**SYEDA TEHREEM**

**FA20-BCS-017**

**SECTION B**

**AI LAB 1 and 2 TASKS:**

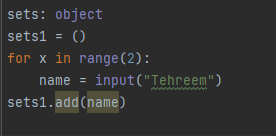
**TASKS 1:**

Dictionary1 = {  
 "name": "Tehreem",  
 "reg\_no": 17,  
 "section": "b"  
}  
print(Dictionary1)

**OUTPUT:**

****

**TASKS 2:**

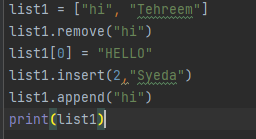
****

**OUTPUT:**

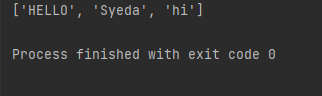
****

**TASK 3:**

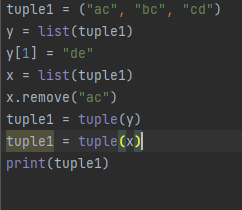
**List**

****

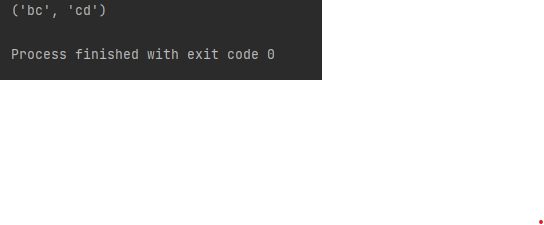
**Output**

****

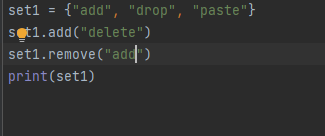
**Tuple**

****

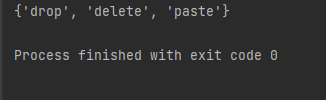
**Output**

****

**Sets**

****

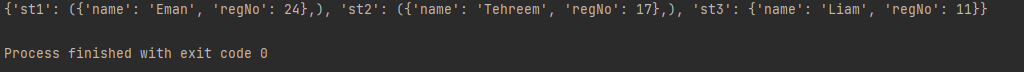
**Output**

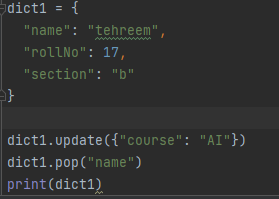
****

**Dictionary**

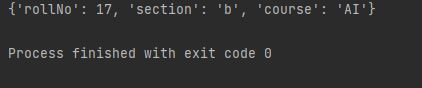
****

**Output**

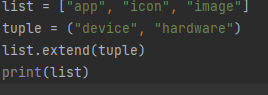
****

****

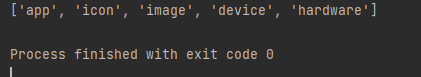
**Output**

****

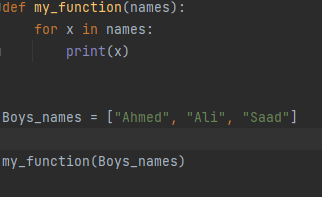
**Merging list and tuple:**

****

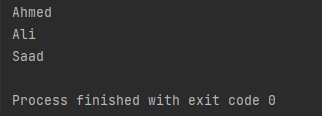
**Output**

****

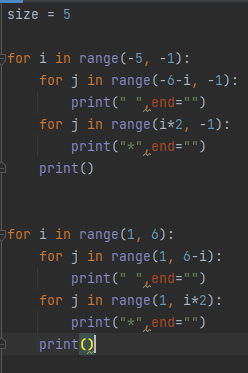
**Functions:**

****

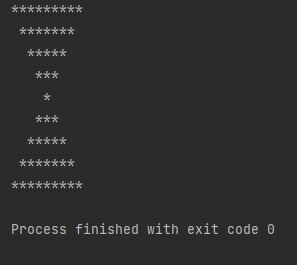
**Output**

****

**Pyramids:**

****

**Output**

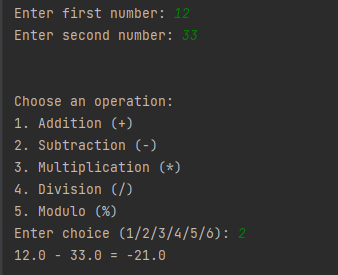
****

**HOME TASK OF LAB 1:**

**CALCULATOR:**

def add(x, y):  
 return x + y  
  
  
def subtract(x, y):  
 return x - y  
  
  
def multiply(x, y):  
 return x \* y  
  
  
def divide(x, y):  
 return x / y  
  
  
def modulo(x, y):  
 return x % y  
  
num1 = float(input("Enter first number: "))  
num2 = float(input("Enter second number: "))  
print("\n")  
print("Choose an operation:")  
print("1. Addition (+)")  
print("2. Subtraction (-)")  
print("3. Multiplication (\*)")  
print("4. Division (/)")  
print("5. Modulo (%)")  
choice = input("Enter choice (1/2/3/4/5/6): ")  
  
if choice == '1':  
