CSL100 Practice Sheet – Whole Syllabus

Instructions

- For each question below, create a Python file named <YourID>_<questio
  e.g., B25MM777_q1.py.
- Each file must contain the specified function/class signature.
- You may add helper functions, but the main signature must remain unchanged.
- Your code will be tested against hidden cases. Follow input/output specs precisely.

Mandatory Submission Rules

- **Do not read input from the user.** No input() calls.
- **Return values instead of printing**, unless a method explicitly says to print/return a string. Prefer returning strings; printing will not be tested unless stated.
- Use exact function names and parameter orders.
- Keep each question self-contained (no cross-imports).

Practice Questions

Q1. FizzBuzz (Easy)
Filename: <YourID>_q1.py
Function:

```python
def fizzbuzz(n):
    """
    Return a list from 1..n where multiples of 3 -> 'Fizz',
    multiples of 5 -> 'Buzz', and multiples of both -> '
   FizzBuzz'.
    Otherwise keep the number.
    """
    pass
```

Input: n: int **Output:** list
Test Cases:

```
fizzbuzz(5)
# [1, 2, 'Fizz', 4, 'Buzz']

fizzbuzz(15)
# [1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8,
#  'Fizz', 'Buzz', 11, 'Fizz', 13, 14, 'FizzBuzz']
```

Q2. Reverse String (Easy)
Filename: <YourID>_q2.py
Function:

```
def reverse_string(s):
    """
    Return the reversed version of 's' using slicing.
    """
    pass
```

Input: s: str **Output:** str
Test Cases:

```
reverse_string("hello")    # "olleh"
reverse_string("Python")   # "nohtyP"
reverse_string("")         # ""
```

Q3. Count Vowels (Easy)
Filename: <YourID>_q3.py
Function:

```
def count_vowels(s):
    """
    Return the number of vowels (a,e,i,o,u) in 's'.
    Case-insensitive.
    """
    pass
```

Input: s: str **Output:** int
Test Cases:

```
count_vowels("Hello World")   # 3
count_vowels("AEIOU")         # 5
count_vowels("Rhythm")        # 0
```

Q4. Prime Number Check (Medium)
Filename: <YourID>_q4.py
Function:

```
def is_prime(n):
    """
    Return True if n is prime, else False. (n > 1 and only
   divisible by 1,n)
    """
    pass
```

Input: n: int **Output:** bool
Test Cases:

```
is_prime(7)    # True
is_prime(10)   # False
is_prime(1)    # False
is_prime(2)    # True
```

Q5. Sum of Digits (Easy)
Filename: <YourID>_q5.py
Function:

```
def sum_digits(n):
    """
    Return the sum of digits of non-negative integer n.
    """
    pass
```

Input: n: int **Output:** int
Test Cases:

```
sum_digits(123)  # 6
sum_digits(90)   # 9
sum_digits(5)    # 5
```

Q6. Find Max in List (Easy)
Filename: <YourID>_q6.py
Function:

```
def find_max(numbers):
    """
    Return the largest element of 'numbers' without using max
   ().
    Return None for empty list.
    """
    pass
```

Input: numbers: list **Output:** int | float | None
Test Cases:

```
find_max([1, 5, 3, 9, 2])  # 9
find_max([-10, -5, -2])    # -2
find_max([])               # None
```

Q7. Remove Duplicates (Easy)
Filename: <YourID>_q7.py
Function:

```
def remove_duplicates(items):
    """
    Return a new list with unique elements from 'items'.
    Order in the result does not matter.
    """
    pass
```

Input: items: list **Output:** list
Test Cases:

```
remove_duplicates([1, 2, 2, 3, 1, 4])  # [1,2,3,4] (any order)
remove_duplicates(["a", "b", "a"])     # ["a","b"] (any order)
```

Q8. Get Odd-Indexed Elements (Easy)
Filename: <YourID>_q8.py
Function:

```
def get_odd_indices(items):
    """
    Return elements from 'items' that are at odd indices
   (1,3,5,...).
    """
    pass
```

