Python

Follow me on Sumit Khanna for more updates

BeautifulSoup Cheat Sheet for Data Scraping

BeautifulSoup is a powerful library used for web scraping purposes to pull the data out of HTML and XML files

Cheat Sheet Table of BeautifulSoup Functions

Function	Brief Explanation
BeautifulSoup	Parses an HTML or XML document into a tree of Python objects.
prettify	Returns a string containing the prettified HTML or XML document.
find	Finds the first tag that matches a given criteria.
find_all	Finds all tags that match a given criteria.
select	Finds all tags that match a CSS selector.
get_text	Extracts all text from a tag.
attrs	Accesses the attributes of a tag.
parent	Navigates to the parent of a tag.
parents	Navigates to all parents of a tag.
children	Navigates to the children of a tag.
descendants	Navigates to all descendants of a tag.
next_sibling	Navigates to the next sibling of a tag.
previous_sibling	Navigates to the previous sibling of a tag.
next_siblings	Navigates to all next siblings of a tag.
previous_siblings	Navigates to all previous siblings of a tag.
decompose	Removes a tag from the tree.
replace_with	Replaces a tag with another tag or string.
new_tag	Creates a new tag.
insert	Inserts a new tag at a specified position.
append	Appends a new tag at the end.
find_parents	Finds all parent tags that match a given criteria.
find_parent	Finds the first parent tag that matches a given criteria.
find_next_siblings	Finds all next siblings that match a given criteria.
find_next_sibling	Finds the first next sibling that matches a given criteria.
find_previous_siblings	Finds all previous siblings that match a given criteria.
find_previous_sibling	Finds the first previous sibling that matches a given criteria.
find_all_next	Finds all tags that match a given criteria after the current tag.
find_next	Finds the first tag that matches a given criteria after the current tag.
find_all_previous	Finds all tags that match a given criteria before the current tag.
find_previous	Finds the first tag that matches a given criteria before the current tag.
get	Retrieves an attribute value of a tag.
string	Accesses the string within a tag.
strings	Accesses all strings within a tag.
stripped_strings	Accesses all stripped strings within a tag.

BeautifulSoup Function

`from bs4 import BeautifulSoup`
The `BeautifulSoup` function is used to parse an HTML or XML document into a tree of Python objects. It takes in the document and a parser as parameters.

Parameters:

- `markup` (str): The HTML or XML document to be parsed.
- `features` (str): The parser to be used. Common options are 'html.parser', 'lxml', and 'xml'.

Returns:

- `soup` (BeautifulSoup object): The parsed document.

Example:

```
html_doc = "<html><head><title>The Dormouse's story</title></head><body><p>Once upon a time...</p></body>
</html>"
soup = BeautifulSoup(html_doc, 'html.parser')
print(soup.prettify())
```

This code will parse the HTML document and print it in a nicely formatted way.

In [ ]:

```
from bs4 import BeautifulSoup

html_doc = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Sample Web Page</title>
</head>
<body>
    <h1>Welcome to the Sample Web Page</h1>
    <p class="intro">This is an example paragraph with a class attribute.</p>
    <p id="main">This is another example paragraph with an ID attribute.</p>
    <div>
        <p>This is a nested paragraph inside a div.</p>
        <a href="https://example.com" title="Example Link">Visit Example</a>
        <ul>
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>
        </ul>
    </div>
</body>
</html>
"""
soup = BeautifulSoup(html_doc, 'html.parser')
print(soup.prettify())
```

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   Sample Web Page
  </title>
 </head>
 <body>
  <h1>
   Welcome to the Sample Web Page
  </h1>
  <p class="intro">
   This is an example paragraph with a class attribute.
  </p>
  <p id="main">
   This is another example paragraph with an ID attribute.
  </p>
  <div>
   <p>
    This is a nested paragraph inside a div.
   </p>
   <a href="https://example.com" title="Example Link">
    Visit Example
   </a>
   <ul>
    <li>
     Item 1
    </li>
    <li>
     Item 2
    </li>
    <li>
     Item 3
    </li>
   </ul>
  </div>
 </body>
</html>
```

prettify Function

The `prettify` function returns a string containing the prettified HTML or XML document. It formats the document with proper indentation and line breaks.

Parameters:

- None

Returns:

- `str`: A prettified string of the HTML or XML document.

Example:

