Codility

CodeCheck Report: training96WZXH-99U

Test Name:

Check out Codility training tasks

Summary | Timeline

Tasks summary

Task	Time spent	Score
Brackets Python	2 min	100%

Total score

100%

Tasks Details

Easy

1. Brackets

Determine whether a given string of parentheses (multiple types) is properly nested.

Task Score	Correctness	Performance
100%	100%	100%

Task description

A string S consisting of N characters is considered to be *properly nested* if any of the following conditions is true:

- S is empty;
- S has the form "(U)" or "[U]" or "{U}" where U is a properly nested string;
- S has the form "VW" where V and W are properly nested strings.

For example, the string "{[()()]}" is properly nested but "([)()]" is not.

Write a function:

```
def solution(S)
```

that, given a string S consisting of N characters, returns 1 if S is properly nested and 0 otherwise.

For example, given S = "{[()()]}", the function should return 1 and given S = "([)()]", the function should return 0, as explained above.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [0..200,000];
- string S consists only of the following characters: "(", "{", "[", "]", "}" and/or ")".

Solution

Programming language used:	Python
Total time used:	2 minutes
Effective time used:	2 minutes
Notes:	*not defined yet*

Task timeline

19:03:31 — 19:04:35

Code: 19:04:35 UTC, py, final, score: 100

show code in pop-up

```
# you can write to stdout for debugging purposes, e.g.
# print("this is a debug message")

def solution(S):
    # write your code in Python 3.6
    if len(S) == 0:
        return 1
    if len(S) == 1 or len(S)%2 != 0:
        return 0
    if not ((S[0] == '(' or S[0] == '[' or S[0] == '{') and (S
        return 0

    stack_ = []
    for c in S:
        if stack_:
            el = stack_.pop()
            if el == '(' and c == '(':
                stack_.append(el)
                stack_.append(c)
            elif el == '(' and c == ')':
                pass
            elif el == '(' and c == '[':
                stack_.append(el)
                stack_.append(c)
            elif el == '(' and c == ']':
                stack_.append(el)
                stack_.append(c)
            elif el == '(' and c == '{':
                stack_.append(el)
                stack_.append(c)
            elif el == '(' and c == '}':
                stack_.append(el)
                stack_.append(c)
            elif el == '[' and c == '(':
                stack_.append(el)
                stack_.append(c)
            elif el == '[' and c == ')':
                stack_.append(el)
                stack_.append(c)
            elif el == '[' and c == '[':
                stack_.append(el)
                stack_.append(c)
            elif el == '[' and c == ']':
                pass
            elif el == '[' and c == '{':
                stack_.append(el)
                stack_.append(c)
            elif el == '[' and c == '}':
                stack_.append(el)
                stack_.append(c)
            elif el == '{' and c == '(':
                stack_.append(el)
                stack_.append(c)
            elif el == '{' and c == ')':
                stack_.append(el)
                stack_.append(c)
            elif el == '{' and c == '[':
                stack_.append(el)
                stack_.append(c)
            elif el == '{' and c == ']':
                stack_.append(el)
                stack_.append(c)
            elif el == '{' and c == '{':
                stack_.append(el)
                stack_.append(c)
            elif el == '{' and c == '}':
                pass
            else:
                stack_.append(el)
                stack_.append(c)
        else:
            stack_.append(c)

    if stack_:
        return 0
    else:
        return 1
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

expand all	Example tests	
▶	example1 example test 1	✔ OK
▶	example2 example test 2	✔ OK
expand all	**Correctness tests**	
▶	negative_match invalid structures	✔ OK
▶	empty empty string	✔ OK
▶	simple_grouped simple grouped positive and negative test, length=22	✔ OK
expand all	**Performance tests**	
▶	large1 simple large positive test, 100K ('s followed by 100K)'s +)(	✔ OK
▶	large2 simple large negative test, 10K+1 ('s followed by 10K)'s +)(+ ()	✔ OK
▶	large_full_ternary_tree tree of the form T=(TTT) and depth 11, length=177K+	✔ OK
▶	multiple_full_binary_trees sequence of full trees of the form T=(TT), depths [1..10..1], with/without some brackets at the end, length=49K+	✔ OK
▶	broad_tree_with_deep_paths string of the form [TTT...T] of 300 T's, each T being '({{...}})' nested 200-fold, length=120K+	✔ OK