 print(num1, "+", num2, "=", add(num1, num2))  
  
elif choice == '2':  
 print(num1, "-", num2, "=", subtract(num1, num2))  
  
elif choice == '3':  
 print(num1, "\*", num2, "=", multiply(num1, num2))  
elif choice == '4':  
 print(num1, "/", num2, "=", divide(num1, num2))  
  
elif choice == '5':  
 print(num1, "%", num2, "=", modulo(num1, num2))  
  
else:  
 print("Invalid input")

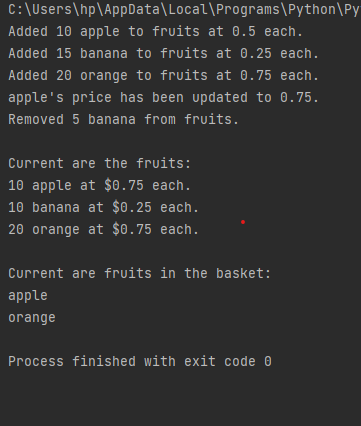
**OUTPUT:**

****

**HOME TASK of lab 2:**

fruits = []  
prices = {}  
basket = set()  
  
# Add function  
def add\_item(name, quantity, price):  
 fruits.append((name, quantity))  
 prices[name] = price  
 print(f"Added {quantity} {name}(s) to fruits at {price} each.")  
  
# Remove function  
def remove\_item(name, quantity):  
 for i, item in enumerate(fruits):  
 if item[0] == name and item[1] >= quantity:  
 fruits[i] = (name, item[1] - quantity)  
 print(f"Removed {quantity} {name}(s) from fruits.")  
 if fruits[i][1] == 0:  
 del fruits[i]  
 del prices[name]  
 print(f"{name} is out of stock and has been removed from fruits.")  
 break  
 else:  
 print(f"Error: {quantity} {name}(s) not found in fruits.")  
  
# Update function  
def update\_price(name, price):  
 if name in prices:  
 prices[name] = price  
 print(f"{name}'s price has been updated to {price}.")  
 else:  
 print(f"Error: {name} not found in fruits.")  
  
# Testing  
add\_item("apple", 10, 0.50)  
add\_item("banana", 15, 0.25)  
add\_item("orange", 20, 0.75)  
  
update\_price("apple", 0.75)  
remove\_item("banana", 5)  
  
# Adding items to cart  
basket.add("apple")  
basket.add("orange")  
  
# Displaying inventory and cart  
print("\nCurrent fruits are:")  
for name, quantity in fruits:  
 print(f"{quantity} {name} at ${prices[name]} each.")  
  
print("\nCurrent basket are:")  
for item in basket:  
 print(item)

