|  |
| --- |
| [Company name] |
| ICT 162 - OOP |
| [Document subtitle] |

|  |
| --- |
| Student Name  [Date] |

Table of Contents

[1. Question 1 4](#_Toc162105743)

[a. Customer Class 4](#_Toc162105744)

[b. Tour Class 4](#_Toc162105745)

[c. Tour objects 5](#_Toc162105746)

[Output 6](#_Toc162105747)

[2. Question 2 6](#_Toc162105748)

[a. ScheduledTour 6](#_Toc162105749)

[c. ScheduledTour objects 8](#_Toc162105750)

[Output 9](#_Toc162105751)

[3. Question 3 9](#_Toc162105752)

[a. BookingException 9](#_Toc162105753)

[b. Booking 9](#_Toc162105754)

[c. 2 subclasses of Booking 11](#_Toc162105755)

[d. main() function with exception handling 12](#_Toc162105756)

[4. Question 4 13](#_Toc162105757)

[a. TourAgency Class 13](#_Toc162105758)

[b. Application for Staff 16](#_Toc162105759)

Table of Figures

[Figure 1: Printing Tour Objects 6](#_Toc162114134)

[Figure 2: Printing ScheduledTour Objects 9](#_Toc162114135)

[Figure 3: Tour Management 23](#_Toc162114136)

[Figure 4: Schedule Tour Option 1 at Schedule Tour 24](#_Toc162114137)

[Figure 5: Open/Close Schedule Option 2 at Tour at Schedule Tour 24](#_Toc162114138)

[Figure 6: Removal Schedule Tour Option 3 at Schedule Tour 25](#_Toc162114139)

[Figure 7: List Scheduled Tours Option 4 at Schedule Tour 25](#_Toc162114140)

[Figure 8: Booking Management 26](#_Toc162114141)

[Figure 9: Create Booking Option 1 at Booking Management 26](#_Toc162114142)

[Figure 10: Cancel Booking Option 2 at Booking Management 27](#_Toc162114143)

[Figure 11: Add Seats to Booking Option 3 at Booking Management 27](#_Toc162114144)

[Figure 12: List Bookings Option 4 at Booking Management 28](#_Toc162114145)

# Question 1

## Customer Class

|  |
| --- |
| from datetime import datetime  class Customer:      def \_\_init\_\_(self, passportNumber, name, dob, contact):          self.\_passportNumber = passportNumber          self.\_name = name          self.\_dob = dob          self.\_contact = contact        @property      def passportNumber(self):          return self.\_passportNumber        @property      def name(self):          return self.\_name        @property      def contact(self):          return self.\_contact        @contact.setter      def contact(self, value):          self.\_contact = value        def getAge(self):          current\_year = datetime.now().year          return current\_year - self.\_dob.year        def \_\_str\_\_(self):          return f"Passport: {self.\_passportNumber} Name: {self.\_name} Age: {self.getAge()} Contact: {self.\_contact}" |

## Tour Class

|  |
| --- |
| class Tour:      def \_\_init\_\_(self, code, name, days, nights, cost):          self.\_code = code          self.\_name = name          self.\_days = days          self.\_nights = nights          self.\_cost = cost        @property      def code(self):          return self.\_code        @property      def name(self):          return self.\_name        @property      def cost(self):          return self.\_cost        @cost.setter      def cost(self, value):          self.\_cost = value        @property      def daysNights(self):          return f"{self.\_days}D/{self.\_nights}N"        def \_\_str\_\_(self):          return f"Tour Code: {self.\_code} Name: {self.\_name} ({self.daysNights}) Base Cost: ${self.\_cost:.2f}" |

## Tour objects

|  |
| --- |
| tour1 = Tour("JPHA08", "Best of Hokkaido", 8, 7, 2699.08)  tour2 = Tour("KMBK08", "Mukbang Korea", 8, 6, 1699.36)  tour3 = Tour("VNDA06", "Discover Vietnam", 6, 5, 999.00)  # Print string representations of Tour objects  print(tour1)  print(tour2)  print(tour3) |

## Output

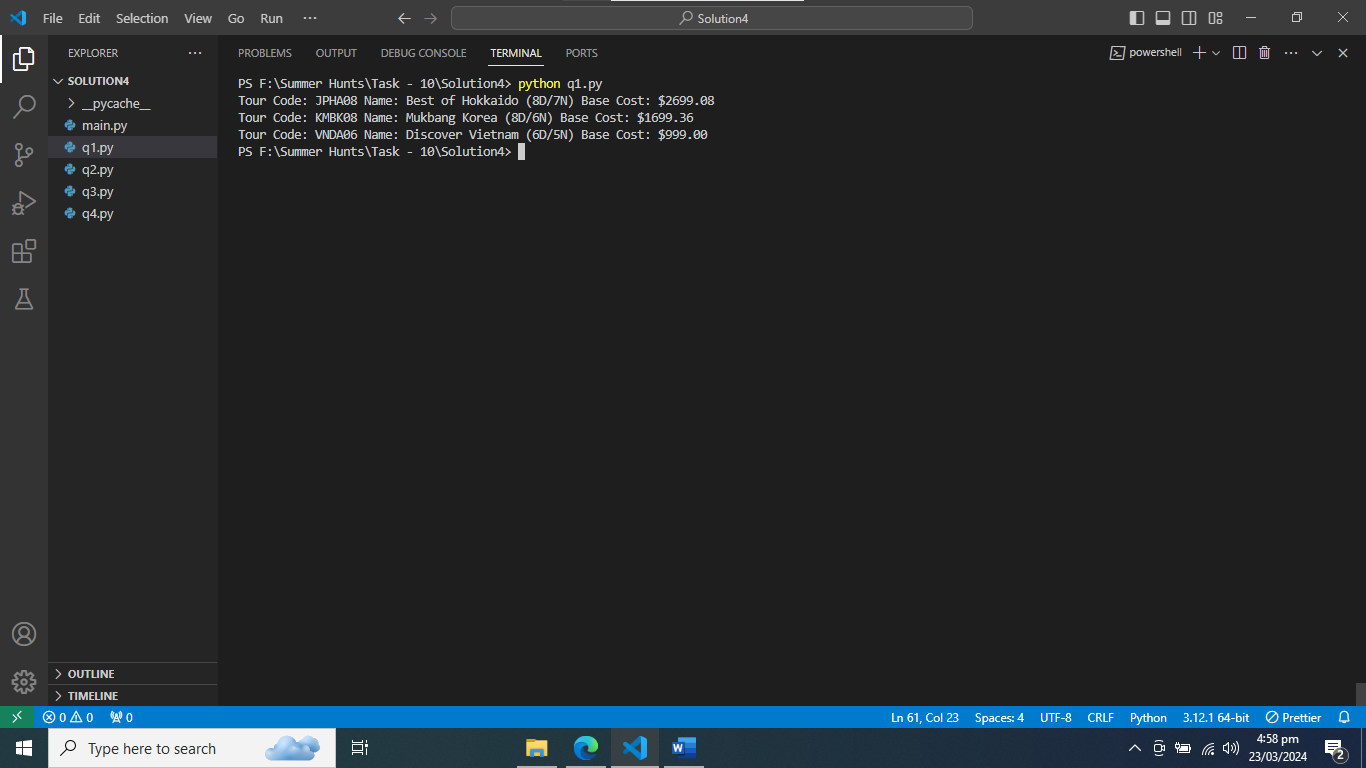


Figure : Printing Tour Objects

# Question 2

## ScheduledTour

|  |
| --- |
| from q1 import tour1  from q1 import tour2  from q1 import tour3  from datetime import datetime, timedelta  class ScheduledTour:      handling\_fee = 120        @classmethod      def getHandlingFee(cls):          return cls.handling\_fee        def \_\_init\_\_(self, scheduleCode, tour, departureDateTime, lang, capacity):          self.\_scheduleCode = scheduleCode          self.\_tour = tour          self.\_departureDateTime = departureDateTime          self.\_lang = lang          self.\_capacity = capacity          self.\_seatsAvailable = capacity          self.\_status = True        @property      def departureDateTime(self):          return self.\_departureDateTime        @property      def capacity(self):          return self.\_capacity        @property      def seatsAvailable(self):          return self.\_seatsAvailable        @property      def status(self):          return self.\_status        @status.setter      def status(self, value):          self.\_status = value        @property      def code(self):          return f"{self.\_tour.code}-{self.\_scheduleCode}"        @property      def cost(self):          return self.\_tour.cost        def bookSeats(self, quantity):          if quantity > self.\_seatsAvailable:              return False          self.\_seatsAvailable -= quantity          return True        def cancelSeats(self, quantity):          if self.\_seatsAvailable + quantity > self.\_capacity:              return False          self.\_seatsAvailable += quantity          return True        def getPenaltyRate(self):          days\_before\_tour = (self.\_departureDateTime - datetime.now()).days          if days\_before\_tour >= 46:              return 0.1          elif 14 <= days\_before\_tour <= 45:              return 0.5          elif 7 <= days\_before\_tour <= 13:              return 1          else:              return 1        def \_\_str\_\_(self):          return f"Name: {self.\_tour.name} ({self.\_tour.daysNights}) Base Cost: ${self.\_tour.cost:.2f}\n" \                 f"Code: {self.code} Departure: {self.\_departureDateTime.strftime('%d-%b-%Y %H:%M')} " \                 f"Language: {self.\_lang}\n" \                 f"Capacity: {self.\_capacity} Available: {self.\_seatsAvailable} Open: {'Yes' if self.\_status else 'No'}" |