Input: items: list **Output:** list
Test Cases:

```
get_odd_indices([10, 20, 30, 40, 50])  # [20, 40]
get_odd_indices(["a", "b", "c"])       # ["b"]
```

Q9. Recursive Factorial (Medium)
Filename: <YourID>_q9.py
Function:

```
def factorial_recursive(n):
    """
    Return n! using recursion. Assume n is non-negative.
    """
    pass
```

Input: n: int **Output:** int
Test Cases:

```
factorial_recursive(5)  # 120
factorial_recursive(0)  # 1
factorial_recursive(1)  # 1
```

Q10. Flexible Sum (Easy)
Filename: <YourID>_q10.py
Function:

```
def flexible_sum(*args):
    """
    Return the sum of all positional numeric arguments.
    If no arguments, return 0.
    """
    pass
```

Input: *args: numbers **Output:** int | float
Test Cases:

```
flexible_sum(1, 2, 3)       # 6
flexible_sum(10.5, 2.5)     # 13.0
flexible_sum()              # 0
```

Q11. Build User Profile (Easy)
Filename: <YourID>_q11.py
Function:

```
def build_profile(first, last, **kwargs):
    """
    Create a user profile dict with required keys
    'first_name' and 'last_name', plus any extra kwargs.
    """
    pass
```

Input: first: str, last: str, **kwargs **Output:** dict
Test Cases:

```
build_profile("John", "Doe")
# {'first_name': 'John', 'last_name': 'Doe'}

build_profile("Jane", "Smith", age=30, city="New York")
# {'first_name': 'Jane', 'last_name': 'Smith', 'age': 30, '
   city': 'New York'}
```

Q12. Word Frequency (Medium)
Filename: <YourID>_q12.py
Function:

```
def word_frequency(sentence):
    """
    Return {word: count} for lowercase, space-separated '
   sentence'
    with no punctuation. Empty string -> {}.
    """
    pass
```

Input: sentence: str **Output:** dict
Test Cases:

```
word_frequency("hello world hello")
# {'hello': 2, 'world': 1}

word_frequency("go spot go")
# {'go': 2, 'spot': 1}

word_frequency("")
# {}
```

Q13. Set Intersection (Easy)
Filename: <YourID>_q13.py
Function:

```
def find_intersection(set1, set2):
    """
    Return a new set with elements present in both set1 and
   set2.
    """
    pass
```

Input: set1: set, set2: set **Output:** set
Test Cases:

```
find_intersection({1, 2, 3}, {2, 3, 4})  # {2, 3}
find_intersection({"a", "b"}, {"c", "d"}) # set()
```

Q14. Invert Dictionary (Medium)
Filename: <YourID>_q14.py
Function:

```
def invert_dict(d):
    """
    Return a dict with keys and values swapped.
    Assume all values are unique and hashable.
    """
    pass
```

Input: d: dict **Output:** dict
Test Cases:

```
invert_dict({'a': 1, 'b': 2})  # {1: 'a', 2: 'b'}
invert_dict({'k1': 'v1'})      # {'v1': 'k1'}
```

Q15. Squares of Evens (Easy)
Filename: <YourID>_q15.py
Function:

```
def squares_of_evens(numbers):
    """
    Return squares of even numbers from 'numbers' using a list
    comp.
    """
    pass
```

Input: numbers: list[int] **Output:** list[int]
Test Cases:

```
squares_of_evens([1, 2, 3, 4, 5, 6])  # [4, 16, 36]
squares_of_evens([1, 3, 5])           # []
squares_of_evens([-2, 0, 4])          # [4, 0, 16]
```

Q16. Nested Sum by Depth (Medium)
Filename: <YourID>_q16.py
Function:

```
def nested_sum(lst, depth=1):
    """
    Each integer contributes value * depth to the total.
    Example: [1,[4,[6]]] = 1*1 + 4*2 + 6*3 = 27.
    Use recursion. Elements are ints or lists of the same form
   .
    """
    pass
```

