urllib.parse — Split URLs into Components

Purpose: Split URL into components

The `urllib.parse` module provides functions for manipulating URLs and their component parts, to either break them down or build them up.

Parsing

The return value from the `urlparse()` function is a `ParseResult` object that acts like a `tuple` with six elements.

```
# urllib_parse_urlparse.py

from urllib.parse import urlparse

url = 'http://netloc/path;param?query=arg#frag'
parsed = urlparse(url)
print(parsed)
```

The parts of the URL available through the tuple interface are the scheme, network location, path, path segment parameters (separated from the path by a semicolon), query, and fragment.

```
$ python3 urllib_parse_urlparse.py

ParseResult(scheme='http', netloc='netloc', path='/path',
params='param', query='query=arg', fragment='frag')
```

Although the return value acts like a tuple, it is really based on a `namedtuple`, a subclass of `tuple` that supports accessing the parts of the URL via named attributes as well as indexes. In addition to being easier to use for the programmer, the attribute API also offers access to several values not available in the `tuple` API.

```
# urllib_parse_urlparseattrs.py

from urllib.parse import urlparse

url = 'http://user:pwd@NetLoc:80/path;param?query=arg#frag'
parsed = urlparse(url)
print('scheme  :', parsed.scheme)
print('netloc  :', parsed.netloc)
print('path    :', parsed.path)
print('params  :', parsed.params)
print('query   :', parsed.query)
print('fragment:', parsed.fragment)
print('username:', parsed.username)
print('password:', parsed.password)
print('hostname:', parsed.hostname)
print('port    :', parsed.port)
```

The `username` and `password` are available when present in the input URL, and set to `None` when not. The `hostname` is the same value as `netloc`, in all lower case and with the port value stripped. And the `port` is converted to an integer when present and `None` when not.

```
$ python3 urllib_parse_urlparseattrs.py

scheme  : http
netloc  : user:pwd@NetLoc:80
path    : /path
params  : param
query   : query=arg
fragment: frag
username: user
password: pwd
```

```
hostname: netloc
port    : 80
```

The `urlsplit()` function is an alternative to `urlparse()`. It behaves a little differently, because it does not split the parameters from the URL. This is useful for URLs following **RFC 2396**, which supports parameters for each segment of the path.

```
# urllib_parse_urlsplit.py

from urllib.parse import urlsplit

url = 'http://user:pwd@NetLoc:80/p1;para/p2;para?query=arg#frag'
parsed = urlsplit(url)
print(parsed)
print('scheme  :', parsed.scheme)
print('netloc  :', parsed.netloc)
print('path    :', parsed.path)
print('query   :', parsed.query)
print('fragment:', parsed.fragment)
print('username:', parsed.username)
print('password:', parsed.password)
print('hostname:', parsed.hostname)
print('port    :', parsed.port)
```

Since the parameters are not split out, the tuple API will show five elements instead of six, and there is no `params` attribute.

```
$ python3 urllib_parse_urlsplit.py

SplitResult(scheme='http', netloc='user:pwd@NetLoc:80',
path='/p1;para/p2;para', query='query=arg', fragment='frag')
scheme  : http
netloc  : user:pwd@NetLoc:80
path    : /p1;para/p2;para
query   : query=arg
fragment: frag
username: user
password: pwd
hostname: netloc
port    : 80
```

To simply strip the fragment identifier from a URL, such as when finding a base page name from a URL, use `urldefrag()`.

```
# urllib_parse_urldefrag.py

from urllib.parse import urldefrag

original = 'http://netloc/path;param?query=arg#frag'
print('original:', original)
d = urldefrag(original)
print('url     :', d.url)
print('fragment:', d.fragment)
```

The return value is a `DefragResult`, based on `namedtuple`, containing the base URL and the fragment.

```
$ python3 urllib_parse_urldefrag.py

original: http://netloc/path;param?query=arg#frag
url     : http://netloc/path;param?query=arg
fragment: frag
```

