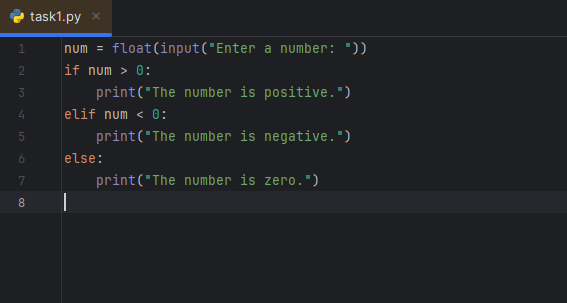
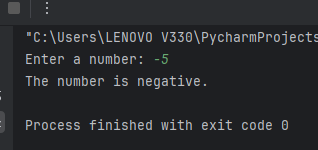
### **Python Control Structures**

1. **If-Else Statement**: Write a program to check if a number is positive, negative, or zero.

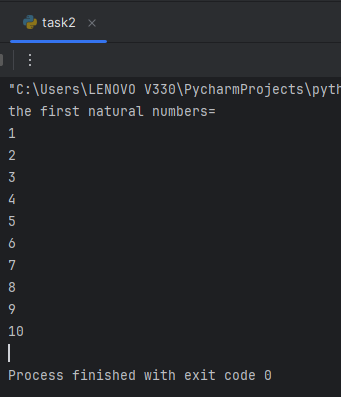


Output:



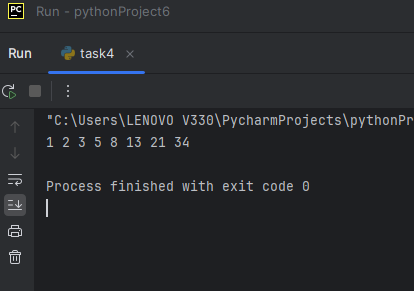
1. **For Loop**: Write a program to print the first 10 natural numbers using a for loop.

print("the first natural numbers=")  
for i in range(1,11):  
 print(i)



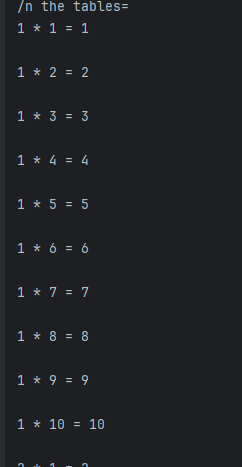
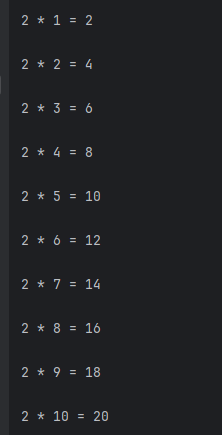
1. **While Loop**: Write a program to print the Fibonacci sequence up to the nth term using a while loop.

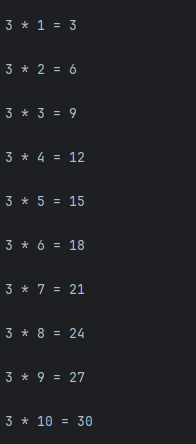
n = 8  
n1 = 0  
n2 = 1  
next\_number = n2  
count = 1  
  
while count <= n:  
 print(next\_number, end=" ")  
 count += 1  
 n1, n2 = n2, next\_number  
 next\_number = n1 + n2  
print()



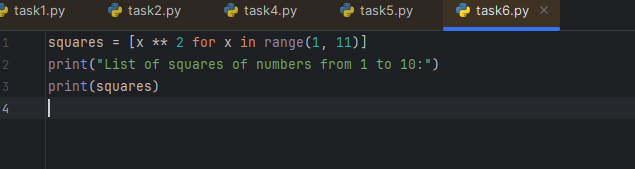
1. **Nested Loop**: Write a program to print a multiplication table from 1 to 10.

