WRITE A PROGRAM TO IMPLEMENT FILE AND DIRECTORIES.

```
import os
import shutil

# Create a file
file_path = "example.txt"
with open(file_path, "w") as f:
    f.write("Hello, world!")

# Read from a file
with open(file_path, "r") as f:
    contents = f.read()
    print(contents)

# Write to a file
with open(file_path, "a") as f:
    f.write("\nHow are you?")
with open(file_path, "r") as f:
    contents = f.read()
    print(contents)

# Remove the file
os.remove(file_path)

# Create a directory
dir_path = "example_dir"
os.mkdir(dir_path)

# Copy a file to the directory
shutil.copy(file_path, dir_path)

# Move the file to the directory
shutil.move(file_path, dir_path)
```

```
# Remove the directory and all of its contents
shutil.rmtree(dir_path)
```

#OUTPUT

```
Hello, world!
Hello, world!
How are you?
```

WRITE A PROGRAM TO IMPLEMENT EXCEPTION HANDLING IN PYTHON

```
try:
    # code that might raise an exception
    x = int(input("Enter a number: "))
    y = int(input("Enter another number: "))
    print(x / y)
except ZeroDivisionError:
    # handle the ZeroDivisionError exception
    print("Cannot divide by zero")
except ValueError:
    # handle the ValueError exception
    print("Invalid input. Please enter a number.")
except:
    # handle any other exception that might occur
    print("An error occurred")
else:
    # code that should run if no exception is raised
    print("Division operation completed successfully")
finally:
    # code that should always run, regardless of whether an exception is raised or not
    print("Program execution complete")
```

#OUTPUT

```
Enter a number: 6
Enter another number: 8
0.75
Division operation completed successfully
Program execution complete
```

WRITE A PROGRAM TO IMPLEMENT MULTILEVEL, MULTIPLE INHERITANCE IN PYTHON.

(Note:in multilevel inheritance also implement method overriding)

```
# single level
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def print_info(self):
        print(f"{self.year} {self.make} {self.model}")

class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
        super().__init__(make, model, year)
        self.num_doors = num_doors

    def print_info(self):
        super().print_info()
        print(f"{self.num_doors} doors")

car1 = Car("Toyota", "Camry", 2021, 4)
car1.print_info()
```

#OUTPUT

```
2021 Toyota Camry
4 doors
```

 # multiple inheritance

```
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def print_info(self):
        print(f"{self.year} {self.make} {self.model}")

class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
        super().__init__(make, model, year)
        self.num_doors = num_doors

    def print_info(self):
        super().print_info()
        print(f"{self.num_doors} doors")

class ElectricCar(Car):
    def __init__(self, make, model, year, num_doors, battery_size):
        super().__init__(make, model, year, num_doors)
        self.battery_size = battery_size

    def describe_battery(self):
        print(f"Battery size: {self.battery_size} kWh")

electric_car1 = ElectricCar("Tesla", "Model S", 2022, 4, 100)
electric_car1.print_info()
electric_car1.describe_battery()
```

#OUTPUT

```
2022 Tesla Model S
4 doors
Battery size: 100 kWh
```

#method overriding using multi level inheritance

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

class Labrador(Dog):
    def speak(self):
        print("Labrador woofs")

animal1 = Animal()
animal1.speak()

dog1 = Dog()
dog1.speak()

labrador1 = Labrador()
labrador1.speak()
```

#OUTPUT

```
Animal speaks
Dog barks
Labrador woofs
```

WRITE A MENU DRIVEN PROGRAM FOR DATA STRUCTURE USING BUILT IN FUNCTION FOR LINK LIST, STACK AND QUEUE.

#link list

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
  def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
   def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node


    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node


    def delete_node(self, key):
        current = self.head
        if current and current.data == key:
            self.head = current.next
```

```
            current = None
            return
        prev = None
        while current and current.data != key:
            prev = current
            current = current.next
        if current is None:
            return
        prev.next = current.next
        current = None
if __name__ == '__main__':
 linked_list = LinkedList()

 while True:
        print("\nLinked List Operations:")
        print("1. Insert at Beginning")
        print("2. Insert at End")
        print("3. Delete Node")
        print("4. Print List")
        print("5. Exit")
  choice = int(input("Enter your choice: "))
    if choice == 1:
            data = input("Enter data: ")
            linked_list.insert_at_beginning(data)
            print("Node inserted at beginning.")
        elif choice == 2:
            data = input("Enter data: ")
            linked_list.insert_at_end(data)
            print("Node inserted at end.")
        elif choice == 3:
            key = input("Enter node data to delete: ")
            linked_list.delete_node(key)
            print("Node deleted.")
        elif choice == 4:
            print("Linked List: ", end="")
```

```
            linked_list.print_list()
        elif choice == 5:
            break
        else:
            print("Invalid choice.")
```

