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())
```