19. Client-Side Network Protocol Modules

Functions

The module `urllib` in v2 supplies a number of functions described in Table 19-1, with `urlopen` being the simplest and most frequently used.

Table 19-1.

quote	`quote(str,safe='/')`
	Returns a copy of `str` where special characters are changed into Internet-standard quoted form `%xx`. Does not quote alphanumeric characters, spaces, any of the characters `_,.-` (underscore, comma, period, hyphen), nor any of the characters in string `safe`. For example:
	`print(urllib.quote('zip&zap'))# prints: zip%26zap`

quote_plus	`quote_plus(str, safe='/')` Like `quote`, but also changes spaces into plus signs.

unquote	`unquote(str)`
	Returns a copy of `str` where each quoted form `%xx` is changed into the corresponding character. For example:
	`print(urllib.unquote('zip%26zap'))# prints: zip&zap`

unquote_plus	`unquote_plus(str)` Like `unquote`, but also changes plus signs into spaces.
urlcleanup	`urlcleanup()` Clears the cache of function `urlretrieve`, covered below.

urlencode `urlencode(query,doseq=False)`

Returns a string with the URL-encoded form of `query`. `query` can be either a sequence of `(name, value)` pairs, or a mapping, in which case the resulting string encodes the mapping's `(key, value)` pairs. For example:

```
urllib.urlencode([('ans',42),('key','val')])# result is:
'ans=42&key=val'urllib.urlencode({'ans':42, 'key':'val'})# result is:
'key=val&ans=42'
```

The order of items in a dictionary is arbitrary. Should you need the URL-encoded form to have key/value pairs in a specific order, use a sequence as the `query` argument, as in the first call in this snippet.

When `doseq` is true, any `value` in `query` that is a sequence and is not a string is encoded as separate parameters, one per item in `value`. For example:

```
urllib.urlencode([('K',('x','y','z'))], True)# result is: 'K=x&K=y&K=z'
```

When `doseq` is false (the default), each value is encoded as the `quote_plus` of its string form given by built-in `str`, regardless of whether the value is a sequence:

```
urllib.urlencode([('K',('x','y','z'))], False)# result is:
'K=%28%27x%27%2C+%27y%27%2C+%27z%27%29'
```

urlopen `urlopen(urlstring,data=None,proxies=None)`

Accesses the given URL and returns a read-only file-like object `f`. `f` supplies file-like methods `read`, `readline`, `readlines`, and `close`, as well as two others:

`f``.geturl()`

> Returns the URL of `f`. This may differ from `urlstring` by normalization (as mentioned for function `urlunsplit` earlier) and because of HTTP redirects (i.e., indications that the requested data is located elsewhere). `urllib` supports redirects transparently, and the method `geturl` lets you check for them if you want.

`f``.info()`

> Returns an instance `m` of the class `Message` of the module `mimetools`, covered in "rfc822 and mimetools Modules (v2)". `m`'s headers provide metadata about `f`. For example, `'Content-Type'` is the MIME type of the data in `f`, and `m`'s methods `m.gettype()`, `m.getmaintype()`, and `m.getsubtype()` provide the same information.

When `data` is `None` and `urlstring`'s scheme is *http*, `urlopen` sends a `GET` request. When `data` is not `None`, `urlstring`'s scheme must be *http*, and `urlopen` sends a `POST` request. `data` must then be in URL-encoded form, and you normally prepare it with the function `urlencode`, covered above.

`urlopen` can use proxies that do not require authentication. Set the environment variables `http_proxy`, `ftp_proxy`, and `gopher_proxy` to the proxies' URLs to exploit this. You normally perform such settings in your system's environment, in platform-dependent ways, before you start Python. On macOS only, `urlopen` transparently and implicitly retrieves proxy URLs from your Internet configuration settings. Alternatively, you can pass as argument `proxies` a mapping whose keys are scheme names, with the corresponding values being proxy URLs. For example:

```
f = urllib.urlopen('http://python.org',
proxies={'http':'http://prox:999'})
```

`urlopen` does not support proxies that require authentication; for such advanced needs, use the richer library module `urllib2`, covered in "The urllib2 Module (v2)".

urlretrieve

`urlretrieve(urlstring,filename=None,reporthook=None, data=None)`

Similar to `urlopen(urlstring,data)`, but meant for downloading large files. Returns a pair `(f,m)`, where `f` is a string that specifies the path to a file on the local filesystem, `m` an instance of class `Message` of module `mimetools`, like the result of method `info` called on the result value of `urlopen`, covered above.

When `filename` is `None`, `urlretrieve` copies retrieved data to a temporary local file, and `f` is the path to the temporary local file. When `filename` is not `None`, `urlretrieve` copies retrieved data to the file named `filename`, and `f` is `filename`. When `reporthook` is not `None`, it must be a callable with three arguments, as in the function:

```
def reporthook(block_count, block_size, file_size):    print(
block_count)
```

`urlretrieve` calls `reporthook` zero or more times while retrieving data. At each call, it passes `block_count`, the number of blocks of data retrieved so far; `block_size`, the size in bytes of each block; and `file_size`, the total size of the file in bytes. `urlretrieve` passes `file_size` as `-1` when it cannot determine file size, which depends on the protocol involved and on how completely the server implements that protocol. The purpose of `reporthook` is to allow your program to give graphical or textual feedback to the user about the progress of the file-retrieval operation that `urlretrieve` performs.