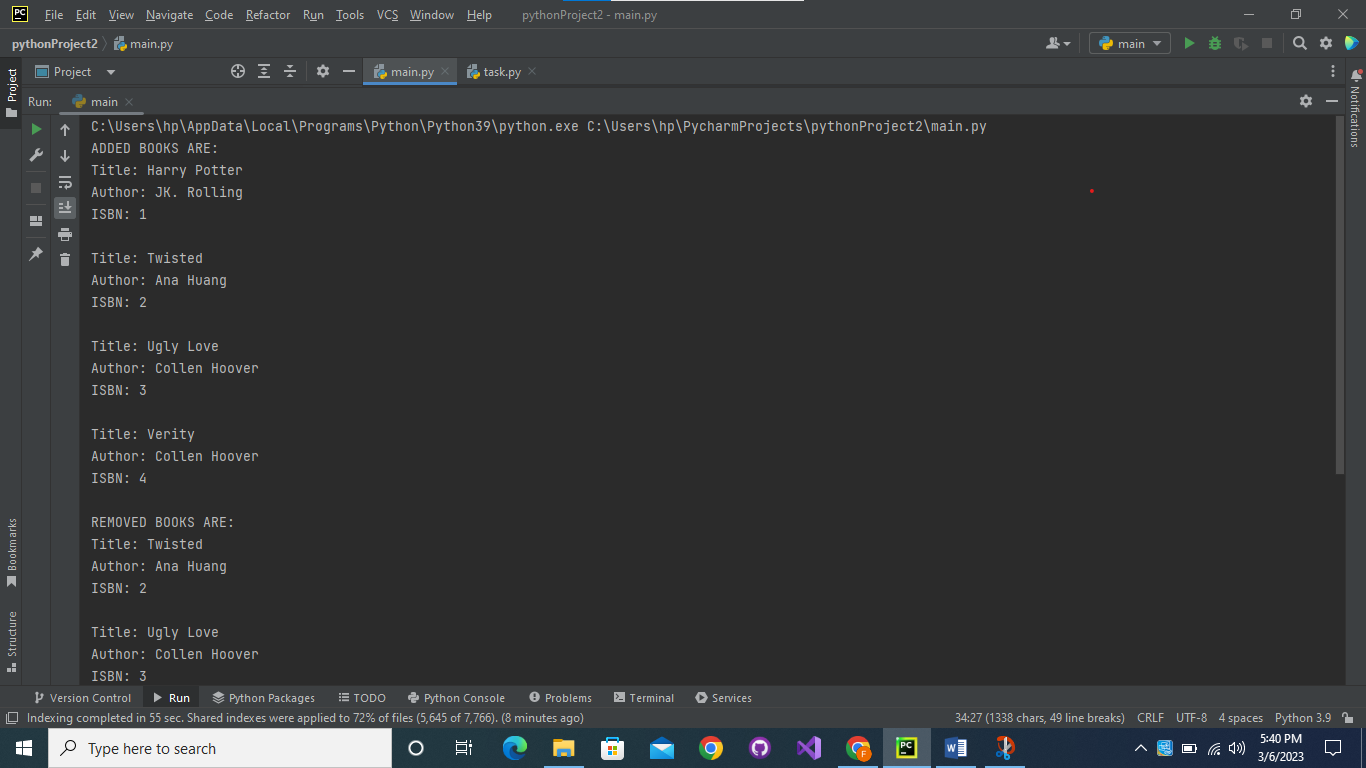
**OUTPUT:**

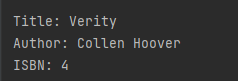
****

**AI LAB 3 TASKS**

class Library\_Books:  
 def \_\_init\_\_(self, title, author, Id):  
 self.title = title  
 self.author = author  
 self.Id = Id  
  
  
class Library:  
 def \_\_init\_\_(self):  
 self.Lib\_books = []  
  
 def add\_book(self, book):  
 self.Lib\_books.append(book)  
  
