BAN401: Mid-Term Assignment Report

Problem 1

Code

```
# Import required packages
import numpy as np

# Create the function
def roe_func(net_income, beg_equity, end_equity):
    # Compute the Average Shareholder's Equity for the year
    avg_equity = (beg_equity+end_equity)/2

    # Compute the Return on Equity according to the given equation
    roe = net_income/avg_equity

    # Return ROE rounded off to 2 decimal places
    return round(roe, 2)

# Print ROE of Apple in the console
apple_ROE = (roe_func(59500000000, 134000000000, 107100000000))
print('ROE of Apple for 2018 is ' + str(apple_ROE))
```

Detailed Explanation

We create a function with 3 variables - net income, shareholders' equity at the beginning of the year and shareholders' equity at the end of the year. The function first calculates the average equity then calculates the ROE before rounding off the ROE to 2 decimal places.

Result

```
ROE of Apple for 2018 is 0.49

Process finished with exit code 0
```

Problem 2

Code

```
# Import required packages
import numpy as np

# Insert given unemployment information into a dictionary
unemp = {'Australia': 5.92,
        'Brazil': 12.77,
        'China': 3.90,
        'Finland': 8.53,
        'Norway': 4.22,
        'South Africa': 27.45,
        'Iran': 11.81}

# Retrieve Norway's unemployment rate using a key
print('Unemployment rate of Norway is ' + str(unemp['Norway']))

# Find out the position of the countries with the highest and lowest unemployment
highest_ue_pos = np.argmax(list(unemp.values()), axis = None)
lowest_ue_pos = np.argmin(list(unemp.values()), axis = None)

# Extract the countries with the highest and lowest unemployment
highest_country = list(unemp.keys())[highest_ue_pos]
lowest_country = list(unemp.keys())[lowest_ue_pos]

# Print the results of which country has highest/lowest unemployment
print(lowest_country,
     "has the lowest and",
     highest_country,
     "has the highest unemployment rate.")
```

Detailed Explanation:

We create a dictionary containing the respective country's unemployment rate. We then retrieve access Norway's unemployment rate by using the key 'Norway'.

To find the countries with the highest and lowest unemployment rate, we first convert the dictionary values to a list. By using numpy package's argmax and argmin functions, we extract the indexes of the highest and lowest unemployment rates. We then convert the dictionary keys

to a list. By using the indexes extracted previously, we can extract the corresponding keys, to access the countries with the highest and lowest unemployment rates.

Result

```
Unemployment rate of Norway is 4.22
China has the lowest and South Africa has the highest unemployment rate.

Process finished with exit code 0
```

Problem 3

Code

```
word = 'Deceptivenesses'

# Create empty list to convert the string to a list
word_list = []

# Store each character of the string in a list
for i in word:
    word_list.append(i)

count_e = 0

# Use a for-loop to determine the number of times 'e' appears in the given string
for i in word_list:
    if i=='e':
        count_e += 1

print("The letter 'e' occurs", count_e, "times.")
```

Detailed Explanation

To convert the string into a list, we first created an empty list “word_list” to store each letter in the word ‘Deceptivenesses’ as an element of that list. We then iterated each letter from the string “word” and added each letter to “word_list” using the append function.

To determine the number of times ‘e’ appears in the word, we first created a variable count_e with a value of 0. We then iterated each element from “word_list” and checked if it is equal to ‘e’. By using the if function, if the letter == ‘e’, the value 1 is added to “count_e”. For each iteration, “count_e”’s value will then increase by 1 if the element is ‘e’.

Result

```
The letter 'e' occurs 5 times.

Process finished with exit code 0
```

Problem 4

Code

```
def armstrong_check(start, stop):
    # Use a for loop to check for armstrong numbers between the given range
    armstrong_count = 0
    armstrong_numbers = []

    for i in range(start, stop+1):
        number = str(i)
        x = len(number)
        total = 0

        # Use another for loop to determine the sum of all digits raised to the power of N
        for n in number:
            total += int(n)**x

        # Check if the sum total of digits raised to power N is the same as the beginning number,
        # print if it is an armstrong number
        if total == i:
            armstrong_numbers.append(str(i))
            armstrong_count += 1

    # Print the number of Armstrong numbers in the given range, and the value of those numbers
    print("There are", armstrong_count, "armstrong numbers, and they are:",
          ', '.join(armstrong_numbers) + ".")

#Print the number of Armstrong numbers between 100 and 1000
armstrong_check(100, 1000)
```

Detailed Explanation

We create the function “armstrong_check” with 2 variables, which will be the first and last number of the desired range. Within that function, we first create the variable “armstrong_count” with a value of 0, which will be updated later on with the number of armstrong numbers in a given range. We also create an empty list “armstrong_number” to store all the armstrong numbers in a given range.

