Python Interview Handbook — Batch 9

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
```

081. Thread Safe Counter

Statement: Implement problem “thread safe counter” in Python.
Explanation: Problem “thread safe counter”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def thread_safe_counter(*args, **kwargs):
    """TODO: implement thread_safe_counter as described in the handbook."""
    return None
```

```
import unittest
from problems.thread_safe_counter import thread_safe_counter
class TestThreadSafeCounter(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(thread_safe_counter())
```

082. Async Fetch Example

Statement: Implement problem “async fetch example” in Python.
Explanation: Problem “async fetch example”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def async_fetch_example(*args, **kwargs):
    """TODO: implement async_fetch_example as described in the handbook."""
    return None
```

```
import unittest
from problems.async_fetch_example import async_fetch_example
class TestAsyncFetchExample(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(async_fetch_example())
```

083. Producer Consumer Queue

Statement: Implement problem “producer consumer queue” in Python.
Explanation: Problem “producer consumer queue”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def producer_consumer_queue(*args, **kwargs):
    """TODO: implement producer_consumer_queue as described in the handbook."""
    return None
```

```
import unittest
from problems.producer_consumer_queue import producer_consumer_queue
class TestProducerConsumerQueue(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(producer_consumer_queue())
```

084. Exponential Backoff Retry

Statement: Implement problem “exponential backoff retry” in Python.
Explanation: Problem “exponential backoff retry”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def exponential_backoff_retry(*args, **kwargs):
    """TODO: implement exponential_backoff_retry as described in the handbook."""
    return None
```

```
import unittest
from problems.exponential_backoff_retry import exponential_backoff_retry
class TestExponentialBackoffRetry(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(exponential_backoff_retry())
```

085. Retry Decorator

Statement: Implement problem “retry decorator” in Python.
Explanation: Problem “retry decorator”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def retry_decorator(*args, **kwargs):
    """TODO: implement retry_decorator as described in the handbook."""
    return None
```

```
import unittest
from problems.retry_decorator import retry_decorator
class TestRetryDecorator(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(retry_decorator())
```

086. Binary Search Tree Insert

Statement: Implement problem “binary search tree insert” in Python.
Explanation: Problem “binary search tree insert”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_search_tree_insert(*args, **kwargs):
    """TODO: implement binary_search_tree_insert as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_search_tree_insert import binary_search_tree_insert
class TestBinarySearchTreeInsert(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_search_tree_insert())
```

087. Binary Search Tree Delete

Statement: Implement problem “binary search tree delete” in Python.
Explanation: Problem “binary search tree delete”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def binary_search_tree_delete(*args, **kwargs):
    """TODO: implement binary_search_tree_delete as described in the handbook."""
    return None
```

```
import unittest
from problems.binary_search_tree_delete import binary_search_tree_delete
class TestBinarySearchTreeDelete(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_search_tree_delete())
```

088. Union Find

Statement: Implement problem “union find” in Python.
Explanation: Problem “union find”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def union_find(*args, **kwargs):
    """TODO: implement union_find as described in the handbook."""
    return None
```

```
import unittest
from problems.union_find import union_find
class TestUnionFind(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(union_find())
```

089. Graph Cycle Detection

Statement: Implement problem “graph cycle detection” in Python.
Explanation: Problem “graph cycle detection”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def graph_cycle_detection(*args, **kwargs):
    """TODO: implement graph_cycle_detection as described in the handbook."""
    return None
```

```
import unittest
from problems.graph_cycle_detection import graph_cycle_detection
class TestGraphCycleDetection(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(graph_cycle_detection())
```

090. Topological Sort

Statement: Implement problem “topological sort” in Python.
Explanation: Problem “topological sort”. Outline brute-force vs optimized approaches, edge cases, and complexity.

```
def topological_sort(*args, **kwargs):
    """TODO: implement topological_sort as described in the handbook."""
    return None
```

```
import unittest
from problems.topological_sort import topological_sort
class TestTopologicalSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(topological_sort())
```