PRACTICAL PROGRAMS

1. Write a program to swap two numbers without taking a temporary variable.

Program:

```
# Swap two numbers using addition and subtraction
def swap_numbers(a, b):
    a = a + b  # Step 1: Add both numbers and assign to a
    b = a - b  # Step 2: Subtract b from a (which is now a + b) to get original value of a
    a = a - b  # Step 3: Subtract b (which is original a) from a to get original value of b
    return a, b

# Example usage
a = 50
b = 100
a, b = swap_numbers(a, b)
print("Swapped numbers: a =", a, ", b =", b)
```

Output: Swapped numbers: a = 100 , b = 50

2. Write a Python program to generate prime numbers less than 50.

Program:

```
# Function to check if a number is prime
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
# Function to generate prime numbers less than 50
def generate_primes(limit):
    primes = []
    for num in range(2, limit):
        if is_prime(num):
            primes.append(num)
    return primes
```

```
# Generate and print prime numbers less than 50
primes = generate_primes(50)
print("Prime numbers less than 50:", primes)
```

Output: Prime numbers less than 50: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

3. Create a program to create, append and remove lists in python.

Program:

```
# Creating a list
my_list = [1, 2, 3, 4, 5]
print("Original list:", my_list)

# Appending elements to the list
my_list.append(6)  # Appends 6 to the end of the list
my_list.append(7)  # Appends 7 to the end of the list
print("List after appending 6 and 7:", my_list)

# Removing elements from the list
my_list.remove(3)  # Removes the first occurrence of 3 from the list
print("List after removing 3:", my_list)

# Removing the last element (using pop)
last_element = my_list.pop()  # Removes and returns the last element of the list
print("List after popping the last element:", my_list)
print("Popped element:", last_element)

# Removing an element by index (e.g., removing the element at index 1)
removed_element = my_list.pop(1)  # Removes and returns the element at index 1
print("List after removing element at index 1:", my_list)
print("Removed element:", removed elements)
```

Output :

Original list: [1, 2, 3, 4, 5]

List after appending 6 and 7: [1, 2, 3, 4, 5, 6, 7]

List after removing 3: [1, 2, 4, 5, 6, 7]

List after popping the last element: [1, 2, 4, 5, 6]

Popped element: 7

List after removing element at index 1: [1, 4, 5, 6]

Removed element: 2

4. Write a program to demonstrate working with tuples in python

Program:

```
# Creating a tuple
my_tuple = (1, 2, 3, 4, 5)
print("Original tuple:", my_tuple)
# Accessing elements of a tuple
print("First element:", my_tuple[0])
print("Last element:", my_tuple[-1])
# Slicing a tuple
print("Sliced tuple (from index 1 to 3):", my_tuple[1:4])
# Tuple concatenation
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = tuple1 + tuple2
print("Concatenated tuple:", concatenated_tuple)
# Tuple repetition
repeated_tuple = tuple1 * 3
print("Repeated tuple:", repeated_tuple)
# Checking if an element exists in a tuple
element_exists = 3 in my_tuple
print("Does 3 exist in the tuple?:", element_exists)
# Finding the length of a tuple
tuple_length = len(my_tuple)
print("Length of the tuple:", tuple_length)
# Iterating through a tuple
print("Iterating through the tuple:")
for item in my_tuple:
print(item)
```

```
# Unpacking a tuple
a, b, c, d, e = my_tuple
print("Unpacked values: a =", a, ", b =", b, ", c =", c, ", d =", d, ", e =", e)
# Tuple with mixed data types
mixed_tuple = (1, "Hello", 3.14, [1, 2, 3], (4, 5))
print("Tuple with mixed data types:", mixed_tuple)
# Nested tuples
nested_tuple = (1, (2, 3), (4, 5, 6))
print("Nested tuple:", nested_tuple)
print("Accessing element in nested tuple (2nd element of 2nd tuple):", nested_tuple[1][1])
# Counting occurrences of an element in a tuple
count_of_twos = my_tuple.count(2)
print("Number of times 2 appears in the tuple:", count_of_twos)
# Finding the index of an element in a tuple
index_of_four = my_tuple.index(4)
print("Index of element 4 in the tuple:", index_of_four)
# Converting a list to a tuple
my_list = [7, 8, 9]
converted_tuple = tuple(my_list)
print("Converted tuple from list:", converted_tuple)
```

