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())
```