Input: lst: list[int | list] **Output:** int
Examples:

```
nested_sum([1, [4, [6]]])          # 27
nested_sum([2, [3, [4, [5]]]])     # 40
nested_sum([])                     # 0
```

Q17. Spiral Matrix Traversal (Medium)
Filename: <YourID>_q17.py
Function:

```
def spiral_order(matrix):
    """
    Return elements of a rectangular 2D matrix in clockwise
   spiral order.
    """
    pass
```

Input: matrix: list[list[int]] **Output:** list[int]
Examples:

```
spiral_order([[1,2,3],[4,5,6],[7,8,9]])
# [1,2,3,6,9,8,7,4,5]

spiral_order([[1,2],[3,4]])
# [1,2,4,3]
```

Q18. Subset with Target Sum (Hard)
Filename: <YourID>_q18.py
Function:

```
def subset_sum(nums, target):
    """
    Return True if some subset of 'nums' sums exactly to '
   target'.
    Use recursion or backtracking.
    """
    pass
```

Input: nums: list[int], target: int **Output:** bool
Examples:

```
subset_sum([3,34,4,12,5,2], 9)   # True
subset_sum([1,2,3], 7)           # False
subset_sum([], 0)                # True
```

Q19. Anagram Grouping (Medium)
Filename: <YourID>_q19.py
Function:

```
def group_anagrams(words):
    """
    Group words that are anagrams.
    Key: sorted-letter signature; Value: list of words.
    """
    pass
```

Input: words: list[str] **Output:** dict[str, list[str]]
Examples:

```
group_anagrams(["eat","tea","tan","ate","nat","bat"])
# {'aet': ['eat','tea','ate'], 'ant': ['tan','nat'], 'abt': ['
   bat']}
```

Q20. Sudoku Validator (Hard)
Filename: <YourID>_q20.py
Function:

```
def valid_sudoku_rows(board):
    """
    Return True iff each row has no duplicate numbers among
   1..9.
    Zeros are blanks and ignored. 'board' is 9x9 (list of
   lists).
    """
    pass
```

Input: board: list[list[int]] **Output:** bool
Examples:

```
valid_sudoku_rows([[5,3,0],[6,0,0],[0,9,8]])  # True
valid_sudoku_rows([[5,3,3],[6,0,0],[0,9,8]])  # False
```

Q21. Student Class (OOP | Easy)
Filename: <YourID>_q21.py
Class:

```
class Student:
    """
    Store and display student details.
    Attributes:
      name (str), roll_number (int), marks (list[int])
    Methods:
      average() -> float         # 0.0 if no marks
      grade() -> str             # 'A' if avg>=75, 'B' if >=60,
    else 'C'
      info() -> str              # "Alice (Roll: 101) | Avg:
   80.0 | Grade: A"
    """
    pass
```

Examples:

```
s1 = Student("Alice", 101, [80,70,90])
s1.average()  # 80.0
s1.grade()    # 'A'
s1.info()     # "Alice (Roll: 101) | Avg: 80.0 | Grade: A"

s2 = Student("Bob", 102, [50,60,55])
s2.grade()    # 'C'
```

Q22. Temperature Converter (OOP | Medium)
Filename: <YourID>_q22.py
Class:

```
class Temperature:
    """
    Store temperature in Celsius internally.
    Methods:
      to_fahrenheit() -> float
      @classmethod from_fahrenheit(f) -> Temperature
      display() -> str   # "37 C  / 98.6 F "
    """
    pass
```

Examples:

```
Temperature(25).to_fahrenheit()                   # 77.0
Temperature.from_fahrenheit(98.6).display()       # "37 C  /
   98.6 F "
```

Q23. Bank Account (OOP | Medium)
Filename: <YourID>_q23.py
Class:

```
class BankAccount:
    """
    Simple bank account.
    Attributes: owner (str), _balance (float)
    Methods:
      deposit(amount) -> None
      withdraw(amount) -> str|None  # "Insufficient funds" on
   failure
      display_balance() -> str      # e.g., "Balance: 1500.0"
    """
    pass
```