Each number in the range is then iterated. For each iteration:

- The number will be converted into a string called “number”

- The length of the string is calculated and stored in the variable “x”
- The variable “total’ will be created with a value of 0
- Each element in the string “number” will be iterated. For each iteration, “total” will be updated by adding the result of each element raised to the power of “x” (where x = length of the number)
- If “total” is equal to the number being iterated, that number is added to the list “armstrong_number” and the value 1 is added to “armstrong_count”.

Result

```
There are 4 armstrong numbers, and they are: 153, 370, 371, 407.

Process finished with exit code 0
```

Problem 5

Code

```
# Define the solution within a function
def shipping_cost():
    # Store all the distances between destinations in a dictionary
    distances = {'B-O': 463.9, 'B-T': 628.7, 'O-T': 493.7}


    # Prompt the user for the origin and destination cities, and volume to be transported, and
    # store them in different variables
    print("To find the shipping cost please enter")
    origin_city = input("What is the origin city? ")
    dest_city = input("What is the destination? ")
    volume = float(input("What is the volume to be transported? "))

    # Remove whitespaces at the start and end of the origin and destination string.
    # Use the first letter of the locations as keys to the dictionary containing the strings.
    # Since the distance between 2 locations are the same regardless of the order,
    # try different orders (origin-destination, destination-origin) to get the distance
    try:
        shipping_path = origin_city.strip()[0].upper() + "-" + dest_city.strip()[0].upper()
        ship_distance = distances[shipping_path]
    except:
        shipping_path = dest_city.strip()[0].upper() + "-" + origin_city.strip()[0].upper()
        ship_distance = distances[shipping_path]

    # Calculate total shipping costs based on the distances and volume
    # and round to the nearest whole number.
    shipping_cost = round((volume*ship_distance/4) + 100)
    print("The total shipping cost is:", shipping_cost, "kr")

# Call the function
shipping_cost()
```

Detailed Explanation

A function “shipping_cost” is created. Within the function, the dictionary “distances” is created containing the different distances between destinations. Users are then required to key in 3 inputs - the origin city, destination city and the volume to be shipped.

Before we are able to access the different distances between cities, we must convert the inputs so that it matches the keys in the “distances” dictionary. To do so, we first create a variable “shipping_path”. We remove any whitespaces in the “origin_city” and “dest_city” inputs, access the first letters and convert them into uppercase letters to follow the format of the keys in the “distances” dictionary.

Since the distance between 2 cities are the same regardless of where the item comes from (e.g. Bergen to Oslo and Oslo to Bergen will have the same distance), “shipping_path” should be able to access the key regardless of the order. This is done using the try-except function. We first try to access the key by the original order given by the user. If an error occurs, Python will move on to the next chunk of code which flips the order. By doing this, the “shipping_path” will definitely match one of the keys in the “distances” dictionary to access the corresponding distance.

Finally, the shipping cost is calculated by plugging in the distance obtained from the “distance” dictionary and the “volume” input into the given formula.

Result

```
To find the shipping cost please enter
What is the origin city? Oslo
What is the destination? Bergen
What is the volume to be transported? 28
The total shipping cost is: 3347 kr

Process finished with exit code 0
```