1. PeakScheduledTour

|  |
| --- |
| class PeakScheduledTour(ScheduledTour):      surcharge = 0.15      handling\_charges = 200        def \_\_init\_\_(self, scheduleCode, tour, departureDateTime, lang, capacity):          super().\_\_init\_\_(scheduleCode, tour, departureDateTime, lang, capacity)          self.\_cost = self.\_tour.cost \* (1 + self.surcharge)        @property      def cost(self):          return self.\_cost        def getPenaltyRate(self):          base\_penalty\_rate = super().getPenaltyRate()          max\_penalty\_rate = 1  # 100%          return min(base\_penalty\_rate + 0.1, max\_penalty\_rate) |

## ScheduledTour objects

|  |
| --- |
| scheduled\_tour1 = ScheduledTour("505", tour1, datetime(2024, 5, 5, 10, 30), "English", 30)  scheduled\_tour2 = ScheduledTour("408", tour1, datetime(2024, 4, 8, 8, 45), "English", 25)  scheduled\_tour3 = ScheduledTour("503", tour2, datetime(2024, 5, 3, 8, 5), "English", 32)  scheduled\_tour4 = ScheduledTour("403", tour2, datetime(2024, 4, 3, 10, 5), "Mandarin", 25)  scheduled\_tour5 = ScheduledTour("503", tour3, datetime(2024, 5, 3, 11, 8), "Mandarin", 28)  # Create PeakScheduledTour objects  peak\_scheduled\_tour1 = PeakScheduledTour("505", tour1, datetime(2024, 5, 5, 10, 30), "English", 30)  peak\_scheduled\_tour2 = PeakScheduledTour("503", tour2, datetime(2024, 5, 3, 8, 5), "English", 32)  peak\_scheduled\_tour3 = PeakScheduledTour("503", tour3, datetime(2024, 5, 3, 11, 8), "Mandarin", 28) |

## Output

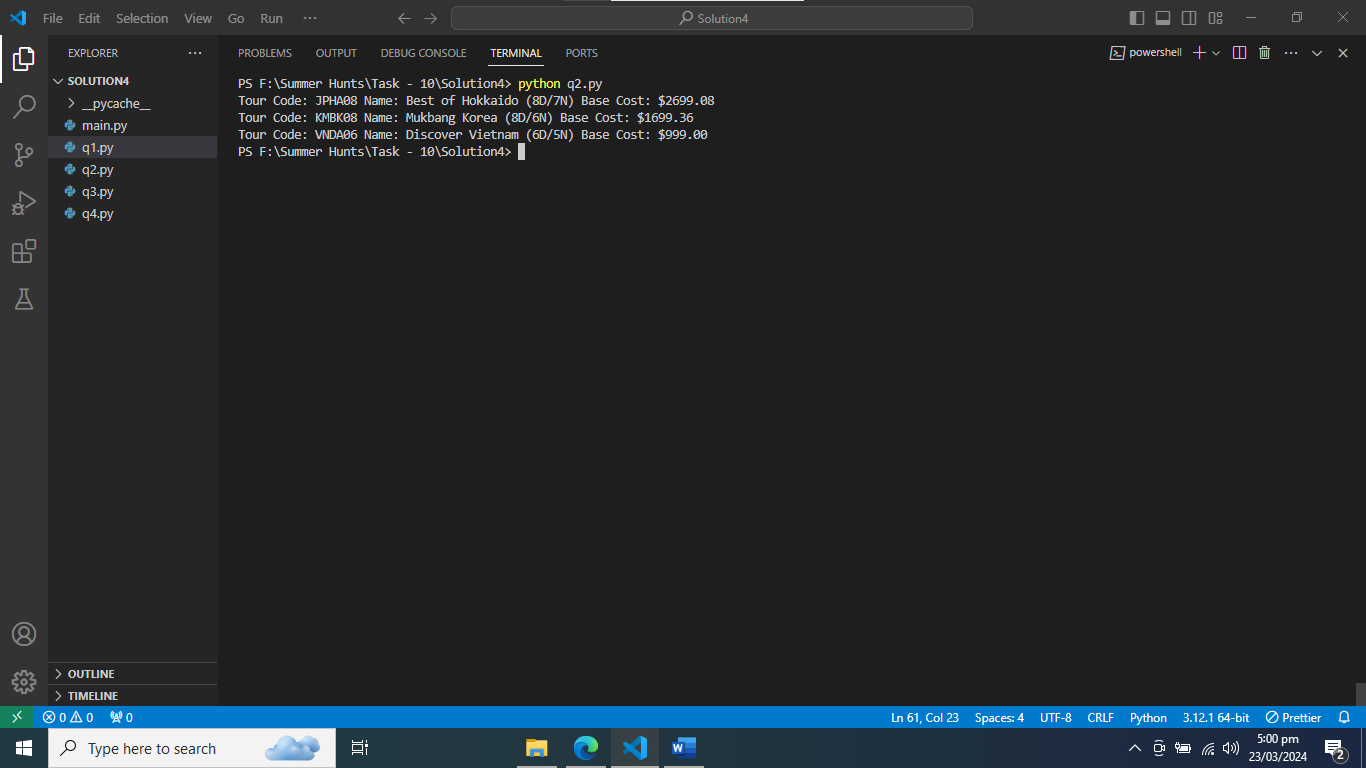


Figure : Printing ScheduledTour Objects

# Question 3

## BookingException

|  |
| --- |
| class BookingException(Exception):      pass |

## Booking

|  |
| --- |
| from q1 import Customer  from q2 import ScheduledTour  from q1 import tour1  from abc import ABC, abstractmethod  from datetime import datetime  from q2 import scheduled\_tour1  class Booking(ABC):      \_NEXT\_ID = 1      def \_\_init\_\_(self, scheduledTour, customers):          if not scheduledTour.bookSeats(len(customers)):              raise BookingException("Not enough seats available for booking.")          self.\_bookingId = Booking.\_NEXT\_ID          Booking.\_NEXT\_ID += 1          self.\_scheduledTour = scheduledTour          self.\_customers = customers      @property      def bookingId(self):          return self.\_bookingId      @property      def scheduledTour(self):          return self.\_scheduledTour      @property      def customers(self):          return self.\_customers      @property      def seats(self):          return len(self.\_customers)      @abstractmethod      def cost(self):          pass      @abstractmethod      def addSeats(self, customers):          pass      def searchCustomer(self, passport\_numbers):          return any(customer.passportNumber in passport\_numbers for customer in self.\_customers)      def getPenaltyAmount(self):          days\_difference = (self.\_scheduledTour.departureDateTime - datetime.now()).days          penalty\_rate = self.\_scheduledTour.getPenaltyRate()          handling\_charges = ScheduledTour.getHandlingFee()          base\_cost = self.cost()          penalty\_amount = handling\_charges + (penalty\_rate \* base\_cost)          return min(penalty\_amount, base\_cost)      def \_\_str\_\_(self):          output = f"Booking Id: {self.\_bookingId} Seats: {self.seats} Final Cost: ${self.cost():,.2f}\n"          output += str(self.\_scheduledTour) + "\n"          output += "\n".join(str(customer) for customer in self.\_customers)          return output |

