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())
```