CSL100 PracticeSheet-5,6

Instructions

- For each question below, create a Python file named <YourID>_<questionID>.py, for example B25MM777_q1.py.
- Each file must contain the specified function signature.
- You may write additional helper functions if needed, but the main function signature must remain unchanged.
- Your code will be tested against hidden test cases, so follow the inputoutput specifications precisely.

Mandatory Submission Rules

- **Do not read input from the user.** Implement the specified functions and return values directly. No calls to input() should appear in your code.
- **Return the result as specified. Do not print inside your functions.** Only print when explicitly instructed; otherwise rely on return values.
- Use the exact function names and parameter orders provided in the problem statements.
- Follow the file naming convention exactly: <YourID>_<questionID>.py. Examples of incorrect names include <yourid>.q1.py, <yourid>_q1.py.py, etc.
- Keep each question self-contained in its own file (no cross-imports between questions).
- Include concise docstrings and comments where necessary to explain your logic.

Practice Questions

Q1. Cost of Coffee (Easy)
Filename: <YourID>_q1.py
Function:

```
def getCostOfCoffee(numberOfCoffees, pricePerCoffee):
    """
    Calculate the total cost of coffee orders where for every
   eight coffees you buy,
    you get one additional coffee free (i.e., nine coffees in
   total per block, but
    you only pay for eight). Return the total cost as a float.
    """
    pass
```

Input: numberOfCoffees: int, pricePerCoffee: float
Output: float
Test Cases:

- getCostOfCoffee(8, 2.50) # 20.0
- getCostOfCoffee(9, 2.50) # 20.0
- getCostOfCoffee(10, 2.50) # 22.5
- getCostOfCoffee(18, 2.50) # 40.0

Q2. Sort Digits (Medium)
Filename: <YourID>_q2.py
Function:

```
def sort_digits(n):
    """
    Return an integer whose decimal digits are sorted in
   ascending order.
    Leading zeros are naturally dropped when converting the
   sorted string back
    to an integer.
    """
    pass
```

Input: n: int
Output: int
Test Cases:

- sort_digits(4312) # 1234
- sort_digits(989) # 899
- sort_digits(10) # 1

Q3. Valid Date Check (Hard)
Filename: <YourID>_q3.py
Functions:

```
def isLeapYear(year):
    """
    Return True if 'year' is a leap year according to Gregorian
   rules, otherwise False.
    """
```

```
    pass

def isValidDate(year, month, day):
    """
    Return True if 'year', 'month', and 'day' form a valid date
   on the Gregorian calendar.
    Month must be 1..12 and day must be valid for that month
   (including leap-year February).
    """
    pass
```

Input: year: int, month: int, day: int
Output: bool
Test Cases:

- isValidDate(1999, 12, 31) # True
- isValidDate(2000, 2, 29) # True
- isValidDate(2001, 2, 29) # False
- isValidDate(2029, 13, 1) # False
- isValidDate(1000000, 1, 1) # True
- isValidDate(2015, 4, 31) # False
- isValidDate(1970, 5, 99) # False
- isValidDate(1981, 0, 3) # False
- isValidDate(1666, 4, 0) # False

Q4. Custom Boolean Function (Medium)
Filename: <YourID>_q4.py
Function:

```
def custom_bool(value):
    """
    Emulate Python's built-in bool() without calling it. Return
   False for None,
    numeric zeros (e.g. 0, 0.0), and empty sequences or
   collections (e.g. '', []),
    and True otherwise.
    """
    pass
```

Input: value: any
Output: bool
Test Cases:

- custom_bool(0) # False
- custom_bool(1) # True
- custom_bool("") # False
- custom_bool("hello") # True

- custom_bool([]) # False
- custom_bool([1]) # True
- custom_bool(None) # False

Q5. Largest Unique Subarray (Medium)
Filename: <YourID>_q5.py
Function:

```
def largest_unique_subarray(nums):
    """
    Return the length of the longest contiguous subarray of
   'nums' in which all
    elements are distinct.
    """
    pass
```

Input: nums: list[int]
Output: int
Test Cases:

- largest_unique_subarray([5, 1, 3, 5, 2, 3, 4, 1]) # 5
- largest_unique_subarray([1, 2, 3, 4]) # 4
- largest_unique_subarray([1, 1, 1]) # 1

Q6. Smart Sum (Medium)
Filename: <YourID>_q6.py
Function:

```
def smart_sum(*args):
    """
    Sum all numeric inputs, including numbers contained inside
   lists or tuples (which
    themselves may be nested arbitrarily). Non-numeric values
   are not expected.
    Return the sum as an int or float.
    """
    pass
```

Input: variadic: numbers andor liststuples of numbers
Output: int or float
Test Cases:

- smart_sum(1, 2, [3, 4], (5, 6)) # 21
- smart_sum(10) # 10
- smart_sum([1, [2, 3]]) # 6

Q7. Maximum Depth of Nested Lists (Easy)
Filename: <YourID>_q7.py
Function:

```
def max_depth(lst):
    """
    Return the maximum nesting depth of lists. A flat list
   (including an empty list)
    has depth 1. If there are nested lists inside, the depth
   increases.
    """
    pass
```

Input: lst: list
Output: int
Test Cases:

- max_depth([1, [2, [3, [4]]]]) # 4
- max_depth([1, 2, 3]) # 1
- max_depth([]) # 1

Q8. Median of Numbers (Easy)
Filename: <YourID>_q8.py
Function:

```
def median(numbers):
    """
    Return the statistical median of the list 'numbers'. If the
   list is empty, return
    None. For an odd-length list, return the middle element.
   For an even-length list,
    return the average of the two middle elements.
    """
    pass
```

Input: numbers: list[int or float]
Output: float or None
Test Cases:

- median([1, 2, 3]) # 2
- median([1, 2, 3, 4]) # 2.5
- median([]) # None
- median([5]) # 5

Q9. Rotate List (Medium)
Filename: <YourID>_q9.py
Function:

```
def rotate_list(nums, k):
    """
    Rotate the list 'nums' to the right by k steps, modifying
   the list in place
    without creating a new list. Return the rotated list.
    """
    pass
```

Input: nums: list[int], k: int
Output: list[int]
Test Cases:

- rotate_list([1, 2, 3, 4, 5, 6], 2) # [5, 6, 1, 2, 3, 4]
- rotate_list([1, 2, 3], 0) # [1, 2, 3]
- rotate_list([1], 5) # [1]

Submission Checklist

- Ensure each function passes the public tests; hidden tests may cover additional edge cases.
- Include docstrings and minimal comments where necessary.
- Zip all .py files into <YourNAME>_<YourID>.zip for submission.