## 2 subclasses of Booking

|  |
| --- |
| class IndividualBooking(Booking):      \_SINGLE = 0.5      def \_\_init\_\_(self, scheduledTour, customer, single=False):          if datetime.now().year - customer.\_dob.year < 20:              raise BookingException("Customer must be at least 20 years old for individual booking.")          super().\_\_init\_\_(scheduledTour, [customer])          self.\_single = single      def cost(self):          base\_cost = self.\_scheduledTour.cost          if self.\_single:              return base\_cost \* (1 + self.\_SINGLE)          return base\_cost      def addSeats(self, customers):          raise BookingException("It is not possible to add seats for individual booking.")  class GroupBooking(Booking):      \_DISCOUNT = {6: 0.05, 10: 0.10}      def \_\_init\_\_(self, scheduledTour, customers):          if len(customers) < 2:              raise BookingException("Group size must be at least 2.")          for customer in customers:              if datetime.now().year - customer.\_dob.year < 20:                  raise BookingException("All customers must be at least 20 years old for group booking.")          super().\_\_init\_\_(scheduledTour, customers)      def getDiscount(self):          group\_size = len(self.\_customers)          if group\_size < 6:              return 0          for size, discount in sorted(self.\_DISCOUNT.items(), reverse=True):              if group\_size >= size:                  return discount      def cost(self):          base\_cost = self.\_scheduledTour.cost          discount = self.getDiscount()          if discount:              return base\_cost \* (1 - discount) \* len(self.\_customers)          return base\_cost \* len(self.\_customers)      def addSeats(self, customers):          existing\_passports = {customer.passportNumber for customer in self.\_customers}          if any(customer.passportNumber in existing\_passports for customer in customers):              raise BookingException("One or more customers are already in this booking.")          if not self.\_scheduledTour.bookSeats(len(customers)):              raise BookingException("The scheduled tour may have reached full capacity.")          self.\_customers.extend(customers)          return True |

## main() function with exception handling

|  |
| --- |
| def main():      try:          print("Individual Booking:")          customer1 = Customer("E2714897X", "Henry Tan", datetime(1980, 5, 10), 97777777)            individual\_booking1 = IndividualBooking(scheduled\_tour1, customer1, single=True)          print(individual\_booking1)          individual\_booking1.addSeats([customer1])      except BookingException as e:          print(e)      try:          print("\nIndividual Booking:")          customer2 = Customer("E1234567G", "Alice Oh", datetime(2005, 1, 15), 98989898)          individual\_booking2 = IndividualBooking(scheduled\_tour1, customer2, single=True)          print(individual\_booking2)      except BookingException as e:          print(e)      try:          print("\nGroup Booking:")          customers3 = [              Customer("E1234567G", "Alice Oh", datetime(2005, 1, 15), 98989898),              Customer("E2323232Y", "Joyce Seetoh", datetime(2007, 9, 23), 62349000)          ]          group\_booking1 = GroupBooking(scheduled\_tour1, customers3)          print(group\_booking1)      except BookingException as e:          print(e)      try:          print("\nGroup Booking:")          customers4 = [              Customer("E1234567G", "Alice Oh", datetime(2005, 1, 15), 98989898),              Customer("E2323232Y", "Joyce Seetoh", datetime(2007, 9, 23), 62349000),              Customer("K1234771H", "Marvin Heng", datetime(1980, 3, 25), 89699090),              Customer("K5324732Y", "Mary Tham", datetime(2009, 7, 8), 87112345),              Customer("E2714897X", "Henry Tan", datetime(1980, 5, 10), 97777777)          ]          group\_booking2 = GroupBooking(scheduled\_tour1, customers4)          print(group\_booking2)          additional\_customers = [              Customer("E1234568G", "Bob Lee", datetime(1987, 11, 20), 99999999),              Customer("E5432101A", "David Lim", datetime(1985, 6, 7), 88888888)          ]          group\_booking2.addSeats(additional\_customers)          print(group\_booking2)      except BookingException as e:          print(e)  if \_\_name\_\_ == "\_\_main\_\_":      main() |

# Question 4

## TourAgency Class

|  |
| --- |
| from datetime import datetime  from q1 import Customer  from q1 import Tour  from q2 import ScheduledTour  from q3 import BookingException  from q3 import IndividualBooking  from q3 import GroupBooking  class TourAgency:      def \_\_init\_\_(self):          self.\_tours = []          self.\_customers = []          self.\_scheduled\_tours = []          self.\_bookings = []      def searchCustomer(self, passport\_number):          for customer in self.\_customers:              if customer.passportNumber == passport\_number:                  return customer          return None      def searchTour(self, tour\_code):          for tour in self.\_tours:              if tour.code == tour\_code:                  return tour          return None      def searchScheduledTour(self, schedule\_code):          for scheduled\_tour in self.\_scheduled\_tours:              if scheduled\_tour.code == schedule\_code:                  return scheduled\_tour          return None      def searchBooking(self, booking\_id):          for booking in self.\_bookings:              if booking.bookingId == booking\_id:                  return booking          return None      def formatTours(self):          return "\n".join(str(tour) for tour in self.\_tours)      def formatBookings(self):          return "\n".join(str(booking) for booking in self.\_bookings)      def formatScheduledTours(self):          return "\n".join(str(scheduled\_tour) for scheduled\_tour in self.\_scheduled\_tours)      def formatOpenScheduledTours(self):          return "\n".join(str(scheduled\_tour) for scheduled\_tour in self.\_scheduled\_tours if scheduled\_tour.status)      def addCustomer(self, customer):          if customer not in self.\_customers:              self.\_customers.append(customer)          else:              raise BookingException("Customer already exists.")      def addTour(self, tour):          if tour not in self.\_tours:              self.\_tours.append(tour)          else:              raise BookingException("Tour already exists.")      def addScheduledTour(self, scheduled\_tour):          if scheduled\_tour not in self.\_scheduled\_tours:              self.\_scheduled\_tours.append(scheduled\_tour)          else:              raise BookingException("Scheduled tour already exists.")      def removeScheduledTour(self, schedule\_code):          scheduled\_tour = self.searchScheduledTour(schedule\_code)          if scheduled\_tour:              if not scheduled\_tour.\_status:                  self.\_scheduled\_tours.remove(scheduled\_tour)              else:                  raise BookingException("Scheduled tour has bookings and cannot be removed.")          else:              raise BookingException("Scheduled tour not found.")        def updateScheduledTourStatus(self, scheduled\_tour\_code, status):          scheduled\_tour = self.searchScheduledTour(scheduled\_tour\_code) #..........          if scheduled\_tour:              scheduled\_tour.status = status              print("Status updated!!")              print(scheduled\_tour)          else:              print("Scheduled Tour not found.")      def addBooking(self, scheduled\_tour, customers, booking\_type, single=False):          if not scheduled\_tour.status:              raise BookingException("Scheduled tour is not open for booking.")            for customer in customers:              if not any(self.searchCustomer(customer.passportNumber) for scheduled\_tour in self.\_scheduled\_tours):                  raise BookingException("Customer already booked for this scheduled tour.")          if booking\_type == "I":              booking = IndividualBooking(scheduled\_tour, customers[0], single)          elif booking\_type == "G":              booking = GroupBooking(scheduled\_tour, customers)          else:              raise ValueError("Invalid booking type.")            self.\_bookings.append(booking)          return booking      def cancelBooking(self, booking\_id):          booking = self.searchBooking(booking\_id)          if booking:              if booking.scheduledTour.status:                  booking.scheduledTour.cancelSeats(booking.seats)                  self.\_bookings.remove(booking)              else:                  raise BookingException("Scheduled tour associated with the booking is not open for booking.")          else:              raise BookingException("Booking not found.")      def addSeats(self, booking\_id, customers):          booking = self.searchBooking(booking\_id)          if booking:              if booking.scheduledTour.status:                  additional\_cost = booking.addSeats(customers)                  return additional\_cost              else:                  raise BookingException("Scheduled tour associated with the booking is not open for booking.")          else:              raise BookingException("Booking not found.") |

