Activity
Persistence & Exception

Learning Outcome

On successful completion of this practical, the students should be able to:

- Explain the concepts of persistence.
- Explain the concepts of exception handling.
- Implement program that persist data and handle exceptions.

1. Implementing exception handling to program to maintain normal flow even when unexpected events occur.
   a) Identify the possible exceptions and explain what might happen if an exception is not handled in the given program:

```
count = int(input("How many number you want to capture? "))
numList = []
for i in range(count):
    msg = f"Enter number #{i+1}: "
    num = int(input(msg))
    numList.append(num)

print(f"The lowest number in the list: {min(numList)}")
print(f"The highest number in the list: {max(numList)}")
print(f"The total of the number in the list: {sum(numList)}")
print(f"The average of the number in the list:
{sum(numList)/len(numList)}")
```

   b) Modify the program, so that the program can handle the exception identify in part a).

   Note: When do we use the else and finally clause?

2. a) Download the BMI.py (from BrightSpace) and explain what the program does.

```
while True:
    name = input("Enter name: ")
    weight = float(input("Enter weight (kg): "))
    height = float(input("Enter height(m): "))

    bmi = weight / height ** 2
    print(bmi)

    command = input("Store your bmi to file? (Y/N): ")
    if command.upper() == "Y":
        bmi_File = open("bmi.txt", "a")
        bmi_File.write(f"{name},{bmi}\n")
        bmi_File.close()

    command = input("Do you want to continue? (Y/N): ")
    if command.upper() == "N":
        break

command = input("Do you want view BMI record in file? (Y/N): ")
if command.upper() == "Y":
    bmi_File = open("bmi.txt", "r")
    contents = bmi_File.read()
    print(contents)
    bmi_File.close()
```

b) Modify the codes to handle all the necessary exceptions.

Sample Output:

```
Enter name: Andrea
Enter weight (kg): 46
Enter height(m): 1.55
19.14672216441207
Store your bmi to file? (Y/N): y
Do you want to continue? (Y/N): y
Enter name: James
Enter weight (kg): 67
Enter height(m): 1.78
21.146319909102385
Store your bmi to file? (Y/N): y
Do you want to continue? (Y/N): n
Do you want view BMI record in file? (Y/N): y
Andrea,19.14672216441207
James,21.146319909102385


Process finished with exit code 0
```

c) Modify the following line of code to **write** mode.

```
bmi_File = open("bmi.txt", "a")
```

Sample Output:

```
Enter name: Karen
Enter weight (kg): 78
Enter height(m): 1.69
27.309968138370508
Store your bmi to file? (Y/N): y
Do you want to continue? (Y/N): n
Do you want view BMI record in file? (Y/N): y
Karen,27.309968138370508


Process finished with exit code 0
```

Note: What will be the output for append mode? Explain the difference you notice from the sample output.

3. Download the data file results.txt (from BrightSpace) that contains student result in Math, Chinese, English and Science separated by commas.

results.txt

```
John, 95, 92, 93, 91
May, 91, 89, 87, 88
James, 94, 93, 93, 92
Janice, 82, 85, 88, 99
Megan, 92, 93, 90, 91
April, 91, 93, 94, 95
June, 88, 98, 89, 90
July, 89, 92, 93, 91
David, 91, 91, 95, 92
Tom, 91, 89, 95, 96
Mike, 89, 92, 90, 90
Larry, 87, 89, 88, 95
Jane, 90, 93, 94, 95
Jenny, 92, 89, 91, 93
Kenny, 91, 88, 90, 92
```

Download the Main.py (from BrightSpace) that consists of class definition for Student and test program with 2 functions.

```
class Student:
    def __init__(self, name):
        self.name = name
        self.math = 0
        self.chinese = 0
        self.english = 0
        self.science = 0
        self.choices = []

    def get_score(self):
        return (self.math + self.chinese + self.english + self.science) / 4

def main():
    students = load_result()
    for s in students:
        print(s.name, s.get_score())

def load_result():
    students = []
    # implement the load result logic here
    return students

# start the test program
main()
```

a) Complete the implementation of load_result() function that loads the student result from results.txt and return a list of Student object.

The program should implement exception handling which displays "*File not found*" when the results.txt file cannot be found.