Output:

```
Original tuple: (1, 2, 3, 4, 5)
First element: 1
Last element: 5
Sliced tuple (from index 1 to 3): (2, 3, 4)
Concatenated tuple: (1, 2, 3, 4, 5, 6)
Repeated tuple: (1, 2, 3, 1, 2, 3, 1, 2, 3)
Does 3 exist in the tuple?: True
Length of the tuple: 5
Iterating through the tuple:
1
2
3
4
5
Unpacked values: a = 1 , b = 2 , c = 3 , d = 4 , e = 5
Tuple with mixed data types: (1, 'Hello', 3.14, [1, 2, 3], (4, 5))
Nested tuple: (1, (2, 3), (4, 5, 6))
```

Accessing element in nested tuple (2nd element of 2nd tuple): 3
Number of times 2 appears in the tuple: 1
Index of element 4 in the tuple: 3
Converted tuple from list: (7, 8, 9)

5. Demonstrate working with dictionaries in python.

Program:

Creating a Dictionary:

```
# Using curly braces
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

# Using dict() function
person_alt = dict(name="Bob", age=25, city="Los Angeles")

Accessing Values:
name = person["name"]  # Accessing the value for the key 'name'
age = person.get("age")  # Another way to access the value

Adding or Updating Items:
person["email"] = "alice@example.com"  # Adding a new key-value pair
person["age"] = 31  # Updating the value of an existing key

Removing Items:
email = person.pop("email")  # Removes 'email' key and returns its value
del person["city"]  # Deletes the key 'city'
last_item = person.popitem()  # Removes and returns the last inserted item
```

Iterating Over a Dictionary:

```
# Looping through keys
for key in person.keys():
    print(key)

# Looping through values
for value in person.values():
    print(value)

# Looping through key-value pairs
for key, value in person.items():
    print(f"{key}: {value}")

Checking if a Key Exists:
if "name" in person:
    print("Name is present")
```

Output:

Keys in person dictionary:

name

age

Values in person dictionary:

Alice

31

Key-Value pairs in person dictionary:

name: Alice

age: 31

Name is present

6. Write a python program to find factorial of a number using function recursion.

Program:

```
def factorial(n):
    # Base case: if n is 0 or 1, the factorial is 1
    if n == 0 or n == 1:
        return 1
    else:
        # Recursive case: n * factorial of (n-1)
        return n * factorial(n - 1)

# Input from the user
num = int(input("Enter a number: "))

# Check if the input is a non-negative integer
if num < 0:
    print("Factorial is not defined for negative numbers.")
else:
    # Calculate and print the factorial
    result = factorial(num)
print(f"The factorial of {num} is {result}")
```

Output:

Enter a number: 5

The factorial of 5 is 120

Enter a number: -3

Factorial is not defined for negative numbers.

Enter a number: 0

The factorial of 0 is 1

7. Write a Python program to demonstrate class, object and accessing class member's example.

Program:

```
# Defining a class
class Dog:
    # Class attribute (shared by all instances)
    species = "Canis familiaris"

    # Instance attributes (unique to each instance)
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # Method to return dog's description
    def description(self):
        return f"{self.name} is {self.age} years old."

    # Method to return dog's sound
    def speak(self, sound):
        return f"{self.name} says {sound}."


# Creating objects (instances of the Dog class)
dog1 = Dog("Buddy", 5)
dog2 = Dog("Max", 3)

# Accessing class attribute
print(f"Dog species: {Dog.species}")

# Accessing instance attributes
print(f"{dog1.name} is {dog1.age} years old.")
print(f"{dog2.name} is {dog2.age} years old.")

# Accessing methods
print(dog1.description())
print(dog2.speak("Woof"))

# Modifying an instance attribute
dog1.age = 6
```

```
print(f"After a birthday, {dog1.name} is now {dog1.age} years old.")
```

Output:

```
Dog species: Canis familiaris
Buddy is 5 years old.
Max is 3 years old.
Buddy is 5 years old.
Max says Woof.
After a birthday, Buddy is now 6 years old.
```

