IE5600 – Applied Programming for Industrial Systems

AY 2023/24 Semester 2

Practical Lab 02 – Suggested Solution

Section A – Tutorial Discussion

Question A1 – Elegant and Efficient Usage of If-Else Statement

1. Write concise and clean if-else statement by using i) math logic and relational operators to streamline Boolean expressions; ii) list of values and membership operators when comparing multiple values; and iii) helper functions.

   Using membership operator:

```
num1 = int(input('Input an integer: '))

if num1 in [1,3,5,7]:
    print('num1 is odd')
elif num1 in [2,4,6,8]:
    print('num1 is even')
else:
    print('num1 is outside valid range')
```

   Using helper functions (and math logic of modulo 2) instead of hardcoding all the values in the list:

```
def isOdd(num):
    return True if num < 100 and num % 2 == 1 else False

def isEven(num):
    return True if num < 100 and num % 2 == 0 else False

num1 = int(input('Input an integer: '))

if isOdd(num1):
    print('num1 is odd')
elif isEven(num1):
    print('num1 is even')
else:
    print('num1 is outside valid range')
```

List is a Python data structure that we would be discussing in Week 05. Functions would be discussed in Week 06.

2. Optimize the order of conditions when having multiple conditions. Consider placing the most likely-to-fail conditions first. Using this approach, if an early condition evaluates to `False`, the subsequent conditions will not be checked thus saving time. This is especially useful if the earlier conditions take longer to execute.

   In the helper functions of (1), the value range check `num < 100` will likely fail more often.

3. Use the shorthand ternary `if` operator only for writing simple value assignment. For multiple branching, the standard if statement is actually more readable.

4. Avoid the use of multiple `if` statements by replacing them with `if-elif` statements for more efficient evaluation. The former requires the interpreter to always check the Boolean expressions of all `if` statements. The latter will stop after the first Boolean expression evaluates to `True`, i.e., skip over the remaining Boolean expressions.

5. Avoid redundant comparisons by simplifying or eliminating unnecessary comparisons. Refer to Week 03 lecture note slide 38 to 40.