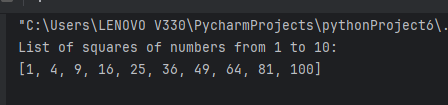
for i in range(1, 11):  
 for j in range(1, 11):  
 print(i, '\*', j, '=', i \* j)  
 print()

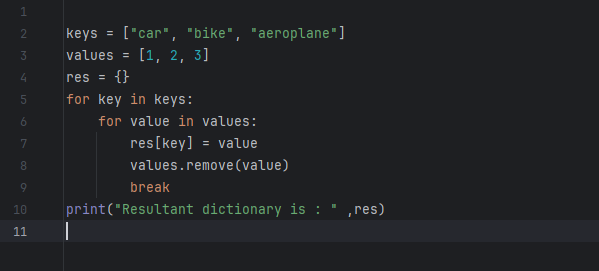


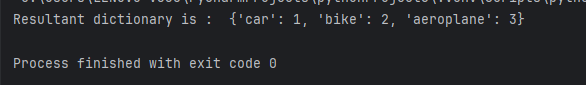
1. **List Comprehension**: Write a program to generate a list of squares of numbers from 1 to 10 using list comprehension.



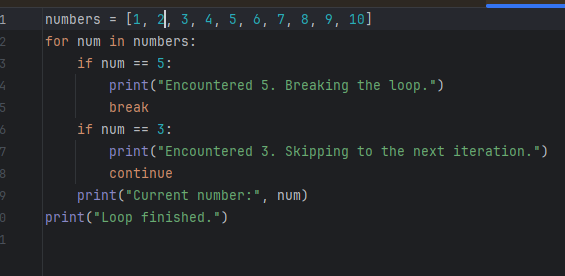


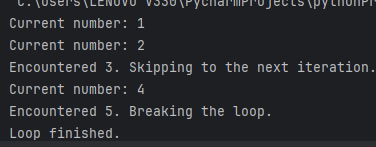
1. **Dictionary Comprehension**: Write a program to create a dictionary from two lists: one of keys and one of values.



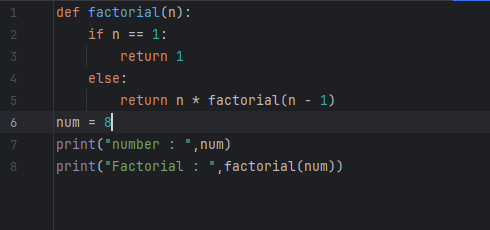


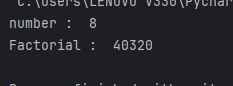
1. **Break and Continue**: Write a program to iterate over a list of numbers and print each number. If you encounter the number 5, break the loop. If you encounter the number 3, skip to the next iteration.



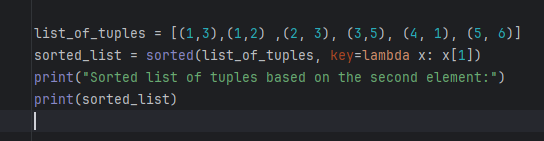


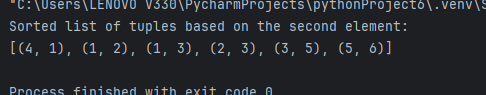
1. **Functions**: Write a function to calculate the factorial of a number using recursion.



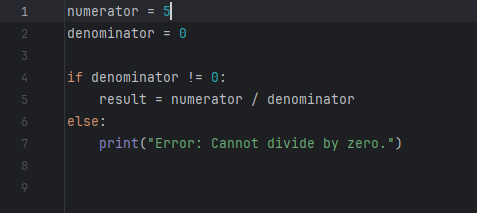


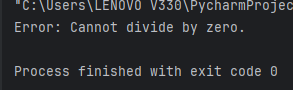
1. **Lambda Function**: Write a program to sort a list of tuples based on the second element using a lambda function.





1. **Exception Handling**: Write a program to handle a division by zero exception.





### **Python Data Structures**

1. **Lists**: Create a list of the first 10 prime numbers. Write functions to add an element, remove an element, and find an element in the list.

def is\_prime(n):  
  
 if n <= 1:  
 return False  
 elif n == 2:  
 return True  
 elif n % 2 == 0:  
 return False  
 else:  
  
 for i in range(3, int(n \*\* 0.5) + 1, 2):  
 if n % i == 0:  
 return False  
 return True  
  
def first\_n\_primes(n):  
   
 primes = []  
 num = 2 # Start checking from 2  
 while len(primes) < n:  
 if is\_prime(num):  
 primes.append(num)  
 num += 1  
 return primes  
  
  
prime\_numbers = first\_n\_primes(10)  
print("List of the first 10 prime numbers:", prime\_numbers)  
  
def add\_element(lst, element):  
   
 if is\_prime(element):  
 lst.append(element)  
 print("Element", element, "added successfully.")  
 else:  
 print("Error: Element is not a prime number.")  
  
def remove\_element(lst, element):  
   
 if element in lst:  
 lst.remove(element)  
 print("Element", element, "removed successfully.")  
 else:  
 print("Error: Element not found in the list.")  
  
def find\_element(lst, element):  
   
 if element in lst:  
 print("Element", element, "found in the list at index", lst.index(element))  
 else:  
 print("Element", element, "not found in the list.")  
add\_element(prime\_numbers, 31)   
add\_element(prime\_numbers, 33)   
remove\_element(prime\_numbers, 13)   
remove\_element(prime\_numbers, 15)   
find\_element(prime\_numbers, 7)   
find\_element(prime\_numbers, 15)

output:

List of the first 10 prime numbers: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

Element 31 added successfully.

Error: Element is not a prime number.

Element 13 removed successfully.

Error: Element not found in the list.

Element 7 found in the list at index 3

Element 15 not found in the list.

1. **Tuples**: Create a tuple with 5 different elements. Write a program to access and print each element in the tuple.

my\_tuple= (12, 'kumar', 28, 'sasi', 'world')  
for i in my\_tuple:  
 print (i)

output:

12

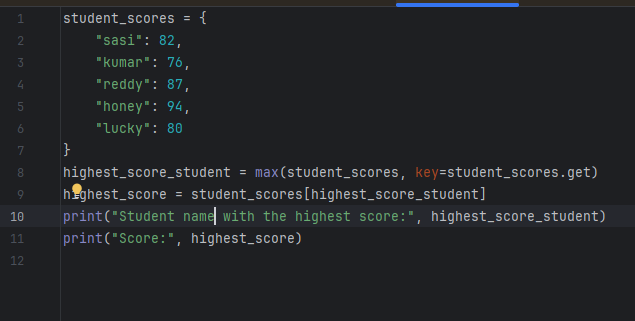
kumar

28

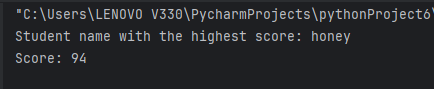
sasi

world

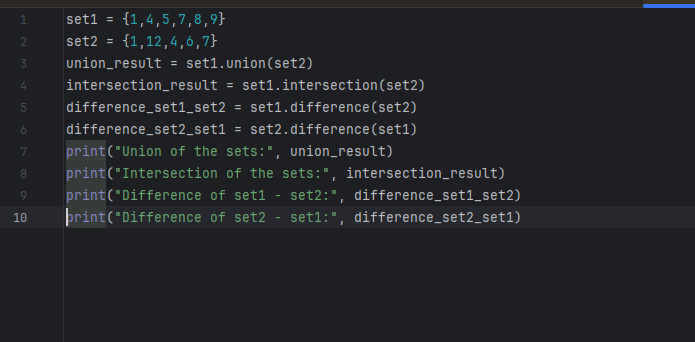
1. **Dictionaries**: Create a dictionary with keys as student names and values as their scores. Write a program to find the student with the highest score.



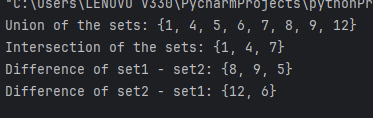
Output:



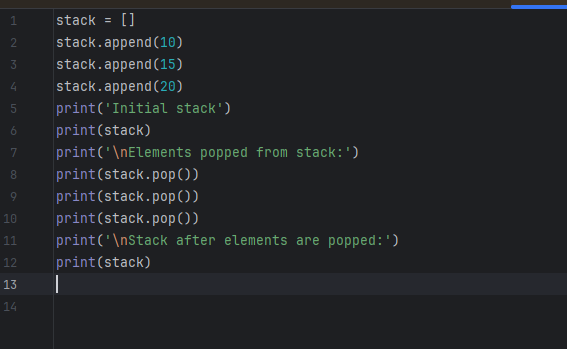
1. **Sets**: Create two sets of integers. Write a program to find their union, intersection, and difference.



