# Python Interview Handbook — Batch 1

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 001. Reverse String

**Statement:** Implement problem “reverse string” in Python.
Explanation: Problem “reverse string”. Outline brute-force vs optimized approaches, edge cases, and complexity. Uses Python reversed() + join(). Time O(n), Space O(n).

```
def reverse_string(s: str) -> str:
    """Return reversed string using join/reversed."""
    return ''.join(reversed(s))
```

```
import unittest
from problems.001_reverse_string import reverse_string
class TestReverseString(unittest.TestCase):
    def test_basic(self):
        self.assertEqual(reverse_string('abc'), 'cba')
```

## 002. Is Palindrome

**Statement:** Implement problem “is palindrome” in Python.
Explanation: Problem “is palindrome”. Outline brute-force vs optimized approaches, edge cases, and complexity. Clean input then compare to reverse. Time O(n), Space O(n).

```
def is_palindrome(s: str) -> bool:
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s == s[::-1]
```

```
import unittest
from problems.002_is_palindrome import is_palindrome
class TestIsPalindrome(unittest.TestCase):
    def test_examples(self):
        self.assertTrue(is_palindrome('A man, a plan, a canal: Panama'))
        self.assertFalse(is_palindrome('hello'))
```

## 003. Char Frequency

**Statement:** Implement problem “char frequency” in Python.
Explanation: Problem “char frequency”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def char_frequency(*args, **kwargs):
    """TODO: implement char_frequency as described in the handbook."""
    return None
```

```
import unittest
from problems.char_frequency import char_frequency
class TestCharFrequency(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(char_frequency())
```

## 004. First Non Repeated Char

**Statement:** Implement problem “first non repeated char” in Python.
Explanation: Problem “first non repeated char”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def first_non_repeated_char(*args, **kwargs):
    """TODO: implement first_non_repeated_char as described in the handbook."""
    return None
```

```
import unittest
from problems.first_non_repeated_char import first_non_repeated_char
class TestFirstNonRepeatedChar(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(first_non_repeated_char())
```

## 005. Second Largest

**Statement:** Implement problem “second largest” in Python.
Explanation: Problem “second largest”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def second_largest(*args, **kwargs):
    """TODO: implement second_largest as described in the handbook."""
    return None
```

```
import unittest
from problems.second_largest import second_largest
class TestSecondLargest(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(second_largest())
```

## 006. Remove Duplicates

**Statement:** Implement problem “remove duplicates” in Python.
Explanation: Problem “remove duplicates”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def remove_duplicates(*args, **kwargs):
    """TODO: implement remove_duplicates as described in the handbook."""
    return None
```

```
import unittest
from problems.remove_duplicates import remove_duplicates
class TestRemoveDuplicates(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(remove_duplicates())
```

## 007. Rotate List K

**Statement:** Implement problem “rotate list k” in Python.
Explanation: Problem “rotate list k”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def rotate_list_k(*args, **kwargs):
    """TODO: implement rotate_list_k as described in the handbook."""
    return None
```

```
import unittest
from problems.rotate_list_k import rotate_list_k
class TestRotateListK(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(rotate_list_k())
```

## 008. Merge Two Sorted Lists

**Statement:** Implement problem “merge two sorted lists” in Python.
Explanation: Problem “merge two sorted lists”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def merge_two_sorted_lists(*args, **kwargs):
    """TODO: implement merge_two_sorted_lists as described in the handbook."""
    return None
```

```
import unittest
from problems.merge_two_sorted_lists import merge_two_sorted_lists
class TestMergeTwoSortedLists(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(merge_two_sorted_lists())
```

## 009. Linked List Cycle

**Statement:** Implement problem “linked list cycle” in Python.
Explanation: Problem “linked list cycle”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def linked_list_cycle(*args, **kwargs):
    """TODO: implement linked_list_cycle as described in the handbook."""
    return None
```

```
import unittest
from problems.linked_list_cycle import linked_list_cycle
class TestLinkedListCycle(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(linked_list_cycle())
```

## 010. Detect Cycle Linked List

**Statement:** Implement problem “detect cycle linked list” in Python.
Explanation: Problem “detect cycle linked list”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def detect_cycle_linked_list(*args, **kwargs):
    """TODO: implement detect_cycle_linked_list as described in the handbook."""
    return None
```

```
import unittest
from problems.detect_cycle_linked_list import detect_cycle_linked_list
class TestDetectCycleLinkedList(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(detect_cycle_linked_list())
```

# Python Interview Handbook — Batch 10

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 091. Serialize Graph Adjlist

**Statement:** Implement problem “serialize graph adjlist” in Python.
Explanation: Problem “serialize graph adjlist”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def serialize_graph_adjlist(*args, **kwargs):
    """TODO: implement serialize_graph_adjlist as described in the handbook."""
    return None
```

```
import unittest
from problems.serialize_graph_adjlist import serialize_graph_adjlist
class TestSerializeGraphAdjlist(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(serialize_graph_adjlist())
```

## 092. Deserialize Graph Adjlist

**Statement:** Implement problem “deserialize graph adjlist” in Python.
Explanation: Problem “deserialize graph adjlist”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def deserialize_graph_adjlist(*args, **kwargs):
    """TODO: implement deserialize_graph_adjlist as described in the handbook."""
    return None
```

```
import unittest
from problems.deserialize_graph_adjlist import deserialize_graph_adjlist
class TestDeserializeGraphAdjlist(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(deserialize_graph_adjlist())
```

