Daffodil International University

Lab Report on
Computer Programming Lab
Sub Code: CS 502

Submitted To

Dr. Rubaiyat Islam
Associate Professor
Department of Software Engineering
Daffodil International University

Submitted By

Name: - Wasique Al Azad Digonta
Roll: - 242-56-006
Semester: - Spring-2025
Department: - SWE (M.Sc. in Cyber Security)

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.

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.

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

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.

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.

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.

Looping Through Dictionary

```
# Loop through a dictionary and print each key and value.
data = {'a': 10, 'b': 25, 'c': 5, 'd': 40}
for key, value in data.items():
    print(key, value)

# Find and print all keys that have values greater than a given
number.
for key, value in data.items():
    if value > 20:
        print("Greater than 20:", key)

# Count the occurrences of each value in a dictionary.
values = list(data.values())
count_dict = {}
for value in values:
    count_dict[value] = count_dict.get(value, 0) + 1
print(count_dict)

# Filter out dictionary items where values are less than a threshold.
filtered = {k: v for k, v in data.items() if v >= 20}
print("Filtered:", filtered)

# Print dictionary items in sorted order of their keys.
for key in sorted(data):
    print(key, data[key])
```

String Slicing & Iteration

```
text = "PythonIsFunAndEasy"

# Extract the first 5 and last 5 characters from a given string.
print(text[:5], text[-5:])

# Remove every alternate character from a string.
print(text[::2])

# Extract the substring from index 2 to 7.
```

```
print(text[2:8])

# Iterate through a string and print each character.
for ch in text:
    print(ch)

# Reverse a string using slicing.
print(text[::-1])
```

String Strip Functions

```
s = "   Hello World   "
s2 = "###Python###"

# Remove leading spaces using lstrip().
print(s.lstrip())

# Remove trailing spaces using rstrip().
print(s.rstrip())

# Strip both leading and trailing spaces from a string.
print(s.strip())

# Remove specific characters (e.g., #) from the start of a string.
print(s2.lstrip('#'))

# Remove specific characters from the end of a string.
print(s2.rstrip('#'))

# Using List comprehension: Generate a list of squares of numbers from
1 to 10.
squares = [x*x for x in range(1, 11)]
print(squares)

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

List Operations

```
lst = [10, 20, 30, 40, 50, 60, 70]

# Slice a list to extract the first 3 and last 3 elements.
print(lst[:3], lst[-3:])

# Add an item to a list, remove an item, and change an item at a given
index.
lst.append(80)
lst.remove(40)
```

```
lst[0] = 15
print(lst)

# Copy a list and verify that modifying the copy does not affect the
original.
copy_lst = lst.copy()
copy_lst.append(99)
print("Original:", lst)
print("Copy:", copy_lst)

# Join a list of words into a sentence using .join().
words = ["Python", "is", "fun"]
sentence = ' '.join(words)
print(sentence)

# Iterate through a list and print each element.
for item in lst:
    print(item)

# Print only the even numbers from a list.
for num in lst:
    if num % 2 == 0:
        print("Even:", num)

# Use all() to check if all numbers in a list are positive.
print(all(x > 0 for x in lst))

# Use any() to check if any number in a list is greater than 50.
print(any(x > 50 for x in lst))

# Use all() and any() to check conditions in a list of strings.
names = ["Alice", "Bob", "Charlie"]
print(all(len(name) > 2 for name in names))
print(any(name.startswith("C") for name in names))
```

Built-in Functions

```
nums = [4, 7, 1, 12, 9, 2]

# Find the minimum, maximum, length, and sum of a list.
print(min(nums), max(nums), len(nums), sum(nums))

# Find the longest word in a list of words.
words = ["apple", "banana", "kiwi"]
print(max(words, key=len))

# Compute the sum of even numbers in a list.
even_sum = sum(x for x in nums if x % 2 == 0)
print(even_sum)
```

```python
# Find the shortest string in a list.
print(min(words, key=len))

# Sort a list of tuples based on the second element.
tuples = [(1, 3), (2, 1), (4, 2)]
sorted_tuples = sorted(tuples, key=lambda x: x[1])
print(sorted_tuples)
```

