CONTENTS

LAB ACTIVITY 1

PYTHON DATA TYPES- II

Objectives

Explore advanced Python data types: lists, tuples, dictionaries, and sets. Understand mutability, immutability, and type casting for effective data manipulation.

Aim

Master advanced Python data types to make informed decisions in structuring and manipulating data.

Data Types Overview

Lists

Lists are dynamic, ordered collections that can hold various data types. They are versatile and allow modifications after creation.

```
my_list = [1, 'hello', 3.14, True]
print(my_list)
```

Tuples

Tuples are similar to lists but immutable, meaning once created, their elements cannot be changed. They provide stability for fixed collections.

```
my_tuple = (1, 'world', 2.71, False)
print(my_tuple)
```

Dictionaries

Dictionaries store key-value pairs, enabling efficient data retrieval. They are useful for representing relationships and mappings.

```
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}
print(my_dict)
```

Sets

Sets are unordered collections of unique elements. They are handy for tasks requiring distinct values.

```
my_set = {1, 2, 3, 4, 5}
print(my_set)
```

Mutability and Immutability

Mutability

Mutable data types, like lists, allow modifications to their elements after creation. This flexibility is suitable for dynamic data.

```
my_list = [1, 2, 3]
my_list[0] = 99
print(my_list)
```

Immutability

Immutable data types, exemplified by tuples, do not permit changes to their elements once defined. This ensures data stability.

```
my_tuple = (1, 2, 3)
# Attempting to modify will raise an error
# my_tuple[0] = 99
```

Type Casting

Type casting involves converting one data type to another. This is crucial for compatibility and seamless data manipulation.

```
num_str = '123'
num_int = int(num_str)
print(num_int)
```

Program

Create a Python program that accepts user input for a list of numbers, converts the input to a tuple, calculates the sum of the numbers, stores the sum in a dictionary with the original list as the key, and finally prints the dictionary along with the unique elements of the list in a sorted set.

```
# Accept user input for a list of numbers
num_list_str = input("Enter a list of numbers separated by spaces: ")

# Convert input to a list of integers
num_list = [int(num) for num in num_list_str.split()]

# Convert the list to a tuple
num_tuple = tuple(num_list)

```

```
# Calculate the sum of the numbers
sum_of_numbers = sum(num_list)

# Store the sum in a dictionary with the original list as the key
result_dict = {num_tuple: sum_of_numbers}

# Print the dictionary
print("Dictionary:")
print(result_dict)

# Print unique elements of the list in a sorted set
unique_elements = set(num_list)
print("Unique Elements (Sorted):")
print(sorted(unique_elements))
```

Output:

```
Enter a list of numbers separated by spaces: 3 5 2 5 1 4 5 6
Dictionary:
{(3, 5, 2, 5, 1, 4, 5, 6): 31}
Unique Elements (Sorted):
[1, 2, 3, 4, 5, 6]
```

Questions

Write a Python program that prompts the user to input information about employees, including their names, ages, departments, and salaries, stores this information in a dictionary where each employee's name is a key and the corresponding values are tuples containing their age, department, and salary, then calculates the average age and salary for all employees, identifies the youngest and oldest employees, prints a list of unique departments, and allows the user to search for an employee by name and view their details.

```
def input_employee_info():
    name = input("Enter employee name: ")
    age = int(input("Enter employee age: "))
    department = input("Enter employee department: ")
    salary = float(input("Enter employee salary: "))
    return name, (age, department, salary)

def calculate_average(data, key):
    total = sum(item[key] for item in data.values())
    return total / len(data)

def find_extremes(data, key, extreme_type):
    extreme_value = None
    extreme_employee = None

    for name, values in data.items():
        value = values[key]

        if extreme_value is None or (extreme_type == "min" and value
           < extreme_value) or (
```

```
                extreme_type == "max" and value > extreme_value):
            extreme_value = value
            extreme_employee = name

    return extreme_employee, extreme_value

def display_employee_details(employee_data, employee_name):
    if employee_name in employee_data:
        details = employee_data[employee_name]
        print(f"\nDetails for {employee_name}:")
        print(f"Age: {details[0]}")
        print(f"Department: {details[1]}")
        print(f"Salary: {details[2]}")
    else:
        print(f"\nEmployee {employee_name} not found.")

if __name__ == "__main__":
    employee_data = {}

    while True:
        print("\nEmployee Information System")
        print("1. Add Employee")
        print("2. Calculate Average Age and Salary")
        print("3. Identify Youngest and Oldest Employees")
        print("4. List Unique Departments")
        print("5. Search for Employee Details")
        print("6. Exit")

        choice = input("Enter your choice (1-6): ")

        if choice == "1":
            employee_name, employee_details = input_employee_info()
            employee_data[employee_name] = employee_details
            print(f"\nEmployee {employee_name} added successfully!")

        elif choice == "2":
            average_age = calculate_average(employee_data, key=0)
            average_salary = calculate_average(employee_data, key=2)
            print(f"\nAverage Age: {average_age:.2f}")
            print(f"Average Salary: {average_salary:.2f}")

        elif choice == "3":
            youngest_employee, youngest_age = find_extremes(
                employee_data, key=0, extreme_type="min")
            oldest_employee, oldest_age = find_extremes(employee_data
                , key=0, extreme_type="max")
            print(f"\nYoungest Employee: {youngest_employee} (Age: {
                youngest_age})")
            print(f"Oldest Employee: {oldest_employee} (Age: {
                oldest_age})")

        elif choice == "4":
```

```
            unique_departments = set(employee_details[1] for
                employee_details in employee_data.values())
            print("\nUnique Departments:")
            print(", ".join(unique_departments))

        elif choice == "5":
            search_name = input("\nEnter employee name to search: ")
            display_employee_details(employee_data, search_name)

        elif choice == "6":
            print("\nExiting Employee Information System. Goodbye!")
            break

        else:
            print("\nInvalid choice. Please enter a number between 1
                and 6.")
```