## 093. Palindrome Partitioning

**Statement:** Implement problem “palindrome partitioning” in Python.
Explanation: Problem “palindrome partitioning”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def palindrome_partitioning(*args, **kwargs):
    """TODO: implement palindrome_partitioning as described in the handbook."""
    return None
```

```
import unittest
from problems.palindrome_partitioning import palindrome_partitioning
class TestPalindromePartitioning(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(palindrome_partitioning())
```

## 094. Min Window Substring

**Statement:** Implement problem “min window substring” in Python.
Explanation: Problem “min window substring”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def min_window_substring(*args, **kwargs):
    """TODO: implement min_window_substring as described in the handbook."""
    return None
```

```
import unittest
from problems.min_window_substring import min_window_substring
class TestMinWindowSubstring(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(min_window_substring())
```

## 095. Max Rectangle Histogram

**Statement:** Implement problem “max rectangle histogram” in Python.
Explanation: Problem “max rectangle histogram”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def max_rectangle_histogram(*args, **kwargs):
    """TODO: implement max_rectangle_histogram as described in the handbook."""
    return None
```

```
import unittest
from problems.max_rectangle_histogram import max_rectangle_histogram
class TestMaxRectangleHistogram(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(max_rectangle_histogram())
```

## 096. Sliding Window Max

**Statement:** Implement problem “sliding window max” in Python.
Explanation: Problem “sliding window max”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def sliding_window_max(*args, **kwargs):
    """TODO: implement sliding_window_max as described in the handbook."""
    return None
```

```
import unittest
from problems.sliding_window_max import sliding_window_max
class TestSlidingWindowMax(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(sliding_window_max())
```

## 097. Design Hashmap Simple

**Statement:** Implement problem “design hashmap simple” in Python.
Explanation: Problem “design hashmap simple”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def design_hashmap_simple(*args, **kwargs):
    """TODO: implement design_hashmap_simple as described in the handbook."""
    return None
```

```
import unittest
from problems.design_hashmap_simple import design_hashmap_simple
class TestDesignHashmapSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(design_hashmap_simple())
```

## 098. Design Cache Ttl

**Statement:** Implement problem “design cache ttl” in Python.
Explanation: Problem “design cache ttl”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def design_cache_ttl(*args, **kwargs):
    """TODO: implement design_cache_ttl as described in the handbook."""
    return None
```

```
import unittest
from problems.design_cache_ttl import design_cache_ttl
class TestDesignCacheTtl(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(design_cache_ttl())
```

## 099. Url Validation

**Statement:** Implement problem “url validation” in Python.
Explanation: Problem “url validation”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def url_validation(*args, **kwargs):
    """TODO: implement url_validation as described in the handbook."""
    return None
```

```
import unittest
from problems.url_validation import url_validation
class TestUrlValidation(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(url_validation())
```

## 100. Email Validation Regex

**Statement:** Implement problem “email validation regex” in Python.
Explanation: Problem “email validation regex”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def email_validation_regex(*args, **kwargs):
    """TODO: implement email_validation_regex as described in the handbook."""
    return None
```

```
import unittest
from problems.email_validation_regex import email_validation_regex
class TestEmailValidationRegex(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(email_validation_regex())
```

# Python Interview Handbook — Batch 11

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 101. Find Kth Smallest Bst

**Statement:** Implement problem “find kth smallest bst” in Python.
Explanation: Problem “find kth smallest bst”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def find_kth_smallest_bst(*args, **kwargs):
    """TODO: implement find_kth_smallest_bst as described in the handbook."""
    return None
```

```
import unittest
from problems.find_kth_smallest_bst import find_kth_smallest_bst
class TestFindKthSmallestBst(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(find_kth_smallest_bst())
```

## 102. Count Set Bits

**Statement:** Implement problem “count set bits” in Python.
Explanation: Problem “count set bits”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def count_set_bits(*args, **kwargs):
    """TODO: implement count_set_bits as described in the handbook."""
    return None
```

```
import unittest
from problems.count_set_bits import count_set_bits
class TestCountSetBits(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(count_set_bits())
```

## 103. Hamming Distance

**Statement:** Implement problem “hamming distance” in Python.
Explanation: Problem “hamming distance”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def hamming_distance(*args, **kwargs):
    """TODO: implement hamming_distance as described in the handbook."""
    return None
```

```
import unittest
from problems.hamming_distance import hamming_distance
class TestHammingDistance(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(hamming_distance())
```

## 104. Gray Code

**Statement:** Implement problem “gray code” in Python.
Explanation: Problem “gray code”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def gray_code(*args, **kwargs):
    """TODO: implement gray_code as described in the handbook."""
    return None
```

```
import unittest
from problems.gray_code import gray_code
class TestGrayCode(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(gray_code())
```

## 105. Power Set Iterative

**Statement:** Implement problem “power set iterative” in Python.
Explanation: Problem “power set iterative”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def power_set_iterative(*args, **kwargs):
    """TODO: implement power_set_iterative as described in the handbook."""
    return None
```

```
import unittest
from problems.power_set_iterative import power_set_iterative
class TestPowerSetIterative(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(power_set_iterative())
```

# Python Interview Handbook — Batch 2

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 011. Binary Search

**Statement:** Implement problem “binary search” in Python.
Explanation: Problem “binary search”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_search(*args, **kwargs):
    """TODO: implement binary_search as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_search import binary_search
class TestBinarySearch(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_search())
```

