")

found = True

print("")

if found == False:

print("There's no upcoming booking.")

#add new booking to the list

def add\_booking(booking\_list):

new\_date\_str, choice = check\_new\_date(booking\_list)

if choice == "2":

return

new\_hour = input("Hour:")

new\_event\_name = input("Event name:")

new\_name = input("Contact Name:")

new\_no = input("Contact number:") #No limit was set. Because the user can enter two numbers for an event or if it is a foreign number, the country code may need to be written at the beginning. For example:+44,+1,+90

new\_email = input("Contact email:")

new\_numberofpeople = input("Number of people:") #No limit was set. Because for example, the user can enter: 15-20

catering\_choice = input("Catering (yes/no):") #learns the catering choice from user and stores it as boolean

while catering\_choice not in ["yes","no"]:

catering\_choice = input("\nInvalid choice.\nCatering (yes/no):").lower() #used .lower() to eliminate the capital difference

if catering\_choice == "yes":

new\_catering = True

else:

new\_catering = False

while True:

while True:

new\_cost = input("Cost:")

if new\_cost.isdigit(): #ensures that the value is only a number

break

else:

print("Please enter valid value.")

while True:

new\_deposit = input("Deposit amount:")

if new\_deposit.isdigit():

break

else:

print("Please enter only number.")

if int(new\_deposit) <= int(new\_cost): #prevents deposit to be entered larger than total cost

break

else:

print("\nDeposit amount can't be higher than total. Please re-enter cost and deposit amounts.\n")

new\_remaining\_payment = int(new\_cost) - int(new\_deposit) #calculates remaining payment

new\_additional\_info = input("Additional info:")

#creates a new booking dictionary with all information

new\_booking = {"Date":new\_date\_str,"Hour":new\_hour,"Event Name":new\_event\_name,"Contact Name":new\_name,"Contact No":new\_no,"Contact Email":new\_email,"Number of people":new\_numberofpeople,"Catering":new\_catering,"Cost":new\_cost,"Deposit":new\_deposit,"Remaining Payment":new\_remaining\_payment,"Additional Info":new\_additional\_info}

booking\_list.append(new\_booking) #Adds the created booking dictionary to the booking\_list

print("")

for key, value in new\_booking.items():

print(f"{key}: {value}", end=" | ")

print("\n\nNew booking added.")

save\_bookings(filename, booking\_list) #saves the new list to the CSV file

#change existing booking info

def change\_booking(booking\_list):

date\_str, choice, found = check\_date(booking\_list) #learns and checks the date of the reservation to be changed

if choice == "2":

return

while True:

print("\n")

counter = 1

#tells the user the categories that can be changed with their numbers

for booking in booking\_list:

if booking["Date"]== date\_str:

for key, value in booking.items():

if key != "Remaining Payment":

print(f"{counter}. {key}: {value}", end="\n") #The "counter" variable helps the user to make easy selections using numbers

counter += 1

while True:

change = input("Type the number you want to change or press enter to return menu:") #asks the user the category they want to change

if change.isdigit(): #checks the user input and asks for an input again if it is invalid choice

change = int(change)

if 1 <= change <= 11:

break

else:

print("Number should between 1 and 11.")

elif change == "":

return

else:

print("Invalid choice.")

for booking in booking\_list:

if booking["Date"] == date\_str:

#takes action according to the change to be made according to the user response and replaces the old data

if change == 1:

new\_date\_str, choice = check\_new\_date(booking\_list)

if choice == "2":

return

booking["Date"] = new\_date\_str

date\_str = new\_date\_str

elif change == 2:

new\_hour = input("New hour:")

booking["Hour"] = new\_hour

elif change == 3:

new\_event\_name = input("New event name:")

booking["Event Name"] = new\_event\_name

elif change == 4:

new\_name = input("New contact name:")

booking["Contact Name"] = new\_name

elif change == 5:

new\_no = input("New number:")

booking["Contact No"] = new\_no

elif change == 6:

new\_email = input("New email:")

booking["Contact Email"] = new\_email

elif change == 7:

new\_numberofpeople = input("New number of people:")

booking["Number of people"] = new\_numberofpeople

elif change == 8:

new\_catering = input("New catering choice (yes/no):").lower()

while new\_catering not in ["yes","no"]:

new\_catering = input("\nInvalid choice.\nCatering (yes/no):").lower() #used .lower() to eliminate the capital difference

if new\_catering == "yes":

new\_catering = True

else:

new\_catering = False

booking["Catering"] = new\_catering

elif change == 9:

while True:

new\_cost = input("Cost:")

if new\_cost.isdigit():

break

else:

print("Please enter valid value.")

booking["Cost"] = new\_cost

booking["Remaining Payment"] = int(booking["Cost"]) - int(booking["Deposit"])

elif change == 10:

while True:

new\_deposit = input("Deposit amount:")

if new\_deposit.isdigit():

if int(new\_deposit) <= int(new\_cost):

break

else:

print("\nDeposit amount can't be higher than total.\n")

else:

print("Please enter only number.")

