Debugging and Exceptions

Solve the following exercises and upload your solutions to Moodle until the specified due date.

Important Information!

Please try to *exactly match the output* given in the examples (naturally, the input can be different). We are running automated tests to aid in the correction and grading process, and deviations from the specified output lead to a significant organizational overhead, which we cannot handle in the majority of the cases due to the high number of submissions.

Make sure to use the *exact filenames* that are specified for each individual exercise. Also, use the provided unit tests to check your scripts before submission (see the slides Handing in Assignments on Moodle).

It is of *particular importance* in this assignment to wrap the printing that you see in the example outputs in `if __name__=='__main__':`, as your exercises essentially only consist of a function definition. Example - let's say the task is to write a function that doubles the float value that is passed:

```
def double(var: float) -> float:
    return var*2

if __name__ == '__main__':
    print(double(3.4))
```

Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are *not allowed* to use any concepts and modules that have not yet been presented in the lecture.

Feel free to copy the example text from the assignment sheet, and then change it according to the exercise task.

Exercise 1 – Submission: `a5_ex1.py` 25 Points

In this exercise, you will debug a provided Python function called `analyze_and_update_collection`. The function takes a list of integers, `my_list`, and an optional set of integers, `my_set`. The function checks certain conditions, adds the mean of `my_list` to `my_set`, and returns the updated set. However, the provided code in `a5_ex1_buggy.py` contains several bugs you need to fix so that it produces the expected output (see below).

Hints

- You can check if an object is of a certain data type with `isinstance(OBJECT, TYPE)`.
- Use `assert` statements to check the validity of some the inputs.

Example usage

Example execution of the program:

```
items1 = [1,2,3,4,5]
items2 = [2,4,6]

s = analyze_and_update_collection(items1)
print('Current set:', s)
s = analyze_and_update_collection(items2)
print('Current set:', s)
s = analyze_and_update_collection(items1, my_set=set(items1))
print('Current set:', s)
s = analyze_and_update_collection(items1, my_set=set(items2))
print('Current set:', s)

try:
    s = analyze_and_update_collection([])
except AssertionError as e:
    print(e)

try:
    s = analyze_and_update_collection([str(i) for i in items1])
except AssertionError as e:
    print(e)
```

Output:

```
The last element of my_list is 5
Current set: {3}
The last element of my_list is 6
Current set: {4}
The last element of my_list is 5
my_set and my_list contain the same elements
Current set: {1, 2, 3, 4, 5}
The last element of my_list is 5
Current set: {2, 3, 4, 6}
Aborted as my_list must not be empty
Aborted as my_list contains non integer values
```

Exercise 2 – Submission: `a5_ex2.py` 25 Points

Write a function `safe_lookup(d, keys, expected_type)` that performs a safe lookup in a nested dictionary and checks whether the final value matches the expected type. The function should:

- Accept three arguments:
  - `d`: A nested dictionary.
  - `keys`: A list of keys to navigate through the nested dictionary.
  - `expected_type`: The expected type of the final value.
- Traverse the dictionary using the list of keys. If a `KeyError` occurs because a key is not found in the (sub)dictionary, the function should return the string `"Key not found"`.
- After traversing the dictionary successfully, check whether the type of the final value matches `expected_type`. If not, raise a `TypeError` with the message `"Expected type <expected_type>, but got <actual_value_type>"` where `<expected_type>` and `<actual_value_type>` are the names of the expected type and the name of the actual value of the final value respectively.
- If everything is correct, return the value.

Hints

- You can check if an object is of a certain data type with `isinstance(OBJECT, TYPE)`.
- Check Unit 1 to find out how to extract the name of a type.