## 012. Merge Sort

**Statement:** Implement problem “merge sort” in Python.
Explanation: Problem “merge sort”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def merge_sort(*args, **kwargs):
    """TODO: implement merge_sort as described in the handbook."""
    return None
```

```
import unittest
from problems.merge_sort import merge_sort
class TestMergeSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(merge_sort())
```

## 013. Quick Sort

**Statement:** Implement problem “quick sort” in Python.
Explanation: Problem “quick sort”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def quick_sort(*args, **kwargs):
    """TODO: implement quick_sort as described in the handbook."""
    return None
```

```
import unittest
from problems.quick_sort import quick_sort
class TestQuickSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(quick_sort())
```

## 014. Kth Largest

**Statement:** Implement problem “kth largest” in Python.
Explanation: Problem “kth largest”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def kth_largest(*args, **kwargs):
    """TODO: implement kth_largest as described in the handbook."""
    return None
```

```
import unittest
from problems.kth_largest import kth_largest
class TestKthLargest(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(kth_largest())
```

## 015. Kadane Max Subarray

**Statement:** Implement problem “kadane max subarray” in Python.
Explanation: Problem “kadane max subarray”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def kadane_max_subarray(*args, **kwargs):
    """TODO: implement kadane_max_subarray as described in the handbook."""
    return None
```

```
import unittest
from problems.kadane_max_subarray import kadane_max_subarray
class TestKadaneMaxSubarray(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(kadane_max_subarray())
```

## 016. Permutations

**Statement:** Implement problem “permutations” in Python.
Explanation: Problem “permutations”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def permutations(*args, **kwargs):
    """TODO: implement permutations as described in the handbook."""
    return None
```

```
import unittest
from problems.permutations import permutations
class TestPermutations(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(permutations())
```

## 017. Combinations

**Statement:** Implement problem “combinations” in Python.
Explanation: Problem “combinations”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def combinations(*args, **kwargs):
    """TODO: implement combinations as described in the handbook."""
    return None
```

```
import unittest
from problems.combinations import combinations
class TestCombinations(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(combinations())
```

## 018. Power Set

**Statement:** Implement problem “power set” in Python.
Explanation: Problem “power set”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def power_set(*args, **kwargs):
    """TODO: implement power_set as described in the handbook."""
    return None
```

```
import unittest
from problems.power_set import power_set
class TestPowerSet(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(power_set())
```

## 019. Fib Memo

**Statement:** Implement problem “fib memo” in Python.
Explanation: Problem “fib memo”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def fib_memo(*args, **kwargs):
    """TODO: implement fib_memo as described in the handbook."""
    return None
```

```
import unittest
from problems.fib_memo import fib_memo
class TestFibMemo(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(fib_memo())
```

## 020. Nth Fibonacci Iterative

**Statement:** Implement problem “nth fibonacci iterative” in Python.
Explanation: Problem “nth fibonacci iterative”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def nth_fibonacci_iterative(*args, **kwargs):
    """TODO: implement nth_fibonacci_iterative as described in the handbook."""
    return None
```

```
import unittest
from problems.nth_fibonacci_iterative import nth_fibonacci_iterative
class TestNthFibonacciIterative(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(nth_fibonacci_iterative())
```

# Python Interview Handbook — Batch 3

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 021. Lru Cache Simple

**Statement:** Implement problem “lru cache simple” in Python.
Explanation: Problem “lru cache simple”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def lru_cache_simple(*args, **kwargs):
    """TODO: implement lru_cache_simple as described in the handbook."""
    return None
```

```
import unittest
from problems.lru_cache_simple import lru_cache_simple
class TestLruCacheSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(lru_cache_simple())
```

## 022. Stack Using List

**Statement:** Implement problem “stack using list” in Python.
Explanation: Problem “stack using list”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def stack_using_list(*args, **kwargs):
    """TODO: implement stack_using_list as described in the handbook."""
    return None
```

```
import unittest
from problems.stack_using_list import stack_using_list
class TestStackUsingList(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(stack_using_list())
```

## 023. Queue Using Deque

**Statement:** Implement problem “queue using deque” in Python.
Explanation: Problem “queue using deque”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def queue_using_deque(*args, **kwargs):
    """TODO: implement queue_using_deque as described in the handbook."""
    return None
```

```
import unittest
from problems.queue_using_deque import queue_using_deque
class TestQueueUsingDeque(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(queue_using_deque())
```

## 024. Min Heap K Smallest

**Statement:** Implement problem “min heap k smallest” in Python.
Explanation: Problem “min heap k smallest”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def min_heap_k_smallest(*args, **kwargs):
    """TODO: implement min_heap_k_smallest as described in the handbook."""
    return None
```

```
import unittest
from problems.min_heap_k_smallest import min_heap_k_smallest
class TestMinHeapKSmallest(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(min_heap_k_smallest())
```

## 025. Top K Frequent Words

**Statement:** Implement problem “top k frequent words” in Python.
Explanation: Problem “top k frequent words”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def top_k_frequent_words(*args, **kwargs):
    """TODO: implement top_k_frequent_words as described in the handbook."""
    return None
```

```
import unittest
from problems.top_k_frequent_words import top_k_frequent_words
class TestTopKFrequentWords(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(top_k_frequent_words())
```

