SOFT40161_Lab03_Solution

October 14, 2024

1 SOFT40161 - Introduction to Computer Programming: Lab 03

Try to buid up the concepts! Don't just execute the codes without knowing it's meaning.

1.1 Data Structure and Error Handling in Jupyter Notebook

Welcome to this week's lab session! Today, we will explore data structures—the backbone of organising data in Python and the basics of error handling to make your code more robust and reliable.

We'll be working with lists, tuples, sets, and dictionaries, learning how to manipulate and utilise these powerful structures. Additionally, you'll discover how to handle common errors gracefully using Python's built-in tools, ensuring your programs can run smoothly even when something goes wrong.

Let's get hands-on, explore, and solidify these key concepts!

1.2 Lab Learning Outcomes

By the end of this lab, you will be able to: - Work with Python's data structures (lists, tuples, sets, and dictionaries). - Understand how to access, modify, and iterate through these data types. - Apply best practices for code readability and maintainability. - Implement error handling techniques to improve code robustness.

1.2.1 Example 1: Lists in Python

A list is a mutable collection of ordered items. You can add, modify, or remove elements from a list.

[ ]:
```python
# Creating a list
fruits = ["apple", "banana", "cherry"]

# Accessing list elements
print(fruits[0])  # Output: apple

# Modifying a list element
fruits[1] = "blueberry"
print(fruits)  # Output: ['apple', 'blueberry', 'cherry']

# Adding an item to the list
fruits.append("orange")
print(fruits)  # Output: ['apple', 'blueberry', 'cherry', 'orange']
```

[ ]:
```python

```

Exercise 1: List Operations

Create a list of five integers. Write code that: - Adds two more numbers to the list. - Removes the last number. - Prints the sum of all the numbers in the list.

Exercise 1: Solution

[ ]:
```python
# Creating a list of five integers
numbers = [10, 20, 30, 40, 50]

# Adding two more numbers
numbers.append(60)
numbers.append(70)

# Removing the last number
numbers.pop()

# Calculating the sum of the numbers
total_sum = sum(numbers)

# Output the list and total sum
print("Updated list:", numbers)
print("Sum of numbers:", total_sum)
```

1.2.2 Example 2: Tuples in Python

A tuple is an immutable collection, meaning you cannot change its contents after creation. Tuples are often used when you want a group of values to remain constant throughout the program.

[ ]:
```python
# Example: Storing geographic coordinates

# Creating a tuple for coordinates (latitude, longitude)
coordinates = (40.7128, -74.0060)
```

```python
# Accessing elements in the tuple
print("Latitude:", coordinates[0])  # Output: 40.7128
print("Longitude:", coordinates[1])  # Output: -74.0060

# Trying to modify a tuple element (this will raise an error)
# coordinates[0] = 41.0  # This would cause a TypeError

# Using tuples in a list
city_coordinates = [
    ("New York", (40.7128, -74.0060)),
    ("Los Angeles", (34.0522, -118.2437)),
    ("Chicago", (41.8781, -87.6298))
]

# Accessing tuple elements within a list
for city, coord in city_coordinates:
    print(f"{city} is located at {coord[0]} latitude and {coord[1]} longitude.")
```

[ ]:

Exercise 2: Tuple Operations

Create a tuple with three colors. - Print the colors one by one using a for loop. - Try to modify one element and see what error you get. - Create a list of countries with their capital coordinates stored as tuples. Some examples are- The United States (38.8977, -77.0365), France (48.8566, 2.3522) and Japan (35.682839, 139.759455)

[ ]:

[ ]:
```python
# Step 1: Create a tuple with three colors
colors = ('red', 'green', 'blue')

# Step 2: Print the colors one by one using a for loop
print("Colors in the tuple:")
for color in colors:
    print(color)

# Step 3: Try to modify one element (will cause an error)
try:
    colors[0] = 'yellow'  # Attempt to modify the tuple
except TypeError as e:
    print(f"Error: {e}")  # Output the error message

# Step 4: Create a list of countries with their capital coordinates stored as␣
 ↪tuples
countries = [
    ('United States', (38.8977, -77.0365)),  # Washington D.C. coordinates
    ('France', (48.8566, 2.3522)),           # Paris coordinates
    ('Japan', (35.682839, 139.759455))       # Tokyo coordinates
]

# Print the list of countries with capital coordinates
print("\nCountries and their capital coordinates:")
for country, coordinates in countries:
    print(f"{country}: {coordinates}")
```

Explanation: - Colors Tuple: A tuple colors is created with three color values: 'red', 'green', and 'blue'. We print each color using a for loop. - Tuple Immutability: Attempting to change a tuple element will raise a TypeError, which we handle using a try-except block. - List of Countries: A list of countries is created where each country's name is paired with its capital's coordinates stored as a tuple.

