# FINAL LAB REPORT

**Course Code:** CS502
**Course Title:** Computer Programming with LAB

**Submitted To:**

**Name:** Dr. Rubaiyat Islam
**Designation:** Associate Professor
**Department:** Department of Software Engineering (SWE)
**Daffodil International University**

**Submitted By:**

**Name:** Md. Tazmirul Islam
**ID:** 251-56-013
**Section:** M. Sc. in Cyber Security
**Semester:** Spring 2025
**Department:** Department of Software Engineering (SWE)
**Daffodil International University**

**Submission Date:** 2025-04-18

***Lacture6:***
***### Class Assignment Problems:***
***1. Bank Account Management System where we create a BankAccount class that allows a user to deposit and withdraw money. Possible variables are account_name, balance***

```
class BankAccount:
    def __init__(self, account_name, balance=0):
        self.account_name = account_name
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"{self.account_name} deposited ${amount}. New balance: ${self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"{self.account_name} withdrew ${amount}. New balance: ${self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        print(f"{self.account_name}'s current balance: ${self.balance}")

# Example usage
account1 = BankAccount("Alice")
account1.deposit(1000)
account1.withdraw(300)
account1.display_balance()
```

Output:

```
Alice deposited $1000. New balance: $1000
Alice withdrew $300. New balance: $700
Alice's current balance: $700
```

***2. Hospital Management System : This system has a base class Person and two subclasses: Doctor and Patient that inherit attributes and define additional ones.***
***Base/parent class (Person) attributes : name , age, gender***
***Person/sub class Doctor attributes : name,age, gender, speciality, salary***
***Person/sub class Patient attributes : name,age, gender, Disease, Fee***

```
# Base class
class Person:
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}, Gender: {self.gender}")

# Subclass: Doctor
class Doctor(Person):
    def __init__(self, name, age, gender, speciality, salary):
        super().__init__(name, age, gender)
        self.speciality = speciality
        self.salary = salary

    def display_info(self):
        super().display_info()
        print(f"Speciality: {self.speciality}, Salary: ${self.salary}")

# Subclass: Patient
class Patient(Person):
    def __init__(self, name, age, gender, disease, fee):
        super().__init__(name, age, gender)
        self.disease = disease
        self.fee = fee

    def display_info(self):
        super().display_info()
        print(f"Disease: {self.disease}, Fee: ${self.fee}")

# Example usage
print("Doctor Info:")
doctor1 = Doctor("Dr. Smith", 45, "Male", "Cardiologist", 120000)
doctor1.display_info()

print("\nPatient Info:")
patient1 = Patient("Jane Doe", 30, "Female", "Flu", 150)
patient1.display_info()
```

**Output:**

```
Doctor Info:
Name: Dr. Smith, Age: 45, Gender: Male
Speciality: Cardiologist, Salary: $120000

Patient Info:
Name: Jane Doe, Age: 30, Gender: Female
Disease: Flu, Fee: $150
```

***### Home Assignemnt***

***1. A system where a SmartDevice class represents a smart home device (like a light bulb) that can be turned on or off.***

```
class SmartDevice:
    def __init__(self, name):
        self.name = name
        self.status = False  # False = off, True = on

    def turn_on(self):
        if not self.status:
            self.status = True
            print(f"{self.name} is now ON.")
        else:
            print(f"{self.name} is already ON.")

    def turn_off(self):
        if self.status:
            self.status = False
            print(f"{self.name} is now OFF.")
        else:
            print(f"{self.name} is already OFF.")

    def get_status(self):
        state = "ON" if self.status else "OFF"
        print(f"{self.name} is currently {state}.")

# Create smart devices
light = SmartDevice("Living Room Light")
fan = SmartDevice("Bedroom Fan")

# Control devices
light.get_status()
light.turn_on()
light.get_status()
light.turn_off()

print()  # just spacing

fan.turn_on()
fan.get_status()
```

**Output:**

```
Living Room Light is currently OFF.
Living Room Light is now ON.
Living Room Light is currently ON.
Living Room Light is now OFF.

Bedroom Fan is now ON.
Bedroom Fan is currently ON.
```

***2. A system where a Book class manages book information and checks if a book is available for borrowing.***

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.is_available = True

    def display_info(self):
        status = "Available" if self.is_available else "Not Available"
        print(f"Title: {self.title}, Author: {self.author}, Status: {status}")

    def borrow_book(self):
        if self.is_available:
            self.is_available = False
            print(f"You have borrowed '{self.title}'.")
        else:
            print(f"Sorry, '{self.title}' is currently not available.")

    def return_book(self):
        if not self.is_available:
            self.is_available = True
            print(f"'{self.title}' has been returned. Thank you!")
        else:
            print(f"'{self.title}' was not borrowed.")

