Python Interview Handbook — Batch 2

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
```

011. Binary Search

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

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

```
import unittest
from problems.binary_search import binary_search
class TestBinarySearch(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(binary_search())
```

012. Merge Sort

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

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

```
import unittest
from problems.merge_sort import merge_sort
class TestMergeSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(merge_sort())
```

013. Quick Sort

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

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

```
import unittest
from problems.quick_sort import quick_sort
class TestQuickSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(quick_sort())
```

014. Kth Largest

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

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

```
import unittest
from problems.kth_largest import kth_largest
class TestKthLargest(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(kth_largest())
```

015. Kadane Max Subarray

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

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

```
import unittest
from problems.kadane_max_subarray import kadane_max_subarray
class TestKadaneMaxSubarray(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(kadane_max_subarray())
```

016. Permutations

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

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

```
import unittest
from problems.permutations import permutations
class TestPermutations(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(permutations())
```

017. Combinations

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

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

```
import unittest
from problems.combinations import combinations
class TestCombinations(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(combinations())
```

018. Power Set

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

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

```
import unittest
from problems.power_set import power_set
class TestPowerSet(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(power_set())
```

019. Fib Memo

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

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

```
import unittest
from problems.fib_memo import fib_memo
class TestFibMemo(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(fib_memo())
```

020. Nth Fibonacci Iterative

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

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

```
import unittest
from problems.nth_fibonacci_iterative import nth_fibonacci_iterative
class TestNthFibonacciIterative(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(nth_fibonacci_iterative())
```