```
1 12 6 35
banana
18
kiwi
[(2, 1), (4, 2), (1, 3)]
```

Dictionary Operation

```python
d = {'x': 100, 'y': 200}

# Access an item in a dictionary by key.
print(d['x'])

# Add a new key-value pair to a dictionary.
d['z'] = 300
print(d)

# Remove an item from a dictionary.
del d['y']
print(d)

# Copy a dictionary and modify it without affecting the original.
d2 = d.copy()
d2['x'] = 999
print("Original:", d)
print("Copy:", d2)

# Loop through a dictionary and print keys and values.
for key, value in d.items():
    print(key, value)
```

```
100
{'x': 100, 'y': 200, 'z': 300}
{'x': 100, 'z': 300}
Original: {'x': 100, 'z': 300}
Copy: {'x': 999, 'z': 300}
x 100
z 300
```

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

prices = [150, 320, 45, 600, 125, 75, 220, 90, 480, 350]
# Filter out expensive products (above $200)
expensive_product  = list(filter(lambda n: n>200, prices))
print("Expensive products : ", expensive_product)

# we ant 15% discount on all sales
discount_prices = list(map(lambda x: x*0.85, prices))
print("Discounted prices :", discount_prices)

# calculate total sale price
total_sales = reduce(lambda x,y : x+y, prices )
print("Total Sale : ", total_sales)
```

```
Expensive products :  [320, 600, 220, 480, 350]
Discounted prices : [127.5, 272.0, 38.25, 510.0, 106.25, 63.75, 187.0,
76.5, 408.0, 297.5]
Total Sale :  2455
```

2. Student Grades Processing

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

```
from functools import reduce

scores = [55, 88, 74, 90, 45, 67, 80, 93, 38, 76]

# Filter passing grade(pass mark : 40)

passing_grade =  list(filter(lambda x: x>40, scores))
print("passing grade: ", passing_grade)

# Apply 5% curve on grades
curved_score =  list(map(lambda x: x * 1.05, scores))
print("Grade after curving scores: ", curved_score)

# The total scores
```

```
highest_socres = reduce(lambda x,y : x if x>y else y, curved_score)
print("Highest score after curving: ", highest_socres)
```

```
passing grade:  [55, 88, 74, 90, 45, 67, 80, 93, 76]
Grade after curving scores:  [57.75, 92.4, 77.7, 94.5, 47.25,
70.35000000000001, 84.0, 97.65, 39.9, 79.8]
Highest score after curving:  97.65
```

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

words = ["lambda", "map", "filter", "reduce", "python", "programming"]

# Long words filter out if greater than 5 characters

long_words = list(filter(lambda x: len(x)>5, words))
print("Long words list : ", long_words)

# upper case words converison

uppercase_words = list(map(lambda  x : x.upper(), words ))
print("Uppercase Words : ", uppercase_words)

# total characters in all words
total_character = reduce(lambda x,y : x + len(y), words, 0)
print("total character: ", total_character)
```

```
Long words list :  ['lambda', 'filter', 'reduce', 'python',
'programming']
Uppercase Words :  ['LAMBDA', 'MAP', 'FILTER', 'REDUCE', 'PYTHON',
'PROGRAMMING']
total character:  38
```

5. Employee Salary Processing

Scenario: Given employee salaries, apply a bonus, filter high earners, and find the highest salary.

```
from functools import reduce

salaries = [2500, 4200, 3800, 5500, 3000, 4800, 2000, 6000, 7200]
# Filter high earners (above $4000)
high_earners = list(filter(lambda x: x > 4000, salaries))
print("High Earners:", high_earners)

# Apply a 7% bonus to all salaries
new_salaries = list(map(lambda x: x * 1.07, salaries))
print("Salaries After Bonus:", new_salaries)
```

```
# Find the highest salary
max_salary = reduce(lambda x, y: x if x > y else y, salaries)
print("Highest Salary:", max_salary)
```

```
High Earners: [4200, 5500, 4800, 6000, 7200]
Salaries After Bonus: [2675.0, 4494.0, 4066.0000000000005, 5885.0,
3210.0, 5136.0, 2140.0, 6420.0, 7704.0]
Highest Salary: 7200
```

6. List of Numbers: Even Filtering and Summation

Scenario: Given a list of numbers, filter even numbers, square all numbers, and sum them up.

