

Mini Parser

Given a nested list of integers represented as a string, implement a parser to deserialize it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Note: You may assume that the string is well-formed:

- String is non-empty.
- String does not contain white spaces.
- String contains only digits `0-9`, `[`, `-`, `,`, `]`.

Example 1:

Given `s = "324"`,

You should return a `NestedInteger` object which contains a single integer 324.

Example 2:

Given `s = "[123,[456,[789]]]"`,

Return a `NestedInteger` object containing a nested list with 2 elements:

1. An integer containing value 123.
2. A nested list containing two elements:
 - i. An integer containing value 456.
 - ii. A nested list with one element:
 - a. An integer containing value 789.

Solution 1

Python using `eval`:

```
def deserialize(self, s):
    def nestedInteger(x):
        if isinstance(x, int):
            return NestedInteger(x)
        lst = NestedInteger()
        for y in x:
            lst.add(nestedInteger(y))
        return lst
    return nestedInteger(eval(s))
```

Python one-liner

```
def deserialize(self, s):
    return NestedInteger(s) if isinstance(s, int) else reduce(lambda a, x: a.add(
self.deserialize(x)) or a, s, NestedInteger()) if isinstance(s, list) else self.d
erialize(eval(s))
```

Python Golf (136 bytes or 31 bytes)

```
class Solution:deserialize=d=lambda S,s,N=NestedInteger:s<[]and N(s)or s<' 'and red
uce(lambda a,x:a.add(S.d(x))or a,s,N())or S.d(eval(s))
```

Or abusing how the judge judges (yes, this gets accepted):

```
class Solution:deserialize=eval
```

Python parsing char by char

Here I turned the input string into a list with sentinel for convenience.

```

def deserialize(self, s):
    def nestedInteger():
        num = ''
        while s[-1] in '1234567890-':
            num += s.pop()
        if num:
            return NestedInteger(int(num))
        s.pop()
        lst = NestedInteger()
        while s[-1] != ']':
            lst.add(nestedInteger())
            if s[-1] == ',':
                s.pop()
        s.pop()
        return lst
    s = list(' ' + s[::-1])
    return nestedInteger()

```

C++ using `istringstream`

```

class Solution {
public:
    NestedInteger deserialize(string s) {
        istringstream in(s);
        return deserialize(in);
    }
private:
    NestedInteger deserialize(istringstream &in) {
        int number;
        if (in >> number)
            return NestedInteger(number);
        in.clear();
        in.get();
        NestedInteger list;
        while (in.peek() != ']') {
            list.add(deserialize(in));
            if (in.peek() == ',')
                in.get();
        }
        in.get();
        return list;
    }
};

```

written by [StefanPochmann](#) original link [here](#)

Solution 2

This approach will just iterate through every char in the string (no recursion).

- If encounters '[', push current NestedInteger to stack and start a new one.
- If encounters ']', end current NestedInteger and pop a NestedInteger from stack to continue.
- If encounters ',', append a new number to curr NestedInteger, if this comma is not right after a brackets.
- Update index l and r, where l shall point to the start of a integer substring, while r shall points to the end+1 of substring.

Java Code:

```
public NestedInteger deserialize(String s) {
    if (s.isEmpty())
        return null;
    if (s.charAt(0) != '[') // ERROR: special case
        return new NestedInteger(Integer.valueOf(s));

    Stack<NestedInteger> stack = new Stack<>();
    NestedInteger curr = null;
    int l = 0; // l shall point to the start of a number substring;
               // r shall point to the end+1 of a number substring
    for (int r = 0; r < s.length(); r++) {
        char ch = s.charAt(r);
        if (ch == '[') {
            if (curr != null) {
                stack.push(curr);
            }
            curr = new NestedInteger();
            l = r+1;
        } else if (ch == ']') {
            String num = s.substring(l, r);
            if (!num.isEmpty())
                curr.add(new NestedInteger(Integer.valueOf(num)));
            if (!stack.isEmpty()) {
                NestedInteger pop = stack.pop();
                pop.add(curr);
                curr = pop;
            }
            l = r+1;
        } else if (ch == ',') {
            if (s.charAt(r-1) != ']') {
                String num = s.substring(l, r);
                curr.add(new NestedInteger(Integer.valueOf(num)));
            }
            l = r+1;
        }
    }

    return curr;
}
```

written by [AlexTheGreat](#) original link [here](#)

Solution 3

```
class Solution {
public:
    NestedInteger deserialize(string s) {
        int index = 0;
        char c = s[index];
        if (c == '[') {
            return parseList(s, index);
        } else {
            // starts with 0-9, '-'
            return parseNumber(s, index);
        }
    }

    NestedInteger parseList(string &s, int &index) {
        index++; // eat '['
        NestedInteger root;
        while (index < s.size()) {
            char c = s[index];
            if (c == '[') {
                root.add(parseList(s, index));
            } else if (isNumber(c) || c == '-') {
                root.add(parseNumber(s, index));
            } else if (c == ',') {
                // skip
                index++;
            } else if (c == ']') {
                break;
            }
        }
        index++; // eat ']'
        return root;
    }

    NestedInteger parseNumber(string &s, int &index) {
        int n = 0;
        int positive = 1; // flag for positive number
        if (s[index] == '-') {
            positive = -1;
            index++;
        }
        while (index < s.size()) {
            char c = s[index];
            if (isNumber(c)) {
                n = 10 * n + c - '0';
                index++;
            } else {
                break;
            }
        }
        return NestedInteger(n * positive);
    }

    bool isNumber(char c) {
        return '0' <= c && c <= '9';
    }
};
```

```
}  
};
```

written by [moevis](#) original link [here](#)

From [Leetcode](#).