Recursion, Generators and Exceptions

Solve the following exercises and upload your solutions to Moodle until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. 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.

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

Write a function `sub_summarize(nested: list, sub_sums: list) -> int` that calculates the sum of the input list *nested* and sums of sub lists arbitrarily nested in the input list (you can assume correct arguments). The sums are stored in the list *sub_sums*. Use recursion to implement this function.

Example function calls and results:

```
nested = [1, 2, 3, [4, [5, 6], 7], 8, [9, 10]]
sub_sums = []
sub_summarize(nested, sub_sums)
sub_sums = [11, 22, 19, 55]
```

Hints:

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

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

Write a function `print_directory(dir_path: str)` that enumerates and prints recursively all files and sub directories in an input directory specified by its path *dir_path*. The function should do the following:

- If *dir_path* is a path to a file, print "*dir_path* is a file not a directory".
- If *dir_path* is a path to a directory, enumerate and print recursively the input directory and all its files and sub directories with a hierarchical format as the below example for the accompanied directory *d0*. Use recursion to implement this functionality as the below hints.
- Else print "*dir_path* is invalid".

```
path_to_the_directory_d0
    d0.1
        d0.1.1
        f0.1.1.txt
        f0.1.2.txt
    d0.2
        d0.2.1
            d0.2.1.1
                f0.2.1.1.1.txt
            f0.2.1.1.txt
        f0.2.1.txt
        f0.2.2.txt
    f0.1.txt
    f0.2.txt
```

With the hierarchical format, files and sub directories are indented from its parent directory a tab character. Use base name for files and sub directories but (absolute or relative) path for the root (input) directory. You are *not allowed* to use built-in functions to recursively list files and sub directories, except the following basic functions:

- `os.path.isfile` to check if a path points to a file.
- `os.path.isdir` to check if a path point to a directory.
- `os.path.basename` to get base name from a path.
- `os.listdir` to list files and sub directories in a directory.
- `os.path.join` to make a path from path components with directory separators.

Hints:

- You can implement a recursive function `print_directory_recursively(dir_path: str, level: int)` for the case *dir_ path* is a path to a directory. *level* indicates the depth of a (sub) directory.

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

Write a generator function `gen_fibonacci(upper_bound)` that yields Fibonacci numbers which are not greater than `upper_bound` (can be included). In addition, your function should do the following:

- If `upper_bound` is neither an integer nor a float, raise a `TypeError`.
- If `upper_bound` < 0, raise a `ValueError`.

The Fibonacci sequence is defined as follows:

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

Example function calls and results:

```
list(gen_fibonacci("3")) -> TypeError
list(gen_fibonacci(-1)) -> ValueError
list(gen_fibonacci(0)) -> [0]
list(gen_fibonacci(1)) -> [0, 1, 1]
list(gen_fibonacci(3)) -> [0, 1, 1, 2, 3]
list(gen_fibonacci(9.2)) -> [0, 1, 1, 2, 3, 5, 8]
```

Hints:

- Create appropriate and useful error/exception messages.
- You can check if some object is of a certain data type with `isinstance(OBJECT, TYPE)`.

Exercise 4 – Submission: a5_ex4.txt 25 Points

Consider the following code with custom exceptions ErrorA, ErrorB and ErrorC (they are all independent, i.e., none of them is a special case of another one):

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

def g(x: int):
    try:
        h(x)
        print("g1")
    except ErrorA:
        print("g2")
    except ErrorB:
        print("g3")
        if x < -10:
            raise ErrorC
            print("g4")
        else:
            print("g5")
        print("g6")

def h(x: int):
    try:
        if x > 10:
            raise ErrorA
        if x < 0:
            raise ErrorB
    finally:
        print("h1")
    print("h2")
```

Determine the output of the function f with the following four arguments without actually running the code (the goal is to understand the program flow): f(5), f(-5), f(11), f(-11). 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. Here is an example file content (the examples are incorrect, they are just for demonstrating purposes!):

```
f(5) -> f1 f2 g1 h1
f(-5) -> f3 h2 ErrorB
f(11) -> h1 h2 f1 f5 g2
f(-11) -> g1 h2 f2 ErrorA
```