**Code: -**

class Node:

    def \_\_init\_\_(self, data):

        self.data = data      # Value of the node

        self.left = None      # Left child

        self.right = None     # Right child

class BinarySearchTree:

    def \_\_init\_\_(self):

        self.root = None      # Initially, the tree is empty

    def insert(self, data):

        if not self.root:

            self.root = Node(data)  # If the tree is empty, make new node the root

        else:

            self.\_insert(self.root, data)

    def \_insert(self, node, data):

        if data < node.data:

            if node.left is None:

                node.left = Node(data)

            else:

                self.\_insert(node.left, data)

        elif data > node.data:

            if node.right is None:

                node.right = Node(data)

            else:

                self.\_insert(node.right, data)

        # Duplicate values are ignored

    def search(self, data):

        return self.\_search(self.root, data)

    def \_search(self, node, data):

        if node is None:

            return False

        if node.data == data:

            return True

        elif data < node.data:

            return self.\_search(node.left, data)

        else:

            return self.\_search(node.right, data)

    def inorder(self):

        return self.\_inorder(self.root)

    def \_inorder(self, node):

        result = []

        if node:

            result.extend(self.\_inorder(node.left))

            result.append(node.data)

            result.extend(self.\_inorder(node.right))

        return result

    def preorder(self):

        return self.\_preorder(self.root)

    def \_preorder(self, node):

        result = []

        if node:

            result.append(node.data)

            result.extend(self.\_preorder(node.left))

            result.extend(self.\_preorder(node.right))

        return result

    def postorder(self):

        return self.\_postorder(self.root)

    def \_postorder(self, node):

        result = []

        if node:

            result.extend(self.\_postorder(node.left))

            result.extend(self.\_postorder(node.right))

            result.append(node.data)

        return result

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    bst = BinarySearchTree()

    for value in [50, 30, 20, 40, 70, 60, 80]:

        bst.insert(value)

    print("Search 40:", bst.search(40))  # Output: True

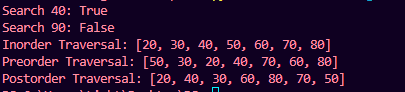
    print("Search 90:", bst.search(90))  # Output: False

    print("Inorder Traversal:", bst.inorder())     # [20, 30, 40, 50, 60, 70, 80]

    print("Preorder Traversal:", bst.preorder())   # [50, 30, 20, 40, 70, 60, 80]

    print("Postorder Traversal:", bst.postorder()) # [20, 40, 30, 60, 80, 70, 50]

**OUTPUT:**

****