## Application for Staff

|  |
| --- |
| # Main function  def main():      # Creating TourAgency object      tour\_agency = TourAgency()        # Adding customers      customer1 = Customer("E2000444N", "John Doe", datetime(1990, 5, 20), "1234567890")      customer2 = Customer("EC4744643", "Jane Smith", datetime(1985, 9, 15), "9876543210")      tour\_agency.customers.extend([customer1, customer2])        # Adding tours      tour1 = Tour("JPHA08", "Best of Hokkaido", 8, 7, 2699.08)      tour2 = Tour("KMBK08", "Mukbang Korea", 8, 6, 1699.36)      tour3 = Tour("VNDA06", "Discover Vietnam", 6, 5, 999.00)      tour\_agency.tours.extend([tour1, tour2, tour3])        # Adding scheduled tours      scheduled\_tour1 = ScheduledTour("505", tour1, datetime(2024, 5, 5, 10, 30), "English", 30, True)      scheduled\_tour2 = ScheduledTour("408", tour1, datetime(2024, 4, 8, 8, 45), "English", 25, False)      scheduled\_tour3 = ScheduledTour("503", tour2, datetime(2024, 5, 3, 8, 5), "English", 32, True)      scheduled\_tour4 = ScheduledTour("403", tour2, datetime(2024, 4, 3, 10, 5), "Mandarin", 25, False)      scheduled\_tour5 = ScheduledTour("503", tour3, datetime(2024, 5, 3, 11, 8), "Mandarin", 28, True)      tour\_agency.scheduled\_tours[tour1.code].extend([scheduled\_tour1, scheduled\_tour2])      tour\_agency.scheduled\_tours[tour2.code].extend([scheduled\_tour3, scheduled\_tour4])      tour\_agency.scheduled\_tours[tour3.code].append(scheduled\_tour5)        # Menu loop      while True:          print("<<<< Main Menu >>>>")          print("1. Tour management")          print("2. Booking management")          print("0. Quit")          choice = input("Enter choice: ")            if choice == "1":              tour\_management\_menu(tour\_agency)          elif choice == "2":              booking\_management\_menu(tour\_agency)          elif choice == "0":              print("Exiting...")              break          else:              print("Invalid choice. Please try again.")  # Tour management menu function  def tour\_management\_menu(tour\_agency):      while True:          print("\n<< Tour Menu >>")          print("1. Schedule Tour")          print("2. Open/Close Scheduled Tour")          print("3. Remove Scheduled Tour")          print("4. List Scheduled Tours")          print("0. Back to main menu")          choice = input("Enter choice: ")            if choice == "1":              schedule\_tour(tour\_agency)          elif choice == "2":              open\_close\_scheduled\_tour(tour\_agency)          elif choice == "3":              remove\_scheduled\_tour(tour\_agency)          elif choice == "4":              list\_scheduled\_tours(tour\_agency)          elif choice == "0":              break          else:              print("Invalid choice. Please try again.")  # Booking management menu function  def booking\_management\_menu(tour\_agency):      while True:          print("\n<< Booking Menu >>")          print("1. Create Booking")          print("2. Cancel Booking")          print("3. Add Seats to Booking")          print("4. List Bookings")          print("0. Back to main menu")          choice = input("Enter choice: ")            if choice == "1":              create\_booking(tour\_agency)          elif choice == "2":              cancel\_booking(tour\_agency)          elif choice == "3":              add\_seats\_to\_booking(tour\_agency)          elif choice == "4":              list\_bookings(tour\_agency)          elif choice == "0":              break          else:              print("Invalid choice. Please try again.")  # Function to schedule a tour  def schedule\_tour(tour\_agency):      print("\n<< Schedule Tour >>")      tour\_code = input("Enter Tour Code: ")      tour = tour\_agency.searchTour(tour\_code)      if not tour:          print("Tour not found.")          return        print("Available scheduled tours:")      print(tour\_agency.listScheduledTours())      schedule\_code = input("Enter Schedule Code: ")      departure\_date = input("Enter Departure Date and Time (YYYY-MM-DD HH:MM): ")      lang = input("Enter Language: ")      capacity = int(input("Enter Capacity: "))      peak = input("Is it Peak (Yes/No): ").lower() == "yes"        try:          departure\_datetime = datetime.strptime(departure\_date, "%Y-%m-%d %H:%M")      except ValueError:          print("Invalid date and time format. Please enter in YYYY-MM-DD HH:MM format.")          return        scheduled\_tour = ScheduledTour(schedule\_code, tour, departure\_datetime, lang, capacity, peak)      tour\_agency.scheduled\_tours[tour\_code].append(scheduled\_tour)      print("Scheduled tour added successfully.")  # Function to open/close scheduled tour  def open\_close\_scheduled\_tour(tour\_agency):      print("\n<< Open/Close Scheduled Tour >>")      schedule\_code = input("Enter Scheduled Tour Code: ")      scheduled\_tour = tour\_agency.searchScheduledTour(schedule\_code)      if not scheduled\_tour:          print("Scheduled tour not found.")          return        if scheduled\_tour.status == "Yes":          action = input("This scheduled tour is currently open. Do you want to close it? (Yes/No): ")          if action.lower() == "yes":              scheduled\_tour.\_status = False              print("Scheduled tour closed successfully.")      else:          action = input("This scheduled tour is currently closed. Do you want to open it? (Yes/No): ")          if action.lower() == "yes":              scheduled\_tour.\_status = True              print("Scheduled tour opened successfully.")  # Function to remove a scheduled tour  def remove\_scheduled\_tour(tour\_agency):      print("\n<< Remove Scheduled Tour >>")      schedule\_code = input("Enter Scheduled Tour Code to remove: ")      scheduled\_tour = tour\_agency.searchScheduledTour(schedule\_code)      if not scheduled\_tour:          print("Scheduled tour not found.")          return        print(scheduled\_tour)      confirmation = input("Are you sure you want to remove this scheduled tour? (Yes/No): ")      if confirmation.lower() == "yes":          if scheduled\_tour.seatsAvailable == scheduled\_tour.capacity:              del tour\_agency.scheduled\_tours[scheduled\_tour.\_tour.code][schedule\_code]              print("Scheduled tour removed successfully.")          else:              print("Cannot remove scheduled tour. Some seats are already booked.")      else:          print("Operation cancelled.")  # Function to list all scheduled tours  def list\_scheduled\_tours(tour\_agency):      print("\n<< List Scheduled Tours >>")      print(tour\_agency.listScheduledTours())  # Function to create a booking  def create\_booking(tour\_agency):      print("\n<< Create Booking >>")      passport\_number = input("Enter Passport Number: ")      customer = tour\_agency.searchCustomer(passport\_number)      if not customer:          print("Customer not found.")          return        print("List of open scheduled tours:")      print(tour\_agency.listOpenScheduledTours())      schedule\_code = input("Enter Scheduled Tour Code: ")      scheduled\_tour = tour\_agency.searchScheduledTour(schedule\_code)      if not scheduled\_tour:          print("Scheduled tour not found.")          return        booking\_type = input("Enter Booking Type (I for Individual, G for Group): ").upper()      if booking\_type == "I":          single = input("Is Single Room required? (Yes/No): ").lower() == "yes"          booking = IndividualBooking(scheduled\_tour, customer, single)      elif booking\_type == "G":          customers = [customer]          while True:              passport\_number = input("Enter Passport Number (<enter> to stop): ")              if not passport\_number:                  break              customer = tour\_agency.searchCustomer(passport\_number)              if not customer:                  print("Customer not found.")                  continue              customers.append(customer)          booking = GroupBooking(scheduled\_tour, customers)      else:          print("Invalid booking type.")          return        tour\_agency.bookings.append(booking)      print("Booking created successfully.")  # Function to cancel a booking  def cancel\_booking(tour\_agency):      print("\n<< Cancel Booking >>")      booking\_id = int(input("Enter Booking Id: "))      booking = tour\_agency.searchBooking(booking\_id)      if not booking:          print("Booking not found.")          return        print(booking)      cancellation\_date = datetime.today()      penalty\_rate = booking.scheduledTour.getPenaltyRate(cancellation\_date)      penalty\_amount = min(penalty\_rate \* booking.cost(), booking.cost())      print(f"Penalty Amount: ${penalty\_amount:.2f}")      confirmation = input("Are you sure you want to cancel this booking? (Yes/No): ")      if confirmation.lower() == "yes":          tour\_agency.bookings.remove(booking)          booking.scheduledTour.cancelSeats(booking.seats)          print("Booking cancelled successfully.")      else:          print("Operation cancelled.")  # Function to add seats to a booking  def add\_seats\_to\_booking(tour\_agency):      print("\n<< Add Seats to Booking >>")      booking\_id = int(input("Enter Booking Id: "))      booking = tour\_agency.searchBooking(booking\_id)      if not booking:          print("Booking not found.")          return        if isinstance(booking, IndividualBooking):          print("Individual booking cannot add more travelers.")          return        print("List of customers already booked:")      print("\n".join(str(customer) for customer in booking.customers))      passport\_numbers = []      while True:          passport\_number = input("Enter Passport Number (<enter> to stop): ")          if not passport\_number:              break          customer = tour\_agency.searchCustomer(passport\_number)          if not customer:              print("Customer not found.")              continue          if customer in booking.customers:              print("Customer already booked.")              continue          passport\_numbers.append(passport\_number)        customers = [booking.scheduledTour.searchCustomer(passport\_number) for passport\_number in passport\_numbers]      if not all(customers):          print("Some customers not found.")          return        if not booking.scheduledTour.bookSeats(len(customers)):          print("No available seats for additional customers.")          return        booking.customers.extend(customers)      print("Seats added to booking successfully.")      additional\_cost = booking.scheduledTour.cost() \* len(customers)      print(f"Additional Cost: ${additional\_cost:.2f}")  # Function to list all bookings  def list\_bookings(tour\_agency):      print("\n<< List Bookings >>")      if not tour\_agency.bookings:          print("No bookings found.")          return      print("\n".join(str(booking) for booking in tour\_agency.bookings))  if \_\_name\_\_ == "\_\_main\_\_":      main() |