Problem 6

Code

```
# Allow the administrator to set the inventory and price in the next line
store_rental = {'Inventory': 30, 'Price': 10}

# Store the initial inventory amount in a variable to reference against for price adjustments
initial_inv = store_rental['Inventory']

def customer_order():
    # Allow the use of the initial inventory variable in the function
    global initial_inv

    # Terminate the program when there are no more board left for rental
    if store_rental["Inventory"] == 0:
        print("There are no more boards")
        return
    else:
        # Display inventory of surfboard and price of rental for customers
        print("Our inventory of surfboards is currently:",
              store_rental["Inventory"])
        print("The price is",
              round(store_rental['Price']),
              "per rental + 10 per board")

        # Request for the amount of surfboard the customer wishes to rent
        order_count = int(input("How many boards would you like to rent? "))

        # Check that the requested order does not exceed the available inventory for order.
        # If it does, request for another order amount from the customer.
        while order_count > store_rental['Inventory']:
            print("Order exceeds inventory, please try again")
            order_count = int(input("How many boards would you like to rent? "))

        # After the order amount is successfully verified and made, update the amount of
        # surfboards in the inventory,
        store_rental['Inventory'] -= order_count

        # Check if the new inventory amount is below half of the initial inventory amount.
        # If it is, increase the rental price of each surfboard to 1.5 times.
        # "initial_inv" variable is set to 0 to prevent further price adjustments.
```

```
        if store_rental['Inventory'] <= (0.5*initial_inv):
            store_rental['Price'] *= 1.5
            initial_inv = 0

        # Function calls itself again for the next customer's order until the inventory is completely
        # depleted.
        customer_order()

# Run the function
customer_order()
```

Detailed Explanation

We created a dictionary “store_rental” where the administrator can set the inventory and price. The variable “initial_inv” is created which accesses the value of ‘Price’ from “store_rental” and will be used as reference for price adjustments.

The “customer_order” function is created which will prompt the user to input an order and the inventory and price will be shown. Within the “customer_order” function:

- If-else function is used to either terminate the function if there are no more boards left, else it will display the current inventory and price.
- “order_count” will ask for the customer to input how many surfboards they wish to rent
- While function is used to ensure that the order does not exceed inventory. The user will keep getting prompted to input another order unless their order is less than the current inventory.
- Once there is a valid order, the inventory will be updated by accessing the value of “Inventory” in “store_rental” and deducting the current order.
- If function is used to update the price if the inventory reaches half the “initial_inv”. If the condition is fulfilled, the “initial-inv” is also set to 0 to prevent any further price adjustments if the customer was to make another order.
- “customer_order” function is called again for the next order until there are no more boards left.

Result

```
Our inventory of surfboards is currently: 30
The price is 10 per rental + 10 per board
How many boards would you like to rent? 45
Order exceeds inventory, please try again
How many boards would you like to rent? 10
Our inventory of surfboards is currently: 20
The price is 10 per rental + 10 per board
How many boards would you like to rent? 10
Our inventory of surfboards is currently: 10
The price is 15 per rental + 10 per board
How many boards would you like to rent?  10
There are no more boards

Process finished with exit code 0
```

Problem 7

Code

```
def draw_square():
  # Request for the length of each side of the square
  square_length = input("Enter n dimension ")

  # Check that the function is an integer, not float, and that it is more than zero to determine a natural number.
  # If not, request for another entry from the user.
  while not(square_length.isdigit() and int(square_length) > 0):
     print("Incorrect dimensions, please enter only natural numbers for n!")
     square_length = input("Enter n dimension ")

  square_length = int(square_length)

  # Use a for-loop to print each line of a square
  for i in range(square_length):
     # Print the first and last row with n number of "#" signs with a space in between since there is a space in between
each newline.
     if i == 0 or i == (square_length - 1):
        print("# "*square_length)
     # For all intermediate rows between the first and the last, print only "#" corresponding to the first and last column.
Whitespaces are printed in between.
     else:
        print("#"+" "*((square_length-2)*2+1)+"#")

# Run the function
draw_square()
```

Detailed explanation

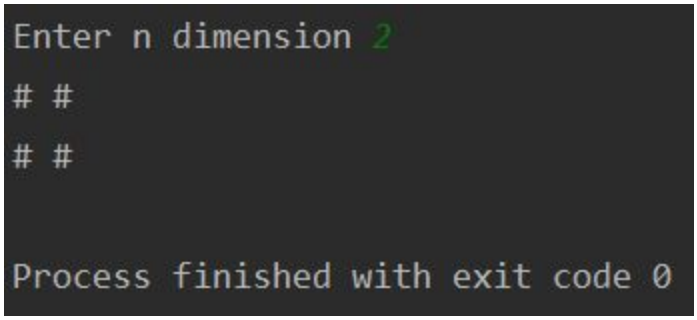
The idea of our designed solution is that firstly, we require the user to input the dimension of the square. This is solved via the input in the second line of the program, requesting the user to input the dimensions of the square.

