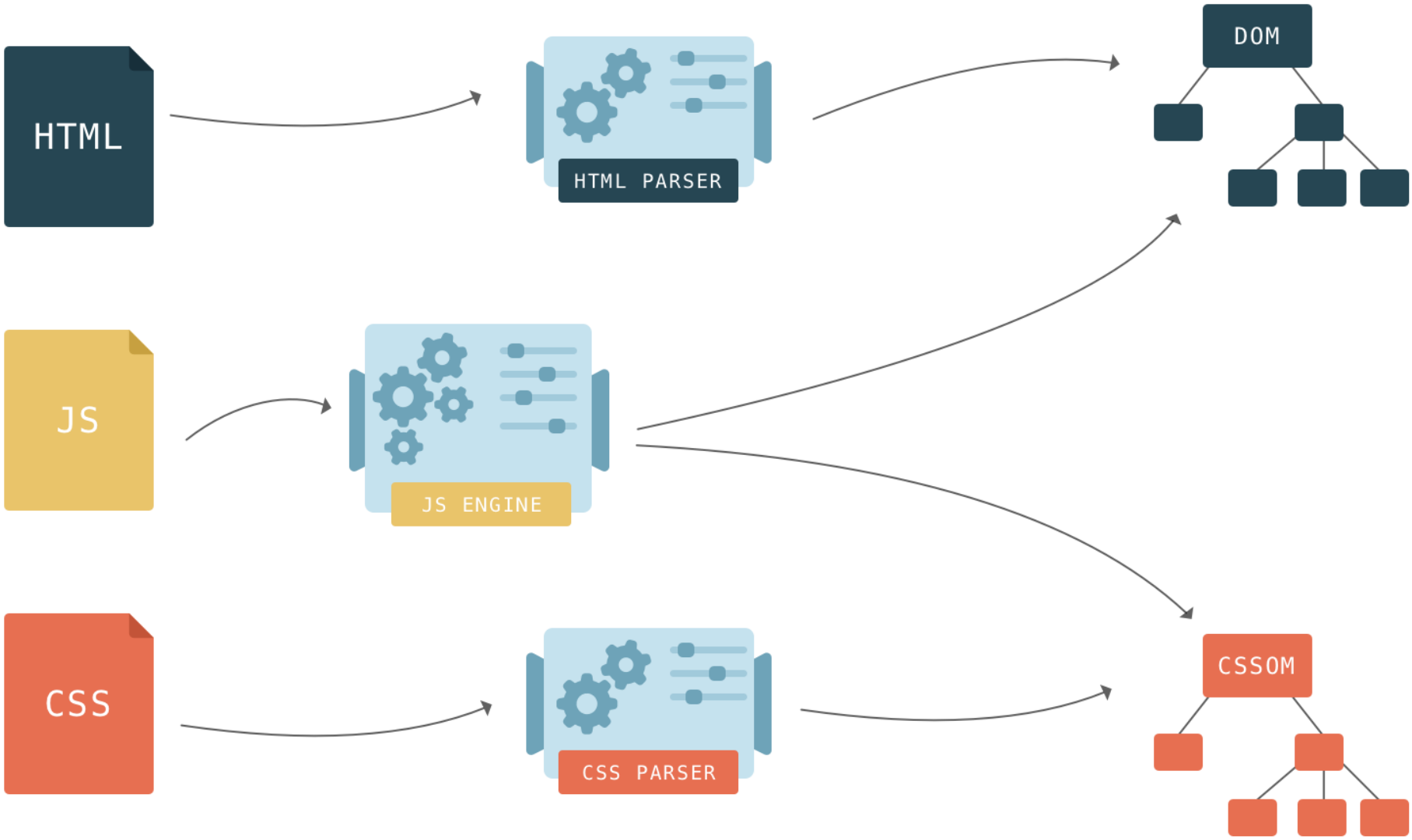