#OUTPUT

Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Delete Node

4. Print List

5. Exit

Enter your choice: 4

Linked List: None

Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Delete Node

4. Print List

5. Exit

Enter your choice: 1

Enter data: 11

Node inserted at beginning.

Linked List Operations:

1. Insert at Beginning

2. Insert at End

3. Delete Node

4. Print List

5. Exit

Enter your choice: 4

Linked List: 11 -> None

Linked List Operations:

1. Insert at Beginning
2. Insert at End
3. Delete Node
4. Print List
5. Exit

Enter your choice: 5**#Stack**

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()

    def is_empty(self):
        return len(self.items) == 0

    def peek(self):
        if not self.is_empty():
            return self.items[-1]

    def size(self):
        return len(self.items)

if __name__ == '__main__':
    stack = Stack()
    while True:
        print("\nStack Operations:")
        print("1. Push")
        print("2. Pop")
        print("3. Peek")
        print("4. Size")
        print("5. Exit")\
        choice = int(input("Enter your choice: "))
        if choice == 1:
            item = input("Enter item to push: ")
            stack.push(item)
            print("Item pushed to stack.")
        elif choice == 2:
            item = stack.pop()
            if item:
                print("Popped item: ", item)
            else:
                print("Stack is empty.")
        elif choice == 3:
            item = stack.peek()
            if item:
                print("Top item: ", item)
            else:
                print("Stack is empty.")
        elif choice == 4:
            print("Size of stack: ", stack.size())
        elif choice == 5:
            break
        else:
            print("Invalid choice.")
```

#OUTPUT

```
Stack Operations:
1. Push
2. Pop
3. Peek
4. Size
5. Exit
Enter your choice: 3
Stack is empty.

Stack Operations:
1. Push
2. Pop
3. Peek
4. Size
5. Exit
Enter your choice: 1
Enter item to push: 10
Item pushed to stack.

Stack Operations:
1. Push
2. Pop
3. Peek
4. Size
5. Exit
Enter your choice: 4
Size of stack:  1

Stack Operations:
1. Push
2. Pop
3. Peek
4. Size
5. Exit
Enter your choice: 5
```

#QUEUE

```
from queue import Queue

queue = Queue()

while True:
    print("\nQueue Operations:")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Size of Queue")
    print("4. Is Queue Empty?")
    print("5. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:
        data = input("Enter data: ")
        queue.put(data)
        print("Data enqueued.")
    elif choice == 2:
        if not queue.empty():
            data = queue.get()
            print("Data dequeued:", data)
        else:
            print("Queue is empty.")
    elif choice == 3:
        print("Size of queue:", queue.qsize())
    elif choice == 4:
        if queue.empty():
            print("Queue is empty.")
        else:
            print("Queue is not empty.")
    elif choice == 5:
        break
    else:
        print("Invalid choice.")
```

#OUTPUT

Queue Operations:

1. Enqueue

2. Dequeue

3. Size of Queue

4. Is Queue Empty?

5. Exit

Enter your choice: 3

Size of queue: 0

Queue Operations:

1. Enqueue

2. Dequeue

3. Size of Queue

4. Is Queue Empty?

5. Exit

Enter your choice: 1

Enter data: 33

Data enqueued.

Queue Operations:

1. Enqueue

2. Dequeue

3. Size of Queue

4. Is Queue Empty?

5. Exit

Enter your choice: 4

Queue is not empty.

Queue Operations:

1. Enqueue

2. Dequeue

3. Size of Queue

4. Is Queue Empty?

5. Exit

Enter your choice: 5