```
from functools import reduce

numbers = [12, 7, 9, 21, 34, 18, 5, 30, 27]

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

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

# Sum of all numbers
sum_numbers = reduce(lambda x, y: x + y, numbers)
print("Sum of Numbers:", sum_numbers)
```

```
Even Numbers: [12, 34, 18, 30]
Squared Numbers: [144, 49, 81, 441, 1156, 324, 25, 900, 729]
Sum of Numbers: 163
```

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
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

```
#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):
        self.account_name = account_name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"{amount} taka deposited. New balance: {self.balance}
taka.")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"{amount} taka withdrawn. Remaining balance:
{self.balance} taka.")
        else:
            print("Insufficient balance.")


account1 = BankAccount("Wasique", 5000)
account1.deposit(2000)
account1.withdraw(1000)
account1.withdraw(7000)
```

```
2000 taka deposited. New balance: 7000 taka.
1000 taka withdrawn. Remaining balance: 6000 taka.
Insufficient balance.
```

1. 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

```
'''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'''

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


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

    def show_info(self):
        print(f"Doctor Name: {self.name}")
        print(f"Age: {self.age}, Gender: {self.gender}")
        print(f"Speciality: {self.speciality}, Salary: {self.salary}
taka")


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

    def show_info(self):
        print(f"Patient Name: {self.name}")
        print(f"Age: {self.age}, Gender: {self.gender}")
        print(f"Disease: {self.disease}, Fee: {self.fee} taka")


doctor1 = Doctor("Dr. Wasique", 40, "Male", "Cardiologist", 80000)
doctor1.show_info()

print()

patient1 = Patient("Akhter", 25, "Female", "Fever", 500)
patient1.show_info()
```

```
Doctor Name: Dr. Wasique
Age: 40, Gender: Male
Speciality: Cardiologist, Salary: 80000 taka
```

```
Patient Name: Akhter
Age: 25, Gender: Female
Disease: Fever, Fee: 500 taka
```

1. A system where a SmartDevice class represents a smart home device (like a light bulb) that can be turned on or off.
2. A system where a Book class manages book information and checks if a book is available for borrowing.
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.

```python
#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

    def turn_on(self):
        self.status = True
        print(f"{self.name} is now ON.")

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

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

bulb = SmartDevice("Wasique's Smart Bulb")
bulb.show_status()
bulb.turn_on()
bulb.show_status()
bulb.turn_off()
bulb.show_status()
```

```
Wasique's Smart Bulb is currently OFF.
Wasique's Smart Bulb is now ON.
Wasique's Smart Bulb is currently ON.
Wasique's Smart Bulb is now OFF.
Wasique's Smart Bulb is currently OFF.
```

```python
#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.available = True

    def borrow(self, reader_name):
        if self.available:
            self.available = False
            print(f"{reader_name} borrowed '{self.title}' by
{self.author}.")
        else:
            print(f"Sorry {reader_name}, '{self.title}' is not
available.")

    def return_book(self):
        self.available = True
        print(f"'{self.title}' has been returned and is now
available.")

    def check_status(self):
        status = "available" if self.available else "not available"
        print(f"'{self.title}' is currently {status}.")

book1 = Book("Python for Beginners", "Azad")
book1.check_status()
book1.borrow("Digonta")
book1.check_status()
book1.borrow("Wasique")
book1.return_book()
book1.check_status()
```

```
'Python for Beginners' is currently available.
Digonta borrowed 'Python for Beginners' by Azad.
'Python for Beginners' is currently not available.
Sorry Wasique, 'Python for Beginners' is not available.
'Python for Beginners' has been returned and is now available.
'Python for Beginners' is currently available.
```

```
#Vehicle Management System: This system has a base class Vehicle and
two subclasses: Car and Bike that inherit common attributes and define
additional ones.
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

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

class Car(Vehicle):
    def __init__(self, brand, model, seats):
        super().__init__(brand, model)
```

```
        self.seats = seats

    def show_info(self):
        super().show_info()
        print(f"Type: Car, Seats: {self.seats}")

class Bike(Vehicle):
    def __init__(self, brand, model, cc):
        super().__init__(brand, model)
        self.cc = cc

    def show_info(self):
        super().show_info()
        print(f"Type: Bike, Engine: {self.cc}cc")

car1 = Car("Toyota", "2025", 5)
bike1 = Bike("Yamaha", "R15", 150)

car1.show_info()
print("----------------------")
bike1.show_info()
```

```
Brand: Toyota, Model: 2025
Type: Car, Seats: 5
----------------------
Brand: Yamaha, Model: R15
Type: Bike, Engine: 150cc
```