## 026. Anagram Groups

**Statement:** Implement problem “anagram groups” in Python.
Explanation: Problem “anagram groups”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def anagram_groups(*args, **kwargs):
    """TODO: implement anagram_groups as described in the handbook."""
    return None
```

```
import unittest
from problems.anagram_groups import anagram_groups
class TestAnagramGroups(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(anagram_groups())
```

## 027. Two Sum

**Statement:** Implement problem “two sum” in Python.
Explanation: Problem “two sum”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def two_sum(*args, **kwargs):
    """TODO: implement two_sum as described in the handbook."""
    return None
```

```
import unittest
from problems.two_sum import two_sum
class TestTwoSum(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(two_sum())
```

## 028. Three Sum

**Statement:** Implement problem “three sum” in Python.
Explanation: Problem “three sum”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def three_sum(*args, **kwargs):
    """TODO: implement three_sum as described in the handbook."""
    return None
```

```
import unittest
from problems.three_sum import three_sum
class TestThreeSum(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(three_sum())
```

## 029. Longest Common Prefix

**Statement:** Implement problem “longest common prefix” in Python.
Explanation: Problem “longest common prefix”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def longest_common_prefix(*args, **kwargs):
    """TODO: implement longest_common_prefix as described in the handbook."""
    return None
```

```
import unittest
from problems.longest_common_prefix import longest_common_prefix
class TestLongestCommonPrefix(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(longest_common_prefix())
```

## 030. Longest Increasing Subsequence

**Statement:** Implement problem “longest increasing subsequence” in Python.
Explanation: Problem “longest increasing subsequence”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def longest_increasing_subsequence(*args, **kwargs):
    """TODO: implement longest_increasing_subsequence as described in the handbook."""
    return None
```

```
import unittest
from problems.longest_increasing_subsequence import longest_increasing_subsequence
class TestLongestIncreasingSubsequence(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(longest_increasing_subsequence())
```

# Python Interview Handbook — Batch 4

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 031. Edit Distance

**Statement:** Implement problem “edit distance” in Python.
Explanation: Problem “edit distance”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def edit_distance(*args, **kwargs):
    """TODO: implement edit_distance as described in the handbook."""
    return None
```

```
import unittest
from problems.edit_distance import edit_distance
class TestEditDistance(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(edit_distance())
```

## 032. Unique Paths Grid

**Statement:** Implement problem “unique paths grid” in Python.
Explanation: Problem “unique paths grid”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def unique_paths_grid(*args, **kwargs):
    """TODO: implement unique_paths_grid as described in the handbook."""
    return None
```

```
import unittest
from problems.unique_paths_grid import unique_paths_grid
class TestUniquePathsGrid(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(unique_paths_grid())
```

## 033. Coin Change Min

**Statement:** Implement problem “coin change min” in Python.
Explanation: Problem “coin change min”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def coin_change_min(*args, **kwargs):
    """TODO: implement coin_change_min as described in the handbook."""
    return None
```

```
import unittest
from problems.coin_change_min import coin_change_min
class TestCoinChangeMin(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(coin_change_min())
```

## 034. Word Break

**Statement:** Implement problem “word break” in Python.
Explanation: Problem “word break”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def word_break(*args, **kwargs):
    """TODO: implement word_break as described in the handbook."""
    return None
```

```
import unittest
from problems.word_break import word_break
class TestWordBreak(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(word_break())
```

## 035. Longest Palindromic Substring

**Statement:** Implement problem “longest palindromic substring” in Python.
Explanation: Problem “longest palindromic substring”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def longest_palindromic_substring(*args, **kwargs):
    """TODO: implement longest_palindromic_substring as described in the handbook."""
    return None
```

```
import unittest
from problems.longest_palindromic_substring import longest_palindromic_substring
class TestLongestPalindromicSubstring(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(longest_palindromic_substring())
```

## 036. Serialize Deserialize Tree

**Statement:** Implement problem “serialize deserialize tree” in Python.
Explanation: Problem “serialize deserialize tree”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def serialize_deserialize_tree(*args, **kwargs):
    """TODO: implement serialize_deserialize_tree as described in the handbook."""
    return None
```

```
import unittest
from problems.serialize_deserialize_tree import serialize_deserialize_tree
class TestSerializeDeserializeTree(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(serialize_deserialize_tree())
```

## 037. Binary Tree Inorder

**Statement:** Implement problem “binary tree inorder” in Python.
Explanation: Problem “binary tree inorder”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_tree_inorder(*args, **kwargs):
    """TODO: implement binary_tree_inorder as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_tree_inorder import binary_tree_inorder
class TestBinaryTreeInorder(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_tree_inorder())
```

## 038. Binary Tree Preorder

**Statement:** Implement problem “binary tree preorder” in Python.
Explanation: Problem “binary tree preorder”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_tree_preorder(*args, **kwargs):
    """TODO: implement binary_tree_preorder as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_tree_preorder import binary_tree_preorder
class TestBinaryTreePreorder(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_tree_preorder())
```

## 039. Binary Tree Postorder

**Statement:** Implement problem “binary tree postorder” in Python.
Explanation: Problem “binary tree postorder”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_tree_postorder(*args, **kwargs):
    """TODO: implement binary_tree_postorder as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_tree_postorder import binary_tree_postorder
class TestBinaryTreePostorder(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_tree_postorder())
```