Output:



1. **Stacks**: Implement a stack using a list. Write functions for push, pop, and peek operations.



Output:

Initial stack

[10, 15, 20]

Elements popped from stack:

20

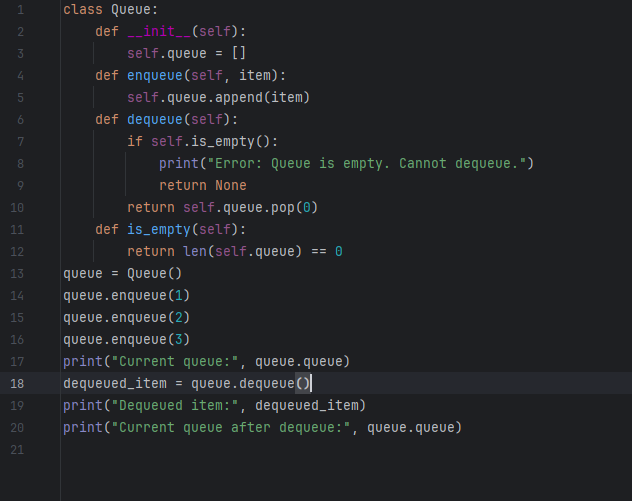
15

10

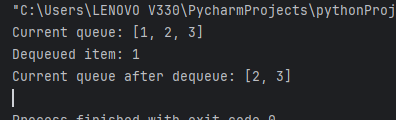
Stack after elements are popped:

[]

1. **Queues**: Implement a queue using a list. Write functions for enqueue and dequeue operations.



Output:



1. **Linked List**: Implement a singly linked list with methods to insert, delete, and display elements.

class Node:  
 def \_\_init\_\_(self, data):  
 self.data = data  
 self.next = None  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
 def insert(self, data):  
 new\_node = Node(data)  
 if self.head is None:  
 self.head = new\_node  
 return  
 last\_node = self.head  
 while last\_node.next:  
 last\_node = last\_node.next  
 last\_node.next = new\_node  
 def delete(self, data):  
 if self.head is None:  
 print("Error: Linked list is empty. Cannot delete.")  
 return  
 if self.head.data == data:  
 self.head = self.head.next  
 return  
 prev\_node = self.head  
 current\_node = self.head.next  
 while current\_node:  
 if current\_node.data == data:  
 prev\_node.next = current\_node.next  
 return  
 prev\_node = current\_node  
 current\_node = current\_node.next  
 print("Error: Node with data", data, "not found.")  
 def display(self):  
 *"""Display the elements of the linked list."""* if self.head is None:  
 print("Linked list is empty.")  
 return  
 current\_node = self.head  
 while current\_node:  
 print(current\_node.data, end=" ")  
 current\_node = current\_node.next  
 print()  
linked\_list = LinkedList()  
linked\_list.insert(10)  
linked\_list.insert(20)  
linked\_list.insert(30)  
linked\_list.insert(40)  
print("Linked list after insertions:")  
linked\_list.display()  
linked\_list.delete(30)  
print("Linked list after deletion of 3:")  
linked\_list.display()  
linked\_list.delete(2)  
linked\_list.delete(10)  
print("Linked list after deletion of 1:")  
linked\_list.display()

output:

Linked list after insertions:

10 20 30 40

Linked list after deletion of 3:

10 20 40

Error: Node with data 2 not found.