```
print(soup.prettify())
```
This will print the HTML document in a nicely formatted way.

find Function

The `find` function searches for the first tag that matches the given criteria. It takes in various parameters to specify the tag and its attributes.

Parameters:

- `name` (str): The name of the tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `recursive` (bool): If True, the search is recursive. Defaults to True.
- `text` (str or callable): A string or a callable to match the text within the tag.

Returns:

- `Tag`: The first tag that matches the given criteria, or None if no match is found.

Example:

```
first_paragraph = soup.find('p')
print(first_paragraph)
```

This code will find and print the first `<p>` tag in the HTML document.

In [ ]:
```
first_paragraph = soup.find('p')
print(first_paragraph)
```

```
<p class="intro">This is an example paragraph with a class attribute.</p>
```

find_all Function

The `find_all` function searches for all tags that match the given criteria. It takes in various parameters to specify the tags and their attributes.

Parameters:

- `name` (str): The name of the tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `recursive` (bool): If True, the search is recursive. Defaults to True.
- `text` (str or callable): A string or a callable to match the text within the tags.
- `limit` (int): The maximum number of tags to return. Defaults to None.

Returns:

- `ResultSet` : A list of tags that match the given criteria.

Example:

```
all_paragraphs = soup.find_all('p')
for p in all_paragraphs:
    print(p)
```

This code will find and print all `<p>` tags in the HTML document.

In [ ]:
```
all_paragraphs = soup.find_all('p')
for p in all_paragraphs:
    print(p)
```

```
<p class="intro">This is an example paragraph with a class attribute.</p>
<p id="main">This is another example paragraph with an ID attribute.</p>
<p>This is a nested paragraph inside a div.</p>
```

select Function

The `select` function searches for all tags that match a given CSS selector. It takes in a CSS selector as a parameter.

Parameters:

- `selector` (str): A string containing the CSS selector.

Returns:

- `list` : A list of tags that match the given CSS selector.

Example:

```
head_title = soup.select('head > title')
for title in head_title:
    print(title)
```

This code will find and print the `<title>` tag inside the `<head>` tag using a CSS selector.

In [ ]:
```
head_title = soup.select('head > title')
for title in head_title:
    print(title)
```

```
<title>Sample Web Page</title>
```

get_text Function

The `get_text` function extracts all the text from a tag. It can take in optional parameters to control the output.

Parameters:

- `separator` (str): A string to be inserted between the pieces of text. Defaults to an empty string.
- `strip` (bool): If True, whitespace will be stripped from the text. Defaults to False.

Returns:

- `str` : The extracted text.

Example:

```
text = soup.get_text()
print(text)
```

This code will extract and print all the text from the HTML document.

In [ ]:
```
text = soup.get_text()
print(text)
```

```
Sample Web Page


Welcome to the Sample Web Page
This is an example paragraph with a class attribute.
This is another example paragraph with an ID attribute.
This is a nested paragraph inside a div.
Visit Example

Item 1
Item 2
Item 3
```

attrs Function

The `attrs` function accesses the attributes of a tag. It returns a dictionary of the attributes and their values.

Parameters:

- None

Returns:

- `dict` : A dictionary of the attributes and their values.

Example:

```
tag = soup.find('p')
print(tag.attrs)
```

This code will print the attributes of the first `<p>` tag.

In [ ]:
```
tag = soup.find('p')
print(tag.attrs)
```

```
{'class': ['intro']}
```

parent Function

The `parent` function navigates to the parent of a tag. It returns the parent tag of the current tag.

Parameters:

- None

Returns:

- `Tag` : The parent tag of the current tag.

Example:

```
parent_tag = tag.parent
print(parent_tag)
```

This code will print the parent tag of the first `<p>` tag.

In [ ]:
```
parent_tag = tag.parent
print(parent_tag)
```

```
<body>
<h1>Welcome to the Sample Web Page</h1>
<p class="intro">This is an example paragraph with a class attribute.</p>
<p id="main">This is another example paragraph with an ID attribute.</p>
<div>
<p>This is a nested paragraph inside a div.</p>
<a href="https://example.com" title="Example Link">Visit Example</a>
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
</div>
</body>
```

parents Function

The `parents` function navigates to all parents of a tag. It returns a generator of the parent tags of the current tag.

Parameters:

- None

Returns:

- `generator` : A generator of the parent tags of the current tag.

Example:

