

20.2. `html.parser` — Simple HTML and XHTML parser

Source code: [Lib/html/parser.py](https://lib.html.parser.py)

This module defines a class `HTMLParser` which serves as the basis for parsing text files formatted in HTML (HyperText Mark-up Language) and XHTML.

`class html.parser.HTMLParser(*, convert_charrefs=True)`

Create a parser instance able to parse invalid markup.

If `convert_charrefs` is `True` (the default), all character references (except the ones in `script/style` elements) are automatically converted to the corresponding Unicode characters.

An `HTMLParser` instance is fed HTML data and calls handler methods when start tags, end tags, text, comments, and other markup elements are encountered. The user should subclass `HTMLParser` and override its methods to implement the desired behavior.

This parser does not check that end tags match start tags or call the end-tag handler for elements which are closed implicitly by closing an outer element.

Changed in version 3.4: `convert_charrefs` keyword argument added.

Changed in version 3.5: The default value for argument `convert_charrefs` is now `True`.

20.2.1. Example HTML Parser Application

As a basic example, below is a simple HTML parser that uses the `HTMLParser` class to print out start tags, end tags, and data as they are encountered:

```
from html.parser import HTMLParser

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print("Encountered a start tag:", tag)

    def handle_endtag(self, tag):
        print("Encountered an end tag :", tag)

    def handle_data(self, data):
```

```
print("Encountered some data :", data)

parser = MyHTMLParser()
parser.feed('<html><head><title>Test</title></head>'
           '<body><h1>Parse me!</h1></body></html>')
```

The output will then be:

```
Encountered a start tag: html
Encountered a start tag: head
Encountered a start tag: title
Encountered some data : Test
Encountered an end tag : title
Encountered an end tag : head
Encountered a start tag: body
Encountered a start tag: h1
Encountered some data : Parse me!
Encountered an end tag : h1
Encountered an end tag : body
Encountered an end tag : html
```

20.2.2. HTMLParser Methods

[HTMLParser](#) instances have the following methods:

`HTMLParser.feed(data)`

Feed some text to the parser. It is processed insofar as it consists of complete elements; incomplete data is buffered until more data is fed or [close\(\)](#) is called. *data* must be [str](#).

`HTMLParser.close()`

Force processing of all buffered data as if it were followed by an end-of-file mark. This method may be redefined by a derived class to define additional processing at the end of the input, but the redefined version should always call the [HTMLParser](#) base class method [close\(\)](#).

`HTMLParser.reset()`

Reset the instance. Loses all unprocessed data. This is called implicitly at instantiation time.

`HTMLParser.getpos()`

Return current line number and offset.

`HTMLParser.get_starttag_text()`

Return the text of the most recently opened start tag. This should not normally be needed for structured processing, but may be useful in dealing with HTML

“as deployed” or for re-generating input with minimal changes (whitespace between attributes can be preserved, etc.).

The following methods are called when data or markup elements are encountered and they are meant to be overridden in a subclass. The base class implementations do nothing (except for `handle_startendtag()`):

`HTMLParser.handle_starttag(tag, attrs)`

This method is called to handle the start of a tag (e.g. `<div id="main">`).

The *tag* argument is the name of the tag converted to lower case. The *attrs* argument is a list of (name, value) pairs containing the attributes found inside the tag's `<>` brackets. The *name* will be translated to lower case, and quotes in the *value* have been removed, and character and entity references have been replaced.

For instance, for the tag ``, this method would be called as `handle_starttag('a', [('href', 'https://www.cwi.nl/')])`.

All entity references from `html.entities` are replaced in the attribute values.

`HTMLParser.handle_endtag(tag)`

This method is called to handle the end tag of an element (e.g. `</div>`).

The *tag* argument is the name of the tag converted to lower case.

`HTMLParser.handle_startendtag(tag, attrs)`

Similar to `handle_starttag()`, but called when the parser encounters an XHTML-style empty tag (``). This method may be overridden by subclasses which require this particular lexical information; the default implementation simply calls `handle_starttag()` and `handle_endtag()`.

`HTMLParser.handle_data(data)`

This method is called to process arbitrary data (e.g. text nodes and the content of `<script>...</script>` and `<style>...</style>`).

`HTMLParser.handle_entityref(name)`

This method is called to process a named character reference of the form `&name;` (e.g. `>`), where *name* is a general entity reference (e.g. `'gt'`). This method is never called if `convert_charrefs` is `True`.