booking["Deposit"] = new\_deposit

booking["Remaining Payment"] = int(booking["Cost"]) - int(booking["Deposit"])

elif change == 11:

info\_choice = input("1.Rewrite info\n2.Add info\nChoice:")

while not info\_choice in ["1","2"]:

info\_choice = input("\nInvalid option. Type 1 or 2:")

new\_additional\_info = input("New additional info:")

if info\_choice == "1":

booking["Additional Info"] = new\_additional\_info

else:

booking["Additional Info"] = booking["Additional Info"] +" "+ new\_additional\_info

print("Booking info succesfully changed.\n")

print("Updated event:")

#prints the changed version of the booking

for key, value in booking.items():

print(f"{key}: {value}", end=" | ")

save\_bookings(filename, booking\_list) #saves the changed booking list back to csv

#delete existing booking

def delete\_booking(booking\_list):

date\_str, choice, found = check\_date(booking\_list) #learns the date to be deleted

if choice == "2":

return

for booking in booking\_list:

if booking["Date"] == date\_str:

print("")

for key, value in booking.items():

print(f"{key}: {value}", end="\n")

#receives confirmation from the user for the last time

delete = input("\nAre you sure you want to delete this event(yes/no):").lower()

while not delete in ["yes","no"]:

delete = input("Type yes or no:").lower()

if delete == "yes":

#performs the deletion operation

for booking in booking\_list:

if booking["Date"] == date\_str:

booking\_list.remove(booking)

save\_bookings(filename, booking\_list)

print("\n"+"Booking succesfully deleted.")

break

return(booking\_list)

#prints all past bookings

def past\_bookings(booking\_list):

found = False

for booking in booking\_list:

event\_date = datetime.strptime(booking["Date"], "%d-%m-%Y")

if event\_date < datetime.now():

print("\n")

for key, value in booking.items():

print(f"{key}: {value}", end=" | ")

found = True

print("")

if found == False:

print("There's no past booking.")

#contains the main menu of our program

def main():

while True:

booking\_list = load\_bookings(filename) #loads data from csv in each loop

print("")

print("1.Search Booking")

print("2.Upcoming Bookings")

print("3.Add Booking")

print("4.Change Booking")

print("5.Delete Booking")

print("6.Past Bookings")

print("7.Exit")

option = input("Type your choice:")

if option == "1":

search\_booking(booking\_list)

elif option == "2":

upcoming\_bookings(booking\_list)

elif option == "3":

add\_booking(booking\_list)

elif option == "4":

change\_booking(booking\_list)

elif option == "5":

delete\_booking(booking\_list)

elif option == "6":

past\_bookings(booking\_list)

elif option == "7":

exit()

else:

print("\nInvalid Choice.")

save\_bookings(filename, booking\_list) #saves the list to csv in each loop

main()

## Section 4: Testing

### Testing for development:

**Problem 1:**

I had to check that the date format entered was correct.

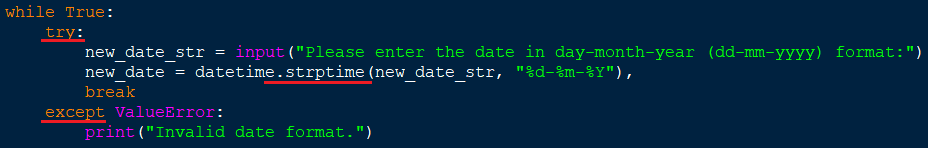
All dates must be in the same format to be able to compare with other dates.

I solved this problem using the datetime library and try-except statement. (Figure 1-2)

I learnt about datetime library from w3 schools website (w3schools.com, Python datetime) and docs.python.org website (docs.python.org, Datetime - basic date and time types).

C:\Users\hakan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Ekran görüntüsü 2024-03-03 140011.png **Figure 1.**

**Figure 2.**



If there is an error in the date format, the except function is activated and the while loop is repeated for user input again.

**Problem 2:**

If the user used "-" or "." or "/" in the date format, the format was confused.

I did a conversion to avoid getting confused and getting an error. (Figure 3)

**Figure 3.**



I changed "." and "/" to "-".

So that if the user enters 30.03.2024 or 30/03/2024, it will make it the same as 30-03-2024.

**Problem 3:**

I wanted the bookings to be kept in the list in order by date.

I used lambda and sorted function. (Figure 4)

I learn sorted and lambda function usage from geeksforgeeks.org (geeksforgeeks.org, 2023)

**Figure 4.**



Lambda function takes "Date" for each item.

Sorted function sorts the items according to the value of the lambda

**Problem 4:**

I didn't want it to save the two as different dates when the user entered 6-5-2024 or 06-05-2024.

So I converted it with the strptime and strftime function and converted it back to dd-mm-yyyy format. (Figure 5)

**Figure 5.**

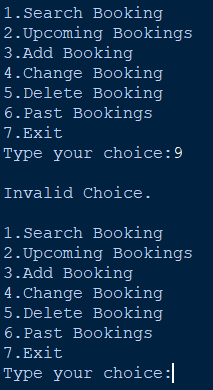


So that if the user enters 6-5-2024 or 06-05-2024, it will save it as 06-05-2024.

