Python Interview Handbook — Batch 7

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
```

061. Heap Sort

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

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

```
import unittest
from problems.heap_sort import heap_sort
class TestHeapSort(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(heap_sort())
```

062. Bucket Sort Simple

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

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

```
import unittest
from problems.bucket_sort_simple import bucket_sort_simple
class TestBucketSortSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(bucket_sort_simple())
```

063. Counting Sort Simple

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

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

```
import unittest
from problems.counting_sort_simple import counting_sort_simple
class TestCountingSortSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(counting_sort_simple())
```

064. Radix Sort Simple

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

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

```
import unittest
from problems.radix_sort_simple import radix_sort_simple
class TestRadixSortSimple(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(radix_sort_simple())
```

065. Flatten Nested List

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

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

```
import unittest
from problems.flatten_nested_list import flatten_nested_list
class TestFlattenNestedList(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(flatten_nested_list())
```

066. Flatten Dict Keys

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

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

```
import unittest
from problems.flatten_dict_keys import flatten_dict_keys
class TestFlattenDictKeys(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(flatten_dict_keys())
```

067. Zip Two Lists

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

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

```
import unittest
from problems.zip_two_lists import zip_two_lists
class TestZipTwoLists(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(zip_two_lists())
```

068. Group By Key

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

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

```
import unittest
from problems.group_by_key import group_by_key
class TestGroupByKey(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(group_by_key())
```

069. Map Reduce Word Count

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

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

```
import unittest
from problems.map_reduce_word_count import map_reduce_word_count
class TestMapReduceWordCount(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(map_reduce_word_count())
```

070. Tail F Like

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

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

```
import unittest
from problems.tail_f_like import tail_f_like
class TestTailFLike(unittest.TestCase):
    def test_placeholder(self):
        self.assertIsNone(tail_f_like())
```