Linked list after deletion of 1:

20 40

1. **Binary Tree**: Implement a binary tree with methods for in-order, pre-order, and post-order traversal.

class Node:  
 def \_\_init\_\_(self, key):  
 self.left = None  
 self.right = None  
 self.val = key  
class BinaryTree:  
 def \_\_init\_\_(self):  
 self.root = None  
 def in\_order\_traversal(self, node):  
 result = []  
 if node:  
 result = self.in\_order\_traversal(node.left)  
 result.append(node.val)  
 result = result + self.in\_order\_traversal(node.right)  
 return result  
 def pre\_order\_traversal(self, node):  
 result = []  
 if node:  
 result.append(node.val)  
 result = result + self.pre\_order\_traversal(node.left)  
 result = result + self.pre\_order\_traversal(node.right)  
 return result  
 def post\_order\_traversal(self, node):  
 result = []  
 if node:  
 result = self.post\_order\_traversal(node.left)  
 result = result + self.post\_order\_traversal(node.right)  
 result.append(node.val)  
 return result  
if \_\_name\_\_ == "\_\_main\_\_":  
 tree = BinaryTree()  
 tree.root = Node(1)  
 tree.root.left = Node(10)  
 tree.root.right = Node(30)  
 tree.root.left.left = Node(40)  
 tree.root.left.right = Node(50)  
 tree.root.right.left = Node(60)  
 tree.root.right.right = Node(70)  
 print("In-order traversal:", tree.in\_order\_traversal(tree.root))  
 print("Pre-order traversal:", tree.pre\_order\_traversal(tree.root))  
 print("Post-order traversal:", tree.post\_order\_traversal(tree.root))

output:

In-order traversal: [40, 10, 50, 1, 60, 30, 70]

Pre-order traversal: [1, 10, 40, 50, 30, 60, 70]

Post-order traversal: [40, 50, 10, 60, 70, 30, 1]

1. **Graphs**: Represent a graph using an adjacency list and write a program to perform a breadth-first search (BFS).

from collections import defaultdict, deque  
class Graph:  
 def \_\_init\_\_(self):  
 self.adjacency\_list = defaultdict(list)  
 def add\_edge(self, u, v):   
 self.adjacency\_list[u].append(v)  
 self.adjacency\_list[v].append(u)   
 def bfs(self, start\_node):  
 visited = set()   
 queue = deque([start\_node])   
 while queue:  
 node = queue.popleft()   
 if node not in visited:  
 print(node, end=" ")  
 visited.add(node)  
 for neighbor in self.adjacency\_list[node]:  
 if neighbor not in visited:  
 queue.append(neighbor)  
graph = Graph()  
graph.add\_edge(0, 1)  
graph.add\_edge(0, 2)  
graph.add\_edge(1, 2)  
graph.add\_edge(2, 3)  
graph.add\_edge(3, 3)  
print("BFS starting from node 2:")  
graph.bfs(2)

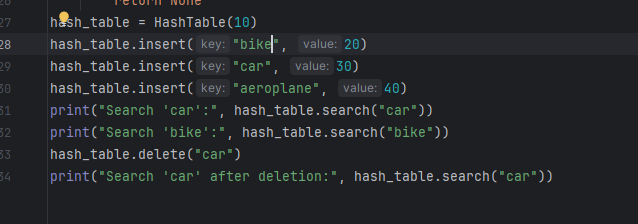
output:

BFS starting from node 2:

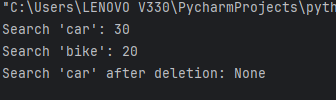
2 0 1 3

1. **Hash Table**: Implement a basic hash table with functions for inserting, deleting, and searching elements.



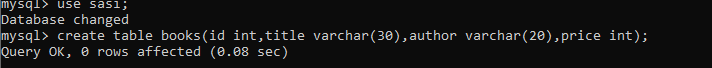


Output:



### **SQL DDL (Data Definition Language)**

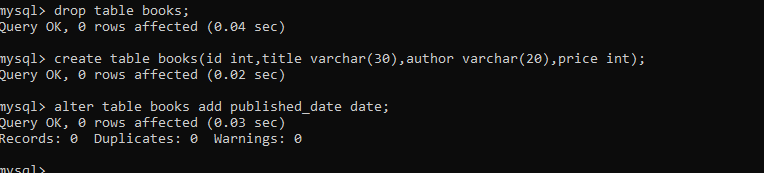
1. **Create Table**: Write a SQL statement to create a table named Books with columns id, title, author, and price.



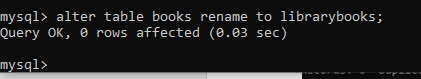
1. **Drop Table**: Write a SQL statement to drop the Books table.



1. **Alter Table**: Write a SQL statement to add a new column published\_date to the Books table.