Unparsing

There are several ways to assemble the parts of a split URL back together into a single string. The parsed URL object has a `geturl()` method.

```
# urllib_parse_geturl.py

from urllib.parse import urlparse
```

```
from urllib.parse import urlparse

original = 'http://netloc/path;param?query=arg#frag'
print('ORIG  :', original)
parsed = urlparse(original)
print('PARSED:', parsed.geturl())
```

`geturl()` only works on the object returned by `urlparse()` or `urlsplit()`.

```
$ python3 urllib_parse_geturl.py

ORIG  : http://netloc/path;param?query=arg#frag
PARSED: http://netloc/path;param?query=arg#frag
```

A regular tuple containing strings can be combined into a URL with `urlunparse()`.

```
# urllib_parse_urlunparse.py

from urllib.parse import urlparse, urlunparse

original = 'http://netloc/path;param?query=arg#frag'
print('ORIG  :', original)
parsed = urlparse(original)
print('PARSED:', type(parsed), parsed)
t = parsed[:]
print('TUPLE :', type(t), t)
print('NEW   :', urlunparse(t))
```

While the `ParseResult` returned by `urlparse()` can be used as a tuple, this example explicitly creates a new tuple to show that `urlunparse()` works with normal tuples, too.

```
$ python3 urllib_parse_urlunparse.py

ORIG  : http://netloc/path;param?query=arg#frag
PARSED: <class 'urllib.parse.ParseResult'>
ParseResult(scheme='http', netloc='netloc', path='/path',
params='param', query='query=arg', fragment='frag')
TUPLE : <class 'tuple'> ('http', 'netloc', '/path', 'param',
'query=arg', 'frag')
NEW   : http://netloc/path;param?query=arg#frag
```

If the input URL included superfluous parts, those may be dropped from the reconstructed URL.

```
# urllib_parse_urlunparseextra.py

from urllib.parse import urlparse, urlunparse

original = 'http://netloc/path;?#'
print('ORIG  :', original)
parsed = urlparse(original)
print('PARSED:', type(parsed), parsed)
t = parsed[:]
print('TUPLE :', type(t), t)
print('NEW   :', urlunparse(t))
```

In this case, `parameters`, `query`, and `fragment` are all missing in the original URL. The new URL does not look the same as the original, but is equivalent according to the standard.

```
$ python3 urllib_parse_urlunparseextra.py

ORIG  : http://netloc/path;?#
PARSED: <class 'urllib.parse.ParseResult'>
ParseResult(scheme='http', netloc='netloc', path='/path',
params='', query='', fragment='')
TUPLE : <class 'tuple'> ('http', 'netloc', '/path', '', '', '')
NEW   : http://netloc/path
```

Joining

In addition to parsing URLs, `urlparse` includes `urljoin()` for constructing absolute URLs from relative fragments.

```
# urllib_parse_urljoin.py

from urllib.parse import urljoin

print(urljoin('http://www.example.com/path/file.html',
              'anotherfile.html'))
print(urljoin('http://www.example.com/path/file.html',
              '../anotherfile.html'))
```

In the example, the relative portion of the path (`"../"`) is taken into account when the second URL is computed.

```
$ python3 urllib_parse_urljoin.py

http://www.example.com/path/anotherfile.html
http://www.example.com/anotherfile.html
```

Non-relative paths are handled in the same way as by `os.path.join()`.

```
# urllib_parse_urljoin_with_path.py

from urllib.parse import urljoin

print(urljoin('http://www.example.com/path/',
              '/subpath/file.html'))
print(urljoin('http://www.example.com/path/',
              'subpath/file.html'))
```

If the path being joined to the URL starts with a slash (`/`), it resets the URL's path to the top level. If it does not start with a slash, it is appended to the end of the path for the URL.

```
$ python3 urllib_parse_urljoin_with_path.py

http://www.example.com/subpath/file.html
http://www.example.com/path/subpath/file.html
```

