21.23. `http.cookies` — HTTP state management

Source code: Lib/http/cookies.py

The `http.cookies` module defines classes for abstracting the concept of cookies, an HTTP state management mechanism. It supports both simple string-only cookies, and provides an abstraction for having any serializable data-type as cookie value.

The module formerly strictly applied the parsing rules described in the **RFC 2109** and **RFC 2068** specifications. It has since been discovered that MSIE 3.0x doesn't follow the character rules outlined in those specs and also many current day browsers and servers have relaxed parsing rules when comes to Cookie handling. As a result, the parsing rules used are a bit less strict.

The character set, `string.ascii_letters`, `string.digits` and `!#$%&'*+-.^_`|~:` denote the set of valid characters allowed by this module in Cookie name (as `key`).

Changed in version 3.3: Allowed ':' as a valid Cookie name character.

> **Note:** On encountering an invalid cookie, `CookieError` is raised, so if your cookie data comes from a browser you should always prepare for invalid data and catch `CookieError` on parsing.

exception `http.cookies.`**`CookieError`**

Exception failing because of **RFC 2109** invalidity: incorrect attributes, incorrect *Set-Cookie* header, etc.

class `http.cookies.`**`BaseCookie`**([*input*])

This class is a dictionary-like object whose keys are strings and whose values are `Morsel` instances. Note that upon setting a key to a value, the value is first converted to a `Morsel` containing the key and the value.

If *input* is given, it is passed to the `load()` method.

class `http.cookies.`**`SimpleCookie`**([*input*])

This class derives from `BaseCookie` and overrides `value_decode()` and `value_encode()` to be the identity and `str()` respectively.

See also:

Module `http.cookiejar`
: HTTP cookie handling for web *clients*. The `http.cookiejar` and `http.cookies` modules do not depend on each other.

RFC 2109 - HTTP State Management Mechanism
: This is the state management specification implemented by this module.

21.23.1. Cookie Objects

`BaseCookie.`**`value_decode`**(*val*)

Return a decoded value from a string representation. Return value can be any type. This method does nothing in `BaseCookie` — it exists so it can be overridden.

`BaseCookie.`**`value_encode`**(*val*)

Return an encoded value. *val* can be any type, but return value must be a string. This method does nothing in `BaseCookie` — it exists so it can be overridden.

In general, it should be the case that `value_encode()` and `value_decode()` are inverses on the range of *value_decode*.

`BaseCookie.`**`output`**(*attrs=None*, *header='Set-Cookie:'*, *sep='\r\n'*)

Return a string representation suitable to be sent as HTTP headers. *attrs* and *header* are sent to each `Morsel`'s `output()` method. *sep* is used to join the headers together, and is by default the combination `'\r\n'` (CRLF).

`BaseCookie.`**`js_output`**(*attrs=None*)

Return an embeddable JavaScript snippet, which, if run on a browser which supports JavaScript, will act the same as if the HTTP headers was sent.

The meaning for *attrs* is the same as in `output()`.

`BaseCookie.`**`load`**(*rawdata*)

If *rawdata* is a string, parse it as an `HTTP_COOKIE` and add the values found there as `Morsel`s. If it is a dictionary, it is equivalent to:

```
for k, v in rawdata.items():
    cookie[k] = v
```

21.23.2. Morsel Objects

class `http.cookies.`**`Morsel`**

Abstract a key/value pair, which has some **RFC 2109** attributes.

Morsels are dictionary-like objects, whose set of keys is constant — the valid **RFC 2109** attributes, which are

- `expires`
- `path`
- `comment`
- `domain`
- `max-age`
- `secure`
- `version`
- `httponly`

The attribute `httponly` specifies that the cookie is only transferred in HTTP requests, and is not accessible through JavaScript. This is intended to mitigate some forms of cross-site scripting.

The keys are case-insensitive and their default value is `''`.

Changed in version 3.5: `__eq__()` now takes `key` and `value` into account.

`Morsel.`**`value`**

The value of the cookie.

