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())
```