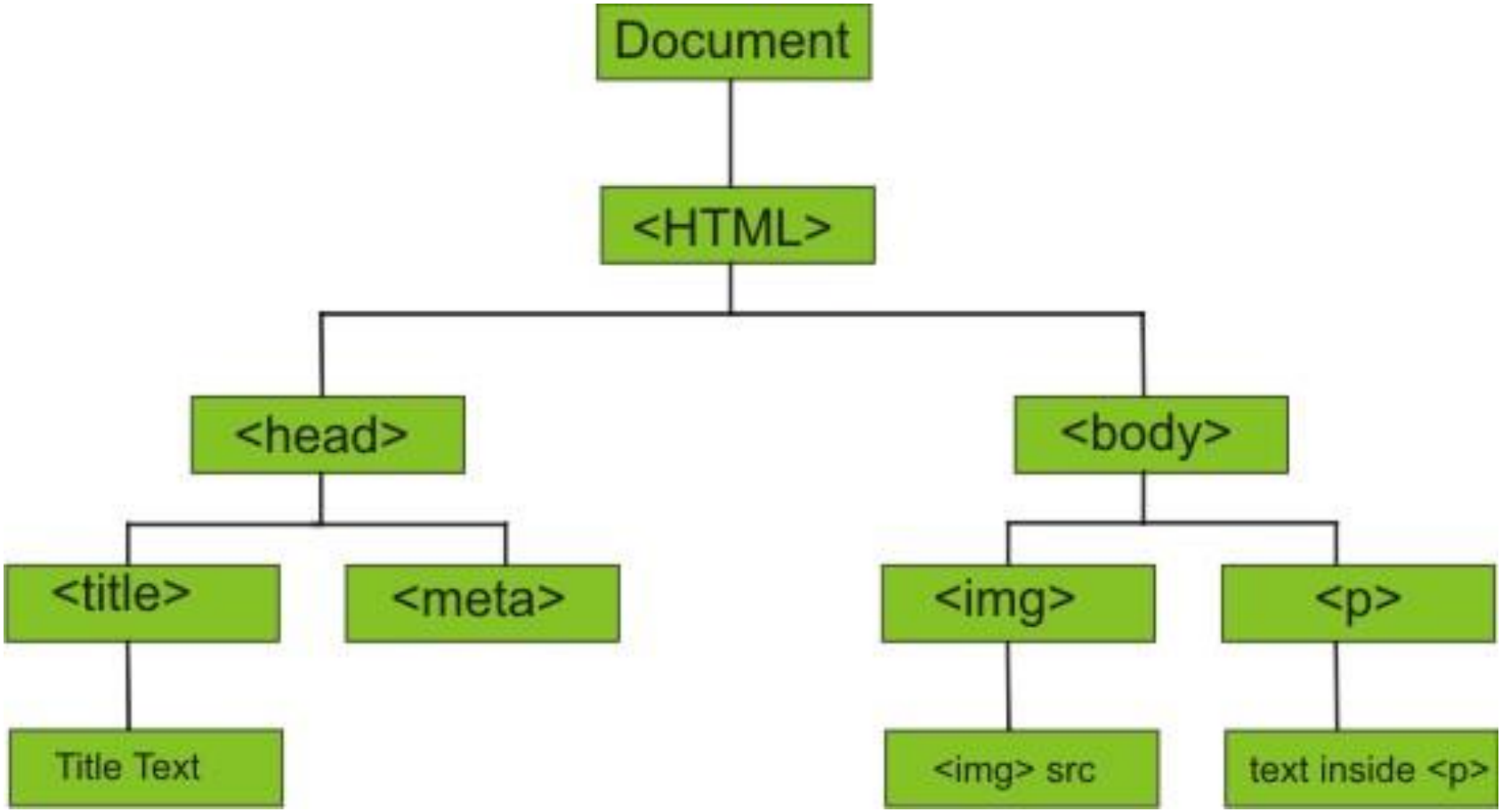
DOM