Deprecated since version 3.5: assigning to `value`; use `set()` instead.

`Morsel.`**`coded_value`**

The encoded value of the cookie — this is what should be sent.

Deprecated since version 3.5: assigning to `coded_value`; use `set()` instead.

`Morsel.`**`key`**

The name of the cookie.

Deprecated since version 3.5: assigning to `key`; use `set()` instead.

`Morsel.`**`set`**(*key*, *value*, *coded_value*)

Set the *key*, *value* and *coded_value* attributes.

Deprecated since version 3.5: The undocumented *LegalChars* parameter is ignored and will be removed in a future version.

`Morsel.`**`isReservedKey`**(*K*)

Whether *K* is a member of the set of keys of a `Morsel`.

`Morsel.`**`output`**(*attrs=None*, *header='Set-Cookie:'*)

Return a string representation of the Morsel, suitable to be sent as an HTTP header. By default, all the attributes are included, unless *attrs* is given, in which case it should be a list of attributes to use. *header* is by default `"Set-Cookie:"`.

`Morsel.`**`js_output`**(*attrs=None*)

Return an embeddable JavaScript snippet, which, if run on a browser which supports JavaScript, will act the same as if the HTTP header was sent.

The meaning for *attrs* is the same as in `output()`.

`Morsel.`**`OutputString`**(*attrs=None*)

Return a string representing the Morsel, without any surrounding HTTP or JavaScript.

The meaning for *attrs* is the same as in `output()`.

`Morsel.`**`update`**(*values*)

Update the values in the Morsel dictionary with the values in the dictionary *values*. Raise an error if any of the keys in the *values* dict is not a valid **RFC 2109** attribute.

Changed in version 3.5: an error is raised for invalid keys.

`Morsel.`**`copy`**(*value*)

Return a shallow copy of the Morsel object.

Changed in version 3.5: return a Morsel object instead of a dict.

`Morsel.`**`setdefault`**(*key*, *value=None*)

Raise an error if key is not a valid **RFC 2109** attribute, otherwise behave the same as `dict.setdefault()`.

21.23.3. Example

The following example demonstrates how to use the `http.cookies` module.

```
>>> from http import cookies
>>> C = cookies.SimpleCookie()
>>> C["fig"] = "newton"
>>> C["sugar"] = "wafer"
>>> print(C) # generate HTTP headers
```

```
Set-Cookie: fig=newton
Set-Cookie: sugar=wafer
>>> print(C.output()) # same thing
Set-Cookie: fig=newton
Set-Cookie: sugar=wafer
>>> C = cookies.SimpleCookie()
>>> C["rocky"] = "road"
>>> C["rocky"]["path"] = "/cookie"
>>> print(C.output(header="Cookie:"))
Cookie: rocky=road; Path=/cookie
>>> print(C.output(attrs=[], header="Cookie:"))
Cookie: rocky=road
>>> C = cookies.SimpleCookie()
>>> C.load("chips=ahoy; vienna=finger") # load from a string (HTTP hea
>>> print(C)
Set-Cookie: chips=ahoy
Set-Cookie: vienna=finger
>>> C = cookies.SimpleCookie()
>>> C.load('keebler="E=everybody; L=\\"Loves\\"; fudge=\\012;";')
>>> print(C)
Set-Cookie: keebler="E=everybody; L=\"Loves\"; fudge=\012;"
>>> C = cookies.SimpleCookie()
>>> C["oreo"] = "doublestuff"
>>> C["oreo"]["path"] = "/"
>>> print(C)
Set-Cookie: oreo=doublestuff; Path=/
>>> C = cookies.SimpleCookie()
>>> C["twix"] = "none for you"
>>> C["twix"].value
'none for you'
>>> C = cookies.SimpleCookie()
>>> C["number"] = 7 # equivalent to C["number"] = str(7)
>>> C["string"] = "seven"
>>> C["number"].value
'7'
>>> C["string"].value
'seven'
>>> print(C)
Set-Cookie: number=7
Set-Cookie: string=seven
```