1. **Rename Table**: Write a SQL statement to rename the Books table to LibraryBooks.

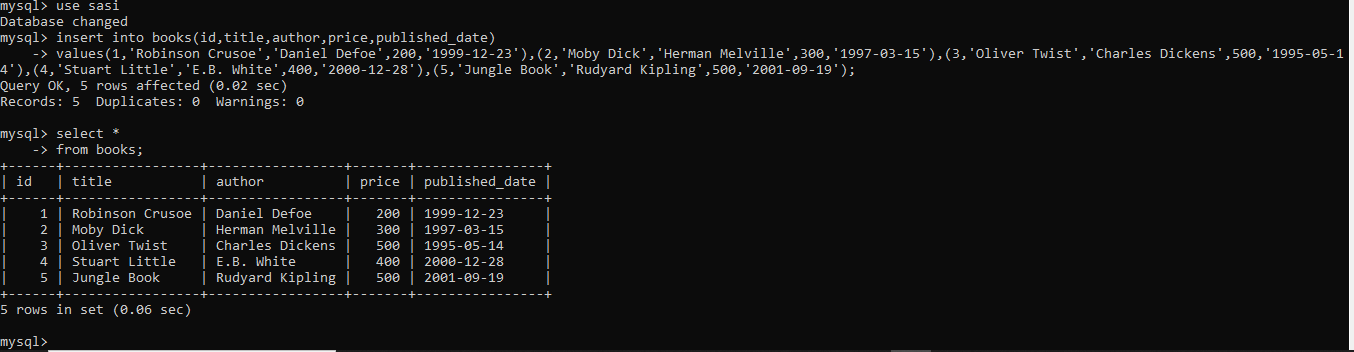


1. **Create Index**: Write a SQL statement to create an index on the author column of the Books table.

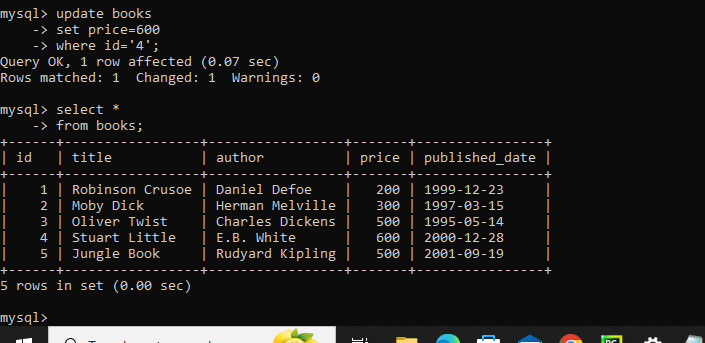


### **SQL DML (Data Manipulation Language)**

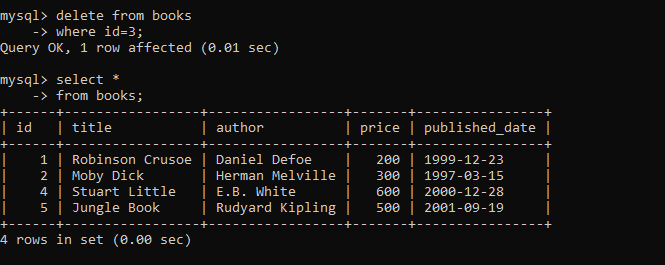
1. **Insert Data**: Write a SQL statement to insert a new record into the Books table.



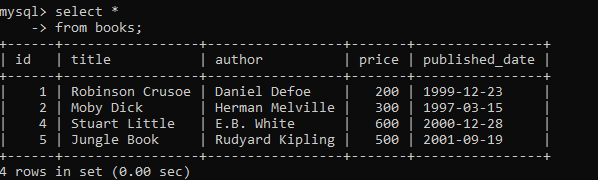
1. **Update Data**: Write a SQL statement to update the price of a book in the Books table based on its id.



1. **Delete Data**: Write a SQL statement to delete a book from the Books table based on its id.



1. **Select Data**: Write a SQL statement to select all records from the Books table.



1. **Join Tables**: Write a SQL statement to join two tables, Books and Authors, on a common column and select relevant data.

