Beautiful Soup 4 Cheatsheet

Basic

```
# https://www.crummy.com/software/BeautifulSoup/bs4/doc/
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')


soup.title # <title>The Dormouse's story</title>
soup.title.name # u'title'
soup.title.string # u'The Dormouse's story'
soup.title.parent.name # u'head'

#various finder
css_soup.select("p.strikeout.body") # css finder
soup.p # <p class="title"><b>The Dormouse's story</b></p>
soup.p['class'] # u'title'
soup.a # <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
soup.find_all('a') # [<a ..>, ..]
soup.find(id="link3") # <a class="sister" href="http://example.com/tillie" id="link3">Till
for link in soup.find_all('a'):
    print(link.get('href')) # http://example.com/elsi, # http://example.com/lacie
```

Make soup

```

from bs4 import BeautifulSoup

soup = BeautifulSoup(open("index.html"))
soup = BeautifulSoup("<html>data</html>")
```

Output

```

# HTML
soup.prettify() #pretty print
str(soup) # non-pretty print
```

```
# String
soup.get_text() #all text under the element
```

Search

```

search.pyhttps://www.crummy.com/software/BeautifulSoup/bs4/doc/
#------------------------
# css selector
#------------------------
css_soup.select("p.strikeout.body")
soup.select("p nth-of-type(3)") # 3rd child
soup.select("head > title")
soup.select("p > a:nth-of-type(2)")
soup.select("p > #link1") # direct child
soup.select("#link1 ~ .sister")  # sibling
soup.select('a[href]') # existence of an attribute
soup.select_one(".sister")

# attribute value
soup.select('a[href="http://example.com/elsie"]') # exact attribute
soup.select('a[href^="http://example.com/"]') # negative match
soup.select('a[href$="tillie"]') # end match
soup.select('a[href*=".com/el"]') # middle match



#------------------------
# basic
#------------------------
soup.find_all('b') # match by tag
soup.find_all(re.compile("^b")) # match by tag using regex
soup.find_all(["a", "b"]) # match by tag in list

# function (complex condition)
def has_class_but_no_id(tag):
  return tag.has_attr('class') and not tag.has_attr('id')
soup.find_all(has_class_but_no_id)



#------------------------
```

Navigation

```
soup.head# <head><title>The Dormouse's story</title></head>
soup.title# <title>The Dormouse's story</title>
soup.body.b # <b>The Dormouse's story</b>
soup.a # <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
# <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
# <a class="sister" href="http:

# children = contents
head_tag.contents # [<title>The Dormouse's story</title>]
head_tag.children # [<title>The Dormouse's story</title>]

# descendants (all of a tag's children, recursively)
for child in head_tag.descendants:
  print(child)

# .string is tricky
head_tag.contents # [<title>The Dormouse's story</title>]
head_tag.string # u'The Dormouse's story' (because head tag has only one child)
print(soup.html.string) # None (because html has many children)

# whitespace removed strings
for string in soup.stripped_strings:
  print(repr(string))



# ----- going up -----
title_tag.parent # <head><title>The Dormouse's story</title></head>
# going up recursively
link.parents # [ p, body, html, [document], None]



# ----- sideway -----
# sibling = include text node
sibling soup.b.next sibling
```

Edit

```


#----------------------------
# change exisitng tag
#----------------------------
tag.name = "blockquote" # modify tag name
tag['class'] = 'verybold' # modify tag attribute
```

```
del tag['class'] # delete attribute
tag.string= 'not too bold' # modify tag contents string
tag.append(" but bolder than usual") # append tag contents

#----------------------------
# insert tag
#----------------------------
new_tag = soup.new_tag("a", href="http://www.example.com")
original_tag.append(new_tag) # create child
new_tag.string = "Link text." # can edit element after creating child

soup.b.string.insert_before(tag)
soup.b.i.insert_after(soup.new_string(" ever "))

#----------------------------
# delete tag
#----------------------------
soup.i.clear() # removes the contents
i_tag = soup.i.extract() # completely removes a tag from tree and returns the element
soup.i.decompose() # completely removes a tag from tree and discard the tag

#----------------------------
# replace/wrap/unwrap tag
#----------------------------
a_tag.i.replace_with(soup.new_tag("b"))
a_tag.i.replace_with(Beautifulsoup("<b>bold element</b>")) # replace inner html
soup.p.string.wrap(soup.new_tag("b"))
a_tag.i.unwrap()
```

Encoding

```

#output
soup.prettify("latin-1")
tag.encode("utf-8")
tag.encode("latin-1")
tag.encode("ascii")
```

Parse only part

```

# The SoupStrainer class allows you to choose which parts of an
# incoming document are parsed
from bs4 import SoupStrainer
```

```python
# conditions
only_a_tags = SoupStrainer("a")
only_tags_with_id_link2 = SoupStrainer(id="link2")

def is_short_string(string):
  return len(string) < 10
only_short_strings = SoupStrainer(string=is_short_string)

# execute parse
BeautifulSoup(html_doc, "html.parser", parse_only=only_a_tags)
BeautifulSoup(html_doc, "html.parser", parse_only=only_tags_with_id_link2)
BeautifulSoup(html_doc, "html.parser", parse_only=only_short_strings)
```