Python for Coding Interviews - Everything You Need to Know

Inspired by **NeetCode**

📑 Table of Contents

Division in Python

- `/` → always returns **float**.
- `//` → floor division.

```
5 // 2   # 2
-5 // 2  # -3 (floored toward -inf)
```

⚠️Gotcha: `//` floors toward negative infinity.

Modulo and fmod

- `%` → result has **same sign as divisor**.
- `math.fmod()` → result has **same sign as dividend**.

```
import math
print(-5 % 2)        # 1
print(math.fmod(-5,2)) # -1
print(5 % -2)        # -1
print(math.fmod(5,-2)) # 1
```

Lists and Arrays

- Python lists = **dynamic arrays**.
- `append()`, `pop()` at end → O(1).
- `insert(i, x)` or `pop(i)` in middle/front → O(n).
- Use `deque` for fast insert/remove at both ends.
- Index read/write → O(1).

```
nums = [1, 2, 3]
nums[2] = 99
```

Enumerate

Adds index to iterable.

```
fruits = ['apple', 'banana', 'cherry']
for i, f in enumerate(fruits, start=1):
    print(i, f)
# 1 apple, 2 banana, 3 cherry
```

Zip

Pairs elements from multiple iterables.

```
names = ['Alice', 'Bob']
ages = [25, 30]
for n, a in zip(names, ages):
    print(n, a)
```

Sorting

- Ascending by default.
- Use `reverse=True` for descending.
- Two ways:
- `list.sort()` → in-place
- `sorted(list)` → returns new list

With key

```
words.sort(key=len)
words.sort(key=lambda w: w[-1])
```

✅**Summary**: Use `.sort()` or `sorted()`, add `key=` and `reverse=` as needed.

List Comprehension

Syntax:

```
[expression for item in iterable if condition]
```

Example:

```
students = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 40},
    {"name": "Charlie", "score": 72}
]
passed = [s["name"].upper() for s in students if s["score"] >= 50]
# ['ALICE', 'CHARLIE']
```

✅**Usage**: loop + filter + transform → one line.

2D Lists

Correct initialization:

```
matrix = [[0 for _ in range(3)] for _ in range(3)]
```

❌Wrong:

```
matrix = [[0]*3]*3  # rows share same object
```

Strings

- **Immutable** → updates create new string (O(n)).

Efficient update

```
s = "Hello"
chars = list(s)
chars[1] = 'a'
s = "".join(chars)  # Hallo
```

Joining

```
words = ["Hello", "World"]
" ".join(words)  # "Hello World"
```

✅Use `join` for efficiency.

HashSet & HashMap

HashSet (set):

```
s = {1, 2, 3}
s.add(4)
print(3 in s)
```

HashMap (dict):

```
d = {"a": 1, "b": 2}
d["c"] = 3
print(d.get("x", "Not found"))
```

✅Summary: `set` = unique elements, `dict` = key-value pairs.

Tuple

- Immutable, can store heterogeneous data.
- Supports indexing, slicing, iteration.
- Usable as dict keys.

```
coords = (10, 20)
x, y = coords
d = {(1,2): "point"}
```

Heap

Use `heapq`.

```
import heapq
heap = []
heapq.heappush(heap, 10)
heapq.heappush(heap, 5)
heapq.heappush(heap, 20)
print(heap)  # [5, 10, 20]
```

Transform list → heap:

```
nums = [10, 5, 20, 1]
heapq.heapify(nums)
```

Pop:

```
val = heapq.heappop(heap)
```

Max heap → use negatives.

Errors: - `IndexError` when popping empty heap. - `TypeError` when mixing incomparable types.

Heap Time Complexity

Operation	Complexity
heapify(list)	O(n)
heappush	O(log n)
heappop	O(log n)
heap[0] (peek)	O(1)
heappushpop / replace	O(log n)
iterate heap	O(n)
pop all elements	O(n log n)

Nested Functions

Wrong (immutable reassignment)

```
def outer():
    x = 10
    def inner():
        x = x + 1  # Error
    inner()
```

Correct

- Using `nonlocal`:

```
def outer():
    x = 10
    def inner():
        nonlocal x
        x += 1
    inner()
    return x
```

- Passing & returning:

```
def outer():
    x = 10
```

```
    def inner(x):
        return x + 1
    x = inner(x)
    return x
```

Arguments & Returning Values

Immutable example

```
def outer(x):
    def inner(x):
        return x + 1
    return inner(x)
```

Mutable example

```
def outer(arr):
    def inner():
        arr.append(4)
        arr[0] = 100
    inner()
    return arr
```

✅Key: - Immutable → need `return` or `nonlocal`. - Mutable → updated in-place.