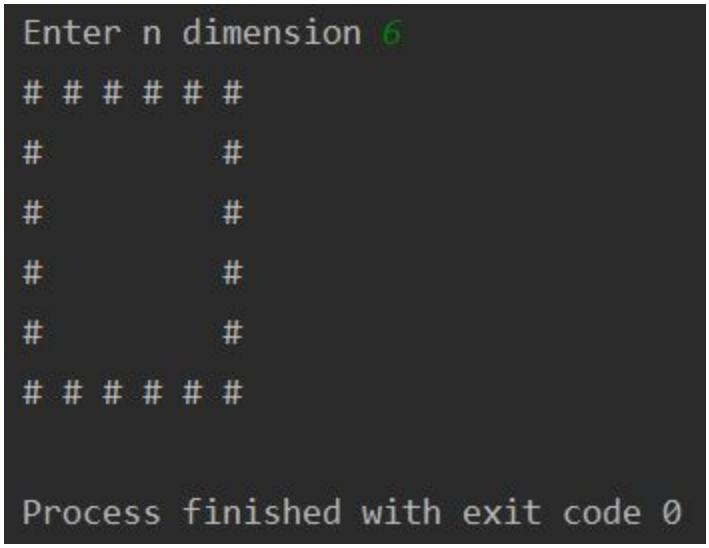
Next, we need our program to check if the input is a natural number. This would mean that the input must satisfy two conditions, the input must be an integer, not a float and the input must be more than zero. If either of these conditions are not met, then we print a message requiring the user to input a number that fulfils the criteria we set out.

Next, we can see from the example output in the assignment that the square is in the form of a matrix, which would require for-loops. In addition, the example output also has spaces between each hash.

From the above requirements, we first fill the first and last rows with hash signs with whitespaces between each hash sign. For the intermediate rows, we output hash signs at position 0 and position square_length – 1, with whitespaces in between.
The final line runs the function upon execution of the program.

Results





```
Enter n dimension 10
# # # # # # # # # #
#                 #
#                 #
#                 #
#                 #
#                 #
#                 #
#                 #
#                 #
# # # # # # # # # #

Process finished with exit code 0
```

Problem 8

Code

```
# Import required packages
from datetime import datetime, time

# Given function to check if a given time is between the start and end time
def isNowInTimePeriod(startTime, endTime, nowTime):
  if startTime < endTime:
    return nowTime >= startTime and nowTime <= endTime
  else: #Over midnight
    return nowTime >= startTime or nowTime <= endTime

def coffee_queue():
  weekdays = ["monday",
          "tuesday",
          "wednesday",
          "thursday",
          "friday",
          "saturday",
          "sunday"]

  # Prompt user for the day and check if its a valid day of the week, if not request for another input.
  day_of_week = input("Please enter the day: ")
  while not(day_of_week.lower() in weekdays):
    print("Please enter a valid day!")
    day_of_week = input("Please enter the day: ")

  # Prompt user for current hour (24-h format) and check if its a valid hour, if not request for another input.
  hour_of_day = input("Please enter the hour, 24h format: ")
  while not(hour_of_day.isdigit() and (int(hour_of_day) in list(range(24)))):
    print("Please enter a valid number!")
    hour_of_day = input("Please enter the hour, 24h format: ")
  hour_of_day = int(hour_of_day)

  # Prompt user for current minute of the hour and check if it is valid (between 0 and 59), if not request for another input.
  minutes = input("Please enter the minutes: ")
  while not(minutes.isdigit() and (int(minutes) in list(range(60)))):
    print("Please enter a valid number!")
    minutes = input("Please enter the minutes: ")
  minutes = int(minutes)

  # Use a few if-else gates to check if the given time falls within the long/medium/no queue periods. Print the
corresponding queue status accordingly.
  if day_of_week.lower() == "saturday" or day_of_week == "sunday":
```

```
        print("There is no queue at this time.")
    elif isNowInTimePeriod(time(10,0), time(10,15), time(hour_of_day,minutes)) or
isNowInTimePeriod(time(12,0), time(12,15), time(hour_of_day,minutes)) or
isNowInTimePeriod(time(14,0), time(14,15), time(hour_of_day, minutes)) or
isNowInTimePeriod(time(7,50), time(8,15), time(hour_of_day, minutes)):
        print("There is long queue at this time.")
    elif isNowInTimePeriod(time(11,30), time(12,0), time(hour_of_day,minutes)) or
isNowInTimePeriod(time(16,0), time(16,15), time(hour_of_day, minutes)):
        print("There is medium queue at this time.")
    else:
        print("There is no queue at this time.")

coffee_queue()
```