8. Create program in Python to demonstrate the example of Inheritance

Program:

```
# Base class
class Animal:
    def __init__(self, name):
        self.name = name

    def sound(self):
        return "Some generic animal sound"

    def eat(self):
        return f"{self.name} is eating."

# Derived class
class Dog(Animal):
    def __init__(self, name, breed):
        # Calling the constructor of the base class
        super().__init__(name)
        self.breed = breed

    # Overriding the sound method
    def sound(self):
        return "Woof! Woof!"
```

```
    def fetch(self, item):
        return f"{self.name} is fetching the {item}."


# Creating an instance of the base class
generic_animal = Animal("Generic Animal")
print(generic_animal.sound())  # Output: Some generic animal sound
print(generic_animal.eat())    # Output: Generic Animal is eating.

# Creating an instance of the derived class
dog = Dog("Buddy", "Golden Retriever")
print(dog.sound())
```
Output: Woof! Woof!
```
print(dog.eat())
```
Output: Buddy is eating.
```
print(dog.fetch("ball"))
```
Output: Buddy is fetching the ball.

9. Program on Functions and Lambda Expressions:

Program:

```
# Regular function to add two numbers
def add(x, y):
    return x + y

# Lambda function to add two numbers
add_lambda = lambda x, y: x + y

# Regular function to find the square of a number
def square(n):
    return n * n

# Lambda function to find the square of a number
square_lambda = lambda n: n * n

# Regular function to filter even numbers from a list
def is_even(n):
    return n % 2 == 0

# Using lambda with filter() to filter even numbers from a list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Using regular function with filter()
even_numbers = list(filter(is_even, numbers))

# Using lambda function with filter()
even_numbers_lambda = list(filter(lambda n: n % 2 == 0, numbers))

# Demonstrating the functions
print("Regular function add(5, 3):", add(5, 3))
print("Lambda function add_lambda(5, 3):", add_lambda(5, 3))

print("Regular function square(4):", square(4))
print("Lambda function square_lambda(4):", square_lambda(4))

print("Even numbers using regular function:", even_numbers)
print("Even numbers using lambda function:", even_numbers_lambda)
```

Output:
Regular function add(5, 3): 8
Lambda function add_lambda(5, 3): 8
Regular function square(4): 16
Lambda function square_lambda(4): 16
Even numbers using regular function: [2, 4, 6, 8, 10]
Even numbers using lambda function: [2, 4, 6, 8, 10]

10. Intersection of Two Arrays:Given two integer arrays nums1 and nums2, returnan array of their intersection. Each element in the result must be unique and youmay return the result in any order.

Program:

```
def intersection(nums1, nums2):
    # Convert both lists to sets to remove duplicates and allow for set operations
    set1 = set(nums1)
    set2 = set(nums2)

    # Find the intersection of the two sets
    result_set = set1.intersection(set2)

    # Convert the set back to a list (if you need the result as a list)
    return list(result_set)

# Example usage
nums1 = [4, 9, 5, 9]
nums2 = [9, 4, 9, 8, 4]

result = intersection(nums1, nums2)
print("Intersection:", result)
```

Output:

Intersection: [9, 4]

11. Third Maximum Number:Given an integer array nums, return the third distinctmaximum number in this array. If the third maximum does not exist, return themaximum number.

Program:

```
def thirdMax(nums):
    # Convert the list to a set to eliminate duplicates
    nums_set = set(nums)

    # If there are fewer than 3 distinct numbers, return the maximum
    if len(nums_set) < 3:
        return max(nums_set)

    # Remove the maximum element twice to get to the third maximum
    nums_set.remove(max(nums_set))  # Remove the first maximum
    nums_set.remove(max(nums_set))  # Remove the second maximum

    # The next maximum is the third maximum
    return max(nums_set)

# Example usage
nums = [3, 2, 1]
result = thirdMax(nums)
print("Third maximum:", result)
```

Output: 1

```
nums = [1, 2]
result = thirdMax(nums)
print("Third maximum:", result)
```

Output: 2 (since there is no third distinct maximum)

```
nums = [2, 2, 3, 1]
result = thirdMax(nums)
print("Third maximum:", result)
```