Sample Output:

```
John 92.75
May 88.75
James 93.0
Janice 88.5
Megan 91.5
April 93.25
June 91.25
July 91.25
David 92.25
Tom 92.75
Mike 90.25
Larry 89.75
Jane 93.0
Jenny 91.25
Kenny 90.25
```

b) The students can choose the 3 schools that they want to apply in the order of their preference. Assume that all students have chosen SchoolA, SchoolB, SchoolC as their choices in the same order. Modify the main() function so that students choices are stored in the data attribute - `choices`. Write the code to display all students' average score and the choices they made.

The sample output is given as shown below.

```
John scores 92.75, the choices are SchoolA, SchoolB, SchoolC
May scores 88.75, the choices are SchoolA, SchoolB, SchoolC
James scores 93.00, the choices are SchoolA, SchoolB, SchoolC
Janice scores 88.50, the choices are SchoolA, SchoolB, SchoolC
Megan scores 91.50, the choices are SchoolA, SchoolB, SchoolC
April scores 93.25, the choices are SchoolA, SchoolB, SchoolC
June scores 91.25, the choices are SchoolA, SchoolB, SchoolC
July scores 91.25, the choices are SchoolA, SchoolB, SchoolC
David scores 92.25, the choices are SchoolA, SchoolB, SchoolC
Tom scores 92.75, the choices are SchoolA, SchoolB, SchoolC
Mike scores 90.25, the choices are SchoolA, SchoolB, SchoolC
Larry scores 89.75, the choices are SchoolA, SchoolB, SchoolC
Jane scores 93.00, the choices are SchoolA, SchoolB, SchoolC
Jenny scores 91.25, the choices are SchoolA, SchoolB, SchoolC
Kenny scores 90.25, the choices are SchoolA, SchoolB, SchoolC

Process finished with exit code 0
```

4. You are going to create a phone inventory system that stores Phone objects in a shelve. The program should present a menu that lets the user perform the following actions:
   - Search for a phone
   - Add a new phone
   - Update an existing phone's make, model, and price
   - Delete a phone
   - Display all phones
   - Quit the program

a) Using the Phone class created from **addition activity in Classes, Objects and Methods** or from **Question 6 in Classes, Objects and Methods** and modify the Phone class:
   - Add in data attribute id for uniquely identifying each phone. initializes it to 'None'
   - Add in accessor and mutator methods for id

b) Download the test program PShelve.py (from BrightSpace). Complete the function for search(), add(), update(), delete() and display_all()

c) The test program must also implement necessary **exception handling** for persistent storage Shelve. You may decide the name of the persistent storage and modify the test program and display_menu() if necessary for codes related to persistent storage

   Note: With the persistent storage, how the Phone Inventory System different from Phone Inventory System in additional activity question in Classes, Objects and Methods? Discuss the differences between them.

   The sample output is given below:

```
Select the program (1-6) to run:
1. Search for a phone
2. Add a new phone
3. Update phone details
4. Delete a phone
5. Display all phones
6. Quit the program
Enter your command (1-6): 2
Enter phone id: 100
Enter phone make: Apple
Enter phone model: iPhone13
Enter price of phone: 1900
Select the program (1-6) to run:
1. Search for a phone
2. Add a new phone
3. Update phone details
4. Delete a phone
5. Display all phones
6. Quit the program
Enter your command (1-6): 3
Enter the phone id to update: 100
What is the new make? (Leave empty to remain unchange):
What is the new model? (Leave empty to remain unchange): iPhone 13 Pro
```

```
What is the new price? (Leave empty to remain unchange):
Phone: 100 model updated
Select the program (1-6) to run:
1. Search for a phone
2. Add a new phone
3. Update phone details
4. Delete a phone
5. Display all phones
6. Quit the program
Enter your command (1-6): 1
Enter the phone id to search: 101
The phone created is Samsung S21 priced at $1700. Now has 2 phone in total
Select the program (1-6) to run:
1. Search for a phone
2. Add a new phone
3. Update phone details
4. Delete a phone
5. Display all phones
6. Quit the program
Enter your command (1-6): 5
The phone created is Samsung S21 priced at $1700. Now has 2 phone in total
The phone created is Apple iPhone 13 Pro priced at $1900. Now has 2 phone in total
Select the program (1-6) to run:
1. Search for a phone
2. Add a new phone
3. Update phone details
4. Delete a phone
5. Display all phones
6. Quit the program
Enter your command (1-6): 6
End of program

Process finished with exit code 0
```

Remarks: Assuming phone - Samsung was added to shelve in the previous run.

-End-