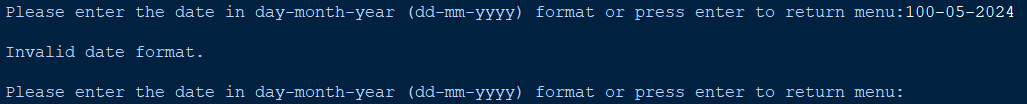
### Testing for evaluation:

|  |  |  |
| --- | --- | --- |
| **Purpose of test** | **How the test was carried out** | **Outcome from that test** |
| **Validate user input for menu choice**  Ensure the program does not accept invalid option. | Invalid option tried as input from user | Program warned the user that they had entered invalid and asked them to type again.  **(Figure 6.)** |
| **Validate user input for date input**  Ensure the program does not accept invalid date format. | Input tried with invalid date format | Program warned the user that they had entered invalid date format and asked them to try again.  **(Figure 7.)** |
| **Validate user input for search menu choice**  Ensuring that only options is accepted. | Entered a choice that is not in the options | Program warned the user that they had entered invalid choice and asked them to type again.  **(Figure 8.)** |
| **Validate user input for add booking date to check for no existing date**  Ensuring double booking is not allowed | Entered a date which is existing. | Program warned the user that they had existing date and asked them to type new date.  **(Figure 9.)** |
| **Validate user input for catering choice**  Ensure the program does not accept invalid option. | Entered a choice that is not in the options | Program warned the user that they had entered invalid choice and asked them to type again.  **(Figure 10.)** |
| **Validate user input for cost and deposit value**  Ensuring that the deposit is not greater than the total cost | Entered a deposit amount which is greater than total cost | Program warned the user and asked them to type cost and deposit again.  **(Figure 11.)** |
| **Validate user input for delete event**  Ensuring that only options is accepted. | Entered a choice that is not in the options | Program warned the user that they had entered invalid choice and asked them to type again.  **(Figure 12.)** |
| **Validate that the programme works properly and without any error** Ensure that outputs are user-friendly and fulfils its purpose | All options and features were tried. all kinds of different inputs were tried to be entered. | The programme fulfils its purpose without error. all functions work properly. |

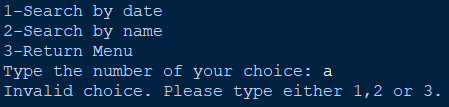
**Figure 6.**



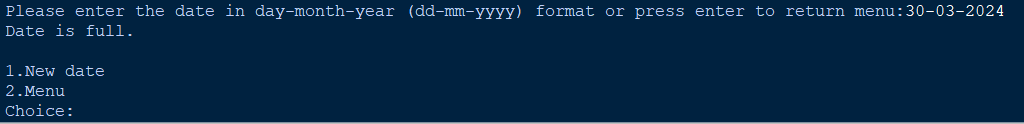
**Figure 7.**



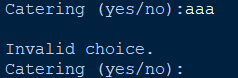
**Figure 8.**



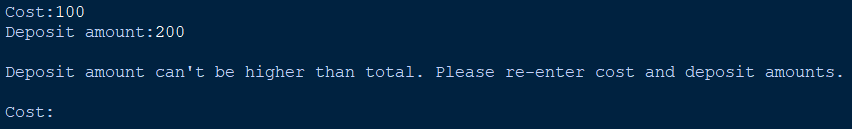
**Figure 9.**



**Figure 10.**



**Figure 11.**



**Figure 12.**



## Section 5: Evaluation and Summary

The program meets all specified criteria and effectively achieves its goal without any error.

Application loads data from csv file or if doesn’t exists it creates new csv file. After, the programme repeats the menu interface which contains search, add, change, delete, show upcoming or past bookings and exit. It runs code or calls the relevant function according to the user's choice.

It check date match on search, add, change or delete function. It wants a new date according to the situation. **For user-friendly interface, each option has the option to go back or return to the menu.** The programme detects invalid entries and warn the user. Program correctly processes date formats. Defined functions were used to make the code look less complex and easier to read.

To summarise, the programme allows us to make a reservation for a venue with a single room and one day block bookings for the dates that are free in the next year (prevent double booking) and store the information. It allows us to view, modify and delete the created reservations.

## Section 6: References

**1-** Used to understand working mechanism of datetime library:

*Python Datetime* (no date) *Python Dates*. Available at: <https://www.w3schools.com/python/python_datetime.asp> (Accessed: 28 February 2024).

**2-**Used to understand working mechanism of datetime library:

*Datetime - basic date and time types* (no date) *Python documentation*. Available at: <https://docs.python.org/3/library/datetime.html> (Accessed: 28 February 2024).

**3-** Used to understand working mechanism of sorted() and lambda function:

*Python program to sort the list according to the column using Lambda* (2023) *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/python-program-to-sort-the-list-according-to-the-column-using-lambda/> (Accessed: 03 March 2024).