Examples:

```
acc = BankAccount("Alice", 1000)
acc.deposit(500)
acc.withdraw(2000)          # "Insufficient funds"
acc.display_balance()       # "Balance: 1500.0"
acc.withdraw(100)
acc.display_balance()       # "Balance: 1400.0"
```

Q24. Time Class (OOP | Medium)
Filename: <YourID>_q24.py
Class:

```
class Time:
    """
    Time with hours and minutes (non-negative).
    Supports:
      t1 + t2, t1 - t2 (non-negative result), display()->"HH:
   MM"
    Normalize minutes to [0,59] with carry/borrow to hours.
    """
    pass
```

Examples:

```
t1 = Time(2,45); t2 = Time(1,30)
(t1 + t2).display()  # "04:15"
(t1 - t2).display()  # "01:15"
Time(0,90).display() # "01:30"
```

Q25. Stack Class (OOP | Easy)
Filename: <YourID>_q25.py
Class:

```
class Stack:
    """
    Implement a stack using a list.
    Methods:
      push(item) -> None
      pop() -> item|None
      peek() -> item|None
      is_empty() -> bool
      size() -> int
    """
    pass
```

Examples:

```
s = Stack()
s.push(10); s.push(20); s.push(30)
s.pop()          # 30
s.peek()         # 20
s.size()         # 2
s.is_empty()     # False
s.pop(); s.pop()
s.is_empty()     # True
```

Q26. Merge Dictionaries with Summation (Medium)
Filename: <YourID>_q26.py
Function:

```
def merge_sum(dict1, dict2):
    """
    Merge two dictionaries whose values are integers.
    For keys present in both, sum their values.
    Return a new dictionary without mutating inputs.
    """
    pass
```

Input: dict1: dict[str,int], dict2: dict[str,int]
Output: dict[str,int]
Test Cases:

```
merge_sum({'a':10,'b':5}, {'b':7,'c':3})
# {'a':10, 'b':12, 'c':3}
```

Q27. Invert Dictionary with Duplicate Values (Medium)
Filename: <YourID>_q27.py
Function:

```
def invert_dict(d):
    """
    Invert a dictionary that may have duplicate values.
    Example: {'a':1,'b':2,'c':1} -> {1:['a','c'], 2:['b']}
    Keys in result lists may be in any order.
    """
    pass
```

Input: d: dict
Output: dict
Test Cases:

```
invert_dict({'a':1,'b':2,'c':1})
# {1: ['a','c'], 2: ['b']}
```

Q28. Longest Consecutive Sequence (Medium)
Filename: <YourID>_q28.py
Function:

```
def longest_consecutive(nums):
    """
    Return the length of the longest consecutive elements
   sequence.
    Order in 'nums' is arbitrary. Runs in O(n) average time.
    """
    pass
```

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

```
longest_consecutive([100,4,200,1,3,2])          # 4  (
   sequence 1,2,3,4)
longest_consecutive([0,3,7,2,5,8,4,6,0,1])      # 9  (
   sequence 0..8)
```

Q29. Pascal's Triangle Generator (Medium)
Filename: <YourID>_q29.py
Function:

```
def pascal_triangle(n):
    """
    Return the first n rows of Pascal's triangle as a list of
   lists.
    For n <= 0, return [].
    """
    pass
```

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

```
pascal_triangle(5)
# [[1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1]]
```

Q30. Matrix Rotation (90° Clockwise) (Medium)
Filename: <YourID>_q30.py
Function:

```python
def rotate_matrix(matrix):
    """
    Rotate a square matrix 90 degrees clockwise in-place.
    Return the same matrix object after rotation.
    """
    pass
```

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

```python
rotate_matrix([[1,2,3],[4,5,6],[7,8,9]])
# [[7,4,1],[8,5,2],[9,6,3]]

rotate_matrix([[1,2],[3,4]])
# [[3,1],[4,2]]
```