Problem: Validate Number in a String

Company: LinkedIn
Difficulty: Medium

Description

You're building an input validator for a form where users can type numbers. The field should accept:

- Integers (e.g., `"10"`, `"-10"`)
- Real numbers with a decimal (e.g., `"10.1"`, `"-10.1"`, `".5"`, `"3."`)
- Scientific notation (e.g., `"1e5"`, `"-3.2E-7"`)

It should **reject** malformed inputs (e.g., `"a"`, `"x 1"`, `"a -2"`, `"-"`, `"1e"`, `"e9"`).

Task: Given a string `s`, return whether it represents a valid number.

Notes:

- Optional leading/trailing spaces may be allowed; internal spaces are not.
- Optional leading sign (+/-) is allowed.
- For scientific notation, an `e`/`E` must be followed by an **integer** (with optional sign).
- Decimal point can appear at most once and not after `e`/`E`.

Input Format

- A single string `s`.

Output Format

- Return `true` if `s` represents a valid number; otherwise `false`.

Constraints

- `1 ≤ len(s) ≤ 10^4`
- Characters may include digits `0-9`, signs `+/-`, decimal point `.`, exponent marker `e/E`, and spaces.
- No thousands separators.

Examples

Example 1

Input: `s = "10"`
Output: `true`
Explanation: Valid positive integer.

Example 2

Input: `s = "-10.1"`
Output: `true`
Explanation: Valid signed real number.

Example 3

Input: `s = "1e5"`
Output: `true`
Explanation: Valid scientific notation; exponent is an integer.

Example 4

Input: `s = "a -2"`
Output: `false`
Explanation: Contains invalid characters/space in between.

More edge cases

Input	Output	Reason
`" -90e3 "`	true	trims OK; `-90e3` valid
`"1e-3"`	true	exponent with sign
`".1"`	true	fractional w/o leading digit
`"3."`	true	trailing dot allowed (fractional part empty)
`"."`	false	needs at least one digit
`"1e"`	false	exponent missing digits
`"e9"`	false	mantissa missing
`"-"`	false	sign alone is not a number
`"+"`	false	same as above
`" 1 2 "`	false	internal space

Hints / Approach

You can solve this by:

1. **One-pass parser with flags**
    - Trim spaces.
    - Track: `seen_digit`, `seen_dot`, `seen_exp`, and `digit_after_exp`.
    - Rules:
        - Digits: update `seen_digit` (and `digit_after_exp` if after exponent).
        - Signs: allowed only at the start *or* immediately after `e/E`.
        - Dot: allowed once, and **not** after `e/E`.
        - `e/E`: allowed once, only if a digit has already appeared; must be followed by digits (optionally after a sign).
2. **DFA (finite-state machine)**
    - More formal but longer to implement.
3. **Regex** (compact but easy to get wrong). Example pattern (anchors + optional spaces):

```
^\s*[+-]?(
    (\d+(\.\d*)?)|(\.\d+)
)([eE][+-]?\d+)?\s*$
```

(Remove whitespace/newlines if you use it.)

Sample Solutions

Python (one-pass flags)

```python
def is_number(s: str) -> bool:
    s = s.strip()
    if not s:
        return False

    seen_digit = False
    seen_dot = False
    seen_exp = False
    digit_after_exp = True  # meaningful only if seen_exp becomes True

    for i, ch in enumerate(s):
        if ch.isdigit():
            seen_digit = True
            if seen_exp:
                digit_after_exp = True
        elif ch in ['+', '-']:
            # sign allowed only at pos 0 or right after e/E
            if i > 0 and s[i-1].lower() != 'e':
                return False
            # cannot be the last character
            if i == len(s) - 1:
                return False
        elif ch == '.':
            # dot not allowed after exponent and only once
            if seen_dot or seen_exp:
                return False
            seen_dot = True
        elif ch in ['e', 'E']:
            # exponent only once and only after at least one digit
            if seen_exp or not seen_digit:
                return False
```

```
            seen_exp = True
            digit_after_exp = False  # must see digits after exponent
            # cannot be the last character
            if i == len(s) - 1:
                return False
        else:
            return False

    return seen_digit and (not seen_exp or digit_after_exp)
```

Java (regex)

```
class Solution {
    public boolean isNumber(String s) {
        String pattern = "^\\s*[+-]?((\\d+(\\.\\d*)?)|(\\.\\d+))([eE][+-
]?\\d+)?\\s*$";
        return s != null && s.matches(pattern);
    }
}
```

Practice Links

- LeetCode: **Valid Number** (Problem 65)
- GeeksforGeeks: **Check if a given string is a valid number**

(Search titles above if direct links aren't allowed in your instructions.)

Video Explanations (YouTube)

- "Valid Number – Parsing / Regex vs DFA"
- "LeetCode 65: Valid Number – Clean One-Pass Approach"

Submission (for your POW instructions)

- Add code and **README.md** with:
  - Approach explanation (flags/DFA/regex)
  - Time/Space complexities (both `O(n)` / `O(1)`)
  - Edge cases tested
- Commit history showing refinements.
- Update your **GitHub repo**