1.2.3 Example 3: Sets in Python

A set is an unordered collection of unique items. You cannot have duplicate items in a set, and the order of items is not preserved.

[ ]:
```
# Example: Managing a set of unique numbers

# Creating a set
unique_numbers = {1, 2, 3, 4, 5, 5, 6}

# Adding and removing elements
unique_numbers.add(7)  # Adding a new number
unique_numbers.remove(3)  # Removing an existing number

print("Updated set of numbers:", unique_numbers)  # Output: {1, 2, 4, 5, 6, 7}

# Checking membership in the set
if 4 in unique_numbers:
    print("Number 4 is in the set")

# Set operations
even_numbers = {2, 4, 6, 8}
common_numbers = unique_numbers.intersection(even_numbers)  # Numbers that are␣
 ↪in both sets
print("Common numbers between two sets:", common_numbers)  # Output: {2, 4, 6}
```

Exercise 3: Set Operations

Create a set of odd numbers from 1 to 10. - Add a new odd number to the set. - Remove a number from the set. - Find the common elements between your set and a set of even numbers.

[ ]:
```
# Step 1: Create a set of odd numbers from 1 to 10
odd_numbers = {1, 3, 5, 7, 9}
```

```python
print("Initial set of odd numbers:", odd_numbers)

# Step 2: Add a new odd number to the set
odd_numbers.add(11)
print("Set after adding 11:", odd_numbers)

# Step 3: Remove a number from the set
odd_numbers.remove(9)
print("Set after removing 9:", odd_numbers)

# Step 4: Find the common elements between the odd number set and a set of even␣
 ↪numbers
even_numbers = {2, 4, 6, 8, 10}
common_elements = odd_numbers.intersection(even_numbers)
print("Common elements between odd and even sets:", common_elements)
```

Explanation: - Step 1: We create a set odd_numbers containing the odd numbers from 1 to 10: {1, 3, 5, 7, 9}. - Step 2: We add the number 11 to the set using the .add() method. Since sets do not allow duplicates, this method only adds unique elements. - Step 3: We remove the number 9 using the .remove() method. - Step 4: We find the common elements between the odd_numbers set and another set of even numbers {2, 4, 6, 8, 10} using the .intersection() method. In this case, there are no common elements, so the result is an empty set.

This exercise helps illustrate how to perform basic set operations like adding, removing, and finding intersections between sets.

1.2.4 Example 4: Dictionaries in Python

A dictionary is a collection of key-value pairs. Each key is unique, and you can access values by using their corresponding keys.

[ ]:
```python
# Example: Storing student details

# Creating a dictionary to store student information
student = {
    "name": "Alice",
    "age": 21,
    "major": "Computer Science"
}

# Accessing dictionary elements
print("Student Name:", student["name"])  # Output: Alice
print("Student Age:", student["age"])  # Output: 21

# Modifying a value
student["age"] = 22
print("Updated Age:", student["age"])  # Output: 22
```

```
# Adding a new key-value pair
student["GPA"] = 3.8
print("Updated Student Info:", student)  # Output: {'name': 'Alice', 'age': 22,␣
  ↪'major': 'Computer Science', 'GPA': 3.8}

# Looping through a dictionary
print("Student Details:")
for key, value in student.items():
    print(f"{key}: {value}")
```

Exercise 4: Dictionary Operations

Create a dictionary to store details of a book (title, author, and year). - Update the year of publication. - Add a new key-value pair for "genre". - Loop through the dictionary and print all key-value pairs.

```
[ ]: # Step 1: Create a dictionary to store details of a book
book_details = {
    "title": "To Kill a Mockingbird",
    "author": "Harper Lee",
    "year": 1960
}

print("Initial book details:", book_details)

# Step 2: Update the year of publication
book_details["year"] = 1961
print("Updated year of publication:", book_details)

# Step 3: Add a new key-value pair for "genre"
book_details["genre"] = "Fiction"
print("Updated book details with genre:", book_details)

# Step 4: Loop through the dictionary and print all key-value pairs
print("\nAll book details:")
for key, value in book_details.items():
    print(f"{key}: {value}")
```

Explanation: - Step 1: We create a dictionary book_details to store details of a book, which includes the title, author, and year of publication. - Step 2: We update the year of publication by modifying the value associated with the "year" key. - Step 3: We add a new key-value pair for the genre using book_details["genre"] = "Fiction". - Step 4: We use a for loop to iterate through the dictionary and print each key-value pair using the .items() method.

1.2.5 Error Handling Section

Errors can occur at any point in a program. To avoid program crashes, you can use error handling techniques such as try-except blocks.

