



Practice > Python > Regex and Parsing > HTML Parser - Part 1

HTML Parser - Part 1 ☆

84/115 challenges solved

Rank: 3074 | Points: 1635



Your HTML Parser - Part 1 submission got 30.00 points.

Share

Tweet

[Try the next challenge](#) | [Try a Random Challenge](#)**Problem**

Submissions

Leaderboard

Discussions

Editorial

HTML

Hypertext Markup Language is a standard markup language used for creating World Wide Web pages.

Parsing

Parsing is the process of syntactic analysis of a string of symbols. It involves resolving a string into its component parts and describing their syntactic roles.

HTMLParser

An *HTMLParser* instance is fed HTML data and calls handler methods when start tags, end tags, text, comments, and other markup elements are encountered.

Example (based on the original Python documentation):

Code

```
from HTMLParser import HTMLParser

# create a subclass and override the handler methods
class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print "Found a start tag :", tag
    def handle_endtag(self, tag):
        print "Found an end tag :", tag
    def handle_startendtag(self, tag, attrs):
        print "Found an empty tag :", tag

# instantiate the parser and fed it some HTML
parser = MyHTMLParser()
parser.feed("<html><head><title>HTML Parser - I</title></head>"
          + "<body><h1>HackerRank</h1><br /></body></html>")
```

Output

```
Found a start tag : html
Found a start tag : head
Found a start tag : title
Found an end tag : title
Found an end tag : head
Found a start tag : body
Found a start tag : h1
Found an end tag : h1
Found an empty tag : br
```

Author

DOSHI

Difficulty

Easy

Max Score

30

Submitted By

4034

NEED HELP?

[View discussions](#)[View editorial](#)[View top submissions](#)

RATE THIS CHALLENGE



MORE DETAILS

[Download problem statement](#)[Download sample test cases](#)[Suggest Edits](#)

```
Found an end tag   : body
Found an end tag   : html
```

`.handle_starttag(tag, attrs)`

This method is called to handle the *start tag* of an element. (For example: <div class='marks'>)

The *tag* argument is the name of the tag converted to lowercase.

The *attrs* argument is a list of (name, value) pairs containing the attributes found inside the tag's <> brackets.

`.handle_endtag(tag)`

This method is called to handle the *end tag* of an element. (For example: </div>)

The *tag* argument is the name of the tag converted to lowercase.

`.handle_startendtag(tag, attrs)`

This method is called to handle the *empty tag* of an element. (For example:
)

The *tag* argument is the name of the tag converted to lowercase.

The *attrs* argument is a list of (name, value) pairs containing the attributes found inside the tag's <> brackets.

Task

You are given an *HTML* code snippet of N lines.

Your task is to print *start tags*, *end tags* and *empty tags* separately.

Format your results in the following way:

```
Start : Tag1
End   : Tag1
Start : Tag2
-> Attribute2[0] > Attribute_value2[0]
-> Attribute2[1] > Attribute_value2[1]
-> Attribute2[2] > Attribute_value2[2]
Start : Tag3
-> Attribute3[0] > None
Empty : Tag4
-> Attribute4[0] > Attribute_value4[0]
End   : Tag3
End   : Tag2
```

Here, the -> symbol indicates that the tag contains an attribute. It is immediately followed by the name of the attribute and the attribute value.

The > symbol acts as a separator of the attribute and the attribute value.

If an *HTML* tag has no attribute then simply print the name of the tag.

If an attribute has no attribute value then simply print the name of the attribute value as `None`.

Note: Do not detect any *HTML* tag, attribute or attribute value inside the *HTML* comment tags (<!-- Comments -->). Comments can be multiline as well.

Input Format

The first line contains integer N , the number of lines in a *HTML* code snippet.

The next N lines contain *HTML* code.

Constraints

- $0 < N < 100$

Output Format

Print the *HTML* tags, attributes and attribute values in order of their occurrence from top to bottom in the given snippet.



Use proper formatting as explained in the problem statement.


Sample Input



```
2
<html><head><title>HTML Parser - I</title></head>
<body data-modal-target class='1'><h1>HackerRank</h1><br /></body></html>
```

Sample Output

```
Start : html
Start : head
Start : title
End  : title
End  : head
Start : body
-> data-modal-target > None
-> class > 1
Start : h1
End  : h1
Empty : br
End  : body
End  : html
```

Current Buffer (saved locally, editable)  

Python 3 

```

1  from html.parser import HTMLParser
2  class myhtmlparser(HTMLParser):
3      def handle_starttag(self,tag,attrs):
4          print("Start :",tag)
5          for attr in attrs:
6              print("-> {} > {}".format(*attr))
7      def handle_endtag(self,tag):
8          print("End  :",tag)
9      def handle_startendtag(self,tag,attrs):
10         print("Empty :",tag)
11         for attr in attrs:
12             print("-> {} > {}".format(*attr))
13     html=""
14     for i in range(int(input())):
15         html=html+input()
16     parser=myhtmlparser()
17     parser.feed(html)
18     parser.close()

```

Line: 10 Col: 18

 Upload Code as File ☐ Test against custom input

Run Code

Submit Code



You have earned 30.00 points!
84/115 challenges solved.

73%

Congratulations

You solved this challenge. Would you like to challenge your friends?



Next
Challenge

- ✔ Testcase 0
- ✔ Testcase 1
- ✔ Testcase 2
- ✔ Testcase 3

6 Testcases

| Input (stdin) | Download | Expected Output | Download |
|--|----------|--|----------|
| 2 <html><head><title>HTML Par ser - I</title></head> <body data-model-target="c1a | | Start : html Start : head Start : title End : title | |

Compiler Message

Success