# Create book objects
book1 = Book("The Alchemist", "Paulo Coelho")
book2 = Book("1984", "George Orwell")

# Display info
book1.display_info()
book2.display_info()

# Borrow and return actions
book1.borrow_book()
book1.display_info()
book1.borrow_book()  # Try borrowing again

book1.return_book()
book1.display_info()
```

**Output:**

```
Title: The Alchemist, Author: Paulo Coelho, Status: Available
Title: 1984, Author: George Orwell, Status: Available
You have borrowed 'The Alchemist'.
Title: The Alchemist, Author: Paulo Coelho, Status: Not Available
Sorry, 'The Alchemist' is currently not available.
'The Alchemist' has been returned. Thank you!
Title: The Alchemist, Author: Paulo Coelho, Status: Available
```

***3. Vehicle Management System: This system has a base class Vehicle and two subclasses: Car and Bike that inherit common attributes and define additional ones.***

```
# Base class
class Vehicle:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    def display_info(self):
        print(f"Brand: {self.brand}, Model: {self.model}, Year: {self.year}")

# Subclass: Car
class Car(Vehicle):
    def __init__(self, brand, model, year, num_doors, fuel_type):
        super().__init__(brand, model, year)
        self.num_doors = num_doors
        self.fuel_type = fuel_type

    def display_info(self):
        super().display_info()
        print(f"Doors: {self.num_doors}, Fuel Type: {self.fuel_type}")

# Subclass: Bike
class Bike(Vehicle):
    def __init__(self, brand, model, year, engine_cc, has_carrier):
        super().__init__(brand, model, year)
        self.engine_cc = engine_cc
        self.has_carrier = has_carrier

    def display_info(self):
        super().display_info()
        carrier = "Yes" if self.has_carrier else "No"
        print(f"Engine: {self.engine_cc}cc, Carrier: {carrier}")

car1 = Car("Toyota", "Corolla", 2020, 4, "Petrol")
bike1 = Bike("Yamaha", "FZ", 2022, 150, False)

print("Car Info:")
car1.display_info()

print("\nBike Info:")
bike1.display_info()
```

**Output:**

```
Car Info:
Brand: Toyota, Model: Corolla, Year: 2020
Doors: 4, Fuel Type: Petrol

Bike Info:
Brand: Yamaha, Model: FZ, Year: 2022
Engine: 150cc, Carrier: No
```

***lacture 5:***
***### 1. Analyzing Sales Data***
***scenario: You are given a list of sales transactions. Each transaction contains a sales amount in dollars. You need to:***
***- Use lambda to define small functions.***
***- Use filter to extract sales above a threshold.***
***- Use map to apply a discount to all sales.***
***- Use reduce to compute the total sales amount.***
***from functools import reduce***

```
# Sample sales data (in dollars)
sales = [120, 450, 320, 80, 150, 600, 230]

# Threshold: extract sales > $200 using filter and lambda
high_sales = list(filter(lambda x: x > 200, sales))
print("Sales above $200:", high_sales)
```

```
# Apply 10% discount to all sales using map and lambda
discounted_sales = list(map(lambda x: x * 0.9, sales))
print("Sales after 10% discount:", discounted_sales)

# Compute total sales amount using reduce and lambda
total_sales = reduce(lambda x, y: x + y, sales)
print("Total sales (original):", total_sales)

# Compute total discounted sales
total_discounted_sales = reduce(lambda x, y: x + y, discounted_sales)
print("Total sales (after discount):", total_discounted_sales)
```

**Output:**

```
Sales above $200: [450, 320, 600, 230]
Sales after 10% discount: [108.0, 405.0, 288.0, 72.0, 135.0, 540.0, 207.0]
Total sales (original): 1950
Total sales (after discount): 1755.0
```

***### 2. Student Grades Processing***

***Scenario: Given a list of student scores, filter out passing grades, curve scores, and find the highest score.***

