LEET Handbook

🐍 Python Basics for DSA

> Purpose of this part:
>
> Make Python feel **familiar and non-scary** before touching LeetCode.
>
> No shortcuts. Clear thinking > fancy code.

1️⃣ How Python Code Is Read (Very Important)

Python runs **line by line**, from **top to bottom**.

```
a = 5
b = 3
c = a + b
print(c)
```

📌 Python doesn’t “guess” — it **executes in order**.

2️⃣ Variables (Storing Values)

A variable is just a **box that stores a value**.

```
x = 10
name = "Alice"
pi = 3.14
```

Python automatically understands the type.

You **don’t need to declare types** like C / Java.

Reassigning Variables

```
x = 10
x = 20   # old value is replaced
```

📌 Latest value always matters.

3️⃣ Data Types (Beginner Level)

🔹 Integer (`int`)

Whole numbers

```
a = 5
b = -3
```

🔹 Float (`float`)

Decimal numbers

```
x = 3.14
```

🔹 String (`str`)

Text inside quotes

```
s = "hello"
```

🔹 Boolean (`bool`)

```
True
False
```

Used in conditions.

4️⃣ Input & Output (Very Common in DSA)

Printing Output

```
print("Hello")
print(5)
```

Taking Input

```
x = input()
```

⚠️ Input is **always a string by default**

Convert it:

```
n = int(input())
```

Multiple Inputs (Very Important)

```
a, b = map(int, input().split())
```

Meaning:

- `input()` → take input
- `split()` → split by space
- `int` → convert to integer

5️⃣ Operators (Basic but Essential)

🔹 Arithmetic

```
+ - * / % **
```

Example:

```
5 % 2   # remainder → 1
2 ** 3  # power → 8
```

🔹 Comparison

```
== != > < >= <=
```

Returns **True or False**

🔹 Assignment

```
x += 1  # same as x = x + 1
```

6️⃣ Conditions (if / elif / else)

Used to **make decisions**

```
if x > 0:
    print("positive")
elif x == 0:
    print("zero")
else:
    print("negative")
```

📌 Indentation (spaces) is **mandatory** in Python.

7️⃣ Loops (Core of DSA)

🔹 for loop

Used when number of steps is known.

```
for i in range(5):
    print(i)
```

Output:

```
0 1 2 3 4
```

🔹 while loop

Used when condition matters more than count.

```
i = 0
while i < 5:
    print(i)
    i +=
```

8️⃣ range() (Beginner Explanation)

```
range(5)       # 0 to 4
range(1, 6)    # 1 to 5
range(0, 10, 2) # step of 2
```

📌 `range()` **does not include the last value.**

9️⃣ Functions (Don’t Be Scared 😄)

A function is a **reusable block of code.**

```
def add(a, b):
    return a + b
```

Calling it:

```
result = add(2, 3)
```

📌 Functions help:

- Organize logic
- Avoid repetition
- Improve readability

🔟 Indentation & Common Errors ⚠️

Correct

```
if x > 0:
    print(x)
```

Wrong

```
if x > 0:
print(x)   # error
```

1️⃣1️⃣ Debugging Habit (Very Important)

When confused → **print values**

```
print(i, x)
```

📌 Debugging is NOT a weakness.

It's a **skill**.

🐍 Python Built-in Functions & Data Structures

> Purpose of this part:
>
> Help you **recall Python quickly while solving DSA problems**
>
> These are the tools you'll use **again and again on LeetCode**

1️⃣ Lists (The Backbone of DSA 🔥)

A **list** stores multiple values in one place.

```
arr = [1, 2, 3, 4]
```

Lists are:

- Ordered
- Mutable (can change)
- Allow duplicates

🔹 Accessing Elements

```
arr[0]   # first element
arr[-1]  # last element
```

🔹 Important List Functions (VERY COMMON)

1. append()

Adds **one element** at the end.

```
arr.append(5)
```

📌 Use when adding elements step by step.

---

2. extend()

Adds **multiple elements**.

```
arr.extend([6, 7])
```

⚠️ Difference:

```
arr.append([6, 7])   # WRONG for most cases
```

---

3. pop()

Removes & returns last element.

```
x = arr.pop()
```

📌 Used in **stack** type problems.

---

4. remove()

Removes first occurrence of a value.

```
arr.remove(3)
```

⚠️ Error if value doesn’t exist.

---

5. sort()

Sorts the list **in-place**.

```
arr.sort()
arr.sort(reverse=True)
```

📌 Used in greedy & two-pointer problems.

---

6. sorted()

Returns a **new sorted list**.

```
new_arr = sorted(arr)
```

📌 Original list stays unchanged.

---

7. reverse()

```
arr.reverse()
```

---

8. len()

```
len(arr)
```

9. count()

```
arr.count(2)
```

📌 Rare, but useful sometimes.

🔹 Looping Through Lists

```
for x in arr:
    print(x)
```

Index + value:

```
for i in range(len(arr)):
    print(i, arr[i])
```

2️⃣ Strings (Immutable but Powerful)

Strings are **text data**.

```
s = "hello"
```

⚠️ Strings **cannot be modified directly**.

🔹 Important String Functions

1. len()

```
len(s)
```

2. Indexing & Slicing

```
s[0]      # 'h'
s[1:4]    # 'ell'
s[::-1]   # reverse string
```

📌 Used in palindrome problems.

3. lower() / upper()

```
s.lower()
s.upper()
```

📌 Helpful for comparison problems.

4. strip()

Removes spaces from both ends.

```
s.strip()
```

5. split()

```
s = "a b c"
s.split()
```

Result:

```
['a', 'b', 'c']
```

📌 Used heavily in input handling.

---

6. join()

```
"".join(['a', 'b', 'c'])
```

Result:

```
"abc"
```

7. replace()

```
s.replace("a", "b")
```

---

8. Character Checks

```
s.isdigit()
s.isalpha()
s.isalnum()
```

3️⃣ Dictionaries (HashMaps ❤️)

Used when you need **frequency** or **fast lookup**.

```
freq = {}
```

🔹 Important Dictionary Functions

1. Add / Update

```
freq['a'] = 1
```

---

2. get() (MOST IMPORTANT)

```
freq.get('a', 0)
```

📌 Prevents KeyError

📌 Default value if key doesn’t exist

---

3. Frequency Counting Pattern

```
for x in arr:
    freq[x] = freq.get(x, 0) + 1
```

📌 Used in:

- Two Sum
- Anagrams
- Majority Element

---

4. Looping Through Dictionary

```
for key, value in freq.items():
    print(key, value)
```

---

5. Checking Key Existence

```
if 'a' in freq:
    print("exists")
```

4️⃣ Sets (Fastest Lookups 🚀)

A **set** stores unique values.

```
s = set(arr)
```

🔹 Set Functions

```
s.add(5)
s.remove(3)
```

Check:

```
if 5 in s:
    print("found")
```

📌 Used to:

- Remove duplicates
- Check existence quickly

5️⃣ Extremely Useful Built-in Functions

```
max(arr)
min(arr)
sum(arr)
```

---

enumerate() (Very Helpful)

```
for index, value in enumerate(arr):
    print(index, value)
```

📌 Cleaner than `range(len())`

---

range() (Revision)

```
range(5)      # 0 to 4
range(1, 6)   # 1 to 5
```

🧠 Important DSA Patterns

> Purpose of this part:
>
> Stop feeling “blank” when you open a LeetCode problem.
>
> Patterns help you recognize **how to approach**, not just code.

1️⃣ Why Patterns Matter in DSA

Most problems are **repeats in disguise**.

If you know patterns:

- You don’t panic
- You know *where to start*
- Coding becomes mechanical

💡 Think:

> “I’ve seen something like this before.”

2️⃣ Linear Scan (Most Basic Pattern)

When to use:

- Check every element
- Find max / min
- Count something

Example:

```
for x in arr:
    print(x)
```

📌 Many Easy problems are just **smart linear scans**.

3️⃣ Two Pointer Pattern 🔥

When to use:

- Sorted array
- Pair problems
- Reverse / compare from both ends

Idea:

Use **two indices**, one from start, one from end.

Template:

```
l = 0
r = len(arr) - 1

while l < r:
    # logic here
    l += 1
    r -= 1
```

Common Problems:

- Two Sum (sorted)
- Palindrome check
- Reverse array

4️⃣ Sliding Window Pattern 🔥🔥

When to use:

- Subarrays
- “Maximum / minimum in a window”
- Continuous range problems

Idea:

Maintain a **window** instead of recalculating again and again.

Template:

```
left = 0
for right in range(len(arr)):
    # include arr[right]

    while condition_not_met:
        # remove arr[left]
        left += 1
```

Common Problems:

- Longest substring
- Max sum subarray
- Window size problems

📌 This pattern saves **time complexity**.

5️⃣ Frequency Map Pattern (Dictionary ❤️)

When to use:

- Counting
- Anagrams
- Repeated elements

Core Idea:

Store **how many times something appears**.

Template:

```
freq = {}

for x in arr:
    freq[x] = freq.get(x, 0) + 1
```

Common Problems:

- Two Sum

- Majority element
- Valid anagram

6️⃣ Prefix Sum Pattern

When to use:

- Range sum queries
- Subarray sum problems

Idea:

Precompute sums so queries are fast.

Template:

```
prefix = [0]
for x in arr:
    prefix.append(prefix[-1] + x)
```

Sum from index `l` to `r`:

```
prefix[r+1] - prefix[l]
```

📌 Converts O(n²) → O(n)

7️⃣ Set for Fast Lookup

When to use:

- Check if element exists
- Remove duplicates

Template:

```
seen = set()

for x in arr:
    if x in seen:
        return True
    seen.add(x)
```

📌 Lookup is **very fast**.

8️⃣ Stack Pattern (Using List)

When to use:

- Previous greater/smaller
- Valid parentheses
- Undo-type logic

Template:

```
stack = []

for x in arr:
```

```
    stack.append(x)
    stack.pop()
```

📌 `append()` = push

📌 `pop()` = pop

9️⃣ Greedy Pattern (Choose Best Now)

When to use:

- Maximize / minimize result
- Sorted + decisions

Idea:

Take the **best option at the moment**.

📌 Often combined with `sort()`.

🔟 How to Identify the Pattern (IMPORTANT)

Ask these questions:

- Sorted array? → **Two Pointer**
- Subarray / window? → **Sliding Window**
- Counting? → **Dictionary**
- Repeated checking? → **Set**
- Range sum? → **Prefix Sum**

~Dibs