Example usage

Example execution of the program:

```
nested_dict = {"level1" : {"level2" : {"key" : "value"}}}
print(safe_lookup(nested_dict, ["level1", "level2", "key"] , str))
```

Output: `value`

Example execution of the program:

```
nested_dict = {"level1" : {"level2" : {"key" : "value"}}}
print(safe_lookup(nested_dict, ["level1", "wrong_key"] , str))
```

Output: `Key not found`

Example execution of the program:

```
nested_dict = {"level1" : {"level2" : {"key" : "value"}}}
try:
    safe_lookup(nested_dict, ["level1", "level2"] , list)
except TypeError as e:
    print(e)
```

Output: `Expected type list, but got dict`

Exercise 3 – Submission: `a5_ex3.py` 25 Points

Write a function `safe_list_access(lst, index)` that safely accesses elements of a list.
The function should:

- Attempt to access the element at the given `index` in the list `lst`.
- If a `TypeError` occurs because `lst` is not a list, return the message `"First argument is not a list"`.
- If a `TypeError` occurs because `index` is not an integer, print `"Converting Index to integer"` and attempt to convert it to an integer. If the conversion fails (raising a `ValueError`), return `"Index cannot be converted to an integer"`.
- If an `IndexError` occurs because the `index` is out of range, return the message `"Index out of range"`.
- Use a `finally` block to print `"Operation completed"` after each access attempt.
- If the attempt was successful, return the element.

Hints:

- You can check if an object is of a certain data type with `isinstance(OBJECT, TYPE)`.

Example usage

Example execution of the program:

```
print(safe_list_access(3,1))
```

Output:

```
Operation completed
First argument is not a list
```

Example execution of the program:

```
numbers = [10, 20, 30, 40, 50]
print(safe_list_access(numbers,1))
```

Output:

```
Operation completed
20
```

Example execution of the program:

```
numbers = [10, 20, 30, 40, 50]
print(safe_list_access(numbers,'1'))
```

Output:

```
Converting Index to integer
Operation completed
20
```

Example execution of the program:

```
numbers = [10, 20, 30, 40, 50]
print(safe_list_access(numbers,'abc'))
```

Output:

```
Converting Index to integer
Operation completed
Index cannot be converted to an integer
```

Example execution of the program:

```
numbers = [10, 20, 30, 40, 50]
print(safe_list_access(numbers,5))
```

Output:

```
Operation completed
Index out of range
```

Example execution of the program:

```
numbers = [10, 20, 30, 40, 50]
print(safe_list_access(numbers,'5'))
```

Output:

```
Converting Index to integer
Operation completed
Index out of range
```

Exercise 4 – Submission: `a5_ex4.txt` 25 Points

Consider the following code with custom exceptions `ErrorX`, `ErrorY` and `ErrorZ` which are all independent, i.e. none is a special case of another:

```
def f(x: int):
    try:
        g(x)
        print("f1")
    except ErrorX:
        print("f2")
    except ErrorZ:
        print("f3")
    finally:
        print("f4")

def g(x: int):
    try:
        h(x)
        print("g1")
    except ErrorY:
        print("g2")
        if x < -10:
            raise ErrorZ
        else:
            print("g3")
        print("g4")
    except ErrorX:
        print("g5")
        raise
    finally:
        print("g6")

def h(x: int):
    try:
        if x > 15:
            raise ErrorX
        elif x == 0:
            raise ErrorY
        elif x < -5:
            raise ErrorZ
    finally:
        print("h1")
    print("h2")
```

To understand the program flow, determine the output of the function `f` with the following four arguments without running the code: `f(0)`, `f(7)`, `f(16)`, `f(-12)`. Write your answers to the text file `a5_ex4.txt` in the following format (one line per answer):

```
f(ARG) -> X1 X2 ... Xn
```

where `ARG` is one of the four input arguments from above and `Xi` are either space-separated print outputs or the error in case the function call ends with an error.

Example file content (the results are incorrect, this is just for demonstrating the format):

```
f(0) -> h1 g2 g3 g4 f1 f4
f(7) -> h1 h2 g1 f1 f4
f(16) -> h1 g5 f2 f4
f(-12) -> h1 ErrorZ f4
```