Python Interview Handbook — Batch 1

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
```

001. Reverse String

Statement: Implement problem “reverse string” in Python.
Explanation: Problem “reverse string”. Outline brute-force vs optimized approaches, edge cases, and complexity. Uses Python reversed() + join(). Time O(n), Space O(n).

```
def reverse_string(s: str) -> str:
    """Return reversed string using join/reversed."""
    return ''.join(reversed(s))
```

```
import unittest
from problems.001_reverse_string import reverse_string
class TestReverseString(unittest.TestCase):
    def test_basic(self):
        self.assertEqual(reverse_string('abc'), 'cba')
```

002. Is Palindrome

Statement: Implement problem “is palindrome” in Python.
Explanation: Problem “is palindrome”. Outline brute-force vs optimized approaches, edge cases, and complexity. Clean input then compare to reverse. Time O(n), Space O(n).

```
def is_palindrome(s: str) -> bool:
    s = ''.join(c.lower() for c in s if c.isalnum())
    return s == s[::-1]
```

```
import unittest
from problems.002_is_palindrome import is_palindrome
class TestIsPalindrome(unittest.TestCase):
    def test_examples(self):
        self.assertTrue(is_palindrome('A man, a plan, a canal: Panama'))
        self.assertFalse(is_palindrome('hello'))
```

003. Char Frequency

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

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

```
import unittest
from problems.char_frequency import char_frequency
class TestCharFrequency(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(char_frequency())
```

004. First Non Repeated Char

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

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

```
import unittest
from problems.first_non_repeated_char import first_non_repeated_char
class TestFirstNonRepeatedChar(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(first_non_repeated_char())
```

005. Second Largest

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

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

```
import unittest
from problems.second_largest import second_largest
class TestSecondLargest(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(second_largest())
```

006. Remove Duplicates

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

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

```
import unittest
from problems.remove_duplicates import remove_duplicates
class TestRemoveDuplicates(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(remove_duplicates())
```

007. Rotate List K

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

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

```
import unittest
from problems.rotate_list_k import rotate_list_k
class TestRotateListK(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(rotate_list_k())
```

008. Merge Two Sorted Lists

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

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

```
import unittest
from problems.merge_two_sorted_lists import merge_two_sorted_lists
class TestMergeTwoSortedLists(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(merge_two_sorted_lists())
```

009. Linked List Cycle

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

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

```
import unittest
from problems.linked_list_cycle import linked_list_cycle
class TestLinkedListCycle(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(linked_list_cycle())
```

010. Detect Cycle Linked List

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

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

```
import unittest
from problems.detect_cycle_linked_list import detect_cycle_linked_list
class TestDetectCycleLinkedList(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(detect_cycle_linked_list())
```