[ ]:
```python
# Example: Handling invalid input with try-except

try:
    user_input = int(input("Enter a number: "))
    print(f"You entered: {user_input}")
except ValueError:
    print("Please enter a valid number.")

# Handling ZeroDivisionError
try:
    numerator = int(input("Enter numerator: "))
    denominator = int(input("Enter denominator: "))
    result = numerator / denominator
    print(f"Result: {result}")
except ZeroDivisionError:
    print("You cannot divide by zero!")
```

[ ]:
```python
numerator = int(input("Enter numerator: "))
denominator = int(input("Enter denominator: "))
result = numerator / denominator
print(f"Result: {result}")
```

Exercise 5: Error Handling

Write a program that prompts the user for two numbers and divides them. - Handle cases where the input is invalid or division by zero is attempted. - Print a relevant error message for each exception.

Solution Exercise 5: Error Handling

[ ]:
```python
# Step 1: Input two numbers from the user
try:
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    # Step 2: Perform division
    result = num1 / num2
    print(f"Result: {num1} divided by {num2} is {result}")

# Step 3: Handle division by zero
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")

# Step 4: Handle invalid input (e.g., entering a non-numeric value)
except ValueError:
    print("Error: Please enter valid numbers.")

# Step 5: Handle any other unexpected errors
```

```
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Explanation: - Step 1: The program prompts the user to input two numbers using input(). These are converted to floats using float() for division. - Step 2: The program attempts to divide the two numbers and prints the result. - Step 3: The ZeroDivisionError is handled with an error message if the second number is zero. - Step 4: The ValueError is handled to catch invalid inputs (e.g., if the user enters non-numeric values). - Step 5: A generic Exception is included to catch any other unexpected errors and print the error message.

2 Challenge Exercise: Student Grade Management System

Task: Create a simple student grade management system that allows users to add, update, and display student grades. The program should also calculate the average grade for the students.

Requirements: Data Structure: Use a dictionary to store student names as keys and their grades (as a list of integers) as values.

Functionality: - Add Student: Prompt the user for a student's name and their grade, then add this information to the dictionary. - Update Grade: Prompt the user for a student's name and a new grade, then update the grade for that student. - Display Grades: Print all student names along with their grades. - Calculate Average: Calculate and print the average grade for all students.

Input Handling: - Ensure that grades are valid integers between 0 and 100. - Handle cases where a student does not exist when updating grades.

Sample I/O:

Welcome to the Student Grade Management System!

1. Add Student
2. Update Grade
3. Display Grades
4. Calculate Average
5. Exit

Choose an option: 1 Enter student's name: Alice Enter grade for Alice (0-100): 85

Student added successfully!

1. Add Student
2. Update Grade
3. Display Grades
4. Calculate Average
5. Exit

Choose an option: 3

Current Grades: Alice: [85]

1. Add Student
2. Update Grade
3. Display Grades

4. Calculate Average
5. Exit

Choose an option: 4

Average Grade: 85.0

2.0.1 Solution:

[ ]:
```python
# Student Grade Management System

# Initialize an empty dictionary to store student grades
student_grades = {}

while True:
    print("\nWelcome to the Student Grade Management System!")
    print("1. Add Student")
    print("2. Update Grade")
    print("3. Display Grades")
    print("4. Calculate Average")
    print("5. Exit")

    choice = input("Choose an option: ")

    if choice == '1':
        name = input("Enter student's name: ")
        try:
            grade = int(input("Enter grade for {} (0-100): ".format(name)))
            if 0 <= grade <= 100:
                # Add student and their grade
                if name in student_grades:
                    student_grades[name].append(grade)
                else:
                    student_grades[name] = [grade]
                print(f"Student {name} added successfully!")
            else:
                print("Error: Grade must be between 0 and 100.")
        except ValueError:
            print("Error: Please enter a valid integer for the grade.")

    elif choice == '2':
        name = input("Enter student's name: ")
        try:
            new_grade = int(input("Enter new grade for {}: ".format(name)))
            if 0 <= new_grade <= 100:
                # Update grade for existing student
                if name in student_grades:
                    student_grades[name].append(new_grade)
                    print(f"Updated {name}'s grade to {new_grade}.")
                else:
                    print(f"Error: Student {name} does not exist.")
            else:
                print("Error: Grade must be between 0 and 100.")
        except ValueError:
            print("Error: Please enter a valid integer for the grade.")

    elif choice == '3':
        # Display all student grades
        print("Current Grades:")
        for name, grades in student_grades.items():
            print(f"{name}: {grades}")

    elif choice == '4':
        # Calculate and display the average grade
        if student_grades:
            total_grades = 0
            total_students = 0
            for grades in student_grades.values():
                total_grades += sum(grades)
                total_students += len(grades)
            average = total_grades / total_students
            print(f"Average Grade: {average:.2f}")
        else:
            print("No grades available to calculate the average.")

    elif choice == '5':
        print("Exiting the program. Goodbye!")
        break

    else:
        print("Error: Invalid choice. Please select a valid option.")
```