```
# List of student scores
scores = [45, 67, 89, 38, 74, 59, 92, 48]

# 1. Filter out passing grades (50 and above)
passing_scores = list(filter(lambda x: x >= 50, scores))
print("Passing Scores:", passing_scores)

# 2. Curve scores: add 5 points to each (but max 100)
curved_scores = list(map(lambda x: min(x + 5, 100), scores))
print("Curved Scores:", curved_scores)

# 3. Find the highest score (after curving)
highest_score = max(curved_scores)
print("Highest Curved Score:", highest_score)
```

**Output:**

```
Passing Scores: [67, 89, 74, 59, 92]
Curved Scores: [50, 72, 94, 43, 79, 64, 97, 53]
Highest Curved Score: 97
```

***### 3. E-commerce Product Price Analysis***

***Scenario: You have a list of product prices and need to apply discounts, filter expensive items, and calculate total revenue.***

***from functools import reduce***

```
# Sample product prices (in dollars)
product_prices = [120, 75, 300, 45, 180, 250, 95]

# 1. Apply 15% discount using map
discounted_prices = list(map(lambda price: round(price * 0.85, 2), product_prices))
print("Discounted Prices:", discounted_prices)
```

```
# 2. Filter expensive items (still over $100 after discount)
expensive_items = list(filter(lambda price: price > 100, discounted_prices))
print("Expensive Items (after discount):", expensive_items)

# 3. Calculate total revenue (sum of all discounted prices)
total_revenue = reduce(lambda x, y: x + y, discounted_prices)
print("Total Revenue:", total_revenue)
```

**Output:**

```
Discounted Prices: [102.0, 63.75, 255.0, 38.25, 153.0, 212.5, 80.75]
Expensive Items (after discount): [102.0, 255.0, 153.0, 212.5]
Total Revenue: 905.25
```

***### 4. Word Processing: Sentence Transformation***
***Scenario: Given a list of words, filter long words, capitalize all words, and count total characters.***
***from functools import reduce***

```
# Sample list of words
words = ["processing", "data", "is", "fun", "and", "educational", "awesome"]

# 1. Filter long words (length > 5)
long_words = list(filter(lambda w: len(w) > 5, words))
print("Long Words:", long_words)

# 2. Capitalize all words
capitalized_words = list(map(lambda w: w.upper(), words))
print("Capitalized Words:", capitalized_words)

# 3. Count total characters in all words (excluding spaces)
total_chars = sum(len(word) for word in words)
print("Total Characters:", total_chars)
```

**Output:**

```
Long Words: ['processing', 'educational', 'awesome']
Capitalized Words: ['PROCESSING', 'DATA', 'IS', 'FUN', 'AND', 'EDUCATIONAL', 'AWESOME']
Total Characters: 47
```

***### 5. Employee Salary Processing***
***Scenario: Given employee salaries, apply a bonus, filter high earners, and find the highest salary.***
***from functools import reduce***

```
# List of employee salaries
salaries = [35000, 50000, 75000, 120000, 45000, 67000, 88000]

# 1. Apply a 10% bonus to each salary using map
bonus_salaries = list(map(lambda salary: round(salary * 1.1, 2), salaries))
print("Salaries after Bonus:", bonus_salaries)

# 2. Filter high earners (e.g., salary above 60,000 after bonus)
high_earners = list(filter(lambda salary: salary > 60000, bonus_salaries))
print("High Earners (after bonus):", high_earners)

# 3. Find the highest salary after bonus
```

```
highest_salary = max(bonus_salaries)
print("Highest Salary (after bonus):", highest_salary)
```

**Output:**

```
Salaries after Bonus: [38500.0, 55000.0, 82500.0, 132000.0, 49500.0, 73700.0, 96800.0]
High Earners (after bonus): [82500.0, 132000.0, 73700.0, 96800.0]
Highest Salary (after bonus): 132000.0
```

***### 6. List of Numbers: Even Filtering and Summation***
***Scenario: Given a list of numbers, filter even numbers, square all numbers, and sum them up.***

```
# List of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# 1. Filter even numbers
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print("Even Numbers:", even_numbers)

# 2. Square all numbers
squared_numbers = list(map(lambda x: x ** 2, numbers))
print("Squared Numbers:", squared_numbers)

