

# Data Types

Absolutely — knowing your **core data types** inside and out is essential for **LeetCode** and **coding interviews**. These are the types you'll **see constantly** and will use to build efficient solutions.

Here's a curated list of the **must-know data types** for LeetCode — including how to use them, when to choose them, and a one-liner to remember what they're best at.

---

## ☐☐ LeetCode Data Types You Should Know (Python)

---

### ☐1. `int` – Whole numbers

Used for counting, math, binary, etc.

```
n = 42
```

☐☐ Use cases:

- Loops, math, binary search, bit manipulation
- 

### ☐2. `float` – Decimal numbers

Used rarely, but important for precision-sensitive problems.

```
pi = 3.14
```

☐☐ Watch out for rounding errors.

---

### ☐3. `str` – Strings

## High-frequency on LeetCode

```
s = "leetcode"
```

Know:

- `s[::-1]` – reverse
  - `s.split()` – split by whitespace
  - `s.isalnum()` – alphanumeric check
  - `ord(c)`, `chr(n)` – ASCII value
- 

## 4. list – Ordered, mutable collection

Most used structure in LeetCode

```
arr = [1, 2, 3]
```

Know:

- `arr.append(x)`
  - `arr.pop()`, `arr.pop(0)`
  - `arr[::-1]`
  - `arr.sort()` VS `sorted(arr)`
  - Slicing, binary search, sliding window
- 

## 5. tuple – Ordered, immutable collection

Used for keys in sets/dicts or returning multiple values.

```
t = (1, 2)
```

Know:

- `x, y = t`
  - Useful as hashable keys: `visited[(x, y)] = True`
- 

## 6. set – Unordered, unique items

Great for fast lookup and duplicates

```
s = set([1, 2, 3])
```

☐ Know:

- `x in s` →  $O(1)$
  - `set(list)` removes duplicates
  - Set operations: `union`, `intersection`, `difference`
- 

## ☐7. `dict` – Key-value pairs

☐ Critical for hash maps / fast lookups

```
d = {"a": 1, "b": 2}
```

☐ Know:

- `d.get(k, default)`
  - `defaultdict(list)`
  - Frequency count: `Counter(s)`
  - Keys/values/items iteration
- 

## ☐8. `collections.deque` – Double-ended queue

✂ Fast `append` and `popleft()` → great for BFS

```
from collections import deque  
q = deque([1, 2])  
q.append(3)  
q.popleft()
```

---

## ☐9. `heapq` / Min Heap (via list)

☐ Use for priority queue, top K elements

```
import heapq
h = [5, 3, 8]
heapq.heapify(h)
heapq.heappop(h)
```

☐ Python only has min-heaps — invert values for max-heap.

---

## ☐ 10. `collections.Counter` – Fast frequency counting

```
from collections import Counter
c = Counter("aabbcc")
```

☐ Useful for:

- Anagram check
  - Majority element
  - Multiset behavior
- 

## ☐ TL;DR: Memorize These for LeetCode

Type	What it's good for
<code>int</code>	Counting, math
<code>float</code>	Precision, rarely needed
<code>str</code>	Manipulation, parsing, search
<code>list</code>	Arrays, stacks, sliding window
<code>tuple</code>	Hashable keys, return multiple items
<code>set</code>	Uniqueness, fast lookup
<code>dict</code>	Key-value, counting, hash maps
<code>deque</code>	BFS, queue (O(1) pops)
<code>heapq</code>	Priority queue, top K problems

Type	What it's good for
Counter	Fast frequency counting

Would you like:

- Anki cards for this?
- Daily drills where you implement key operations from memory?
- A printable reference sheet or PDF?

Revision #1  
Created 5 April 2025 22:46:12 by victor  
Updated 5 April 2025 22:46:23 by victor