HTML
HTML PARSER
DOM
JS
JS ENGINE
CSS
CSS PARSER
CSSOM


Document
<HTML>
<head>
<body>
<title>
<meta>
<img>
<p>
Title Text
<img> src
text inside <p>

BeautifulSoup

BeautifulSoup

BeautifulSoup Python library for pulling data out of HTML and XML files

Provide idiomatic ways of navigating, searching, and modifying DOM

Saves programmers hours or days of work

```
!pip install beautifulsoup4
```

```
from bs4 import BeautifulSoup
```

Parser

lxml If you can, I recommend you install and use **lxml** for speed

Parser	Typical usage	Advantages	Disadvantages
Python's html.parser	`BeautifulSoup(markup, "html.parser")`	• Batteries included • Decent speed • Lenient (as of Python 2.7.3 and 3.2.)	• Not very lenient (before Python 2.7.3 or 3.2.2)
lxml's HTML parser	`BeautifulSoup(markup, "lxml")`	• Very fast • Lenient	• External C dependency
lxml's XML parser	`BeautifulSoup(markup, "lxml-xml")` `BeautifulSoup(markup, "xml")`	• Very fast • The only currently supported XML parser	• External C dependency
html5lib	`BeautifulSoup(markup, "html5lib")`	• Extremely lenient • Parses pages the same way a web browser does • Creates valid HTML5	• Very slow • External Python dependency

Not Well-formed

Every open tag must explicitly be closed

Empty elements in XML are written closed

Child markup must nest completely within parent markup

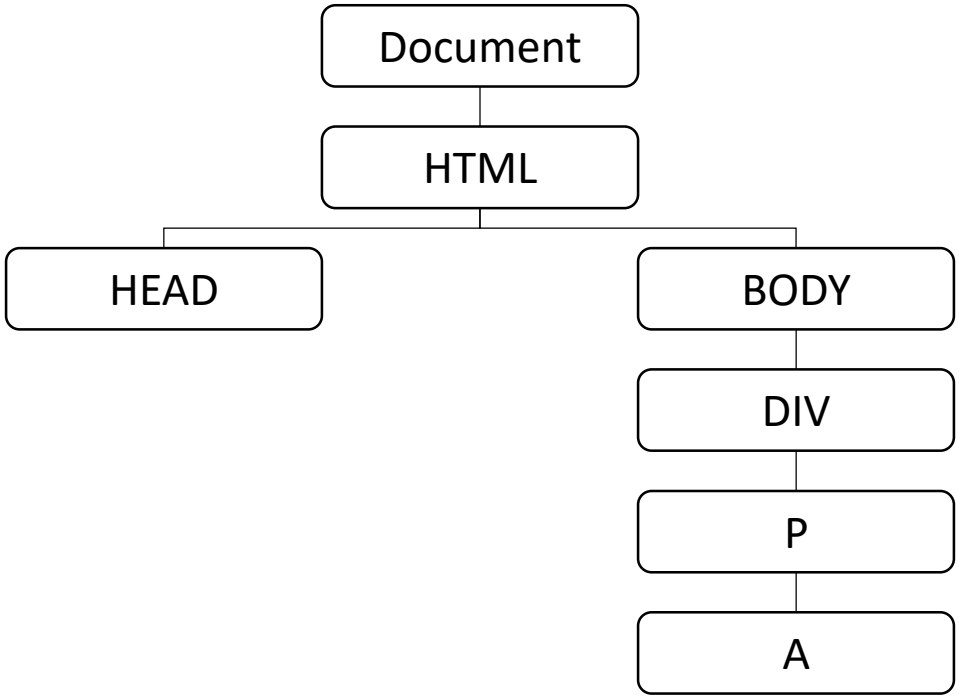
```
<html>
  <head><head>
  <body>
    <div>
      <ul>
        <li>
        <li>
    </div>
      </ul>
  </body>
</HTML>
```

not closed / unclosed tag (meta, br, img, …)

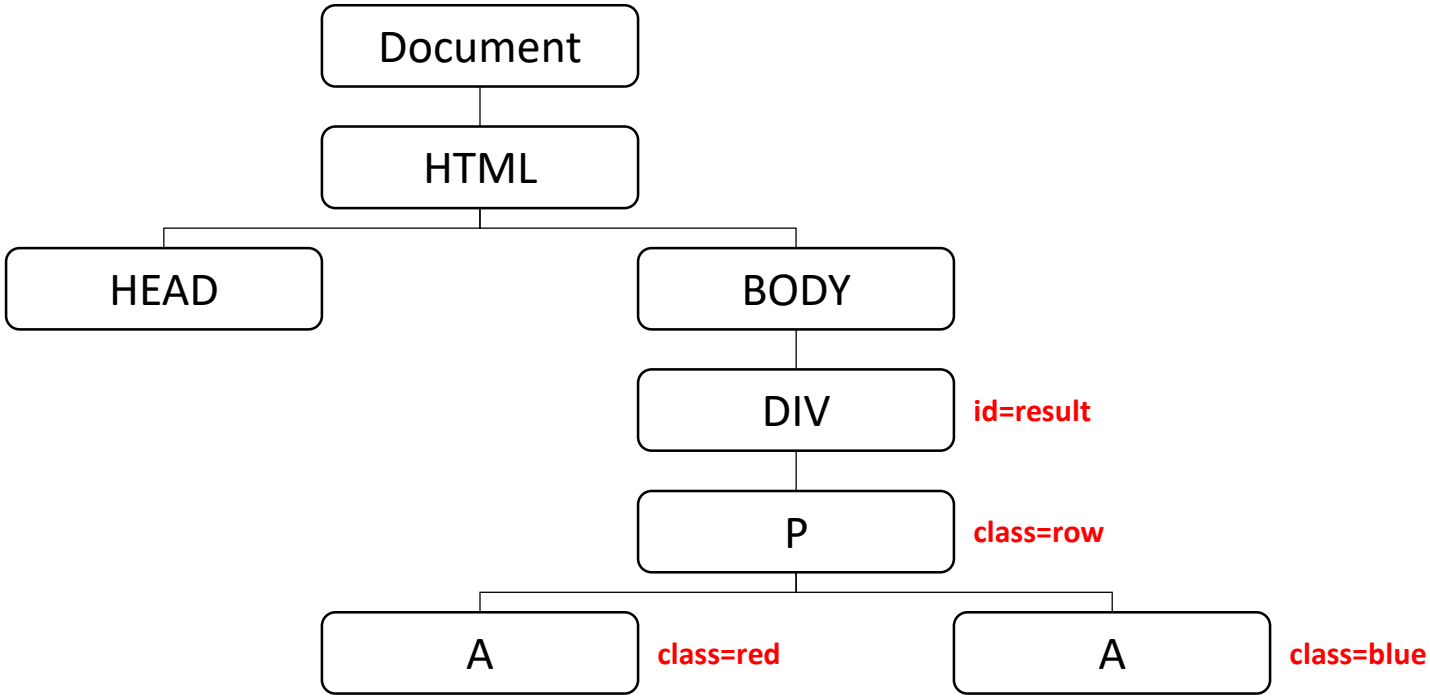
overlap with other elements

case mismatch

```
html = """
<html>
  <head></head>
  <body>
    <div>
      <p>
        <a>go to page</a>
      </p>
    </div>
  </body>
</html>
"""
```



```
html = """
<html>
  <head></head>
  <body>
    <div id="result">
      <p class="row">
        <a class="red">go to page1</a>
        <a class="blue">go to page2</a>
      </p>
    </div>
  </body>
</html>
"""
```



Scans the entire document looking for results

find (name, attrs, recursive, string, **kwargs)

find_all (name, attrs, recursive, string, limit, **kwargs)

```
dom.find_all('a')
dom.find_all({'div', 'p'})
dom.find_all('div', {'id':'result'})
dom.find_all('', attrs={'class':'red'})
dom.find_all('a', recursive=False)
dom.find_all(text='go to page1')
dom.find_all('a', limit=1)
```

find_parent You can access an element's parent with the ***.parent*** attribute

find_parents You can iterate over all of an element's parents with ***.parents***

find_all The ***.descendants*** attribute lets you iterate over all of a tag's children

find_all(recursive=false) The ***.children*** attributes only consider a tag's direct children

find_next_sibling, find_previous_sibling

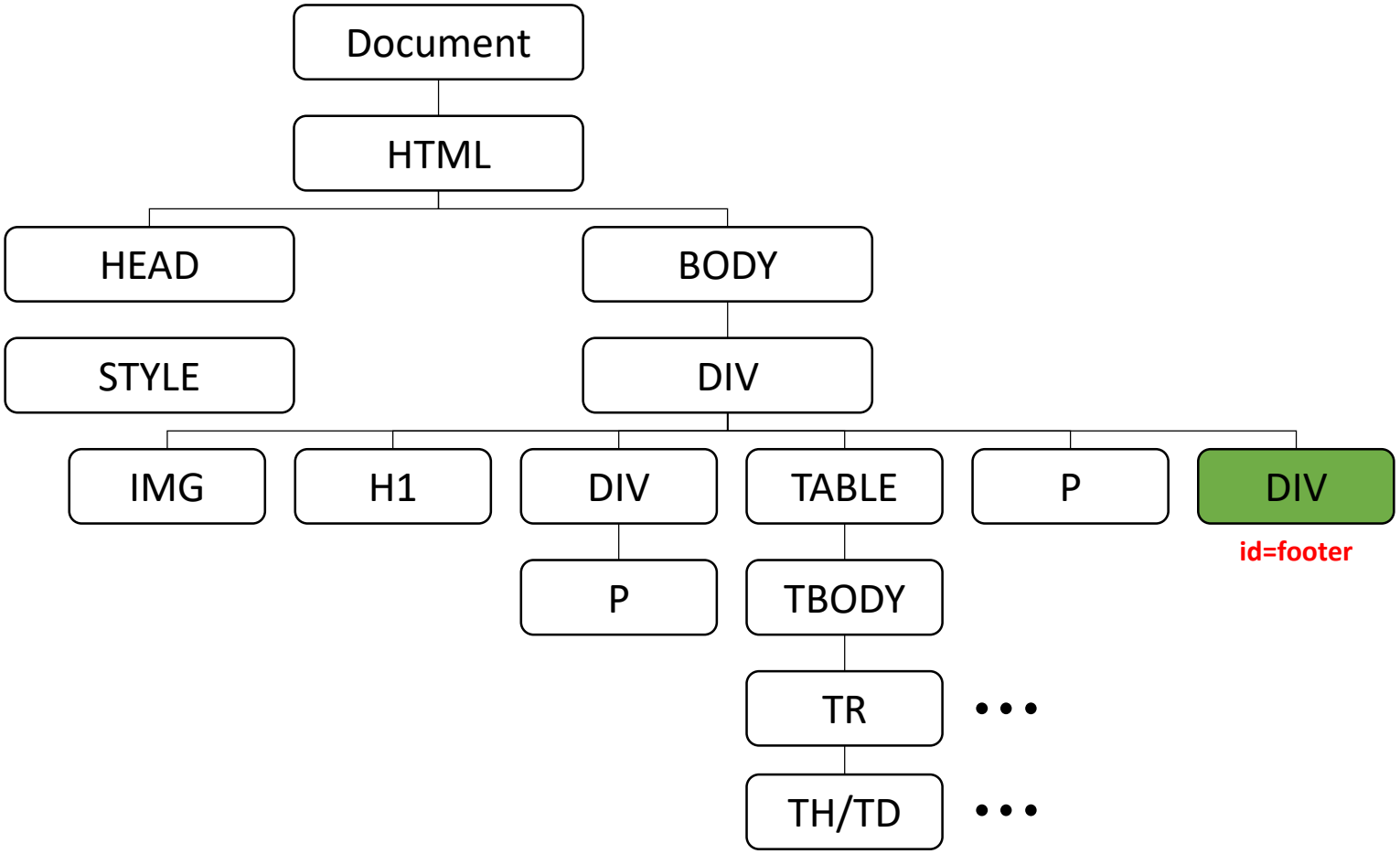
find_next_siblings, find_previous_siblings

Navigate between page elements

on the same level of the parse tree

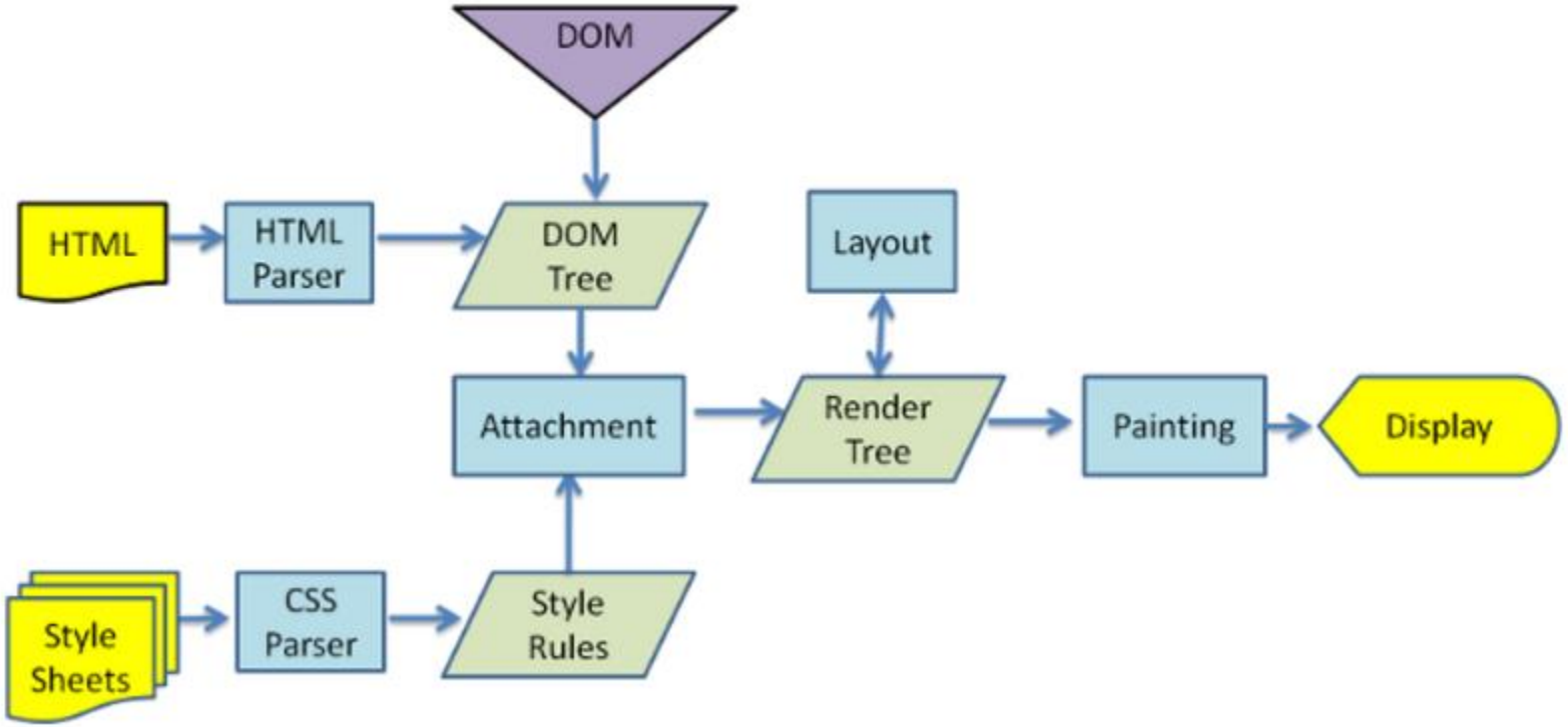
exercises

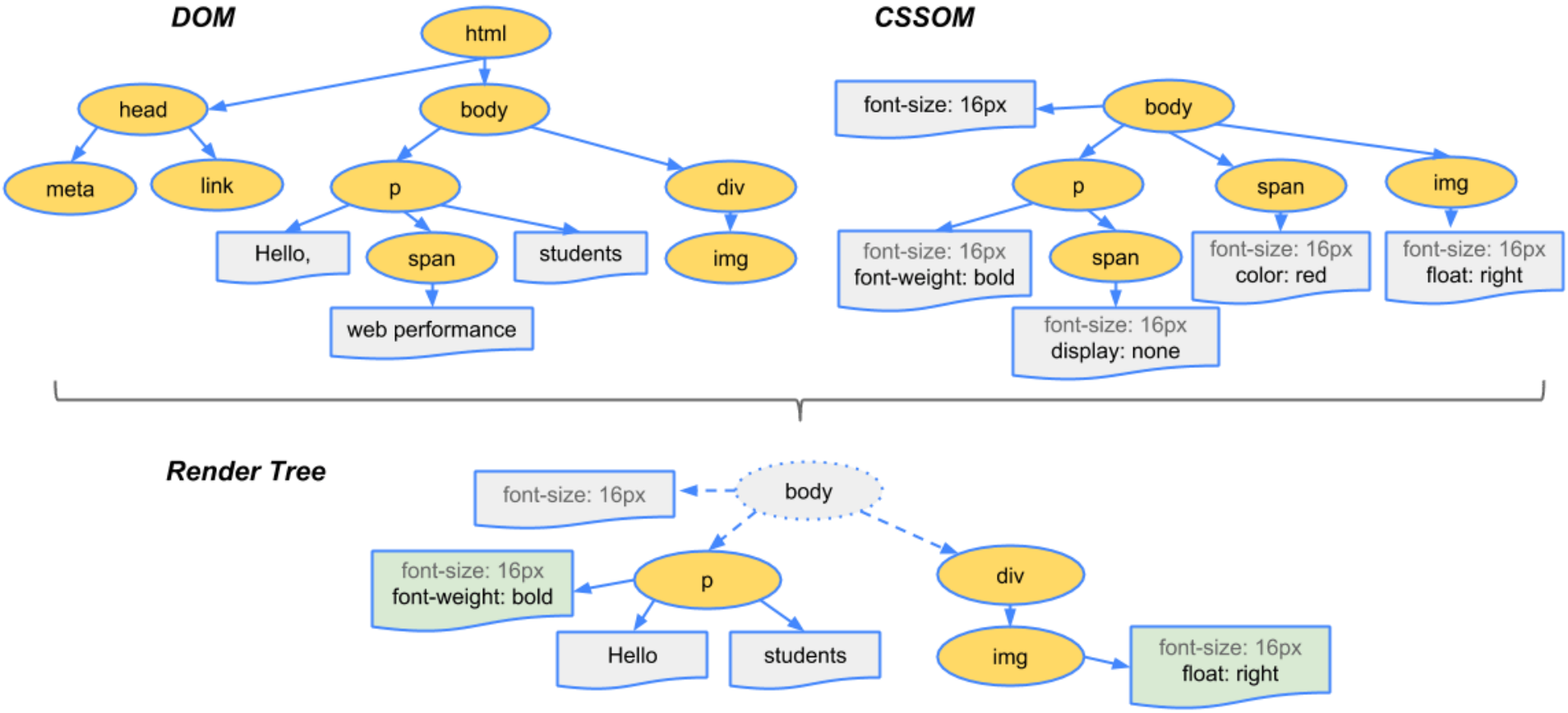
http://pythonscraping.com/pages/page3.html