Detailed explanation

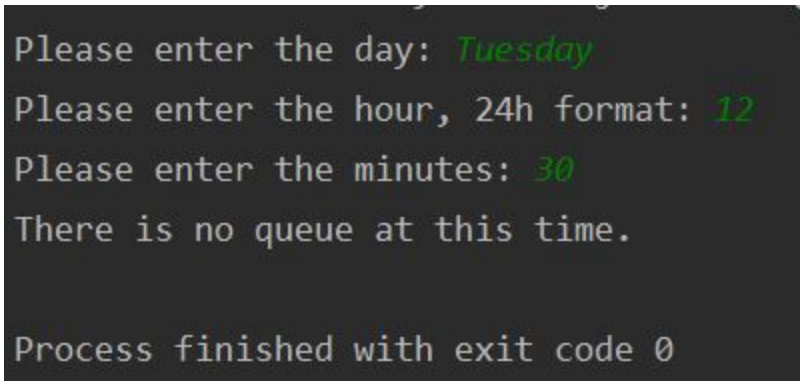
For problem 8, we need 3 inputs from the user.

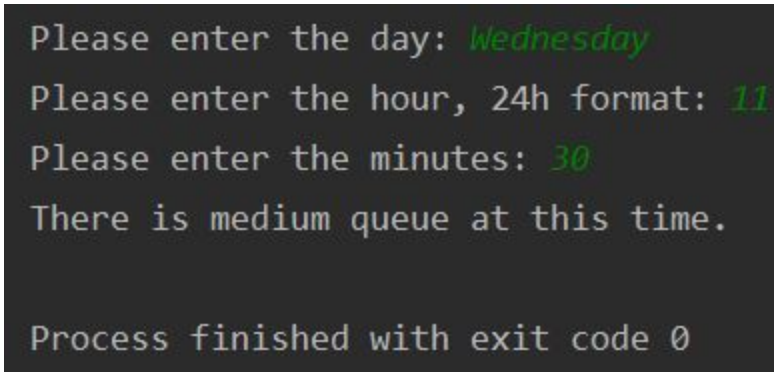
First is the day of the week. We can define days of the week in our program in a list, in order to prevent incorrect inputs or misspellings on the part of the user. After the user inputs the day of the week, our program checks if it corresponds to an element in our list weekdays. If it does not, the user will be prompted for a new input. This input is then recorded under the variable day_of_week.

Second is the hour input. Here our program checks if the input is valid via limiting the input from 0 to 23. This is done using the code list(range(24)) If it is not a valid hour, the user will be prompted to input a valid hour. This input is then recorded under the variable hour_of_day.

Last is the minute input. We limit the input to numbers from 0 to 59, using the same method as the hour input. If the given value is not a number or is not from 0 to 59, the user will be prompted to input a valid minute. This input is then recorded under the variable minutes.
Finally, we use the given assumptions in the problem, along with if-else logical gates to determine the final output of the program.