Output: 1

12. Set Mismatch: You have a set of integers s, which originally contains all the numbers from 1 to n. Unfortunately, due to some error, one of the numbers in got duplicated to another number in the set, which results in repetition of one number and loss of another number.You are given an integer array nums representing the data status of this set after the error.Find the number that occurs twice and the number that is missing and return them in the form of an array

Program:
```
def findErrorNums(nums):
    # Variables to store the duplicated and missing numbers
    duplicated = -1
    n = len(nums)

    # Calculate the expected sum of the first n natural numbers
    expected_sum = n * (n + 1) // 2

    # Initialize a set to find the duplicated number
    seen = set()

    # Calculate the actual sum and identify the duplicated number
    actual_sum = 0
    for num in nums:
        if num in seen:
            duplicated = num
        seen.add(num)
        actual_sum += num

    # The missing number can be found by:
    missing = expected_sum - (actual_sum - duplicated)
    return [duplicated, missing
# Example usage
nums = [1, 2, 2, 4]
result = findErrorNums(nums)
print("Duplicated and missing numbers:", result)
```
Output: [2, 3]

```
nums = [3, 2, 2]
result = findErrorNums(nums)
print("Duplicated and missing numbers:", result)
```
Output: [2, 1]

SKILLING PROGRAMS

1. Valid Parentheses: Given a string s containing just the characters '(', ')', '{', '}', '['and ']', determine if the input string is valid. An input string is valid if: a. Open brackets must be closed by the same type of brackets. b. Open brackets must be closed in the correct order.

Program:

```
def isValid(s: str) -> bool:
    # Dictionary to hold mappings of closing brackets to opening brackets
    bracket_map = {')': '(', '}': '{', ']': '['}
    stack = []

    # Iterate through each character in the string
    for char in s:
        if char in bracket_map:
            # Pop the topmost element from the stack if it exists, otherwise a
dummy value
            top_element = stack.pop() if stack else '#'

            # If the mapped bracket doesn't match the top element, it's invalid
            if bracket_map[char] != top_element:
                return False
        else:
            # It's an opening bracket, so push onto the stack
            stack.append(char)

    # If the stack is empty, all opening brackets were matched properly
    return not stack

# Example usage:
print(isValid("()"))      # True
print(isValid("()[]{}"))  # True
print(isValid("(]"))      # False
print(isValid("([)]"))    # False
print(isValid("{[]}"))    # True
```

Output:

```
True    # for the input "()"
True    # for the input "()[]{}"
False   # for the input "(]"
False   # for the input "([)]"
True    # for the input "{[]}"
```

2. Binary Tree in Order Traversal: Given the root of a binary tree, return the in order traversal of its nodes' values.

Program:

```
Definition for a binary tree node.
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def inorderTraversal(root):
    result = []

    def traverse(node):
        if node:
            traverse(node.left)  # Traverse the left subtree
            result.append(node.val)  # Visit the root node
            traverse(node.right)  # Traverse the right subtree

    traverse(root)
    return result

example usage:
# Example of constructing a binary tree
#    1
#     \
#      2
#     /
#    3

root = TreeNode(1)
root.right = TreeNode(2)
root.right.left = TreeNode(3)

# Get the in-order traversal
print(inorderTraversal(root))
```

Output: [1, 3, 2]

3. Roman to Integer: Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Given a roman numeral, convert it to an integer.

Program:

```
def romanToInt(s: str) -> int:
    # Mapping of Roman numerals to their corresponding integer values
    roman_map = {
        'I': 1, 'V': 5, 'X': 10,
        'L': 50, 'C': 100, 'D': 500, 'M': 1000
    }

    total = 0
    prev_value = 0
    # Iterate through the string in reverse order
    for char in reversed(s):
        current_value = roman_map[char]

        # If the current value is less than the previous one, subtract it
        if current_value < prev_value:
            total -= current_value
        else:
            # Otherwise, add it to the total
            total += current_value

        # Update the previous value for the next iteration
        prev_value = current_value

    return total

# Example usage:
print(romanToInt("III"))
```

Output: 3

```
print(romanToInt("IV"))
```

Output: 4

```
print(romanToInt("IX"))
```

Output: 9

```
print(romanToInt("LVIII"))
```

Output: 58

```
print(romanToInt("MCMXCIV"))
```

Output: 1994

4. Program Creating Classes/Objects and Inheritance

Program:

Step 1: Define the Base Class Animal

```
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species

    def make_sound(self):
        return "Some generic sound"

    def info(self):
        return f"I am {self.name}, a {self.species}."

# Example usage
generic_animal = Animal("Generic", "Unknown")
print(generic_animal.info())
```

Output: I am Generic, a Unknown.

```
print(generic_animal.make_sound())
```

Output: Some generic sound

Create Derived Classes Dog and Cat

```
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name, species="Dog")
        self.breed = breed

    def make_sound(self):
        return "Woof!"

    def info(self):
        return f"I am {self.name}, a {self.breed} dog."

class Cat(Animal):
    def __init__(self, name, color):
        super().__init__(name, species="Cat")
        self.color = color
```

```
    def make_sound(self):
        return "Meow!"

    def info(self):
        return f"I am {self.name}, a {self.color} cat."

# Example usage
dog = Dog("Buddy", "Golden Retriever")
print(dog.info())         # Output: I am Buddy, a Golden Retriever dog.
print(dog.make_sound())   # Output: Woof!

cat = Cat("Whiskers", "black")
print(cat.info())         # Output: I am Whiskers, a black cat.
print(cat.make_sound())   # Output: Meow.

Using Inheritance and Polymorphism:
animals = [
    Dog("Buddy", "Golden Retriever"),
    Cat("Whiskers", "black"),
    Animal("Generic", "Unknown")
]

for animal in animals:
    print(animal.info())
    print(animal.make_sound())
    print()  # Print a blank line for readability
```

Output:

```
I am Buddy, a Golden Retriever dog.
Woof!

I am Whiskers, a black cat.
Meow!

I am Generic, a Unknown.
Some generic sound
```

5. Program on Arithmetic Operations Using Modules in Python

Program:

Step 1: Create a Module (arithmetic_operations.py)

```
# arithmetic_operations.py

def add(a, b):
    return a + b

def subtract(a, b):
 return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b != 0:
        return a / b
    else:
        return "Division by zero is not allowed"
```

Step 2: Create the Main Program

```
# main_program.py

import arithmetic_operations as ao

def main():
    a = float(input("Enter the first number: "))
    b = float(input("Enter the second number: "))

    print(f"Addition: {a} + {b} = {ao.add(a, b)}")
    print(f"Subtraction: {a} - {b} = {ao.subtract(a, b)}")
    print(f"Multiplication: {a} * {b} = {ao.multiply(a, b)}")
    print(f"Division: {a} / {b} = {ao.divide(a, b)}")

if __name__ == "__main__":
    main()
```

Step 3: Run the Program

Output:

Enter the first number: 10

Enter the second number: 5

Addition: 10.0 + 5.0 = 15.0

Subtraction: 10.0 - 5.0 = 5.0

Multiplication: 10.0 * 5.0 = 50.0

Division: 10.0 / 5.0 = 2.0

6. Program on File Handling and Exception Handling Concepts

Program:

```
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            print("File content:")
            print(content)
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' does not exist.")
    except IOError:
        print(f"Error: Could not read file '{file_path}'.")

def write_file(file_path, content):
    try:
        with open(file_path, 'w') as file:
            file.write(content)
            print(f"Content written to '{file_path}' successfully.")
    except IOError:
        print(f"Error: Could not write to file '{file_path}'.")

def append_file(file_path, content):
    try:
        with open(file_path, 'a') as file:
```

```
            file.write(content)
            print(f"Content appended to '{file_path}' successfully.")
        except IOError:
            print(f"Error: Could not append to file '{file_path}'.")

    def main():
        file_path = 'example.txt'

        # Write to the file
        write_content = "Hello, this is a test file.\n"
        write_file(file_path, write_content)

        # Append to the file
        append_content = "This line is appended to the file.\n"
        append_file(file_path, append_content)

        # Read from the file
        read_file(file_path)

        # Attempt to read a non-existent file
        read_file('non_existent_file.txt')

    if __name__ == "__main__":
        main()
```

Output: Hello, this is a test file.This line is appended to the file.

