



# DOM tree traversal

## Description:

One of the techniques we use at Medallia for automated regression testing consist on comparing the HTML output of different versions of our code. To do this, we convert the HTML to a "DOM" (Document Object Model) and compare the output. We also filter expected differences from the output via a simple whitelist.

Assume that a "DOM" representation is a tree containing Element and Content nodes:

- A Content node contains:
  - only its content
    - e.g. if the input is a "<body>some content</body>" then "some content" is the content of the content node
- An Element node contains:
  - a tag
    - e.g. if the input is a "<html>...</html>" then "html" is the tag of the element node
  - an optional id
    - e.g. if the input is "<div id='foo'>" then "foo" is the id of the element node
  - an optional list of children nodes
- The content of an Element node is the concatenation of all its **direct** Content children, separated by spaces
  - e.g. if the input is "<div>some<a>other</a>content</div>" then the content of "div" is "some content"

## Problem:

Given a DOM tree, write out its content subject to the following constraints:

- **One line per level of the tree**

- Each line should include the **tag, id** (if present) and **content** (if present) of all Element nodes in that level, in the **same order they appear** in the tree, with each term **separated by spaces**.
- If a whitelisted string is a **substring** of the tag, id or content of an Element node, then **that node and all its children should be ignored** and excluded from the output.

## Example

For the input DOM:

```
<html>
  <body id='content'>
    This
    <div id='wrapper1'>
      is a
      <div id='container1'>
        <div id='container2'>
          funny
        </div>
      </div>
      <div id='wrapper2'>
        enjoyable
      </div>
      little
    </div>
    good
    <div id='wrapper3'>
      harmless
    </div>
    example.
    <div id='wrapper4'>
    </div>
    <a id='link'>
      And a link.
    </a>
  </body>
</html>
```

with a whitelist:

```
["wrapper4", "a little"]
```

**Then the output should be:**

```
html
body content This good example.
div wrapper3 harmless a link And a link.
```

**Explanation:**

- "html" is the only element on the top of the tree and it doesn't have any content (everything is below "body").
  - "body" has the id "content" and it is the only element on that level. Its content ("This good example") is the concatenation of all its Content nodes.
  - "div wrapper3" is the first element in the third level because "div wrapper1" has been white listed (its combined content "is a little" contains the whitelisted string "a little"). Then "a link" is the second element of that level, given that "div wrapper4" has been white listed (its "id" contains "wrapper4"). Finally "And a link." is the content of the "<a id='link'>" element.
- 

**YOUR ANSWER**

*Draft saved 07:14 pm*

Python 2



```

1  #!/bin/python
2
3  import re
4  import collections
5
6  """
7  An 'Element' node of the DOM tree.
8  @param tag: string
9  @param id: string
10 @param children: array of nodes which could be either
    'Element' or 'Content' instances
11 """
12 class Element:
13     def __init__(self, tag, id, children):
14         self.tag = tag
15         self.id = id
16         self.children = children
17
18     """
19     A 'Content' node of the DOM tree
20     @param content: string
21     """
22 class Content:
23     def __init__(self, content):
24         self.content = content
25
26     """
27     A 'DOM' tree
28     @param root: an instance of 'Element' which represents to
        the root of the tree
29     """
30 class Dom:
31     def __init__(self, root):
32         self.root = root
33
34 def format(dom, whiteList):
35     """
36     Complete the function below.
37     @param dom: An instance of the Dom class
38     @param whiteList: An array of strings
39     @return the output that will be sent to the STDOUT
40     """
41     if dom is not None:
42         output = ""
43         queue = []
44         queue.append(dom.root)

```

```
44 ▼         while queue:
45             cur = queue.pop(0)
46 ▼         if cur.children:
47             output += "\n"
48             if cur.tag:
49                 output += cur.tag + " "
50             if cur.id:
51                 output += cur.id + " "
52 ▼         for child in cur.children:
53             if isinstance(child, Content):
54                 output += child.content + " "
55             elif isinstance(child, Element):
56                 queue.append(child)
57         return output.strip()
58     else:
59         return ""
```

```

60 """
61 The following code is not relevant for this exercise. It
62 is only here to help
63 running the tests.
64 """
65 ▼def main():
66     _html = raw_input()
67     _whiteList = raw_input()
68     _whiteListArray = [] if not _whiteList else
        _whiteList.split(",")
69     _dom = Dom(Parser(_html).parse())
70     _output = format(_dom, _whiteListArray)
71     print _output
72
73 ▼class Token:
74     def __init__(self, type, tag, id, content):
75         self.type = type
76         self.tag = tag
77         self.id = id
78         self.content = content
79
80 ▼class Parser:
81     START_ELEMENT_PATTERN = re.compile('^<(.*)>')
82     END_ELEMENT_PATTERN = re.compile('<\/(.*)>')
83     CLASS_PATTERN = re.compile("(.* id='(.*?)'")
84     CONTENT_PATTERN = re.compile('^(.*)<')
85
86     def __init__(self, html):
87         self.position = 0
88         self.tokens = self.tokenise(html)
89
90     def parse(self):
91         return self._parse()
92
93     def _parse(self):
94         _token = self.tokens[self.position]
95         _children = []
96         self.position += 1
97
98         while self.position < len(self.tokens):
99             _currentToken = self.tokens[self.position]
100             if _currentToken.type == 'START':
101                 _children.append(self._parse())
102             elif _currentToken.type == 'CONTENT':

```

```


        _children.append(Content(_currentToken.content))
104         else:
105             break
106         self.position += 1
107         return Element(_token.tag, _token.id, _children)
108
109     def tokenise(self, input):
110         _tokens = []
111         while input:
112             _endElement =
self.END_ELEMENT_PATTERN.match(input)
113             if _endElement:
114                 _tokens.append(Token('END',
_endElement.group(1), None, None))
115                 input =
input[len(_endElement.group(0)):]
116             else:
117                 _startElement =
self.START_ELEMENT_PATTERN.match(input)
118                 if _startElement:
119                     _classElement =
self.CLASS_PATTERN.match(_startElement.group(1))
120                     if _classElement:
121                         _tokens.append(Token('START',
_classElement.group(1), _classElement.group(2), None))
122                     else:
123                         _tokens.append(Token('START',
_startElement.group(1), None, None))
124                     input =
input[len(_startElement.group(0)):]
125                 else:
126                     _content =
self.CONTENT_PATTERN.match(input)
127                     if _content:
128                         _tokens.append(Token('CONTENT',
None, None, _content.group(1)))
129                     input =
input[len(_content.group(1)):]
130                 else:
131                     _tokens.append(Token('CONTENT',
None, None, input))
132                     input = ''
133         return _tokens
134
135     main()

```

☐ Test against custom input

Run Code

Submit code &amp; Continue

 [Download sample testcases](#) The input/output files have Unix line endings.  
Do not use Notepad to edit them on windows.

**Status: Compiled successfully. 1/4 sample test cases passed.**

### Testcase 1: Wrong Answer

#### Your Output

```
html
body content This good example.
div wrapper1 is a little
div wrapper3 harmless
div container1
div wrapper2 enjoyable
div container2 funny
```

#### Expected Output

```
html
body content This good example.
div wrapper3 harmless
```

### Testcase 2: Wrong Answer

#### Your Output

#### Expected Output

```
html foo
```

### Testcase 3: Success

#### Your Output



### Testcase 4: *Wrong Answer*

#### Your Output

```
html
body this problem is nice
div foo is an ugly
div bar little
div foo not
```

#### Expected Output

```
html
body this problem is nice
div bar little div bar
```