```
for parent in tag.parents:
    print(parent.name)
```

This code will print the names of all parent tags of the first `<p>` tag.

In [ ]:
```
for parent in tag.parents:
    print(parent.name)
```

```
body
html
[document]
```

children Function

The `children` function navigates to the children of a tag. It returns a list of the children tags of the current tag.

Parameters:

- None

Returns:

- `list` : A list of the children tags of the current tag.

Example:

```
body_tag = soup.body
for child in body_tag.children:
    print(child)
```

This code will print the children tags of the `<body>` tag.

In [ ]:
```
body_tag = soup.body
for child in body_tag.children:
    print(child)
```

```
<h1>Welcome to the Sample Web Page</h1>

<p class="intro">This is an example paragraph with a class attribute.</p>

<p id="main">This is another example paragraph with an ID attribute.</p>

<div>
<p>This is a nested paragraph inside a div.</p>
<a href="https://example.com" title="Example Link">Visit Example</a>
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
</div>
```

descendants Function

The `descendants` function navigates to all descendants of a tag. It returns a generator of the descendant tags of the current tag.

Parameters:

- None

Returns:

- `generator` : A generator of the descendant tags of the current tag.

Example:

```
for descendant in body_tag.descendants:
    print(descendant)
```

This code will print the descendant tags of the `<body>` tag.

In [ ]:
```
for descendant in body_tag.descendants:
    print(descendant)
```

```
<h1>Welcome to the Sample Web Page</h1>
Welcome to the Sample Web Page


<p class="intro">This is an example paragraph with a class attribute.</p>
This is an example paragraph with a class attribute.


<p id="main">This is another example paragraph with an ID attribute.</p>
This is another example paragraph with an ID attribute.


<div>
<p>This is a nested paragraph inside a div.</p>
<a href="https://example.com" title="Example Link">Visit Example</a>
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
</div>


<p>This is a nested paragraph inside a div.</p>
This is a nested paragraph inside a div.


<a href="https://example.com" title="Example Link">Visit Example</a>
Visit Example


<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>


<li>Item 1</li>
Item 1


<li>Item 2</li>
Item 2


<li>Item 3</li>
Item 3
```

next_sibling Function

The `next_sibling` function navigates to the next sibling of a tag. It returns the next sibling tag of the current tag.

Parameters:

- None

Returns:

- `Tag` : The next sibling tag of the current tag, or None if no sibling is found.

Example:

```
next_sibling = tag.next_sibling
print(next_sibling)
```

This code will print the next sibling of the first `<p>` tag.

In [ ]:
```
tag = soup.find('p')
tag
```

Out[ ]:
```
<p class="intro">This is an example paragraph with a class attribute.</p>
```

In [ ]:
```
next_sibling = tag.next_sibling
print(next_sibling)
```

previous_sibling Function

The `previous_sibling` function navigates to the previous sibling of a tag. It returns the previous sibling tag of the current tag.

Parameters:

- None

Returns:

- `Tag` : The previous sibling tag of the current tag, or None if no sibling is found.

Example:

```
previous_sibling = tag.previous_sibling
print(previous_sibling)
```

This code will print the previous sibling of the first `<p>` tag.

In [ ]:
```
previous_sibling = tag.previous_sibling
print(previous_sibling)
```

next_siblings Function

The `next_siblings` function navigates to all next siblings of a tag. It returns a generator of the next sibling tags of the current tag.

Parameters:

- None

Returns:

- `generator` : A generator of the next sibling tags of the current tag.

Example:

```
for sibling in tag.next_siblings:
    print(sibling)
```

This code will print the next siblings of the first `<p>` tag.

In [ ]:
```
for sibling in tag.next_siblings:
    print(sibling)
```

```
<p id="main">This is another example paragraph with an ID attribute.</p>


<div>
<p>This is a nested paragraph inside a div.</p>
<a href="https://example.com" title="Example Link">Visit Example</a>
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
</div>
```

previous_siblings Function

The `previous_siblings` function navigates to all previous siblings of a tag. It returns a generator of the previous sibling tags of the current tag.

Parameters:

- None

Returns:

- `generator` : A generator of the previous sibling tags of the current tag.

Example:

```
for sibling in tag.previous_siblings:
    print(sibling)
```

This code will print the previous siblings of the first `<p>` tag.

In [ ]:
```
for sibling in tag.previous_siblings:
    print(sibling)
```

```
<h1>Welcome to the Sample Web Page</h1>
```

decompose Function

The `decompose` function removes a tag from the tree. It is useful for removing unwanted elements from the document.

Parameters:

- None

Returns:

- None

Example:

```
tag.decompose()
print(soup.prettify())
```

This code will remove the first `<p>` tag from the document and print the updated document.

In [ ]:
```
tag.decompose()
print(soup.prettify())
```

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   Sample Web Page
  </title>
 </head>
 <body>
  <h1>
   Welcome to the Sample Web Page
  </h1>
  <p id="main">
   This is another example paragraph with an ID attribute.
  </p>
  <div>
   <p>
    This is a nested paragraph inside a div.
   </p>
   <a href="https://example.com" title="Example Link">
    Visit Example
   </a>
   <ul>
    <li>
     Item 1
    </li>
    <li>
     Item 2
    </li>
    <li>
     Item 3
    </li>
   </ul>
  </div>
 </body>
</html>
```

In [ ]:
```
tag.get_text()
```

Out[ ]:
```
''
```

new_tag Function

The `new_tag` function creates a new tag. It is useful for dynamically adding new elements to the document.

Parameters:

- `name` (str): The name of the new tag.

- `attrs` (dict): A dictionary of attributes and their values to set on the new tag.

Returns:

- `Tag` : The new tag.

Example:

```
new_tag = soup.new_tag('div', id='new-div', class_='new-class')
print(new_tag)
```

This code will create a new `<div>` tag with the attributes `id='new-div'` and `class='new-class'` and print it.

In [ ]:
```
new_tag = soup.new_tag('div', id='new-div', class_='new-class')
print(new_tag)
```

```
<div class_="new-class" id="new-div"></div>
```

find_parents Function

The `find_parents` function searches for all parent tags that match the given criteria. It takes in various parameters to specify the tags and their attributes.

Parameters:

- `name` (str): The name of the parent tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `recursive` (bool): If True, the search is recursive. Defaults to True.
- `text` (str or callable): A string or a callable to match the text within the parent tags.
- `limit` (int): The maximum number of parent tags to return. Defaults to None.

Returns:

- `ResultSet` : A list of parent tags that match the given criteria.

Example:

```
parent_tags = tag.find_parents('body')
for parent in parent_tags:
    print(parent)
```

This code will find and print all parent `<body>` tags of the first `<p>` tag.

In [ ]:
```
parent_tags = tag.find_parents('body')
for parent in parent_tags:
    print(parent)
```

find_parent Function

The `find_parent` function searches for the first parent tag that matches the given criteria. It takes in various parameters to specify the tag and its attributes.

Parameters:

- `name` (str): The name of the parent tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `recursive` (bool): If True, the search is recursive. Defaults to True.
- `text` (str or callable): A string or a callable to match the text within the parent tag.

Returns:

- `Tag` : The first parent tag that matches the given criteria, or None if no match is found.

Example:

```
parent_tag = tag.find_parent('body')
print(parent_tag)
```

This code will find and print the first parent `<body>` tag of the first `<p>` tag.

In [ ]:
```
parent_tag = tag.find_parent('body')
print(parent_tag)
```

```
None
```

find_next_siblings Function

The `find_next_siblings` function searches for all next sibling tags that match the given criteria. It takes in various parameters to specify the tags and their attributes.

Parameters:

- `name` (str): The name of the next sibling tags to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the next sibling tags.

Returns:

- `ResultSet` : A list of next sibling tags that match the given criteria.

Example:

```
next_siblings = tag.find_next_siblings('p')
for sibling in next_siblings:
    print(sibling)
```

This code will find and print all next sibling `<p>` tags of the first `<p>` tag.

In [ ]:
```
next_siblings = tag.find_next_siblings('p')
for sibling in next_siblings:
    print(sibling)
```

find_next_sibling Function

The `find_next_sibling` function searches for the first next sibling tag that matches the given criteria. It takes in various parameters to specify the tag and its attributes.

