SOFT40161_Lab03

October 14, 2024

Nottingham Trent University

Department of Computer Science

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

```

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)

# 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)

Exercise 2: Solution

[ ]:

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.

Exercise 3: Solution

[ ]:

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.

[ ]:
```
# 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.

Exercise 4: Solution

[ ]:

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.

Exercise 5: Solution

[ ]:
```python

```

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:

[ ]: