`!binascii` --- Convert between binary and ASCII

binascii

pair: module; base64

The `!binascii` module contains a number of methods to convert between binary and various ASCII-encoded binary representations. Normally, you will not use these functions directly but use wrapper modules like `base64` instead. The `!binascii` module contains low-level functions written in C for greater speed that are used by the higher-level modules.

> [!NOTE] `a2b_*` functions accept Unicode strings containing only ASCII characters. Other functions only accept `bytes-like objects <bytes-like object>` (such as `bytes`, `bytearray` and other objects that support the buffer protocol).
>
> 3.3 ASCII-only unicode strings are now accepted by the `a2b_*` functions.

The `!binascii` module defines the following functions:

a2b_uu(string)

Convert a single line of uuencoded data back to binary and return the binary data. Lines normally contain 45 (binary) bytes, except for the last line. Line data may be followed by whitespace.

b2a_uu(data, *, backtick=False)

Convert binary data to a line of ASCII characters, the return value is the converted line, including a newline char. The length of *data* should be at most 45. If *backtick* is true, zeros are represented by `'`'` instead of spaces.

3.7 Added the *backtick* parameter.

a2b_base64(string, /, , *padded=True, alphabet=BASE64_ALPHABET, strict_mode=False, canonical=False) a2b_base64(string, /,*, ignorechars, padded=True, alphabet=BASE64_ALPHABET, strict_mode=True, canonical=False)

Convert a block of base64 data back to binary and return the binary data. More than one line may be passed at a time.

Optional *alphabet* must be a `bytes` object of length 64 which specifies an alternative alphabet.

If *padded* is true, the last group of 4 base 64 alphabet characters must be padded with the '=' character. If *padded* is false, padding is neither required nor recognized: the '=' character is not treated as padding but as a non-alphabet character, which means it is silently discarded when *strict_mode* is false, or causes an `~binascii.Error` when *strict_mode* is true unless b'=' is included in *ignorechars*.

If *ignorechars* is specified, it should be a `bytes-like object` containing characters to ignore from the input when *strict_mode* is true. If *ignorechars* contains the pad character `'='`, the pad characters presented before the end of the encoded data and the excess pad characters will be ignored. The default value of *strict_mode* is `True` if *ignorechars* is specified, `False` otherwise.

If *strict_mode* is true, only valid base64 data will be converted. Invalid base64 data will raise `binascii.Error`.

Valid base64:

- Conforms to `4648`.
- Contains only characters from the base64 alphabet.
- Contains no excess data after padding (including excess padding, newlines, etc.).
- Does not start with a padding.

If *canonical* is true, non-zero padding bits in the last group are rejected with `binascii.Error`, enforcing canonical encoding as defined in `4648` section 3.5. This check is independent of *strict_mode*.

3.11 Added the *strict_mode* parameter.

3.15 Added the *alphabet*, *canonical*, *ignorechars*, and *padded* parameters.

b2a_base64(data, *, padded=True, alphabet=BASE64_ALPHABET, wrapcol=0, newline=True)

Convert binary data to a line(s) of ASCII characters in base64 coding, as specified in `4648`.

If *padded* is true (default), pad the encoded data with the '=' character to a size multiple of 4. If *padded* is false, do not add the pad characters.

If *wrapcol* is non-zero, insert a newline (`b'\n'`) character after at most every *wrapcol* characters. If *wrapcol* is zero (default), do not insert any newlines.

If *newline* is true (default), a newline character will be added at the end of the output.

3.6 Added the *newline* parameter.

3.15 Added the *alphabet*, *padded* and *wrapcol* parameters.

a2b_ascii85(string, /, *, foldspaces=False, adobe=False, ignorechars=b'', canonical=False)

Convert Ascii85 data back to binary and return the binary data.

Valid Ascii85 data contains characters from the Ascii85 alphabet in groups of five (except for the final group, which may have from two to five characters). Each group encodes 32 bits of binary data in the range from `0` to `2 ** 32 - 1`, inclusive. The special character `z` is accepted as a short form of the group `!!!!!`, which encodes four consecutive null bytes. A single-character final group is always rejected as an encoding violation.

foldspaces is a flag that specifies whether the 'y' short sequence should be accepted as shorthand for 4 consecutive spaces (ASCII 0x20). This feature is not supported by the "standard" Ascii85 encoding.

adobe controls whether the encoded byte sequence is framed with `<~` and `~>`, as in a PostScript base-85 string literal. If *adobe* is true, a leading `<~` is optionally accepted, while a trailing `~>` is *required*, and `binascii.Error` is raised if it is not found.

ignorechars should be a `bytes-like object` containing characters to ignore from the input. This should only contain whitespace characters.

If *canonical* is true, non-canonical encodings are rejected with `binascii.Error`. Here "canonical" means the encoding that `b2a_ascii85` would produce: the `z` abbreviation must be used for all-zero groups (rather than `!!!!!`), and partial final groups must use the same padding digits as the encoder.