## Tour Management

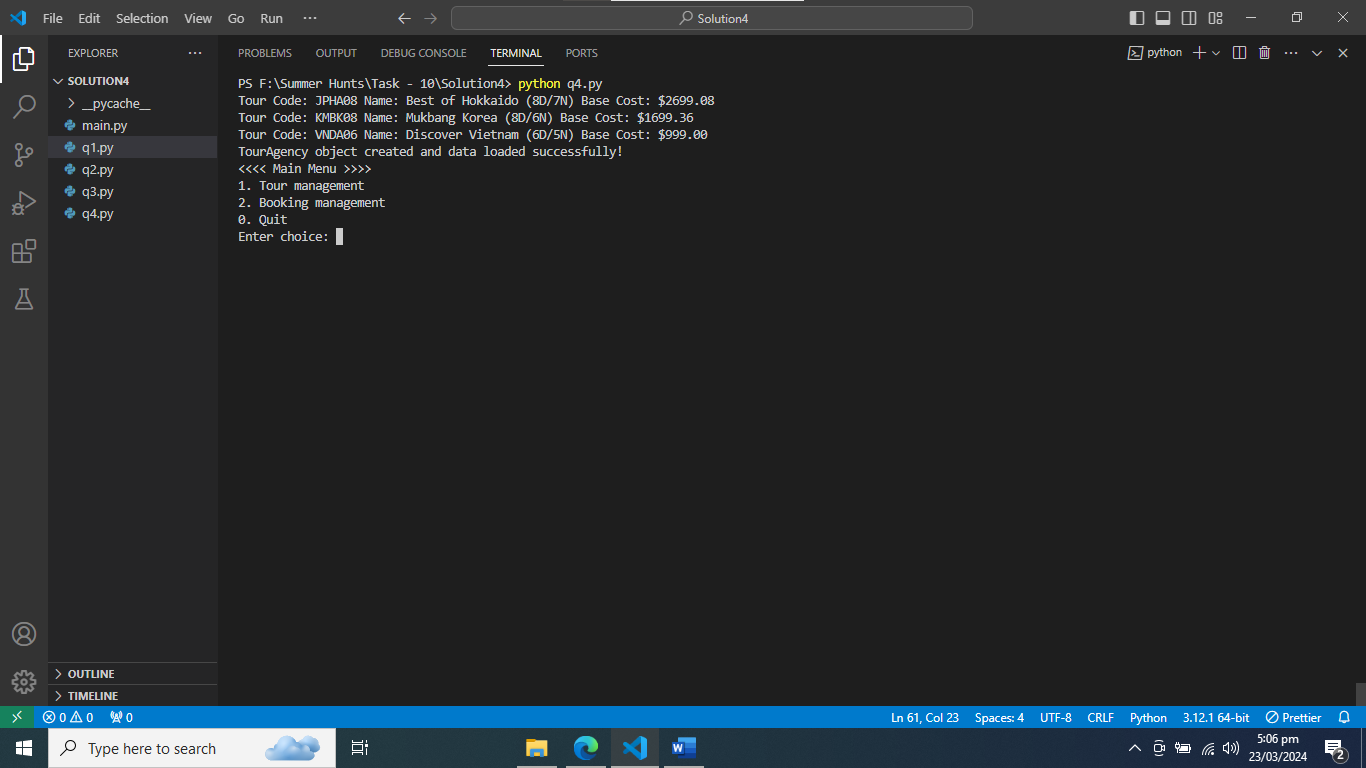


Figure : Tour Management

## Tour Menu Option 1: Schedule Tour

### Schedule Tour Option 1 at Schedule Tour

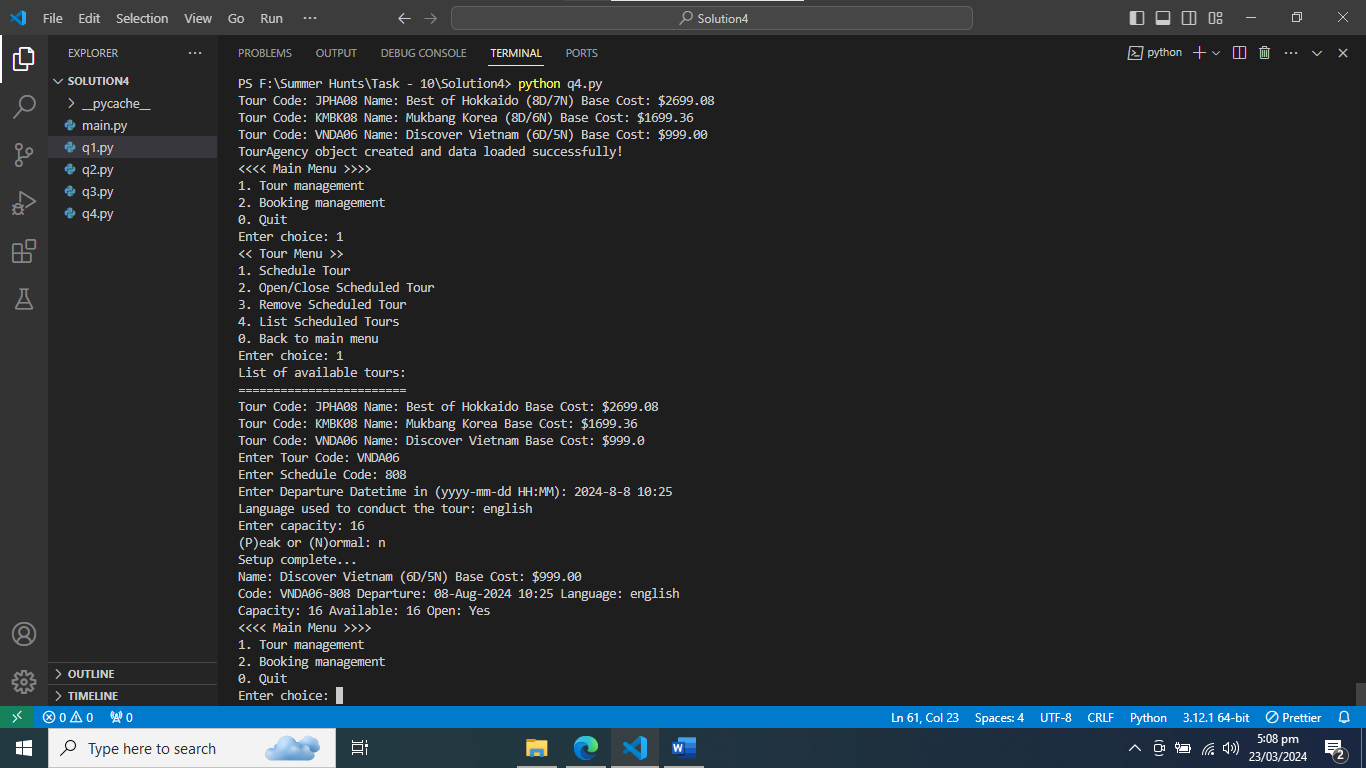


Figure : Schedule Tour Option 1 at Schedule Tour

### Open/Close Schedule Option 2 at Tour at Schedule Tour

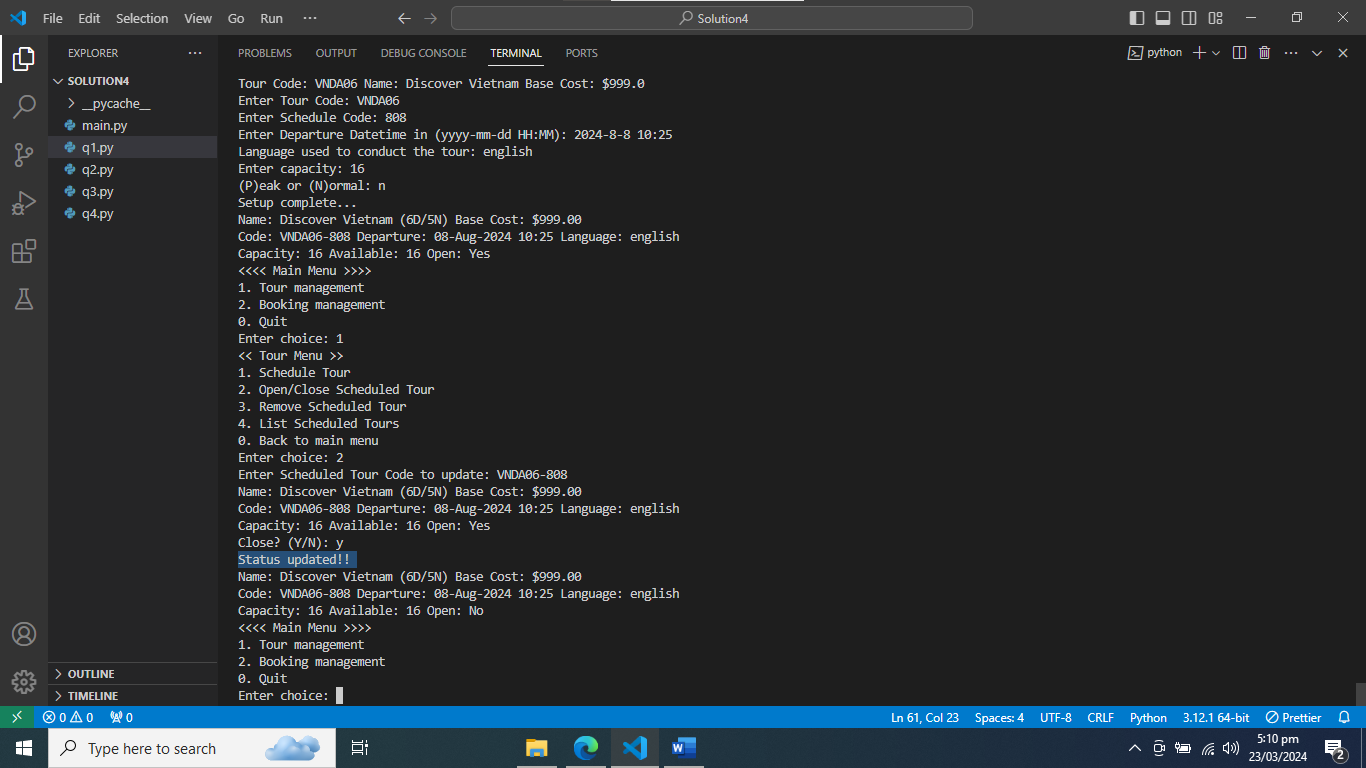