Parameters:

- `name` (str): The name of the next sibling tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the next sibling tag.

Returns:

- `Tag` : The first next sibling tag that matches the given criteria, or None if no match is found.

Example:

```
next_sibling = tag.find_next_sibling('p')
print(next_sibling)
```

This code will find and print the first next sibling `<p>` tag of the first `<p>` tag.

In [ ]:
```
next_sibling = tag.find_next_sibling('p')
print(next_sibling)
```

```
None
```

find_previous_siblings Function

The `find_previous_siblings` function searches for all previous sibling tags that match the given criteria. It takes in various parameters to specify the tags and their attributes.

Parameters:

- `name` (str): The name of the previous sibling tags to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the previous sibling tags.

Returns:

- `ResultSet` : A list of previous sibling tags that match the given criteria.

Example:

```
previous_siblings = tag.find_previous_siblings('p')
for sibling in previous_siblings:
    print(sibling)
```

This code will find and print all previous sibling `<p>` tags of the first `<p>` tag.

In [ ]:
```
previous_siblings = tag.find_previous_siblings('p')
for sibling in previous_siblings:
    print(sibling)
```

find_previous_sibling Function

The `find_previous_sibling` function searches for the first previous sibling tag that matches the given criteria. It takes in various parameters to specify the tag and its attributes.

Parameters:

- `name` (str): The name of the previous sibling tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the previous sibling tag.

Returns:

- `Tag` : The first previous sibling tag that matches the given criteria, or None if no match is found.

Example:

```
previous_sibling = tag.find_previous_sibling('p')
print(previous_sibling)
```
This code will find and print the first previous sibling `<p>` tag of the first `<p>` tag.

In [ ]:
```
previous_sibling = tag.find_previous_sibling('p')
print(previous_sibling)
```

```
None
```

find_all_next Function

The `find_all_next` function searches for all tags that match the given criteria after the current tag. It takes in various parameters to specify the tags and their attributes.

Parameters:

- `name` (str): The name of the tags to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the tags.
- `limit` (int): The maximum number of tags to return. Defaults to None.

Returns:

- `ResultSet` : A list of tags that match the given criteria after the current tag.

Example:

```
next_tags = tag.find_all_next('p')
for next_tag in next_tags:
    print(next_tag)
```
This code will find and print all `<p>` tags that come after the first `<p>` tag.

In [ ]:
```
next_tags = tag.find_all_next('p')
for next_tag in next_tags:
    print(next_tag)
```

find_next Function

The `find_next` function searches for the first tag that matches the given criteria after the current tag. It takes in various parameters to specify the tag and its attributes.

Parameters:

- `name` (str): The name of the tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the tag.

Returns:

- `Tag` : The first tag that matches the given criteria after the current tag, or None if no match is found.

Example:

```
next_tag = tag.find_next('p')
print(next_tag)
```

This code will find and print the first `<p>` tag that comes after the first `<p>` tag.

In [ ]:
```
next_tag = tag.find_next('p')
print(next_tag)
```

```
None
```

find_all_previous Function

The `find_all_previous` function searches for all tags that match the given criteria before the current tag. It takes in various parameters to specify the tags and their attributes.

Parameters:

- `name` (str): The name of the tags to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the tags.
- `limit` (int): The maximum number of tags to return. Defaults to None.

Returns:

- `ResultSet` : A list of tags that match the given criteria before the current tag.

Example:

```
previous_tags = tag.find_all_previous('p')
for previous_tag in previous_tags:
    print(previous_tag)
```

This code will find and print all `<p>` tags that come before the first `<p>` tag.

In [ ]:
```
previous_tags = tag.find_all_previous('p')
for previous_tag in previous_tags:
    print(previous_tag)
```

find_previous Function

The `find_previous` function searches for the first tag that matches the given criteria before the current tag. It takes in various parameters to specify the tag and its attributes.

Parameters:

- `name` (str): The name of the tag to search for.
- `attrs` (dict): A dictionary of attributes and their values to match.
- `text` (str or callable): A string or a callable to match the text within the tag.

Returns:

- `Tag` : The first tag that matches the given criteria before the current tag, or None if no match is found.

Example:

```
previous_tag = tag.find_previous('p')
print(previous_tag)
```

This code will find and print the first `<p>` tag that comes before the first `<p>` tag.