7. Number of 1 Bits: Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).

Program:

```
Function: Counting the Number of '1' Bits
def hamming_weight(n):
    count = 0
    while n:
        count += n & 1  # Add 1 to count if the least significant bit is 1
        n >>= 1  # Right shift n by 1 to check the next bit
    return count

Example Usage:
# Example usage:
n = 11  # Binary representation: 1011
print(hamming_weight(n))
```

Output: 3

8. Write an algorithm to determine if a number n is happy. A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits. Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy. Return true if n is a happy number, and false if not

Program:

```
def is_happy(n: int) -> bool:
    def get_next(number: int) -> int:
        """Helper function to get the next number in the sequence."""
        return sum(int(digit) ** 2 for digit in str(number))

    seen_numbers = set()

    while n != 1 and n not in seen_numbers:
        seen_numbers.add(n)
        n = get_next(n)

    return n == 1

Example Usage:
print(is_happy(19))
```

Output: True

9. Contains Duplicate Given an integer array nums, return true if any value appearsat least twice in the array, and return false if every element is distinct.

Program:

```
def contains_duplicate(nums: list[int]) -> bool:
    seen = set()

    for num in nums:
        if num in seen:
            return True
        seen.add(num)

    return Fals

Example Usage:
print(contains_duplicate([1, 2, 3, 1]))
```

Output: True

```
print(contains_duplicate([1, 2, 3, 4]))
```

Output: False

10. Reverse String Write a function that reverses a string. The input string is givenas an array of characters s.You must do this by modifying the input array in-place with O(1) extra memory.

Program:

```
def reverse_string(s: list[str]) -> None:
    left, right = 0, len(s) - 1

    while left < right:
        # Swap the characters at indices left and right
        s[left], s[right] = s[right], s[left]
        # Move towards the middle
        left += 1
        right -= 1
Example Usage:
s = ['h', 'e', 'l', 'l', 'o']
reverse_string(s)
print(s)
```

Output: ['o', 'l', 'l', 'e', 'h']

11. Add Digits :Given an integer num, repeatedly add all its digits until the resulthas only one digit, and return it.

Program:

```
def add_digits(num: int) -> int:
    while num >= 10:
        num = sum(int(digit) for digit in str(num))
    return num

Example Usage:
print(add_digits(38))
```

Output: 2

```
print(add_digits(123))
```

Output: 6

12. Create Program on JSON Application: User Data Management and regular Expressions Application: Email Validator

Program:

```
User Data Management and Email Validator Program

import json
import re

# Define the email regex pattern
email_pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'

def is_valid_email(email: str) -> bool:
    """Validate the email using regex."""
    return re.match(email_pattern, email) is not None

def load_user_data(file_path: str) -> dict:
    """Load user data from a JSON file."""
    try:
        with open(file_path, 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return {}
```

```
def save_user_data(file_path: str, data: dict) -> None:
    """Save user data to a JSON file."""
    with open(file_path, 'w') as file:
        json.dump(data, file, indent=4)

def add_user(file_path: str, name: str, email: str) -> bool:
    """Add a new user to the JSON file if the email is valid."""
    if not is_valid_email(email):
        print("Invalid email address.")
        return False

    user_data = load_user_data(file_path)

    if email in user_data:
        print("User with this email already exists.")
        return False

    user_data[email] = {"name": name, "email": email}
    save_user_data(file_path, user_data)
    print(f"User '{name}' added successfully.")
    return True

def display_users(file_path: str) -> None:
    """Display all users from the JSON file."""
    user_data = load_user_data(file_path)
    if not user_data:
        print("No users found.")
    else:
        for email, details in user_data.items():
            print(f"Name: {details['name']}, Email: {email}")

def main():
    file_path = 'user_data.json'

    while True:
        print("\nUser Data Management")
        print("1. Add User")
        print("2. Display Users")
        print("3. Exit")
```

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

        if choice == '1':
            name = input("Enter name: ")
            email = input("Enter email: ")
            add_user(file_path, name, email)
        elif choice == '2':
            display_users(file_path)
        elif choice == '3':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

```
User Data Management
1. Add User
2. Display Users
3. Exit
Enter your choice: 1
Enter name: Alice
Enter email: alice@example.com
User 'Alice' added successfully.

User Data Management
1. Add User
2. Display Users
3. Exit
Enter your choice: 3
Exiting the program.
```