Figure : Open/Close Schedule Option 2 at Tour at Schedule Tour

### Remove Schedule Tour Option 3 at Schedule Tour

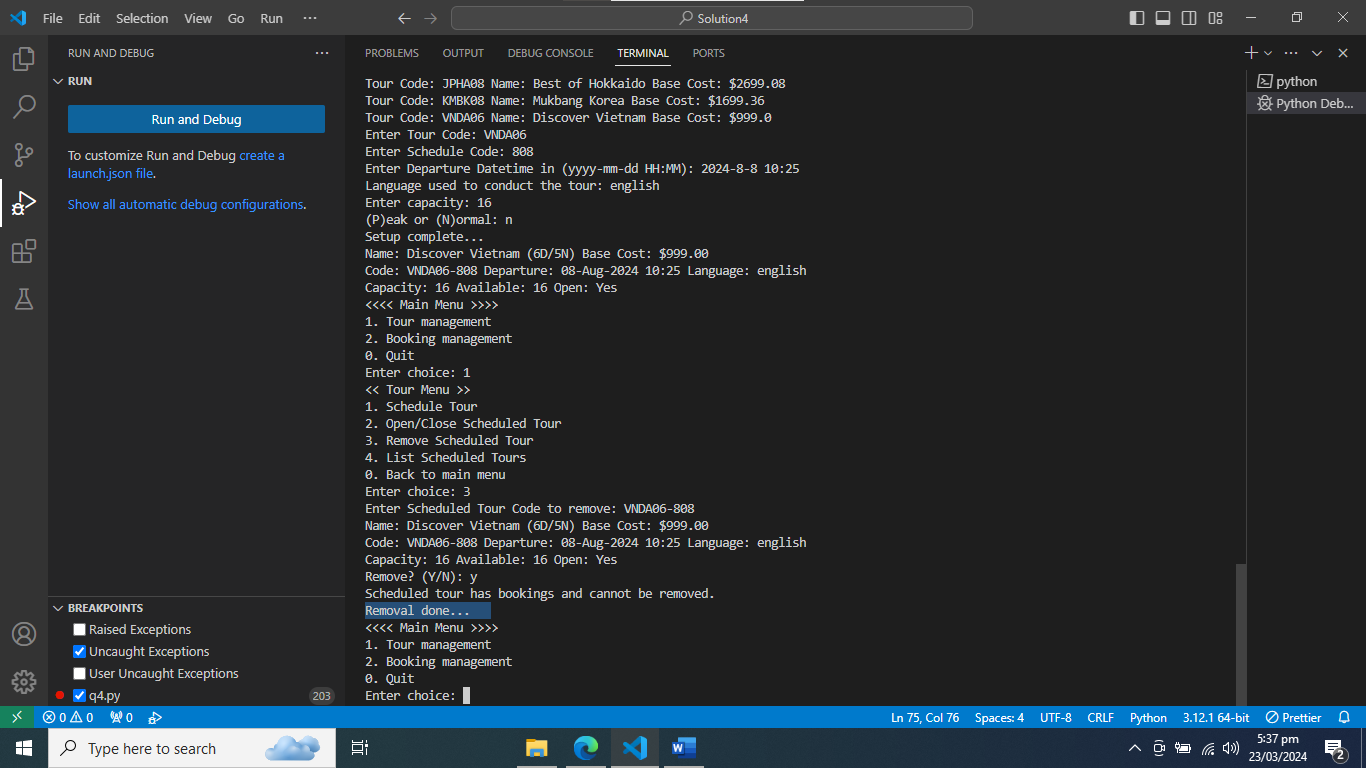


Figure : Removal Schedule Tour Option 3 at Schedule Tour

### List Scheduled Tours

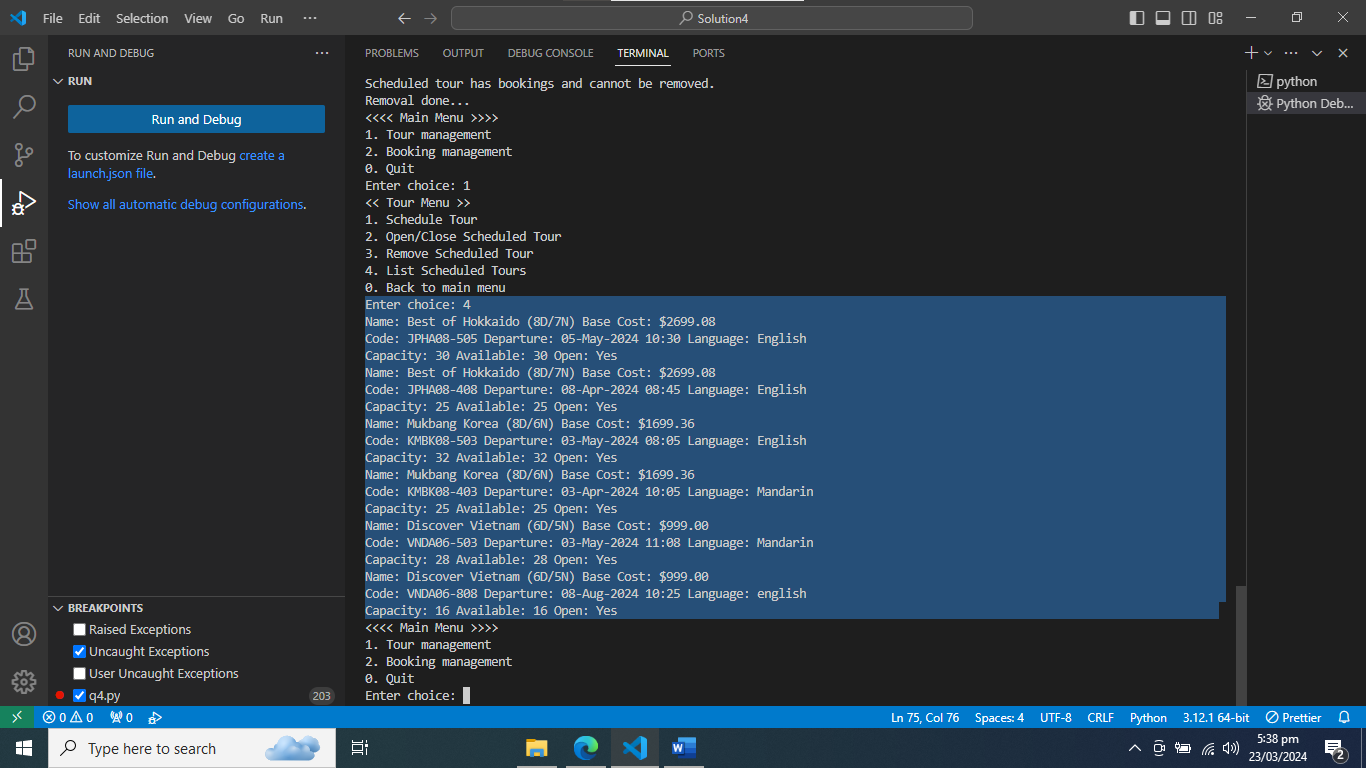


Figure : List Scheduled Tours Option 4 at Schedule Tour