# 3. Sum of all squared numbers
sum_of_squares = sum(squared_numbers)
print("Sum of Squared Numbers:", sum_of_squares)
```

**Output:**

```
Even Numbers: [2, 4, 6, 8, 10]
Squared Numbers: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Sum of Squared Numbers: 385
```

***Lacture3:***
***### Looping Through Dictionary***
***- Loop through a dictionary and print each key and value.***
***- Find and print all keys that have values greater than a given number.***
***- Count the occurrences of each value in a dictionary.***
***- Filter out dictionary items where values are less than a threshold.***
***- Print dictionary items in sorted order of their keys.***

```
# Sample dictionary
data = {
   'apple': 50,
   'banana': 120,
   'cherry': 90,
   'date': 30,
   'elderberry': 150,
   'fig': 110
}

# 1. Loop through a dictionary and print each key and value
print("Key-Value Pairs:")
for key, value in data.items():
   print(f"{key}: {value}")
```

```
# 2. Find and print all keys with values greater than a given number (e.g., 100)
threshold = 100
keys_above_threshold = [key for key, value in data.items() if value > threshold]
print(f"\nKeys with values greater than {threshold}:", keys_above_threshold)

# 3. Count the occurrences of each value in the dictionary
# In case you want to count how many times each value appears:
from collections import Counter
value_counts = Counter(data.values())
print("\nOccurrences of each value:")
for value, count in value_counts.items():
    print(f"Value {value} appears {count} times")

# 4. Filter out dictionary items where values are less than a threshold (e.g., 100)
filtered_data = {key: value for key, value in data.items() if value >= threshold}
print(f"\nFiltered dictionary (values >= {threshold}):", filtered_data)

# 5. Print dictionary items in sorted order of their keys
sorted_data = dict(sorted(data.items()))
print("\nDictionary sorted by keys:")
for key, value in sorted_data.items():
    print(f"{key}: {value}")
```

**Output:**

```
Key-Value Pairs:
apple: 50
banana: 120
cherry: 90
date: 30
elderberry: 150
fig: 110

Keys with values greater than 100: ['banana', 'elderberry', 'fig']

Occurrences of each value:
Value 50 appears 1 times
Value 120 appears 1 times
Value 90 appears 1 times
Value 30 appears 1 times
Value 150 appears 1 times
Value 110 appears 1 times

Filtered dictionary (values >= 100): {'banana': 120, 'elderberry': 150, 'fig': 110}

Dictionary sorted by keys:
apple: 50
banana: 120
cherry: 90
date: 30
elderberry: 150
fig: 110
```

***### String Slicing & Iteration***

***- Extract the first 5 and last 5 characters from a given string.***
***- Remove every alternate character from a string.***
***- Extract the substring from index 2 to 7.***
***- Iterate through a string and print each character.***
***- Reverse a string using slicing.***

```
# Sample string
text = "Hello, welcome to Python programming!"

# 1. Extract the first 5 and last 5 characters
first_5 = text[:5]
last_5 = text[-5:]
print("First 5 characters:", first_5)
print("Last 5 characters:", last_5)

# 2. Remove every alternate character
alternate_removed = text[::2]
print("\nString after removing every alternate character:", alternate_removed)

# 3. Extract substring from index 2 to 7 (inclusive of index 2, exclusive of index 7)
substring = text[2:7]
print("\nSubstring from index 2 to 7:", substring)

# 4. Iterate through the string and print each character
print("\nIterating through the string:")
for char in text:
    print(char)

# 5. Reverse the string using slicing
reversed_text = text[::-1]
print("\nReversed string:", reversed_text)
```

**Output:**

```
First 5 characters: Hello
Last 5 characters: amming!

String after removing every alternate character: Hlo ec ot yhn rgamn

Substring from index 2 to 7: llo,

Iterating through the string:
H
e
l
l
o
,

w
e
l
c
o
m
```

```
e

t
o

P
y
t
h
o
n

p
r
o
g
r
a
m
m
i
n
g
!
```

```
Reversed string: !gnimmargorp nohtyP ot emoclew ,olleH
```

***### String Strip Functions***
***- Remove leading spaces using lstrip().***
***- Remove trailing spaces using rstrip().***
***- Strip both leading and trailing spaces from a string.***
***- Remove specific characters (e.g., #) from the start of a string.***
***- Remove specific characters from the end of a string.***
***- Using List comprehension Generate a list of squares of numbers from 1 to 10.***
***- Using List Comprehension Create a list of even numbers from 1 to 2***