 def display\_books(self):  
 for book in self.Lib\_books:  
 print(f"Title: {book.title}\nAuthor: {book.author}\nISBN: {book.Id}\n")  
  
 def remove\_books(self, book):  
 self.Lib\_books.remove(book)  
  
 def update\_books(self, previous\_book, new\_book):  
 index = self.Lib\_books.index(previous\_book)  
 self.book[index] = new\_book  
  
  
newLibrary = Library()  
book1 = Library\_Books("Harry Potter", "JK. Rolling", "1")  
book2 = Library\_Books("Twisted", "Ana Huang", "2")  
book3 = Library\_Books("Ugly Love", "Collen Hoover", "3")  
book4 = Library\_Books("Verity", "Collen Hoover", "4")  
# add book display  
print("ADDED BOOKS ARE:")  
newLibrary.add\_book(book1)  
newLibrary.add\_book(book2)  
newLibrary.add\_book(book3)  
newLibrary.add\_book(book4)  
newLibrary.display\_books()  
# removed books  
print("REMOVED BOOKS ARE: ")  
newLibrary.remove\_books(book1)  
newLibrary.display\_books()  
  
# updated books  
print("UPDATED BOOKS ARE: ")  
new\_book = Library\_Books("LILY", "Collen Hoover", "5")  
  
newLibrary.update\_books(book3, new\_book)  
newLibrary.display\_books()

**OUTPUT:**

****

****

**AI LAB 4:**

**HOMETASK:**

**PACKAGE1---MODULE1**

def functionAdd(x, y):  
 return x + y  
def subtract(x, y):  
 return x - y

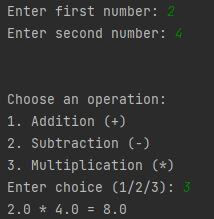
**PACKAGE2---MODULE2**

def multiply(x, y):  
 return x \* y

**MAIN.PY:**

from Package1 import Module1;  
from Package1.Module1 import functionAdd, subtract  
from Package2 import Module2;  
from Package2.Module2 import multiply  
  
num1 = float(input("Enter first number: "))  
num2 = float(input("Enter second number: "))  
print("\n")  
print("Choose an operation:")  
print("1. Addition (+)")  
print("2. Subtraction (-)")  
print("3. Multiplication (\*)")  
choice = input("Enter choice (1/2/3): ")  
  
if choice == '1':  
 print(num1, "+", num2, "=", functionAdd(num1, num2))  
elif choice == '2':  
 print(num1, "-", num2, "=", subtract(num1, num2))  
elif choice == '3':  
 print(num1, "\*", num2, "=", multiply(num1, num2))  
else:  
 print("Invalid input")  
  
Module1.functionAdd()  
Module1.subtract()  
Module2.multiply()

**OUTPUT:**

****

**LAB TASK:**

class Person:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 def display(self):  
 print(f"Name: {self.name}, Age: {self.age}")  
  
  
class Student(Person):  
 def \_\_init\_\_(self, name, age, student\_id):  
 super().\_\_init\_\_(name, age)  
 self.student\_id = student\_id  
  
 def display(self):  
 super().display()  
 print(f"Student ID: {self.student\_id}")  
  
person = Person("Alice", 25)  
student = Student("Bob", 20, 12345)  
person.display()  
student.display()

**OUTPUT:**

**C:\Users\hp\AppData\Local\Programs\Python\Python39\python.exe C:\Users\hp\PycharmProjects\pythonProject1\inheritance\file2.py**

