19.12. base64 — RFC 3548: Base16, Base32, Base64 Data Encodings¶

This module provides data encoding and decoding as specified in **RFC 3548**. This standard defines the Base16, Base32, and Base64 algorithms for encoding and decoding arbitrary binary strings into text strings that can be safely sent by email, used as parts of URLs, or included as part of an HTTP POST request. The encoding algorithm is not the same as the **uuencode** program.

There are two interfaces provided by this module. The modern interface supports encoding and decoding string objects using all three alphabets. The legacy interface provides for encoding and decoding to and from file-like objects as well as strings, but only using the Base64 standard alphabet.

The modern interface, which was introduced in Python 2.4, provides:

`base64.b64encode`(*s*[, *altchars*])¶

Encode a string use Base64.

s is the string to encode. Optional *altchars* must be a string of at least length 2 (additional characters are ignored) which specifies an alternative alphabet for the `+` and `/` characters. This allows an application to e.g. generate URL or filesystem safe Base64 strings. The default is `None`, for which the standard Base64 alphabet is used.

The encoded string is returned.

`base64.b64decode`(*s*[, *altchars*])¶

Decode a Base64 encoded string.

s is the string to decode. Optional *altchars* must be a string of at least length 2 (additional characters are ignored) which specifies the alternative alphabet used instead of the `+` and `/` characters.

The decoded string is returned. A `TypeError` is raised if *s* were incorrectly padded or if there are non-alphabet characters present in the string.

`base64.standard_b64encode`(*s*)¶

Encode string *s* using the standard Base64 alphabet.

`base64.standard_b64decode`(*s*)¶

Decode string *s* using the standard Base64 alphabet.

`base64.urlsafe_b64encode`(*s*)¶

Encode string *s* using a URL-safe alphabet, which substitutes `-` instead of `+` and `_` instead of `/` in the standard Base64 alphabet. The result can still contain `=`.

`base64.urlsafe_b64decode`(*s*)¶

Decode string *s* using a URL-safe alphabet, which substitutes `-` instead of `+` and `_` instead of `/` in the standard Base64 alphabet.

`base64.b32encode`(*s*)¶

Encode a string using Base32. *s* is the string to encode. The encoded string is returned.

`base64.b32decode`(*s*[, *casefold*[, *map01*]])¶

Decode a Base32 encoded string.

s is the string to decode. Optional *casefold* is a flag specifying whether a lowercase alphabet is acceptable as input. For security purposes, the default is `False`.

RFC 3548 allows for optional mapping of the digit 0 (zero) to the letter O (oh), and for optional mapping of the digit 1 (one) to either the letter I (eye) or letter L (el). The optional argument *map01* when not `None`, specifies which letter the digit 1 should be mapped to (when *map01* is not `None`, the digit 0 is always mapped to the letter O). For security purposes the default is `None`, so that 0 and 1 are not allowed in the input.

The decoded string is returned. A `TypeError` is raised if *s* were incorrectly padded or if there are non-alphabet characters present in the string.

`base64.b16encode`(*s*)¶

Encode a string using Base16.

s is the string to encode. The encoded string is returned.

`base64.b16decode`(*s*[, *casefold*])¶

Decode a Base16 encoded string.

s is the string to decode. Optional *casefold* is a flag specifying whether a lowercase alphabet is acceptable as input. For security purposes, the default is `False`.

The decoded string is returned. A `TypeError` is raised if *s* were incorrectly padded or if there are non-alphabet characters present in the string.

The legacy interface:

`base64.decode`(*input*, *output*)¶

Decode the contents of the *input* file and write the resulting binary data to the *output* file. *input* and *output* must either be file objects or objects that mimic the file object interface. *input* will be read until `input.read()` returns an empty string.

`base64.decodestring`(*s*)¶

Decode the string *s*, which must contain one or more lines of base64 encoded data, and return a string containing the resulting binary data.

`base64.encode`(*input*, *output*)¶

Encode the contents of the *input* file and write the resulting base64 encoded data to the *output* file. *input* and *output* must either be file objects or objects that mimic the file object interface. *input* will be read until `input.read()` returns an empty string. `encode()` returns the encoded data plus a trailing newline character (`'\n'`).

`base64.encodestring`(*s*)¶

Encode the string *s*, which can contain arbitrary binary data, and return a string containing one or more lines of base64-encoded data. `encodestring()` returns a string containing one or more lines of base64-encoded data always including an extra trailing newline (`'\n'`).

An example usage of the module:

```
>>> import base64
>>> encoded = base64.b64encode('data to be encoded')
>>> encoded
'ZGF0YSB0byBiZSBlbmNvZGVk'
>>> data = base64.b64decode(encoded)
>>> data
'data to be encoded'
```

See also

Module `binascii`

Support module containing ASCII-to-binary and binary-to-ASCII conversions.

RFC 1521 - MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies

Section 5.2, “Base64 Content-Transfer-Encoding,” provides the definition of the base64 encoding.

Previous topic

19.11. `rfc822` — Parse RFC 2822 mail headers

Next topic

19.13. `binhex` — Encode and decode binhex4 files

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 19. Internet Data Handling »