## 040. Is Balanced Tree

**Statement:** Implement problem “is balanced tree” in Python.
Explanation: Problem “is balanced tree”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def is_balanced_tree(*args, **kwargs):
    """TODO: implement is_balanced_tree as described in the handbook."""
    return None
```

```
import unittest
from problems.is_balanced_tree import is_balanced_tree
class TestIsBalancedTree(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(is_balanced_tree())
```

# Python Interview Handbook — Batch 5

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 041. Lowest Common Ancestor

**Statement:** Implement problem “lowest common ancestor” in Python.
Explanation: Problem “lowest common ancestor”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def lowest_common_ancestor(*args, **kwargs):
    """TODO: implement lowest_common_ancestor as described in the handbook."""
    return None
```

```
import unittest
from problems.lowest_common_ancestor import lowest_common_ancestor
class TestLowestCommonAncestor(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(lowest_common_ancestor())
```

## 042. Trie Insert Search

**Statement:** Implement problem “trie insert search” in Python.
Explanation: Problem “trie insert search”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def trie_insert_search(*args, **kwargs):
    """TODO: implement trie_insert_search as described in the handbook."""
    return None
```

```
import unittest
from problems.trie_insert_search import trie_insert_search
class TestTrieInsertSearch(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(trie_insert_search())
```

## 043. Dijkstra Simple

**Statement:** Implement problem “dijkstra simple” in Python.
Explanation: Problem “dijkstra simple”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def dijkstra_simple(*args, **kwargs):
    """TODO: implement dijkstra_simple as described in the handbook."""
    return None
```

```
import unittest
from problems.dijkstra_simple import dijkstra_simple
class TestDijkstraSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(dijkstra_simple())
```

## 044. Bfs Graph

**Statement:** Implement problem “bfs graph” in Python.
Explanation: Problem “bfs graph”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def bfs_graph(*args, **kwargs):
    """TODO: implement bfs_graph as described in the handbook."""
    return None
```

```
import unittest
from problems.bfs_graph import bfs_graph
class TestBfsGraph(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(bfs_graph())
```

## 045. Dfs Graph Recursive

**Statement:** Implement problem “dfs graph recursive” in Python.
Explanation: Problem “dfs graph recursive”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def dfs_graph_recursive(*args, **kwargs):
    """TODO: implement dfs_graph_recursive as described in the handbook."""
    return None
```

```
import unittest
from problems.dfs_graph_recursive import dfs_graph_recursive
class TestDfsGraphRecursive(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(dfs_graph_recursive())
```

## 046. Sieve Eratosthenes

**Statement:** Implement problem “sieve eratosthenes” in Python.
Explanation: Problem “sieve eratosthenes”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def sieve_eratosthenes(*args, **kwargs):
    """TODO: implement sieve_eratosthenes as described in the handbook."""
    return None
```

```
import unittest
from problems.sieve_eratosthenes import sieve_eratosthenes
class TestSieveEratosthenes(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(sieve_eratosthenes())
```

## 047. N Queens Count

**Statement:** Implement problem “n queens count” in Python.
Explanation: Problem “n queens count”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def n_queens_count(*args, **kwargs):
    """TODO: implement n_queens_count as described in the handbook."""
    return None
```

```
import unittest
from problems.n_queens_count import n_queens_count
class TestNQueensCount(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(n_queens_count())
```

## 048. Sudoku Solver

**Statement:** Implement problem “sudoku solver” in Python.
Explanation: Problem “sudoku solver”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def sudoku_solver(*args, **kwargs):
    """TODO: implement sudoku_solver as described in the handbook."""
    return None
```

```
import unittest
from problems.sudoku_solver import sudoku_solver
class TestSudokuSolver(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(sudoku_solver())
```

## 049. Rotate Matrix 90

**Statement:** Implement problem “rotate matrix 90” in Python.
Explanation: Problem “rotate matrix 90”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def rotate_matrix_90(*args, **kwargs):
    """TODO: implement rotate_matrix_90 as described in the handbook."""
    return None
```

```
import unittest
from problems.rotate_matrix_90 import rotate_matrix_90
class TestRotateMatrix90(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(rotate_matrix_90())
```

## 050. Matrix Multiply

**Statement:** Implement problem “matrix multiply” in Python.
Explanation: Problem “matrix multiply”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def matrix_multiply(*args, **kwargs):
    """TODO: implement matrix_multiply as described in the handbook."""
    return None
```

```
import unittest
from problems.matrix_multiply import matrix_multiply
class TestMatrixMultiply(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(matrix_multiply())
```

# Python Interview Handbook — Batch 6

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 051. Transpose Matrix

**Statement:** Implement problem “transpose matrix” in Python.
Explanation: Problem “transpose matrix”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def transpose_matrix(*args, **kwargs):
    """TODO: implement transpose_matrix as described in the handbook."""
    return None
```

```
import unittest
from problems.transpose_matrix import transpose_matrix
class TestTransposeMatrix(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(transpose_matrix())
```

## 052. Spiral Matrix

**Statement:** Implement problem “spiral matrix” in Python.
Explanation: Problem “spiral matrix”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def spiral_matrix(*args, **kwargs):
    """TODO: implement spiral_matrix as described in the handbook."""
    return None
```