## Booking management

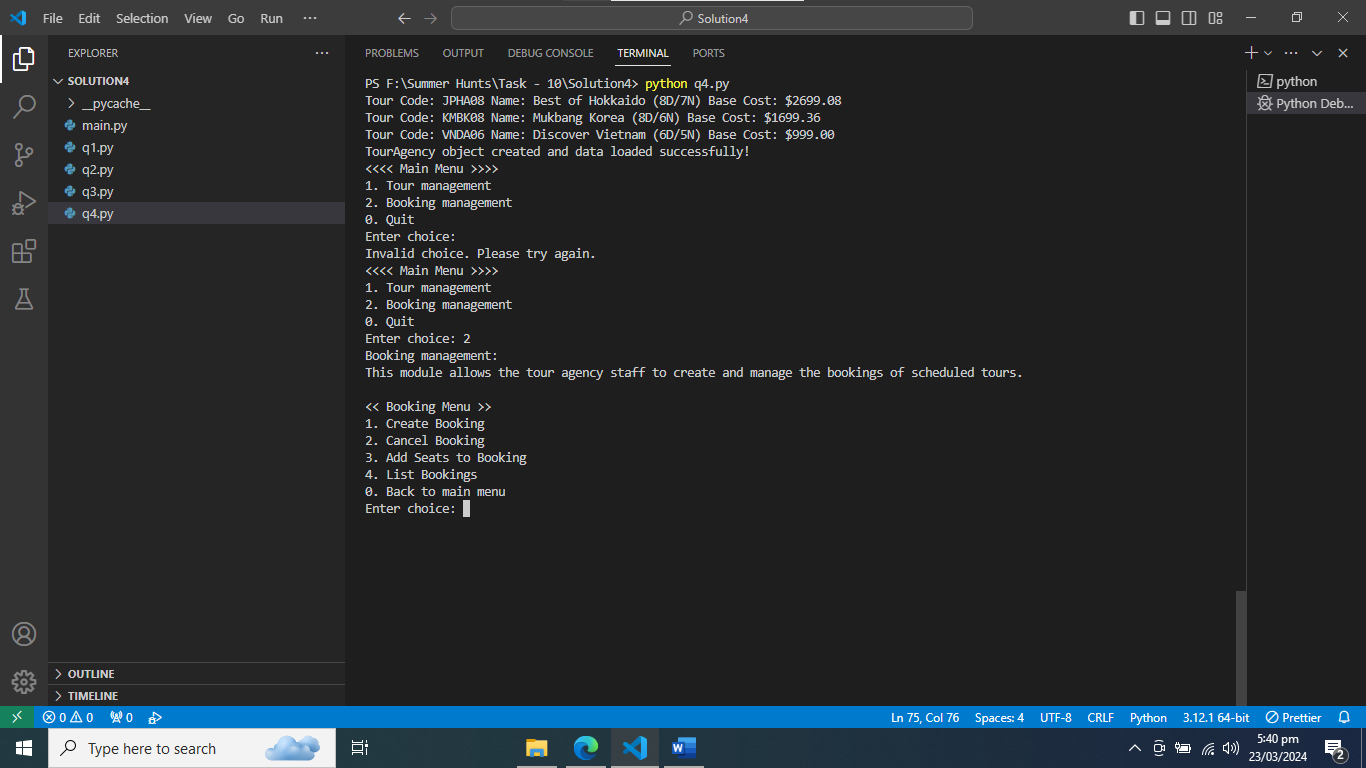


Figure : Booking Management

### Create Booking Option 1 at Booking Management

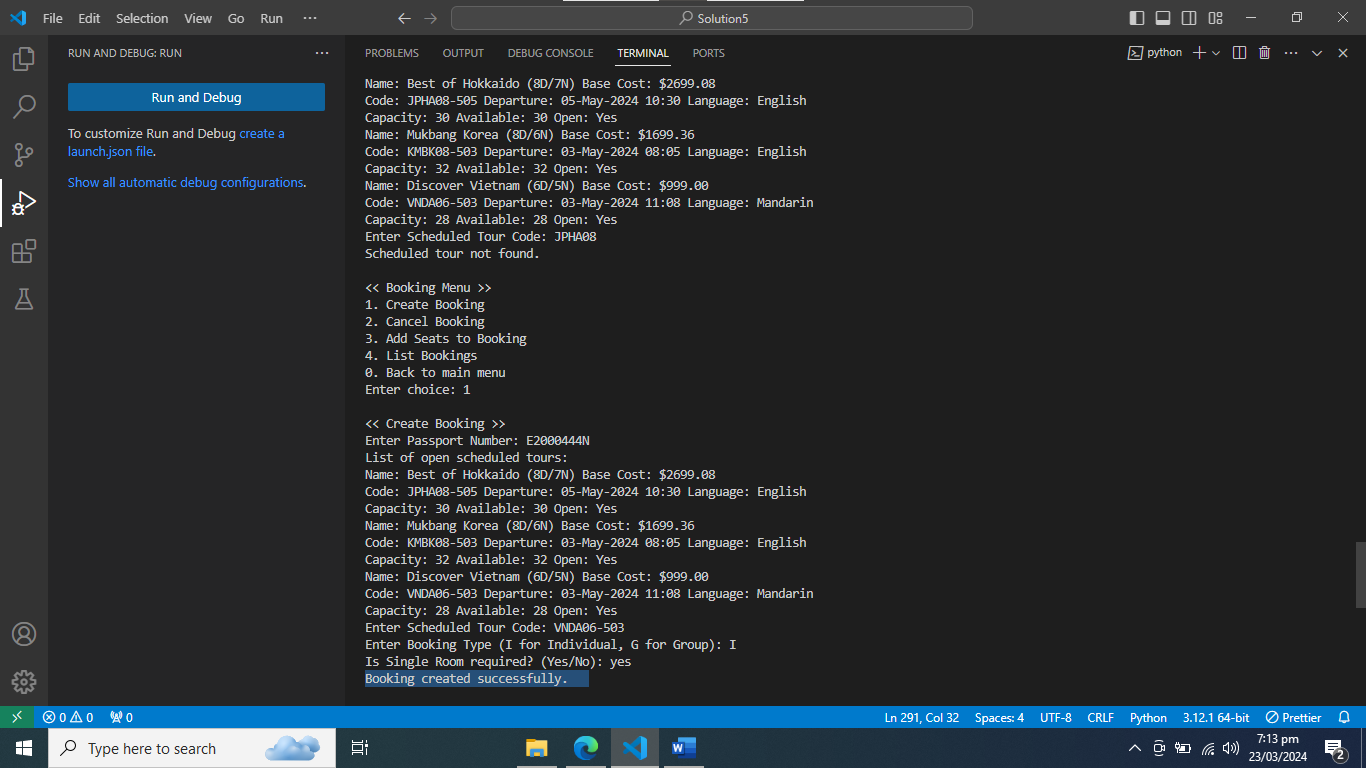


Figure : Create Booking Option 1 at Booking Management

### Cancel Booking Option 2 at Booking Management

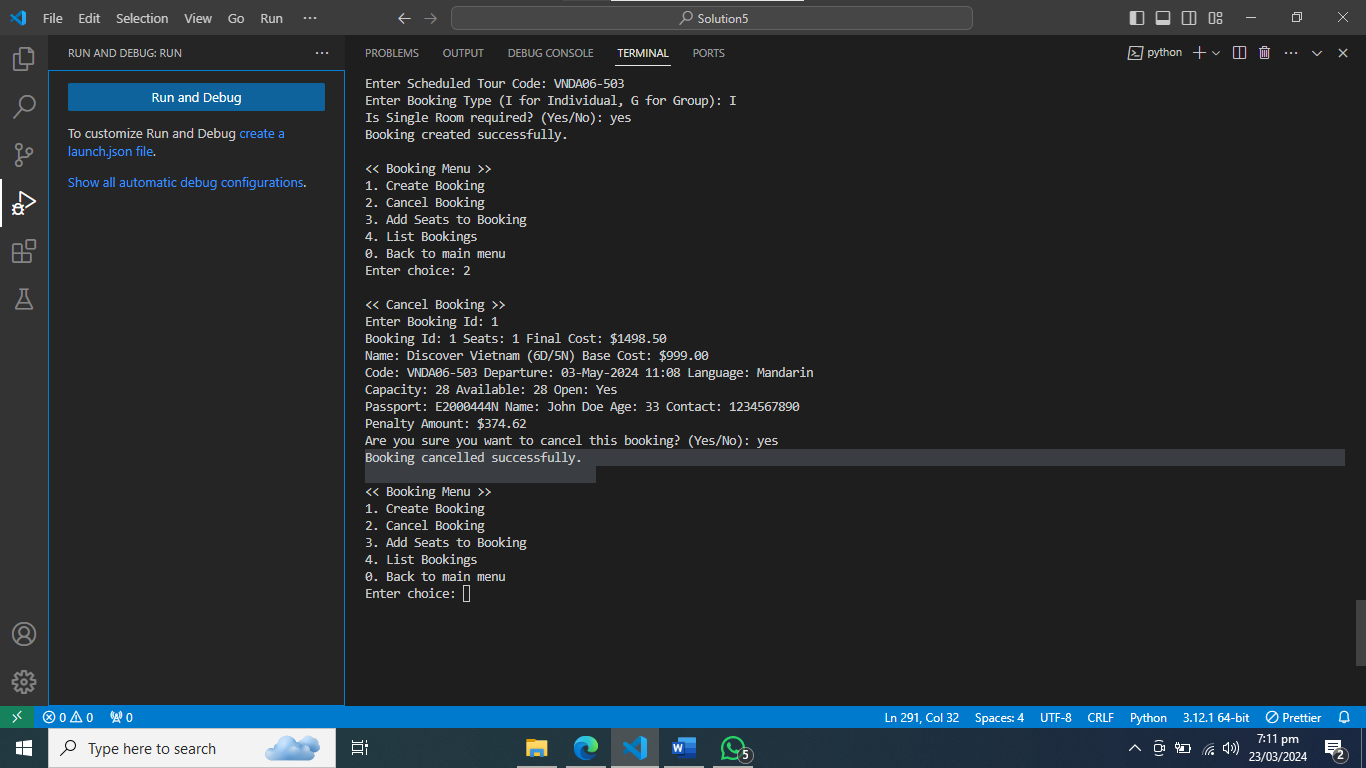