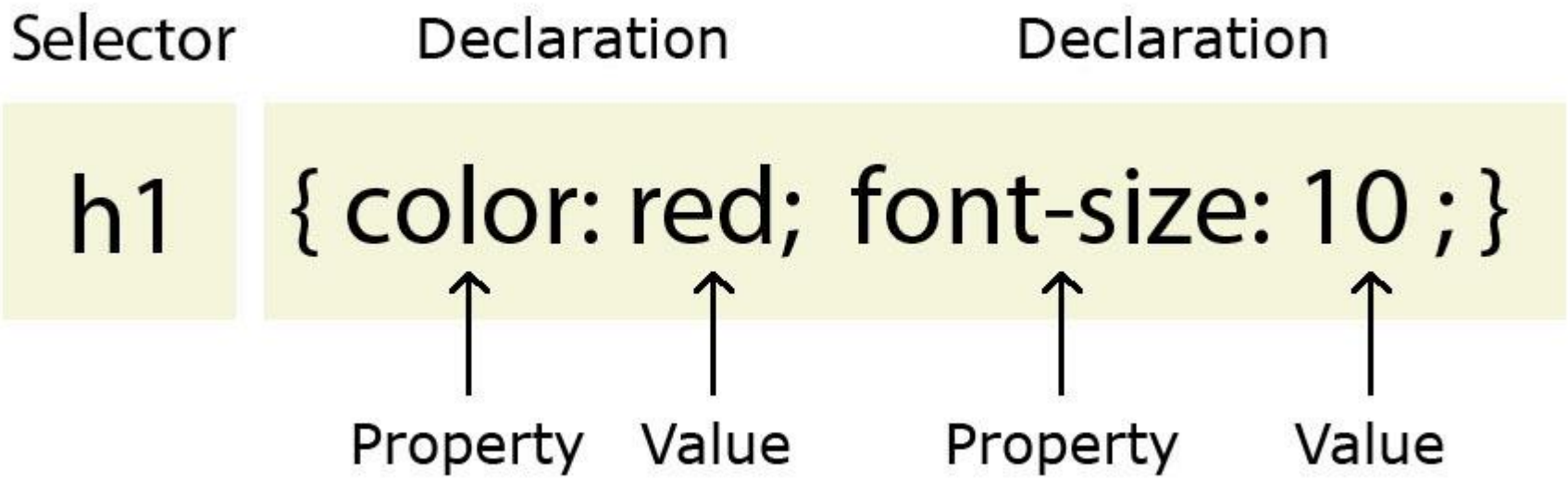
Selector

CSSOM


DOM
HTML
HTML Parser
DOM Tree
Layout
Attachment
Render Tree
Painting
Display
Style Sheets
CSS Parser
Style Rules


DOM
html
head
body
meta
link
p
div
Hello,
span
students
img
web performance
CSSOM
font-size: 16px
body
p
span
img
font-size: 16px
font-weight: bold
span
font-size: 16px
color: red
font-size: 16px
float: right
font-size: 16px
display: none
Render Tree
font-size: 16px
body
font-size: 16px
font-weight: bold
p
div
Hello
students
img
font-size: 16px
float: right

CSS Selector



XPath

- XPath uses path expressions to select nodes in an XML document.
  The node is selected by following steps.

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

exercises

4대 검색사이트 검색결과 가져오기