```
import unittest
from problems.spiral_matrix import spiral_matrix
class TestSpiralMatrix(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(spiral_matrix())
```

## 053. Find Missing Number

**Statement:** Implement problem “find missing number” in Python.
Explanation: Problem “find missing number”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def find_missing_number(*args, **kwargs):
    """TODO: implement find_missing_number as described in the handbook."""
    return None
```

```
import unittest
from problems.find_missing_number import find_missing_number
class TestFindMissingNumber(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(find_missing_number())
```

## 054. Find Duplicate

**Statement:** Implement problem “find duplicate” in Python.
Explanation: Problem “find duplicate”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def find_duplicate(*args, **kwargs):
    """TODO: implement find_duplicate as described in the handbook."""
    return None
```

```
import unittest
from problems.find_duplicate import find_duplicate
class TestFindDuplicate(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(find_duplicate())
```

## 055. Median Two Sorted Arrays Small

**Statement:** Implement problem “median two sorted arrays small” in Python.
Explanation: Problem “median two sorted arrays small”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def median_two_sorted_arrays_small(*args, **kwargs):
    """TODO: implement median_two_sorted_arrays_small as described in the handbook."""
    return None
```

```
import unittest
from problems.median_two_sorted_arrays_small import median_two_sorted_arrays_small
class TestMedianTwoSortedArraysSmall(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(median_two_sorted_arrays_small())
```

## 056. Search Insert Position

**Statement:** Implement problem “search insert position” in Python.
Explanation: Problem “search insert position”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def search_insert_position(*args, **kwargs):
    """TODO: implement search_insert_position as described in the handbook."""
    return None
```

```
import unittest
from problems.search_insert_position import search_insert_position
class TestSearchInsertPosition(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(search_insert_position())
```

## 057. Interval Merge

**Statement:** Implement problem “interval merge” in Python.
Explanation: Problem “interval merge”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def interval_merge(*args, **kwargs):
    """TODO: implement interval_merge as described in the handbook."""
    return None
```

```
import unittest
from problems.interval_merge import interval_merge
class TestIntervalMerge(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(interval_merge())
```

## 058. Best Time To Buy Sell Stock

**Statement:** Implement problem “best time to buy sell stock” in Python.
Explanation: Problem “best time to buy sell stock”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def best_time_to_buy_sell_stock(*args, **kwargs):
    """TODO: implement best_time_to_buy_sell_stock as described in the handbook."""
    return None
```

```
import unittest
from problems.best_time_to_buy_sell_stock import best_time_to_buy_sell_stock
class TestBestTimeToBuySellStock(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(best_time_to_buy_sell_stock())
```

## 059. Product Of Array Except Self

**Statement:** Implement problem “product of array except self” in Python.
Explanation: Problem “product of array except self”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def product_of_array_except_self(*args, **kwargs):
    """TODO: implement product_of_array_except_self as described in the handbook."""
    return None
```

```
import unittest
from problems.product_of_array_except_self import product_of_array_except_self
class TestProductOfArrayExceptSelf(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(product_of_array_except_self())
```

## 060. Max Profit K Transactions

**Statement:** Implement problem “max profit k transactions” in Python.
Explanation: Problem “max profit k transactions”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def max_profit_k_transactions(*args, **kwargs):
    """TODO: implement max_profit_k_transactions as described in the handbook."""
    return None
```

```
import unittest
from problems.max_profit_k_transactions import max_profit_k_transactions
class TestMaxProfitKTransactions(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(max_profit_k_transactions())
```

# Python Interview Handbook — Batch 7

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 061. Heap Sort

**Statement:** Implement problem “heap sort” in Python.
Explanation: Problem “heap sort”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def heap_sort(*args, **kwargs):
    """TODO: implement heap_sort as described in the handbook."""
    return None
```

```
import unittest
from problems.heap_sort import heap_sort
class TestHeapSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(heap_sort())
```

## 062. Bucket Sort Simple

**Statement:** Implement problem “bucket sort simple” in Python.
Explanation: Problem “bucket sort simple”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def bucket_sort_simple(*args, **kwargs):
    """TODO: implement bucket_sort_simple as described in the handbook."""
    return None
```

```
import unittest
from problems.bucket_sort_simple import bucket_sort_simple
class TestBucketSortSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(bucket_sort_simple())
```

## 063. Counting Sort Simple

**Statement:** Implement problem “counting sort simple” in Python.
Explanation: Problem “counting sort simple”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def counting_sort_simple(*args, **kwargs):
    """TODO: implement counting_sort_simple as described in the handbook."""
    return None
```

```
import unittest
from problems.counting_sort_simple import counting_sort_simple
class TestCountingSortSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(counting_sort_simple())
```

## 064. Radix Sort Simple

**Statement:** Implement problem “radix sort simple” in Python.
Explanation: Problem “radix sort simple”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def radix_sort_simple(*args, **kwargs):
    """TODO: implement radix_sort_simple as described in the handbook."""
    return None
```

```
import unittest
from problems.radix_sort_simple import radix_sort_simple
class TestRadixSortSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(radix_sort_simple())
```

## 065. Flatten Nested List

**Statement:** Implement problem “flatten nested list” in Python.
Explanation: Problem “flatten nested list”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def flatten_nested_list(*args, **kwargs):
    """TODO: implement flatten_nested_list as described in the handbook."""
    return None
```

```
import unittest
from problems.flatten_nested_list import flatten_nested_list
class TestFlattenNestedList(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(flatten_nested_list())
```