Figure : Cancel Booking Option 2 at Booking Management

### Add Seats to Booking

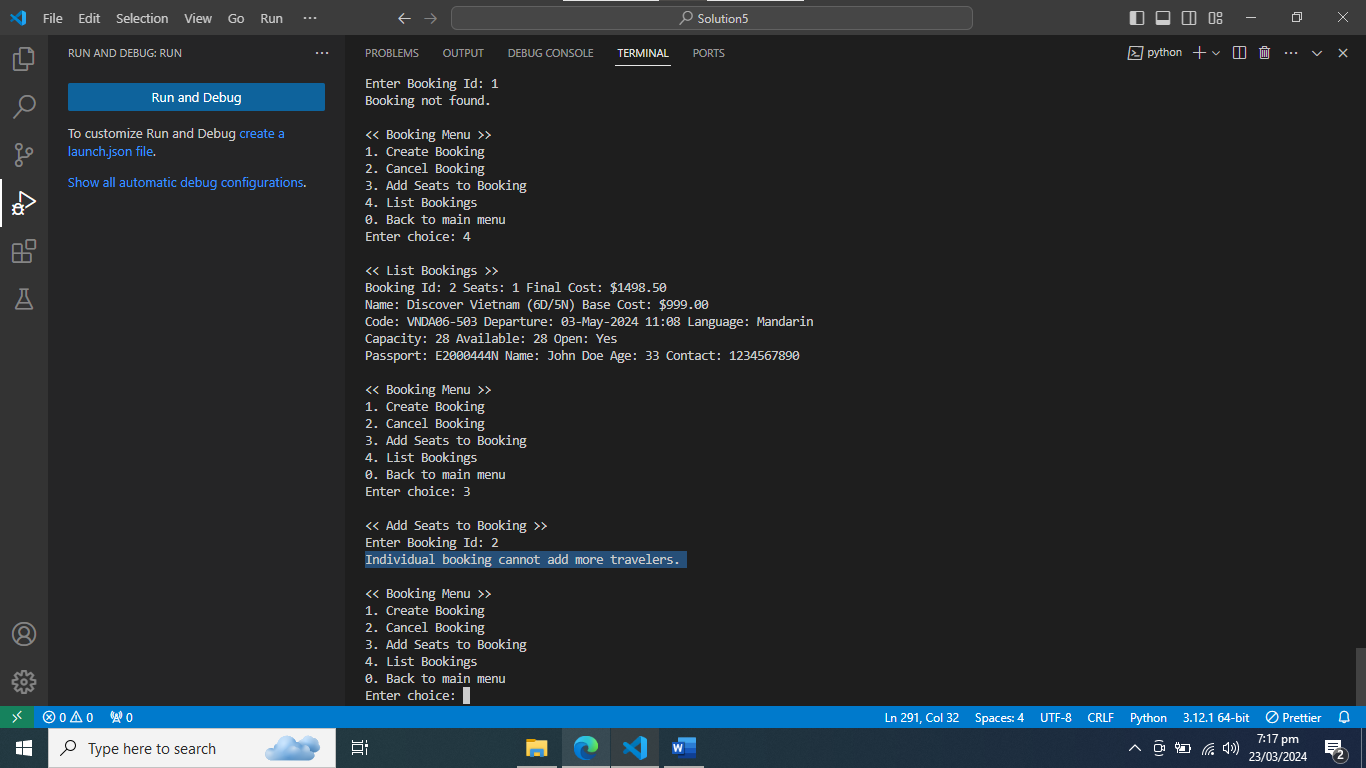


Figure : Add Seats to Booking Option 3 at Booking Management

### List Bookings Option 4 at Booking Management

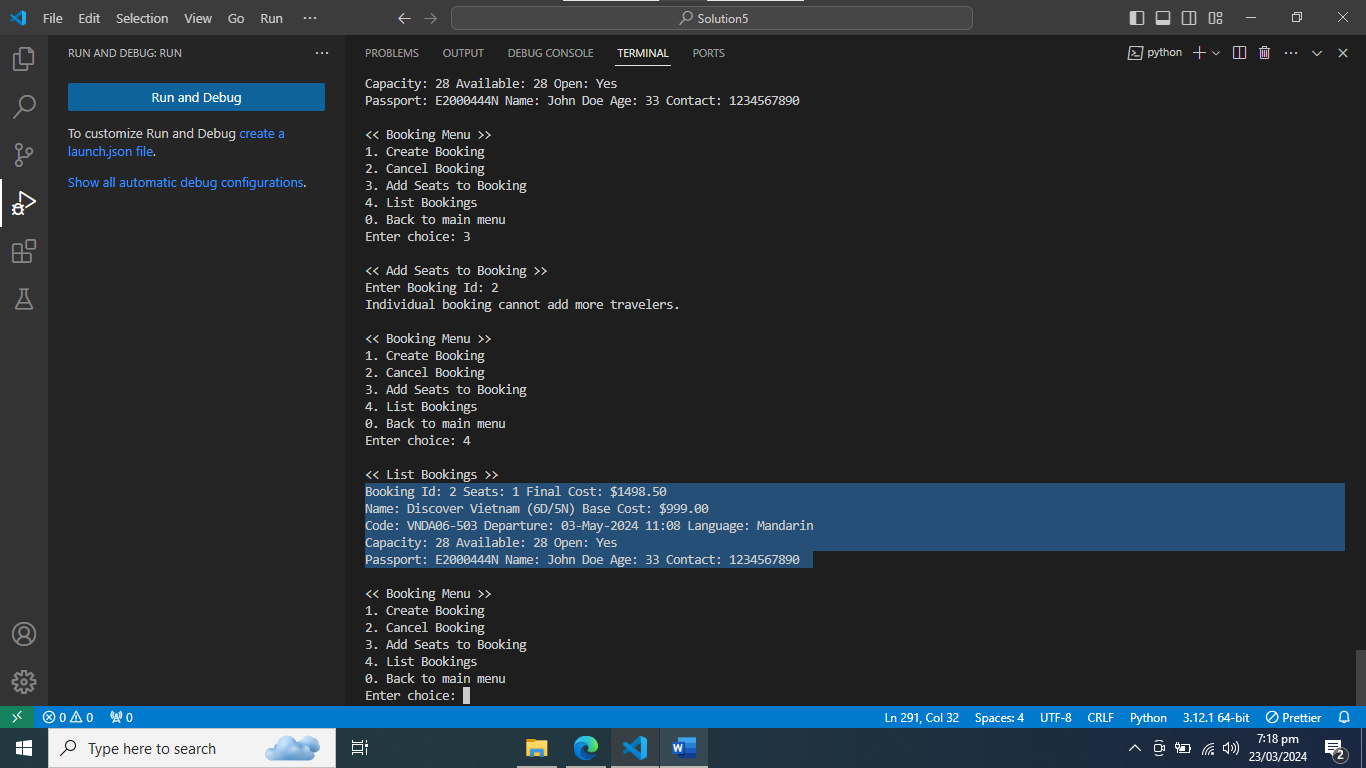


Figure : List Bookings Option 4 at Booking Management