Output:

```
Employee Information System
1. Add Employee
2. Calculate Average Age and Salary
3. Identify Youngest and Oldest Employees
4. List Unique Departments
5. Search for Employee Details
6. Exit
Enter your choice (1-6): 1
Enter employee name: Rupali
Enter employee age: 33
Enter employee department: Computer
Enter employee salary: 35000

Employee Rupali added successfully!

Employee Information System
1. Add Employee
2. Calculate Average Age and Salary
3. Identify Youngest and Oldest Employees
4. List Unique Departments
5. Search for Employee Details
6. Exit
Enter your choice (1-6): 2

Average Age: 33.00
Average Salary: 35000.00

Employee Information System
1. Add Employee
2. Calculate Average Age and Salary
3. Identify Youngest and Oldest Employees
4. List Unique Departments
5. Search for Employee Details
6. Exit
```

```
Enter your choice (1-6): 6

Exiting Employee Information System. Goodbye!
```

Programming Exercise Questions

List Manipulation

1. Write a Python program that takes a list of numbers as input and returns a new list containing only the even numbers.
2. Implement a function that reverses the elements of a given list in-place.

Tuple Operations

1. Create a Python script that accepts a tuple of strings, sorts them alphabetically, and prints the sorted tuple.
2. Write a function that finds the index of a specific element in a tuple.

Dictionary Operations

1. Develop a program that merges two dictionaries, where the keys are names, and the values are ages. If a key exists in both dictionaries, combine the ages.
2. Implement a function that checks if a given key exists in a dictionary.

Set Operations

1. Write a Python function that takes two sets as input and returns a new set containing the common elements.
2. Create a program that removes duplicates from a list and converts it to a set.

Mutability and Immutability

1. Develop a function that modifies a list by squaring each element.
2. Write a Python script that attempts to modify an element in a tuple and handles the error gracefully.

Type Casting

1. Create a program that prompts the user to enter a number as a string, converts it to an integer, squares the result, and prints the squared value.
2. Implement a function that takes a list of numbers as strings and returns their sum as an integer.

Employee Information Management

1. Extend the employee information program to allow the user to update the details of an existing employee.
2. Write a function to calculate the average salary for employees in a specific department.

LAB ACTIVITY 2

CONTROL STATEMENTS IN PYTHON-I

Learning Objectives

In this chapter, we aim to achieve the following learning objectives:

- Understand the concept of control statements in Python.
- Explore the syntax and usage of Python's conditional statements, including `if`, `elif`, and `else`.
- Grasp the role of control statements in directing the flow of a program based on specified conditions.
- Learn to implement and apply control statements for effective decision-making in Python programs.

Aim

The primary aim of this chapter is to provide a comprehensive understanding of control statements in Python, laying the foundation for logical decision-making in programming.

Theoretical Background

Control Statements Overview

Control statements in Python are essential for altering the flow of a program based on certain conditions. They include:

- `if`: Allows the execution of a block of code if a specified condition is true.
- `elif`: Stands for "else if" and is used to check additional conditions if the previous `if` or `elif` conditions are false.
- `else`: Executes a block of code if none of the preceding `if` or `elif` conditions are true.