## 066. Flatten Dict Keys

**Statement:** Implement problem “flatten dict keys” in Python.
Explanation: Problem “flatten dict keys”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def flatten_dict_keys(*args, **kwargs):
    """TODO: implement flatten_dict_keys as described in the handbook."""
    return None
```

```
import unittest
from problems.flatten_dict_keys import flatten_dict_keys
class TestFlattenDictKeys(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(flatten_dict_keys())
```

## 067. Zip Two Lists

**Statement:** Implement problem “zip two lists” in Python.
Explanation: Problem “zip two lists”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def zip_two_lists(*args, **kwargs):
    """TODO: implement zip_two_lists as described in the handbook."""
    return None
```

```
import unittest
from problems.zip_two_lists import zip_two_lists
class TestZipTwoLists(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(zip_two_lists())
```

## 068. Group By Key

**Statement:** Implement problem “group by key” in Python.
Explanation: Problem “group by key”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def group_by_key(*args, **kwargs):
    """TODO: implement group_by_key as described in the handbook."""
    return None
```

```
import unittest
from problems.group_by_key import group_by_key
class TestGroupByKey(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(group_by_key())
```

## 069. Map Reduce Word Count

**Statement:** Implement problem “map reduce word count” in Python.
Explanation: Problem “map reduce word count”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def map_reduce_word_count(*args, **kwargs):
    """TODO: implement map_reduce_word_count as described in the handbook."""
    return None
```

```
import unittest
from problems.map_reduce_word_count import map_reduce_word_count
class TestMapReduceWordCount(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(map_reduce_word_count())
```

## 070. Tail F Like

**Statement:** Implement problem “tail f like” in Python.
Explanation: Problem “tail f like”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def tail_f_like(*args, **kwargs):
    """TODO: implement tail_f_like as described in the handbook."""
    return None
```

```
import unittest
from problems.tail_f_like import tail_f_like
class TestTailFLike(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(tail_f_like())
```

# Python Interview Handbook — Batch 8

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 071. Parse Csv Line

**Statement:** Implement problem “parse csv line” in Python.
Explanation: Problem “parse csv line”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def parse_csv_line(*args, **kwargs):
    """TODO: implement parse_csv_line as described in the handbook."""
    return None
```

```
import unittest
from problems.parse_csv_line import parse_csv_line
class TestParseCsvLine(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(parse_csv_line())
```

## 072. Url Shortener Encode Decode

**Statement:** Implement problem “url shortener encode decode” in Python.
Explanation: Problem “url shortener encode decode”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def url_shortener_encode_decode(*args, **kwargs):
    """TODO: implement url_shortener_encode_decode as described in the handbook."""
    return None
```

```
import unittest
from problems.url_shortener_encode_decode import url_shortener_encode_decode
class TestUrlShortenerEncodeDecode(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(url_shortener_encode_decode())
```

## 073. Rate Limiter Fixed Window

**Statement:** Implement problem “rate limiter fixed window” in Python.
Explanation: Problem “rate limiter fixed window”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def rate_limiter_fixed_window(*args, **kwargs):
    """TODO: implement rate_limiter_fixed_window as described in the handbook."""
    return None
```

```
import unittest
from problems.rate_limiter_fixed_window import rate_limiter_fixed_window
class TestRateLimiterFixedWindow(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(rate_limiter_fixed_window())
```

## 074. Token Bucket Limiter

**Statement:** Implement problem “token bucket limiter” in Python.
Explanation: Problem “token bucket limiter”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def token_bucket_limiter(*args, **kwargs):
    """TODO: implement token_bucket_limiter as described in the handbook."""
    return None
```

```
import unittest
from problems.token_bucket_limiter import token_bucket_limiter
class TestTokenBucketLimiter(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(token_bucket_limiter())
```

## 075. Debounce Function

**Statement:** Implement problem “debounce function” in Python.
Explanation: Problem “debounce function”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def debounce_function(*args, **kwargs):
    """TODO: implement debounce_function as described in the handbook."""
    return None
```

```
import unittest
from problems.debounce_function import debounce_function
class TestDebounceFunction(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(debounce_function())
```

## 076. Throttle Function

**Statement:** Implement problem “throttle function” in Python.
Explanation: Problem “throttle function”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def throttle_function(*args, **kwargs):
    """TODO: implement throttle_function as described in the handbook."""
    return None
```

```
import unittest
from problems.throttle_function import throttle_function
class TestThrottleFunction(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(throttle_function())
```

## 077. Decorator Timing

**Statement:** Implement problem “decorator timing” in Python.
Explanation: Problem “decorator timing”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def decorator_timing(*args, **kwargs):
    """TODO: implement decorator_timing as described in the handbook."""
    return None
```

```
import unittest
from problems.decorator_timing import decorator_timing
class TestDecoratorTiming(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(decorator_timing())
```

## 078. Context Manager Example

**Statement:** Implement problem “context manager example” in Python.
Explanation: Problem “context manager example”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def context_manager_example(*args, **kwargs):
    """TODO: implement context_manager_example as described in the handbook."""
    return None
```

