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())
```