In [ ]:
```
previous_tag = tag.find_previous('p')
print(previous_tag)
```

```
None
```

get Function

The `get` function retrieves an attribute value of a tag. It takes in the attribute name as a parameter.

Parameters:

- `key` (str): The name of the attribute.

Returns:

- `str` : The value of the attribute, or None if the attribute is not found.

Example:

```
attr_value = tag.get('id')
print(attr_value)
```

This code will retrieve and print the value of the `id` attribute of the first `<p>` tag.

In [ ]:
```
tag = soup.find('p')
attr_value = tag.get('id')
print(attr_value)
```

```
main
```

replace_with Function

The `replace_with` function replaces a tag with another tag or string.

Parameters:

- `new_tag` (Tag or str): The new tag or string to replace the current tag with.

Returns:

- `Tag` : The new tag or string that replaced the current tag.

Example:

```
new_tag = soup.new_tag('p')
new_tag.string = 'A new paragraph.'
soup.body.replace_with(new_tag)
print(soup.prettify())
```

This code will replace the `<body>` tag with a new `<p>` tag containing the text 'A new paragraph.' and print the updated document.

In [ ]:
```
new_tag = soup.new_tag('p')
new_tag.string = 'A new paragraph.'
soup.body.replace_with(new_tag)
print(soup.prettify())
```

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   Sample Web Page
  </title>
 </head>
 <p>
  A new paragraph.
 </p>
</html>
```

insert Function

The `insert` function inserts a new tag at a specified position within a parent tag.

Parameters:

- `position` (int): The position at which to insert the new tag.
- `new_tag` (Tag): The new tag to insert.

Returns:

- `Tag` : The new tag that was inserted.

Example:

```
parent_tag = soup.head
parent_tag.insert(1, new_tag)
print(soup.prettify())
```

This code will insert the new `<div>` tag as the second child of the `<head>` tag and print the updated document.

In [ ]:
```
parent_tag = soup.head
parent_tag.insert(1, new_tag)
print(soup.prettify())
```

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <p>
   A new paragraph.
  </p>
  <meta charset="utf-8"/>
  <title>
   Sample Web Page
  </title>
 </head>
</html>
```

append Function

The `append` function appends a new tag at the end of a parent tag.

Parameters:

- `new_tag` (Tag): The new tag to append.

Returns:

- `Tag` : The new tag that was appended.

Example:

```
parent_tag = soup.head
parent_tag.append(new_tag)
print(soup.prettify())
```

This code will append the new `<div>` tag at the end of the `<head>` tag and print the updated document.

In [ ]:
```
parent_tag = soup.head
parent_tag.append(new_tag)
print(soup.prettify())
```

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   Sample Web Page
  </title>
  <p>
   A new paragraph.
  </p>
 </head>
</html>
```

string Function

The `string` function accesses the string within a tag. It returns the string content of the tag.

Parameters:

- None

Returns:

- `NavigableString` : The string content of the tag, or None if the tag contains more than one string.

Example:

```
tag_string = tag.string
print(tag_string)
```

This code will retrieve and print the string content of the first `<p>` tag.

In [ ]:
```
tag_string = tag.string
print(tag_string)
```

```
This is another example paragraph with an ID attribute.
```

strings Function

The `strings` function accesses all strings within a tag. It returns a generator of the string contents of the tag.

Parameters:

- None

Returns:

- `generator` : A generator of the string contents of the tag.

Example:

```
for string in tag.strings:
    print(string)
```

This code will retrieve and print all string contents of the first `<p>` tag.

In [ ]:
```
for string in tag.strings:
    print(string)
```

```
This is another example paragraph with an ID attribute.
```

stripped_strings Function

The `stripped_strings` function accesses all stripped strings within a tag. It returns a generator of the stripped string contents of the tag.

Parameters:

- None

Returns:

- `generator` : A generator of the stripped string contents of the tag.

Example:

```
for string in tag.stripped_strings:
    print(string)
```

This code will retrieve and print all stripped string contents of the first `<p>` tag.

In [ ]:
```
for string in tag.stripped_strings:
    print(string)
```

```
This is another example paragraph with an ID attribute.
```

Follow me on Sumit Khanna for more updates