```
import unittest
from problems.context_manager_example import context_manager_example
class TestContextManagerExample(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(context_manager_example())
```

## 079. Pickle Serialize

**Statement:** Implement problem “pickle serialize” in Python.
Explanation: Problem “pickle serialize”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def pickle_serialize(*args, **kwargs):
    """TODO: implement pickle_serialize as described in the handbook."""
    return None
```

```
import unittest
from problems.pickle_serialize import pickle_serialize
class TestPickleSerialize(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(pickle_serialize())
```

## 080. Json Serialize Custom

**Statement:** Implement problem “json serialize custom” in Python.
Explanation: Problem “json serialize custom”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def json_serialize_custom(*args, **kwargs):
    """TODO: implement json_serialize_custom as described in the handbook."""
    return None
```

```
import unittest
from problems.json_serialize_custom import json_serialize_custom
class TestJsonSerializeCustom(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(json_serialize_custom())
```

# Python Interview Handbook — Batch 9

Generated: 2025-09-13 02:00:06Z (UTC)

## Python Theory & Cheatsheet

```
PYTHON FUNDAMENTALS & CHEATSHEET
--------------------------------
- Data Types: int, float, str, bool, list, tuple, dict, set
- Comprehensions: [x for x in ...], {k:v for ...}, {x for ...}
- Functions: def, *args, **kwargs, closures
- OOP: classes, inheritance, dunder methods (__init__, __repr__)
- Decorators: @decorator, wraps; Context Managers: with, __enter__/__exit__
- Errors: try/except/else/finally; raise
- Iterators & Generators: iter(), next(), yield
- Useful libs: itertools, functools, collections, heapq, bisect
- Tips: enumerate, zip, sorted key=, slicing, unpacking
```

## 081. Thread Safe Counter

**Statement:** Implement problem “thread safe counter” in Python.
Explanation: Problem “thread safe counter”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def thread_safe_counter(*args, **kwargs):
    """TODO: implement thread_safe_counter as described in the handbook."""
    return None
```

```
import unittest
from problems.thread_safe_counter import thread_safe_counter
class TestThreadSafeCounter(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(thread_safe_counter())
```

## 082. Async Fetch Example

**Statement:** Implement problem “async fetch example” in Python.
Explanation: Problem “async fetch example”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def async_fetch_example(*args, **kwargs):
    """TODO: implement async_fetch_example as described in the handbook."""
    return None
```

```
import unittest
from problems.async_fetch_example import async_fetch_example
class TestAsyncFetchExample(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(async_fetch_example())
```

## 083. Producer Consumer Queue

**Statement:** Implement problem “producer consumer queue” in Python.
Explanation: Problem “producer consumer queue”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def producer_consumer_queue(*args, **kwargs):
    """TODO: implement producer_consumer_queue as described in the handbook."""
    return None
```

```
import unittest
from problems.producer_consumer_queue import producer_consumer_queue
class TestProducerConsumerQueue(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(producer_consumer_queue())
```

## 084. Exponential Backoff Retry

**Statement:** Implement problem “exponential backoff retry” in Python.
Explanation: Problem “exponential backoff retry”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def exponential_backoff_retry(*args, **kwargs):
    """TODO: implement exponential_backoff_retry as described in the handbook."""
    return None
```

```
import unittest
from problems.exponential_backoff_retry import exponential_backoff_retry
class TestExponentialBackoffRetry(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(exponential_backoff_retry())
```

## 085. Retry Decorator

**Statement:** Implement problem “retry decorator” in Python.
Explanation: Problem “retry decorator”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def retry_decorator(*args, **kwargs):
    """TODO: implement retry_decorator as described in the handbook."""
    return None
```

```
import unittest
from problems.retry_decorator import retry_decorator
class TestRetryDecorator(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(retry_decorator())
```

## 086. Binary Search Tree Insert

**Statement:** Implement problem “binary search tree insert” in Python.
Explanation: Problem “binary search tree insert”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_search_tree_insert(*args, **kwargs):
    """TODO: implement binary_search_tree_insert as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_search_tree_insert import binary_search_tree_insert
class TestBinarySearchTreeInsert(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_search_tree_insert())
```

## 087. Binary Search Tree Delete

**Statement:** Implement problem “binary search tree delete” in Python.
Explanation: Problem “binary search tree delete”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_search_tree_delete(*args, **kwargs):
    """TODO: implement binary_search_tree_delete as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_search_tree_delete import binary_search_tree_delete
class TestBinarySearchTreeDelete(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_search_tree_delete())
```

## 088. Union Find

**Statement:** Implement problem “union find” in Python.
Explanation: Problem “union find”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def union_find(*args, **kwargs):
    """TODO: implement union_find as described in the handbook."""
    return None
```

```
import unittest
from problems.union_find import union_find
class TestUnionFind(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(union_find())
```

## 089. Graph Cycle Detection

**Statement:** Implement problem “graph cycle detection” in Python.
Explanation: Problem “graph cycle detection”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def graph_cycle_detection(*args, **kwargs):
    """TODO: implement graph_cycle_detection as described in the handbook."""
    return None
```

```
import unittest
from problems.graph_cycle_detection import graph_cycle_detection
class TestGraphCycleDetection(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(graph_cycle_detection())
```

## 090. Topological Sort

**Statement:** Implement problem “topological sort” in Python.
Explanation: Problem “topological sort”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def topological_sort(*args, **kwargs):
    """TODO: implement topological_sort as described in the handbook."""
    return None
```

```
import unittest
from problems.topological_sort import topological_sort
class TestTopologicalSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(topological_sort())
```