**Task No. 1:** Objective: Familiarize yourself with the immutable nature of certain data types. Instructions:

* Create a string variable with the value "Hello".
* Write a function to attempt to change the first character of the string to "J".
* Observe and note the error message you receive.

**Solution:**

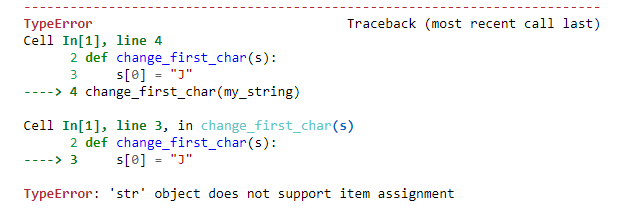
my\_string = "Hello"

def change\_first\_char(s):

s[0] = "J"

change\_first\_char(my\_string)

**Output:**

****

**Task No. 2:** Objective: Understand the mutable nature of lists. Instructions:

* Create a list of integers.
* Write a function that accepts this list as an argument, appends a new integer to it, and returns the modified list.
* After calling the function, print the original list to observe the changes. Discuss why the original list was modified outside of the function.

**Solution:**

my\_list = [1, 2, 3, 4, 5]

def append\_to\_list(lst, new\_item):

modified\_list = lst.copy() # Create a copy of the original list

modified\_list.append(new\_item) # Append the new item to the copy

return modified\_list

modified\_list = append\_to\_list(my\_list, 6)

print("Modified List:", modified\_list)

print("Original List:", my\_list)

**Output:**

****

**Task No. 3:** Grasp the concept that while tuples are immutable, their elements can be mutable. Instructions:

• Create a tuple containing a list.

• Attempt to modify the list inside the tuple (e.g., append a new element).

• Print the tuple to see the changes. Discuss your findings.

**Solution:**

my\_tuple = ([1, 2, 3], 'immutable\_element')

my\_tuple[0].append(4)

print("Modified Tuple:", my\_tuple)

**Output:**

****

**Task No. 4:** Understand the differences between a deep copy and a shallow copy with mutable types. Instructions:

* Create a list containing sub-lists.
* Make a shallow copy of this list.
* Modify the sub-list in the original list and observe the changes in both the original and the copied list.
* Now, make a deep copy of the list and repeat the modification. Observe the changes In the deep-copied list.

**Solution:**

import copy

original\_list = [[1, 2, 3], [4, 5, 6]]

shallow\_copy = copy.copy(original\_list)

original\_list[0].append(4)

print("Original List:", original\_list)

print("Shallow Copy:", shallow\_copy)

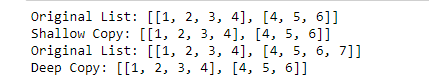
deep\_copy = copy.deepcopy(original\_list)

original\_list[1].append(7)

print("Original List:", original\_list)

print("Deep Copy:", deep\_copy)

**Output:**

****

**Task No. 5:** Recognize the pitfalls of using mutable default arguments in functions. Instructions:

* Create a function that takes a list as an argument with a default value of an empty list.
* In the function, append a new Item to the list every time the function is called.
* Call the function multiple times without providing a list argument and print the list after each call. Observe and discuss the behavior.

**Solution:**

def append\_to\_list(item, my\_list=[]):

my\_list.append(item)

return my\_list

print("Call 1:", append\_to\_list(1))

print("Call 2:", append\_to\_list(2))

print("Call 3:", append\_to\_list(3))

**Output:**

****

You might expect each call to return a list with only the item appended in that particular call. However, the default argument “my\_list=[]” is evaluated only once when the function is defined. As a result, all the calls to the function use the same default list, which accumulates items across multiple function calls. This behavior is generally not what you'd expect or want.

To avoid this pitfall, you should use an immutable default argument or explicitly create a new mutable object within the function to ensure that you're working with a fresh instance every time. Here's an updated version of the function to avoid this issue:

def append\_to\_list(item, my\_list=None):

if my\_list is None:

my\_list = []

my\_list.append(item)

return my\_list

print("Call 1:", append\_to\_list(1))

print("Call 2:", append\_to\_list(2))

print("Call 3:", append\_to\_list(3))

**Output:**

**Task No. 6:** Create a python application of "Book Management System" to implement mutability and immutability in real world application and paste screenshot of the output:

**Features**

* **Add Book:** Users can add a book with details like title, author, and publication year.
* **List Books:** Users can view the list of all books.
* **Borrow Book:** Users can mark a book as borrowed.
* **Return Book:** Users can mark a borrowed book as returned.
* **Search Book:** Users can search for a book by its title or author.

**Data Structures:**

**'books\_ list':** A mutable list to store books\_ Each book is a dictionary with details like "id', 'title", "author', "year', and "borrowed'.

**' Porrowed\_books\_archive’:** An immutable tuple to archive books that have been borrowed and returned.

**Solution:**

books\_list = []

borrowed\_books\_archive = ()

def add\_book(title, author, year):

book\_id = len(books\_list) + 1

book = {

'id': book\_id,

'title': title,

'author': author,

'year': year,

'borrowed': False

}

books\_list.append(book)

print(f"Book '{title}' added to the library.")

def list\_books():

print("List of Books:")

for book in books\_list:

print(f"ID: {book['id']}, Title: {book['title']}, Author: {book['author']}, Year: {book['year']}, Borrowed: {book['borrowed']}")

def borrow\_book(book\_id):

for book in books\_list:

if book['id'] == book\_id:

if not book['borrowed']:

book['borrowed'] = True

print(f"Book '{book['title']}' has been borrowed.")

else:

print(f"Book '{book['title']}' is already borrowed.")

return

print("Book not found.")

def return\_book(book\_id):

for book in books\_list:

if book['id'] == book\_id:

if book['borrowed']:

book['borrowed'] = False

print(f"Book '{book['title']}' has been returned.")

global borrowed\_books\_archive

borrowed\_books\_archive += (book,)

else:

print(f"Book '{book['title']}' is not borrowed.")

return

print("Book not found.")

def search\_book(query):

result = [book for book in books\_list if query in book['title'] or query in book['author']]

if result:

print("Search Results:")

for book in result:

print(f"ID: {book['id']}, Title: {book['title']}, Author: {book['author']}, Year: {book['year']}, Borrowed: {book['borrowed']}")

else:

print("No matching books found.")

add\_book("The Great Gatsby", "F. Scott Fitzgerald", 1925)

add\_book("To Kill a Mockingbird", "Harper Lee", 1960)

add\_book("1984", "George Orwell", 1949)

list\_books()

borrow\_book(1)

borrow\_book(2)

return\_book(2)

list\_books()

search\_book("George Orwell")

**Output:**