Results


Please enter the day: Tuesday
Please enter the hour, 24h format: 12
Please enter the minutes: 30
There is no queue at this time.
Process finished with exit code 0


Please enter the day: Wednesday
Please enter the hour, 24h format: 11
Please enter the minutes: 30
There is medium queue at this time.
Process finished with exit code 0

Problem 9

Code

```
def check_prime():
  # Prompt the user for an integer
  number = int(input("Enter an integer: "))
  divisor_count = 0
  divisors = []

  # Check for all the divisors for that number
  for i in range(1,number+1):
     if number % i == 0:
        divisor_count += 1
        divisors.append(i)


  # If there are only 2 divisors (1 and the number itself), the number is a prime number.
  # Print the result of the prime number check.
  if divisor_count == 2:
     print(number, "is a prime number!")
  else:
     print(number, "is not a prime number, and it's smallest divisor is " + str(divisors[1]))

check_prime()
```

Detailed explanation

The idea for our solution is that prime numbers are defined as having divisors of 1 and itself only. Therefore any number with only 2 divisors, besides 1, will be considered a prime number.

Using this condition, we check if the input is a prime number.

We first create a variable to count the number of divisors, divisor_count. Then, we create a list in which to store the divisors.

If the number has only 2 divisors, essentially having only 1 and itself as divisors, the number will be considered a prime number, and the user will be notified.

If the number has more than 2 divisors, that would mean that the number is not a prime number. In this case, our program would notify the user that the number is not a prime number, and print our position 1 of list “divisors”, thereby displaying the smallest divisor of the non-prime number.

Results

```
Enter an integer: 17
17 is a prime number!

Process finished with exit code 0
```

```
Enter an integer: 33
33 is not a prime number, and it's smallest divisor is 3

Process finished with exit code 0
```

Problem 10

Code

```
def word_in_sentence():
  # Request the user to input a sentence for checking
  sentence_input = input("Please enter a sentence (without any numbers, punctuation, or special characters): ")

  # Prompt for the word to check in the sentence
  word_input = input("Enter the word to check if it is in the sentence: ")

  # Generate a list of all the words in the sentence by splitting based on whitespaces in the sentence
  sentence_list = sentence_input.split(sep = " ")

  if word_input in sentence_list:
     # Return the position of the word in the sentence
     print("The word \"" + word_input + "\" is at position", sentence_list.index(word_input)+1, "in the sentence.")
  else:
     print("The word is not in the sentence")

word_in_sentence()
```

Detailed explanation

The idea of our solution is that our program will first require 2 inputs, a sentence and a word. Then, the program will check if the word is in the sentence. If it is, then the program should print the position of the respective word.

First, our program requests 2 inputs from our user, the sentence, which is assigned under the variable “sentence_input” and the word, which is assigned under the variable “word_input”.
The inputted sentence is then split into its constituent words using the string split method.

Next, the word_input is checked against the list created by sentence_input.split(). If the word is not inside the sentence, the program prints that the word is not in the sentence.

If the word is in the sentence, then the program prints that the word and the position of the word. Here, since Python has zero-based numbering, we would need to add 1 to the displayed position of the word, in order to conform to the required one-based numbering convention.

Results

```
Please enter a sentence (without any numbers, punctuation, or special characters): hello there
Enter the word to check if it is in the sentence: hello
The word "hello" is at position 1 in the sentence.

Process finished with exit code 0
```

```
Please enter a sentence (without any numbers, punctuation, or special characters): hello there
Enter the word to check if it is in the sentence: kenobi
The word is not in the sentence

Process finished with exit code 0
```

Problem 11

Code

```
# Import required packages
import math

# Prompt for the number of people attending the party
people = input("Enter the number of people that are going for the party: ")

def pizza(no_of_people):
  people = no_of_people

  # Check that the input is a digit and is in the range of 1 to 50
  while not(people.isdigit() and (int(people) in list(range(1,51)))):
     print("Incorrect input, please try again.")
     people = input("Enter the number of people that are going for the party: ")

  people = int(people)

  # Determine the number of pizza to order based on 4 person per pizza
  pizza_order = math.ceil(people/4)

  # Calculate the total price of the pizza order
  if 1 <= pizza_order <= 5:
     expected_expense = (pizza_order*20) + 5
  elif 6 <= pizza_order <= 10:
     expected_expense = (pizza_order*15) + 5
  elif pizza_order > 10:
     expected_expense = (pizza_order*10) + 5

  # Display the printed results of the total order price
  print("Estimated pizza cost for the party from Little Sicily is $" + str(expected_expense))

  return expected_expense

pizza(people)
```

Detailed explanation

The idea of our solution is based off the idea that each pizza has a variable cost depending on the total number of pizzas ordered and the given assumption that one pizza feeds four people.

First, our program prompts the user to input the number of people coming to the party. Our program then check if the input is a number, before using the input to derive the number of

pizzas needed for the party, using the variable “pizza_order”. This variable is derived using the math ceiling function, in order for the ordered pizzas to be at least enough for everyone.

Next, the variable is used to calculate the total price of the pizza order, inclusive of delivery fee, via a series of if-else statements.

Finally, the total price of the pizza order is printed for the user.

Results

```
Enter the number of people that are going for the party: 50
Estimated pizza cost for the party from Little Sicily is $135

Process finished with exit code 0
```

```
Enter the number of people that are going for the party: 9
Estimated pizza cost for the party from Little Sicily is $65

Process finished with exit code 0
```

```
Enter the number of people that are going for the party: 4
Estimated pizza cost for the party from Little Sicily is $25

Process finished with exit code 0
```

Problem 12

Code

```
# Import required packages
import math
people = input("Enter the number of people that are going for the party: ")
# Reuse pizza() from Q11 to determine price from Little Sicily
def pizza(no_of_people):
  people = no_of_people

  # Check that the input is a digit and is in the range of 1 to 50
  while not(people.isdigit() and (int(people) in list(range(1,51)))):
    print("Incorrect input, please try again.")
    people = input("Enter the number of people that are going for the party: ")

  people = int(people)

  # Determine the number of pizza to order based on 4 person per pizza
  pizza_order = math.ceil(people/4)

  # Calculate the total price of the pizza order
  if 1 <= pizza_order <= 5:
    expected_expense = (pizza_order*20) + 5
  elif 6 <= pizza_order <= 10:
    expected_expense = (pizza_order*15) + 5
  elif pizza_order > 10:
    expected_expense = (pizza_order*10) + 5

  # Display the printed results of the total order price
  # print("Estimated pizza cost for the party from Little Sicily is $" + str(expected_expense))

  return expected_expense

# Define function to determine price from Big Tuscany
def pizza_competitor(no_of_people):
  people = no_of_people

  # Check that the input is a digit and is in the range of 1 to 50
  while not (people.isdigit() and (int(people) in list(range(1, 51)))):
    print("Incorrect input, please try again.")
    people = input("Enter the number of people that are going for the party: ")

  people = int(people)

  # Determine the number of pizza to order based on 4 person per pizza
```

```
    pizza_order = math.ceil(people / 4)

    # Calculate total cost of the pizza order
    expected_expense = pizza_order*15

    # Add $15 delivery fee for orders that are below $75
    if expected_expense < 75:
        expected_expense += 15

    # print("Estimated pizza cost for the party from Big Tuscany is $" + str(expected_expense))

    return expected_expense

def cheapest(no_of_people):

    little_sicily = pizza(no_of_people)
    big_tuscany = pizza_competitor(no_of_people)

    # Compare prices between Little Sicily and Big Tuscany and return which is the cheaper option as well
as the price of the order
    if little_sicily < big_tuscany:
        print("If",
              no_of_people,
              "people came to the party, then the cheapest option would be Little Sicily and the cost would be $" +
              str(little_sicily))
    elif big_tuscany < little_sicily:
        print("If",
              no_of_people,
              "people came to the party, then the cheapest option would be Big Tuscany and the cost would be $" +
              str(big_tuscany))
    else:
        print("If",
              no_of_people,
              "people came to the party, then both Little Sicily and Big Tuscany costs the same, and the cost would be $" +
              str(little_sicily))

# Test the cheapest() function for 5, 20 and 50 people
# cheapest("5")
# cheapest("20")
# cheapest("50")

cheapest(people)
```

Detailed explanation

The idea of our solution is based off Problem 11. We recreate a similar function to calculate the total price of the pizza order for Big Tuscany, but with different if-else gates due to the different pricing structure of Big Tuscany.

Next, our function “cheapest” then compares the prices of Little Sicily and Big Tuscany. If both are the same price, our program would print that both pizza stores would cost the same, as well as the price for the pizza order. Otherwise, our program would state which option is the cheaper one, along with the total price of the pizza order.

Results