Invalid Ascii85 data will raise `binascii.Error`.

3.15

b2a_ascii85(data, /, *, foldspaces=False, wrapcol=0, pad=False, adobe=False)

Convert binary data to a formatted sequence of ASCII characters in Ascii85 coding. The return value is the converted data.

foldspaces is an optional flag that uses the special short sequence 'y' instead of 4 consecutive spaces (ASCII 0x20) as supported by 'btoa'. This feature is not supported by the "standard" Ascii85 encoding.

If *wrapcol* is non-zero, insert a newline (`b'\n'`) character after at most every *wrapcol* characters. If *wrapcol* is zero (default), do not insert any newlines.

If *pad* is true, the zero-padding applied to the end of the input is fully retained in the output encoding, as done by `btoa`, producing an exact multiple of 5 bytes of output. This is not part of the standard encoding used in PDF, as it does not preserve the length of the data.

adobe controls whether the encoded byte sequence is framed with `<~` and `~>`, as in a PostScript base-85 string literal. Note that while ASCII85Decode streams in PDF documents *must* be terminated with `~>`, they *must not* use a leading `<~`.

3.15

a2b_base85(string, /, *, alphabet=BASE85_ALPHABET, ignorechars=b'', canonical=False)

Convert Base85 data back to binary and return the binary data. More than one line may be passed at a time.

Valid Base85 data contains characters from the Base85 alphabet in groups of five (except for the final group, which may have from two to five characters). Each group encodes 32 bits of binary data in the range from `0` to `2 ** 32 - 1`, inclusive. A single-character final group is always rejected as an encoding violation.

Optional *alphabet* must be a `bytes` object of length 85 which specifies an alternative alphabet.

ignorechars should be a `bytes-like object` containing characters to ignore from the input.

If *canonical* is true, non-canonical encodings are rejected with `binascii.Error`. Here "canonical" means the encoding that `b2a_base85` would produce: partial final groups must use the same padding digits as the encoder.

Invalid Base85 data will raise `binascii.Error`.

3.15

b2a_base85(data, /, *, alphabet=BASE85_ALPHABET, wrapcol=0, pad=False)

Convert binary data to a line of ASCII characters in Base85 coding. The return value is the converted line.

Optional *alphabet* must be a `bytes-like object` of length 85 which specifies an alternative alphabet.

If *wrapcol* is non-zero, insert a newline (`b'\n'`) character after at most every *wrapcol* characters. If *wrapcol* is zero (default), do not insert any newlines.

If *pad* is true, the zero-padding applied to the end of the input is retained in the output, which will always be a multiple of 5 bytes, and thus the length of the data may not be preserved on decoding.

3.15

a2b_base32(string, /, *, padded=True, alphabet=BASE32_ALPHABET, ignorechars=b'', canonical=False)

Convert base32 data back to binary and return the binary data.

Valid base32 data contains characters from the base32 alphabet specified in `4648` in groups of eight (if necessary, the final group is padded to eight characters with `=`). Each group encodes 40 bits of binary data in the range from `0` to `2 ** 40 - 1`, inclusive.

> [!NOTE] This function does not map lowercase characters (which are invalid in standard base32) to their uppercase counterparts, nor does it contextually map `0` to `O` and `1` to `I`/`L` as `4648` allows.

Optional *alphabet* must be a `bytes` object of length 32 which specifies an alternative alphabet.

If *padded* is true, the last group of 8 base 32 alphabet characters must be padded with the '=' character. If *padded* is false, the '=' character is treated as other non-alphabet characters (depending on the value of *ignorechars*).

ignorechars should be a `bytes-like object` containing characters to ignore from the input. If *ignorechars* contains the pad character `'='`, the pad characters presented before the end of the encoded data and the excess pad characters will be ignored.

If *canonical* is true, non-zero padding bits in the last group are rejected with `binascii.Error`, enforcing canonical encoding as defined in `4648` section 3.5.

Invalid base32 data will raise `binascii.Error`.

3.15

b2a_base32(data, /, *, padded=True, alphabet=BASE32_ALPHABET, wrapcol=0)

Convert binary data to a line of ASCII characters in base32 coding, as specified in `4648`. The return value is the converted line.

Optional *alphabet* must be a `bytes-like object` of length 32 which specifies an alternative alphabet.

If *padded* is true (default), pad the encoded data with the '=' character to a size multiple of 8. If *padded* is false, do not add the pad characters.

If *wrapcol* is non-zero, insert a newline (`b'\n'`) character after at most every *wrapcol* characters. If *wrapcol* is zero (default), do not insert any newlines.