Syntax

The basic syntax for an `if` statement is as follows:

```
if condition:
    # Code to execute if the condition is true
```

The `elif` and `else` statements can be added as needed:

```
if condition:
    # Code to execute if the condition is true
elif another_condition:
    # Code to execute if the previous conditions are false and this
       condition is true
else:
    # Code to execute if none of the conditions are true
```

Example

```
# Example of control statements
num = 10

if num > 0:
    print("Number is positive")
elif num == 0:
    print("Number is zero")
else:
    print("Number is negative")
```

Programming Exercise

You are tasked with creating a Python program that simulates a simple grading system using control statements. The program should accomplish the following:

1. Prompt the user to enter the student's name and their exam score.
2. Implement a grading system based on the following criteria:
    - If the exam score is greater than or equal to 90, assign the grade 'A'.
    - If the exam score is between 80 and 89 (inclusive), assign the grade 'B'.
    - If the exam score is between 70 and 79 (inclusive), assign the grade 'C'.
    - If the exam score is between 60 and 69 (inclusive), assign the grade 'D'.
    - If the exam score is below 60, assign the grade 'F'.
3. Display the student's name, exam score, and the corresponding grade.
4. Additionally, provide feedback based on the grade:
    - For 'A' grade, print "Excellent! Keep up the good work."
    - For 'B' grade, print "Well done! Continue to strive for excellence."
    - For 'C' grade, print "Good job! Focus on improvement."
    - For 'D' grade, print "Work harder to achieve a better result."
    - For 'F' grade, print "Unfortunately, you did not pass. Seek assistance for improvement."

Write a Python program that accomplishes the above requirements. Include proper control statements (`if`, `elif`, `else`) for grade determination and feedback.

```python
# Python program for grading system
student_name = input("Enter student's name: ")
exam_score = float(input("Enter exam score: "))

# Grading system
if exam_score >= 90:
    grade = 'A'
    feedback = "Excellent! Keep up the good work."
elif 80 <= exam_score <= 89:
    grade = 'B'
    feedback = "Well done! Continue to strive for excellence."
elif 70 <= exam_score <= 79:
    grade = 'C'
    feedback = "Good job! Focus on improvement."
elif 60 <= exam_score <= 69:
    grade = 'D'
    feedback = "Work harder to achieve a better result."
else:
    grade = 'F'
    feedback = "Unfortunately, you did not pass. Seek assistance for
       improvement."

# Display results and feedback
print(f"\nStudent Name: {student_name}")
print(f"Exam Score: {exam_score}")
print(f"Grade: {grade}")
print(f"Feedback: {feedback}")
```

Output:

```
Enter student's name: Ayaan
Enter exam score: 75

Student Name: Ayaan
Exam Score: 75.0
Grade: C
Feedback: Good job! Focus on improvement.
```

Questions

1. **Leap Year Checker:** Write a Python program that takes a year as input and determines whether it is a leap year. Consider the rules for leap years (divisible by 4, but not divisible by 100 unless divisible by 400).

2. **Temperature Converter:** Implement a temperature converter program in Python. Prompt the user to enter a temperature in Celsius and convert it to Fahrenheit. Provide feedback based on whether the temperature is considered hot, moderate, or cold.

3. **Number Classifier:** Create a Python program that classifies a given integer as positive, negative, or zero. Additionally, check if the number is even or odd and provide appropriate feedback.

4. **Vowel or Consonant:** Write a Python program that takes a single character as input and determines whether it is a vowel or a consonant. Handle both uppercase and lowercase characters.

5. **Simple Calculator:** Develop a simple calculator program in Python that takes two numbers and an operator (+, -, *, /) as input and performs the corresponding operation. Handle division by zero gracefully.

6. **Discount Calculator:** Create a Python program that calculates the final price after applying a discount. Prompt the user to enter the original price and the discount percentage. Display the discounted price.

7. **Guess the Number:** Implement a number guessing game in Python. Generate a random number between 1 and 100 and ask the user to guess the number. Provide hints (higher/lower) until the correct number is guessed.

8. **Triangle Classifier:** Write a Python program that takes three sides of a triangle as input and classifies it as equilateral, isosceles, or scalene. Handle cases where the input sides cannot form a valid triangle.

9. **BMI Calculator:** Develop a Python program that calculates the Body Mass Index (BMI) based on the user's weight (in kilograms) and height (in meters). Provide feedback on whether the user is underweight, normal weight, overweight, or obese.

10. **Time Converter:** Create a time converter program in Python that converts seconds into hours, minutes, and seconds. Prompt the user to enter the total seconds and display the converted time.