

Tasks summary

Task	Time spent	Score
Brackets Python	2 min	100%

Total score



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](#)

```
1  # you can write to stdout for debugging purposes, e.g.
2  # print("this is a debug message")
3
4  def solution(S):
5      # write your code in Python 3.6
6      if len(S) == 0:
7          return 1
8      if len(S) == 1 or len(S)%2 != 0:
9          return 0
10     if not ((S[0] == '(' or S[0] == '[' or S[0] == '{') and (S
11         return 0
12
13     stack_ = []
14     for c in S:
15         if stack_:
16             el = stack_.pop()
17             if el == '(' and c == ')':
18                 stack_.append(el)
19                 stack_.append(c)
20             elif el == '(' and c == ')':
21                 pass
22             elif el == '(' and c == '[':
23                 stack_.append(el)
24                 stack_.append(c)
25             elif el == '(' and c == ']':
26                 stack_.append(el)
27                 stack_.append(c)
28             elif el == '(' and c == '{':
29                 stack_.append(el)
30                 stack_.append(c)
31             elif el == '(' and c == '}':
32                 stack_.append(el)
33                 stack_.append(c)
34             elif el == '[' and c == ')':
35                 stack_.append(el)
36                 stack_.append(c)
37             elif el == '[' and c == ')':
38                 stack_.append(el)
39                 stack_.append(c)
40             elif el == '[' and c == '[':
41                 stack_.append(c)
42                 stack_.append(c)
43             elif el == '[' and c == ']':
44                 pass
45             elif el == '[' and c == '{':
46                 stack_.append(el)
47                 stack_.append(c)
48             elif el == '[' and c == '}':
49                 stack_.append(el)
50                 stack_.append(c)
51             elif el == '{' and c == ')':
52                 stack_.append(el)
53                 stack_.append(c)
54             elif el == '{' and c == ')':
55                 stack_.append(el)
56                 stack_.append(c)
57             elif el == '{' and c == '[':
58                 stack_.append(el)
59                 stack_.append(c)
60             elif el == '{' and c == ']':
61                 stack_.append(el)
62                 stack_.append(c)
63             elif el == '{' and c == '{':
64                 stack_.append(c)
65                 stack_.append(c)
66             elif el == '{' and c == '}':
67                 pass
68             else:
69                 stack_.append(el)
70                 stack_.append(c)
71         else:
72             stack_.append(c)
73
74     if stack_:
75         return 0
76     else:
77         return 1
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: <b>O(N)</b>	
expand all	Example tests
▶ example1	OK
example test 1	
▶ example2	OK
example test 2	
expand all	Correctness tests
▶ negative_match	OK
invalid structures	
▶ empty	OK
empty string	
▶ simple_grouped	OK
simple grouped positive and negative test, length=22	
expand all	Performance tests
▶ large1	OK
simple large positive test, 100K ('s followed by 100K )s + )(	
▶ large2	OK
simple large negative test, 10K+1 ('s followed by 10K )s + )( + 0)	
▶ large_full_ternary_tree	OK
tree of the form T=(TTT) and depth 11, length=177K+	
▶ multiple_full_binary_trees	OK
sequence of full trees of the form T=(TT), depths [1..10..1], with/without some brackets at the end, length=49K+	
▶ broad_tree_with_deep_paths	OK
string of the form [TTT...T] of 300 T's, each T being '{{{...}}}' nested 200-fold, length=120K+	