3.15

a2b_qp(data, header=False, *, strip_ws=False)

Convert a block of quoted-printable data back to binary and return the binary data. More than one line may be passed at a time. If the optional argument *header* is present and true, underscores will be decoded as spaces. If the optional argument *strip_ws* is true, trailing whitespace is stripped from each line, as required by `2045`.

next Added the *strip_ws* parameter.

b2a_qp(data, quotetabs=False, istext=True, header=False)

Convert binary data to a line(s) of ASCII characters in quoted-printable encoding. The return value is the converted line(s). If the optional argument *quotetabs* is present and true, all tabs and spaces will be encoded. If the optional argument *istext* is present and true, newlines are not encoded but trailing whitespace will be encoded. If the optional argument *header* is present and true, spaces will be encoded as underscores per `1522`. If the optional argument *header* is present and false, newline characters will be encoded as well; otherwise linefeed conversion might corrupt the binary data stream.

crc_hqx(data, value)

Compute a 16-bit CRC value of *data*, starting with *value* as the initial CRC, and return the result. This uses the CRC-CCITT polynomial $x^{16} + x^{12} + x^5 + 1$, often represented as 0x1021. This CRC is used in the binhex4 format.

crc32(data[, value])

Compute CRC-32, the unsigned 32-bit checksum of *data*, starting with an initial CRC of *value*. The default initial CRC is zero. The algorithm is consistent with the ZIP file checksum. Since the algorithm is designed for use as a checksum algorithm, it is not suitable for use as a general hash algorithm. Use as follows:

```
print(binascii.crc32(b"hello world"))
# Or, in two pieces:
crc = binascii.crc32(b"hello")
crc = binascii.crc32(b" world", crc)
print('crc32 = {:#010x}'.format(crc))
```

3.0 The result is always unsigned.

b2a_hex(data[, sep[, bytes_per_sep=1]]) hexlify(data[, sep[, bytes_per_sep=1]])

Return the hexadecimal representation of the binary *data*. Every byte of *data* is converted into the corresponding 2-digit hex representation. The returned bytes object is therefore twice as long as the length of *data*.

Similar functionality (but returning a text string) is also conveniently accessible using the `bytes.hex` method.

If *sep* is specified, it must be a single character str or bytes object. It will be inserted in the output after every *bytes_per_sep* input bytes. Separator placement is counted from the right end of the output by default, if you wish to count from the left, supply a negative *bytes_per_sep* value.

> >>> import binascii >>> binascii.b2a_hex(b'xb9x01xef') b'b901ef' >>> binascii.hexlify(b'xb9x01xef', '-') b'b9-01-ef' >>> binascii.b2a_hex(b'xb9x01xef', b'_', 2) b'b9_01ef' >>> binascii.b2a_hex(b'xb9x01xef', b' ', -2) b'b901 ef'

3.8 The *sep* and *bytes_per_sep* parameters were added.

a2b_hex(hexstr, *, ignorechars=b'') *unhexlify(hexstr,*, ignorechars=b'')

Return the binary data represented by the hexadecimal string *hexstr*. This function is the inverse of `b2a_hex`. *hexstr* must contain an even number of hexadecimal digits (which can be upper or lower case), otherwise an `Error` exception is raised.

ignorechars should be a `bytes-like object` containing characters to ignore from the input.

Similar functionality (but more liberal towards whitespace) is also accessible using the `bytes.fromhex` class method.

3.15 Added the *ignorechars* parameter.

Error

Exception raised on errors. These are usually programming errors.

Incomplete

Exception raised on incomplete data. These are usually not programming errors, but may be handled by reading a little more data and trying again.

BASE64_ALPHABET

The Base 64 alphabet according to `4648`.

3.15

URLSAFE_BASE64_ALPHABET

The "URL and filename safe" Base 64 alphabet according to `4648`.

3.15

UU_ALPHABET

The uuencoding alphabet.

3.15

CRYPT_ALPHABET

The Base 64 alphabet used in the `crypt(3)` routine and in the GEDCOM format.

3.15

BINHEX_ALPHABET

The Base 64 alphabet used in BinHex 4 (HQX) within the classic Mac OS.

3.15

BASE85_ALPHABET

The Base85 alphabet.

3.15

ASCII85_ALPHABET

The Ascii85 alphabet.

3.15

Z85_ALPHABET

The Z85 alphabet.

3.15

BASE32_ALPHABET

The Base 32 alphabet according to `4648`.

3.15

BASE32HEX_ALPHABET

The "Extended Hex" Base 32 alphabet according to `4648`. Data encoded with this alphabet maintains its sort order during bitwise comparisons.

3.15

Module `base64`
Support for RFC compliant base64-style encoding in base 16, 32, 64, and 85.

Module `quopri`
Support for quoted-printable encoding used in MIME email messages.

□□□ #2
□ □□□ □□□ 2026-10-31 17:04:28 UTC
□ □□□ □□□ 2026-10-31 17:04:31 UTC