Encoding Query Arguments

Before arguments can be added to a URL, they need to be encoded.

```
# urllib_parse_urlencode.py

from urllib.parse import urlencode

query_args = {
    'q': 'query string',
    'foo': 'bar',
}
encoded_args = urlencode(query_args)
print('Encoded:', encoded_args)
```

Encoding replaces special characters like spaces to ensure they are passed to the server using a format that complies with the standard.

```
$ python3 urllib_parse_urlencode.py

Encoded: q=query+string&foo=bar
```

To pass a sequence of values using separate occurrences of the variable in the query string, set `doseq` to `True` when calling `urlencode()`.

```
# urllib_parse_urlencode_doseq.py

from urllib.parse import urlencode
```

```
query_args = {
    'foo': ['foo1', 'foo2'],
}
print('Single  :', urlencode(query_args))
print('Sequence:', urlencode(query_args, doseq=True))
```

The result is a query string with several values associated with the same name.

```
$ python3 urllib_parse_urlencode_doseq.py

Single  : foo=%5B%27foo1%27%2C+%27foo2%27%5D
Sequence: foo=foo1&foo=foo2
```

To decode the query string, use `parse_qs()` or `parse_qsl()`.

```
# urllib_parse_parse_qs.py
```

```
from urllib.parse import parse_qs, parse_qsl

encoded = 'foo=foo1&foo=foo2'

print('parse_qs :', parse_qs(encoded))
print('parse_qsl:', parse_qsl(encoded))
```

The return value from `parse_qs()` is a dictionary mapping names to values, while `parse_qsl()` returns a list of tuples containing a name and a value.

```
$ python3 urllib_parse_parse_qs.py

parse_qs : {'foo': ['foo1', 'foo2']}
parse_qsl: [('foo', 'foo1'), ('foo', 'foo2')]
```

Special characters within the query arguments that might cause parse problems with the URL on the server side are “quoted” when passed to `urlencode()`. To quote them locally to make safe versions of the strings, use the `quote()` or `quote_plus()` functions directly.

```
# urllib_parse_quote.py
```

```
from urllib.parse import quote, quote_plus, urlencode

url = 'http://localhost:8080/~hellmann/'
print('urlencode() :', urlencode({'url': url}))
print('quote()     :', quote(url))
print('quote_plus():', quote_plus(url))
```

The quoting implementation in `quote_plus()` is more aggressive about the characters it replaces.

```
$ python3 urllib_parse_quote.py

urlencode() : url=http%3A%2F%2Flocalhost%3A8080%2F~hellmann%2F
quote()     : http%3A//localhost%3A8080/~hellmann/
quote_plus(): http%3A%2F%2Flocalhost%3A8080%2F~hellmann%2F
```

To reverse the quote operations, use `unquote()` or `unquote_plus()`, as appropriate.

```
# urllib_parse_unquote.py
```

```
from urllib.parse import unquote, unquote_plus

print(unquote('http%3A//localhost%3A8080/%7Ehellmann/'))
print(unquote_plus(
    'http%3A%2F%2Flocalhost%3A8080%2F%7Ehellmann%2F'
))
```

The encoded value is converted back to a normal string URL.

```
$ python3 urllib_parse_unquote.py

http://localhost:8080/~hellmann/
http://localhost:8080/~hellmann/
```

See also

- Standard library documentation for urllib.parse
- `urllib.request` – Retrieve the contents of a resource identified by a URL.
- **RFC 1738** – Uniform Resource Locator (URL) syntax
- **RFC 1808** – Relative URLs
- **RFC 2396** – Uniform Resource Identifier (URI) generic syntax
- **RFC 3986** – Uniform Resource Identifier (URI) syntax

The Internet

urllib.request — Network Resource Access

Quick Links

Parsing
Unparsing
Joining
Encoding Query Arguments

This page was last updated 2018-12-09.

Navigation

The Internet
urllib.request — Network Resource Access

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example