```
# Sample string with leading and trailing spaces
text = "   Hello, World!   "

# 1. Remove leading spaces using lstrip()
leading_stripped = text.lstrip()
print("After lstrip():", leading_stripped)

# 2. Remove trailing spaces using rstrip()
trailing_stripped = text.rstrip()
print("After rstrip():", trailing_stripped)

# 3. Strip both leading and trailing spaces using strip()
strip_both = text.strip()
print("After strip():", strip_both)

# 4. Remove specific characters (e.g., '#') from the start of a string
has_hash_start = "#Hello, World!"
```

```
remove_hash_start = has_hash_start.lstrip('#')
print("After removing '#' from the start:", remove_hash_start)

# 5. Remove specific characters (e.g., '#') from the end of a string
has_hash_end = "Hello, World!#"
remove_hash_end = has_hash_end.rstrip('#')
print("After removing '#' from the end:", remove_hash_end)

# 6. Generate a list of squares of numbers from 1 to 10 using list comprehension
squares = [x**2 for x in range(1, 11)]
print("\nList of squares from 1 to 10:", squares)

# 7. Create a list of even numbers from 1 to 20 using list comprehension
evens = [x for x in range(1, 21) if x % 2 == 0]
print("List of even numbers from 1 to 20:", evens)
```

Output:

```
After lstrip(): Hello, World!
After rstrip():    Hello, World!
After strip(): Hello, World!
After removing '#' from the start: Hello, World!
After removing '#' from the end: Hello, World!

List of squares from 1 to 10: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
List of even numbers from 1 to 20: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

***### List Operations***
***- Slice a list to extract the first 3 and last 3 elements.***
***- Add an item to a list, remove an item, and change an item at a given index.***
***- Copy a list and verify that modifying the copy does not affect the original.***
***- Join a list of words into a sentence using .join().***
***- Iterate through a list and print each element.***
***Print only the even numbers from a list.***
***- Use all() to check if all numbers in a list are positive.***
***- Use any() to check if any number in a list is greater than 50.***
***- Use all() and any() to check conditions in a list of strings.***

```
# Sample list
numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90]
words = ["hello", "world", "this", "is", "python"]

# 1. Slice a list to extract the first 3 and last 3 elements
first_3 = numbers[:3]
last_3 = numbers[-3:]
print("First 3 elements:", first_3)
print("Last 3 elements:", last_3)

# 2. Add an item to a list, remove an item, and change an item at a given index
numbers.append(100)  # Add item
print("\nList after adding 100:", numbers)

numbers.remove(20)  # Remove item
```

```
print("List after removing 20:", numbers)

numbers[2] = 99  # Change item at index 2
print("List after changing index 2 to 99:", numbers)

# 3. Copy a list and verify that modifying the copy does not affect the original
numbers_copy = numbers.copy()
numbers_copy[0] = 999  # Modify the copy
print("\nOriginal List after modifying copy:", numbers)
print("Modified Copy of List:", numbers_copy)

# 4. Join a list of words into a sentence using .join()
sentence = " ".join(words)
print("\nJoined sentence:", sentence)

# 5. Iterate through a list and print each element
print("\nIterating through the list:")
for num in numbers:
    print(num)

# 6. Print only the even numbers from a list
even_numbers = [num for num in numbers if num % 2 == 0]
print("\nEven numbers in the list:", even_numbers)

# 7. Use all() to check if all numbers in a list are positive
all_positive = all(num > 0 for num in numbers)
print("\nAre all numbers positive?", all_positive)

# 8. Use any() to check if any number in a list is greater than 50
any_above_50 = any(num > 50 for num in numbers)
print("Is any number greater than 50?", any_above_50)

# 9. Use all() and any() to check conditions in a list of strings
strings = ["apple", "banana", "cherry", "date"]
all_start_with_a = all(word.startswith('a') for word in strings)
any_start_with_b = any(word.startswith('b') for word in strings)
print("\nDo all strings start with 'a'?", all_start_with_a)
print("Does any string start with 'b'?", any_start_with_b)
```