`HTMLParser.handle_charref(name)`

This method is called to process decimal and hexadecimal numeric character references of the form `&#NNN;` and `&#xNNN;`. For example, the decimal equiva-

lent for `>` is `>`, whereas the hexadecimal is `>`; in this case the method will receive `'62'` or `'x3E'`. This method is never called if `convert_charrefs` is `True`.

`HTMLParser.handle_comment(data)`

This method is called when a comment is encountered (e.g. `<!--comment-->`).

For example, the comment `<!-- comment -->` will cause this method to be called with the argument `' comment '`.

The content of Internet Explorer conditional comments (condcoms) will also be sent to this method, so, for `<!--[if IE 9]>IE9-specific content<![endif]-->`, this method will receive `'[if IE 9]>IE9-specific content<![endif]'`.

`HTMLParser.handle_decl(decl)`

This method is called to handle an HTML doctype declaration (e.g. `<!DOCTYPE html>`).

The `decl` parameter will be the entire contents of the declaration inside the `<!...>` markup (e.g. `'DOCTYPE html'`).

`HTMLParser.handle_pi(data)`

Method called when a processing instruction is encountered. The `data` parameter will contain the entire processing instruction. For example, for the processing instruction `<?proc color='red'>`, this method would be called as `handle_pi("proc color='red'")`. It is intended to be overridden by a derived class; the base class implementation does nothing.

Note: The `HTMLParser` class uses the SGML syntactic rules for processing instructions. An XHTML processing instruction using the trailing `'?'` will cause the `'?'` to be included in `data`.

`HTMLParser.unknown_decl(data)`

This method is called when an unrecognized declaration is read by the parser.

The `data` parameter will be the entire contents of the declaration inside the `<![...]>` markup. It is sometimes useful to be overridden by a derived class. The base class implementation does nothing.

20.2.3. Examples

The following class implements a parser that will be used to illustrate more examples:

```

from html.parser import HTMLParser
from html.entities import name2codepoint

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print("Start tag:", tag)
        for attr in attrs:
            print("    attr:", attr)

    def handle_endtag(self, tag):
        print("End tag  :", tag)

    def handle_data(self, data):
        print("Data      :", data)

    def handle_comment(self, data):
        print("Comment  :", data)

    def handle_entityref(self, name):
        c = chr(name2codepoint[name])
        print("Named ent:", c)

    def handle_charref(self, name):
        if name.startswith('x'):
            c = chr(int(name[1:], 16))
        else:
            c = chr(int(name))
        print("Num ent  :", c)

    def handle_decl(self, data):
        print("Decl      :", data)

parser = MyHTMLParser()

```

Parsing a doctype:

```

>>> parser.feed('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...             "http://www.w3.org/TR/html4/strict.dtd">')
Decl      : DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www
<

```

Parsing an element with a few attributes and a title:

```

>>> parser.feed('')
Start tag: img
      attr: ('src', 'python-logo.png')
      attr: ('alt', 'The Python logo')
>>>
>>> parser.feed('<h1>Python</h1>')
Start tag: h1

```

```
Data      : Python
End tag   : h1
```

The content of script and style elements is returned as is, without further parsing:

```
>>> parser.feed('<style type="text/css">#python { color: green }</style>')
Start tag: style
  attr: ('type', 'text/css')
Data      : #python { color: green }
End tag   : style

>>> parser.feed('<script type="text/javascript">'
...             'alert("<strong>hello!</strong>");</script>')
Start tag: script
  attr: ('type', 'text/javascript')
Data      : alert("<strong>hello!</strong>");
End tag   : script
```

Parsing comments:

```
>>> parser.feed('<!-- a comment -->'
...             '<!--[if IE 9]>IE-specific content<![endif]-->')
Comment   : a comment
Comment   : [if IE 9]>IE-specific content<![endif]
```

Parsing named and numeric character references and converting them to the correct char (note: these 3 references are all equivalent to '>'):

```
>>> parser.feed('&gt;&#62;&#x3E;')
Named ent: >
Num ent   : >
Num ent   : >
```

Feeding incomplete chunks to `feed()` works, but `handle_data()` might be called more than once (unless `convert_charrefs` is set to True):

```
>>> for chunk in ['<sp', 'an>buff', 'ered ', 'text</s', 'pan>']:
...     parser.feed(chunk)
...
Start tag: span
Data      : buff
Data      : ered
Data      : text
End tag   : span
```

Parsing invalid HTML (e.g. unquoted attributes) also works:

```
>>> parser.feed('<p><a class=link href=#main>tag soup</p ></a>') >>>
Start tag: p
Start tag: a
  attr: ('class', 'link')
  attr: ('href', '#main')
Data      : tag soup
End tag   : p
End tag   : a
```