**Name: Alice, Age: 25**

**Name: Bob, Age: 20**

**Student ID: 12345**

**LAB TASK 2:**

**SIMPLE GRAPHS:**

class graph:  
 def \_\_init\_\_(self,gdict=None):  
 if gdict is None:  
 gdict = []  
 self.gdict = gdict  
  
 def getVertices(self):  
 return list(self.gdict.keys())  
  
 def edges(self):  
 return self.findedges()  
  
 def AddEdge(self, edge):  
 edge = set(edge)  
 (vrtx1, vrtx2) = tuple(edge)  
 if vrtx1 in self.gdict:  
 self.gdict[vrtx1].append(vrtx2)  
 else:  
 self.gdict[vrtx1] = [vrtx2]  
 def findedges(self):  
 edgename = []  
 for vrtx in self.gdict:  
 for nxtvrtx in self.gdict[vrtx]:  
 if {nxtvrtx, vrtx} not in edgename:  
 edgename.append({vrtx, nxtvrtx})  
 return edgename  
  
graph\_elements = {  
 "a" : ["b","c"],  
 "b" : ["a", "d"],  
 "c" : ["a", "d"],  
 "d" : ["e"],  
 "e" : ["d"]  
}  
g = graph(graph\_elements)  
print(g.getVertices())  
g.AddEdge({'a','e'})  
g.AddEdge({'a','c'})  
print(g.edges())

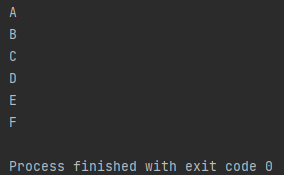
**OUTPUT:**

****

**SIMPLE BFS:**

from collections import deque  
  
  
def bfs(graph, start):  
 visited = set() # keep track of visited nodes  
 queue = deque([start]) # initialize the queue with the start node  
 visited.add(start) # mark start node as visited  
  
 while queue: # while there are nodes in the queue  
 node = queue.popleft() # remove the first node from the queue  
 print(node) # print the node (or process it in some other way)  
 for neighbor in graph[node]: # iterate over the neighbors of the node  
 if neighbor not in visited: # if neighbor has not been visited  
 visited.add(neighbor) # mark it as visited  
 queue.append(neighbor) # add it to the queue  
  
  
# Example usage  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
bfs(graph, 'A')

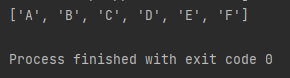
**OUTPUT:**

****

**TRAVERSAL BFS:**

from collections import deque  
  
# define graph as a dictionary  
graph = {'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []}  
  
def bfs(graph, start):  
 visited = [] # list to keep track of visited nodes  
 queue = deque([start]) # queue for BFS  
  
 while queue:  
 node = queue.popleft()  
 if node not in visited:  
 visited.append(node)  
 neighbors = graph[node]  
 for neighbor in neighbors:  
 queue.append(neighbor)  
 return visited  
  
# example usage  
print(bfs(graph, 'A'))

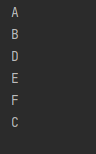
**OUTPUT:**

****

**DFS:**

def dfs(graph, start, visited=None):  
 if visited is None:  
 visited = set() # keep track of visited nodes  
 visited.add(start) # mark the start node as visited  
 print(start) # print the node (or process it in some other way)  
 for neighbor in graph[start]: # iterate over the neighbors of the node  
 if neighbor not in visited: # if neighbor has not been visited  
 dfs(graph, neighbor, visited) # recursively call dfs on the neighbor  
  
  
# Example usage  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
dfs(graph, 'A')

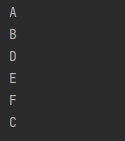
**OUTPUT:**

****

**DFS TRAVERSAL**

def dfs\_traversal(graph, start):  
 visited = set() # keep track of visited nodes  
 stack = [start] # initialize the stack with the start node  
  
 while stack: # while there are nodes in the stack  
 node = stack.pop() # remove the last node from the stack  
 if node not in visited:  
 visited.add(node) # mark the node as visited  
 print(node) # print the node (or process it in some other way)  
 neighbors = graph[node] # get the neighbors of the node  
 stack.extend(reversed(neighbors)) # add the neighbors to the stack in reverse order  
  
  
# Example usage  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
dfs\_traversal(graph, 'A')

**OUTPUT:**

****