**Output:**

```
First 3 elements: [10, 20, 30]
Last 3 elements: [70, 80, 90]

List after adding 100: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
List after removing 20: [10, 30, 40, 50, 60, 70, 80, 90, 100]
List after changing index 2 to 99: [10, 30, 99, 50, 60, 70, 80, 90, 100]

Original List after modifying copy: [10, 30, 99, 50, 60, 70, 80, 90, 100]
Modified Copy of List: [999, 30, 99, 50, 60, 70, 80, 90, 100]

Joined sentence: hello world this is python

Iterating through the list:
```

```
10
30
99
50
60
70
80
90
100

Even numbers in the list: [10, 30, 50, 60, 70, 80, 90, 100]

Are all numbers positive? True
Is any number greater than 50? True

Do all strings start with 'a'? False
Does any string start with 'b'? True
```

***### Built-in Functions***
***- Find the minimum, maximum, length, and sum of a list.***
***- Find the longest word in a list of words.***
***- Compute the sum of even numbers in a list.***
***- Find the shortest string in a list.***
***- Sort a list of tuples based on the second element.***

```python
# Sample list
numbers = [10, 20, 30, 40, 50]
words = ["apple", "banana", "cherry", "date", "elderberry"]
tuples = [(1, 10), (2, 30), (3, 20), (4, 50), (5, 40)]

# 1. Find the minimum, maximum, length, and sum of a list
min_num = min(numbers)
max_num = max(numbers)
length = len(numbers)
sum_of_numbers = sum(numbers)

print("Minimum:", min_num)
print("Maximum:", max_num)
print("Length:", length)
print("Sum:", sum_of_numbers)

# 2. Find the longest word in a list of words
longest_word = max(words, key=len)
print("\nLongest Word:", longest_word)

# 3. Compute the sum of even numbers in a list
even_sum = sum(num for num in numbers if num % 2 == 0)
print("\nSum of Even Numbers:", even_sum)

# 4. Find the shortest string in a list
shortest_string = min(words, key=len)
print("\nShortest String:", shortest_string)

# 5. Sort a list of tuples based on the second element
```

```
sorted_tuples = sorted(tuples, key=lambda x: x[1])
print("\nSorted Tuples based on second element:", sorted_tuples)
```

**Output:**

```
Minimum: 10
Maximum: 50
Length: 5
Sum: 150

Longest Word: elderberry

Sum of Even Numbers: 120

Shortest String: date

Sorted Tuples based on second element: [(1, 10), (3, 20), (2, 30), (5, 40), (4, 50)]
```

***### Dictionary Operations***
***Access an item in a dictionary by key.***
***- Add a new key-value pair to a dictionary.***
***Remove an item from a dictionary.***
***- Copy a dictionary and modify it without affecting the original.***
***- Loop through a dictionary and print keys and values.***

```
# Sample dictionary
person = {
    'name': 'John',
    'age': 30,
    'city': 'New York'
}

# 1. Access an item in a dictionary by key
name = person['name']
print("Accessed Name:", name)

# 2. Add a new key-value pair to a dictionary
person['job'] = 'Engineer'
print("\nUpdated Dictionary after adding new key-value pair:", person)

# 3. Remove an item from a dictionary
removed_item = person.pop('city')  # Removes 'city' and returns the value
print("\nRemoved 'city' item:", removed_item)
print("Updated Dictionary after removal:", person)

# 4. Copy a dictionary and modify it without affecting the original
person_copy = person.copy()
person_copy['age'] = 35  # Modify the copied dictionary
print("\nOriginal Dictionary after copy modification:", person)
print("Modified Copy of Dictionary:", person_copy)

# 5. Loop through a dictionary and print keys and values
print("\nLoop through Dictionary and Print Keys and Values:")
for key, value in person.items():
    print(f"{key}: {value}")
```

**Output:**

```
Accessed Name: John

Updated Dictionary after adding new key-value pair: {'name': 'John', 'age': 30, 'city': 'New York', 'job': 'Engineer'}

Removed 'city' item: New York
Updated Dictionary after removal: {'name': 'John', 'age': 30, 'job': 'Engineer'}

Original Dictionary after copy modification: {'name': 'John', 'age': 30, 'job': 'Engineer'}
Modified Copy of Dictionary: {'name': 'John', 'age': 35, 'job': 'Engineer'}

Loop through Dictionary and Print Keys and Values:
name: John
age: 30
job: Engineer
```