base64 — Encode Binary Data with ASCII

Purpose: The base64 module contains functions for translating binary data into a subset of ASCII suitable for transmission using plaintext protocols.

The base64, base32, base16, and base85 encodings convert 8 bit bytes to values that fit inside the ASCII range of printable characters, trading more bits to represent the data for compatibility with systems that only support ASCII data, such as SMTP. The *base* values correspond to the length of the alphabet used in each encoding. There are also URL-safe variations of the original encodings that use slightly different alphabets.

Base 64 Encoding

This is a basic example of encoding some text.

```
# base64_b64encode.py

import base64
import textwrap

# Load this source file and strip the header.
with open(__file__, 'r', encoding='utf-8') as input:
    raw = input.read()
    initial_data = raw.split('#end_pymotw_header')[1]

byte_string = initial_data.encode('utf-8')
encoded_data = base64.b64encode(byte_string)

num_initial = len(byte_string)

# There will never be more than 2 padding bytes.
padding = 3 - (num_initial % 3)

print('{} bytes before encoding'.format(num_initial))
print('Expect {} padding bytes'.format(padding))
print('{} bytes after encoding\n'.format(len(encoded_data)))
print(encoded_data)
```

The input must be a byte string, so the unicode string is first encoded to UTF-8. The output shows the 185 bytes of the UTF-8 source expand to 248 bytes after being encoded.

Note

There are no carriage returns in the encoded data produced by the library, but the output has been wrapped artificially to make it fit better on the page.

```
$ python3 base64_b64encode.py

184 bytes before encoding
Expect 2 padding bytes
248 bytes after encoding

b'CmltcG9ydCBiYXNlNjQKaW1wb3J0IHRleHR3cmFwCgojIExvYWQgdGhpcyBzb3
VyY2UgZmlsZSBhbmQgc3RyaXAgdGhlIGhlYWRlci4Kd2l0aCBvcGVuKF9fZmlsZV
9fLCAncicsIGVuY29kaW5nPSd1dGYtOCcpIGFzIGlucHV0OgogICAgcmF3ID0gaW
5wdXQucmVhZCgpCiAgICBpbml0aWFsX2RhdGEgPSByYXcuc3BsaXQoJyNlbmRfcH
ltb3R3X2hlYWRlcicpWzFdCgpieXRlX3N0cmluZyA='
```

Base 64 Decoding

`b64decode()` converts an encoded string back to the original form by taking four bytes and converting them to the original three, using a lookup table.

```
# base64_b64decode.py

import base64

encoded_data = b'VGhpcyBpcyB0aGUgZGF0YSwgaW4gdGhlIGNsZWFyLg=='
decoded_data = base64.b64decode(encoded_data)
print('Encoded :', encoded_data)
print('Decoded :', decoded_data)
```

The encoding process looks at each sequence of 24 bits in the input (three bytes) and encodes those same 24 bits spread over four bytes in the output. The equal signs at the end of the output are padding inserted because the number of bits in the original string was not evenly divisible by 24, in this example.

```
$ python3 base64_b64decode.py

Encoded : b'VGhpcyBpcyB0aGUgZGF0YSwgaW4gdGhlIGNsZWFyLg=='
Decoded : b'This is the data, in the clear.'
```

The value returned from `b64decode()` is a byte string. If the contents are known to be text, the byte string can be converted to a unicode object. However, the point of using base 64 encoding is to be able to transmit binary data, and so it is not always safe to assume that the decoded value is text.

URL-safe Variations

Because the default base64 alphabet may use `+` and `/`, and those two characters are used in URLs, it is often necessary to use an alternate encoding with substitutes for those characters.

```
# base64_urlsafe.py
```

```
import base64

encodes_with_pluses = b'\xfb\xef'
encodes_with_slashes = b'\xff\xff'

for original in [encodes_with_pluses, encodes_with_slashes]:
    print('Original         :', repr(original))
    print('Standard encoding:',
          base64.standard_b64encode(original))
    print('URL-safe encoding:',
          base64.urlsafe_b64encode(original))
    print()
```

The `+` is replaced with a `-`, and `/` is replaced with underscore (`_`). Otherwise, the alphabet is the same.

```
$ python3 base64_urlsafe.py

Original         : b'\xfb\xef'
Standard encoding: b'++8='
URL-safe encoding: b'--8='

Original         : b'\xff\xff'
Standard encoding: b'//8='
URL-safe encoding: b'__8='
```

Other Encodings

Besides Base64, the module provides functions for working with Base85, Base32, and Base16 (hex) encoded data.

```
# base64_base32.py

import base64

original_data = b'This is the data, in the clear.'
print('Original:', original_data)

encoded_data = base64.b32encode(original_data)
print('Encoded :', encoded_data)
```

```
print('Encoded :', encoded_data)

decoded_data = base64.b32decode(encoded_data)
print('Decoded :', decoded_data)
```

The Base32 alphabet includes the 26 uppercase letters from the ASCII set and the digits 2 through 7.

```
$ python3 base64_base32.py

Original: b'This is the data, in the clear.'
Encoded : b'KRUGS4ZANFZSA5DIMUQGIYLUMEWCA2LOEB2GQZJAMNWGKYLSFY==
====='
Decoded : b'This is the data, in the clear.'
```

The Base16 functions work with the hexadecimal alphabet.

```
# base64_base16.py

import base64

original_data = b'This is the data, in the clear.'
print('Original:', original_data)

encoded_data = base64.b16encode(original_data)
print('Encoded :', encoded_data)

decoded_data = base64.b16decode(encoded_data)
print('Decoded :', decoded_data)
```

Each time the number of encoding bits goes down, the output in the encoded format takes up more space.

```
$ python3 base64_base16.py

Original: b'This is the data, in the clear.'
Encoded : b'546869732069732074686520646174612C20696E207468652063
6C6561722E'
Decoded : b'This is the data, in the clear.'
```

The Base85 functions use an expanded alphabet that is more space-efficient than base 64.

```
# base64_base85.py

import base64

original_data = b'This is the data, in the clear.'
print('Original    : {} bytes {!r}'.format(
    len(original_data), original_data))

b64_data = base64.b64encode(original_data)
print('b64 Encoded : {} bytes {!r}'.format(
    len(b64_data), b64_data))

b85_data = base64.b85encode(original_data)
print('b85 Encoded : {} bytes {!r}'.format(
    len(b85_data), b85_data))

a85_data = base64.a85encode(original_data)
print('a85 Encoded : {} bytes {!r}'.format(
    len(a85_data), a85_data))
```

There are several Base85 encodings and different variations are used in Mercurial, git, and the PDF file format. Python includes two implementations, `b85encode()` implements the version used in Git and Mercurial while `a85encode()` implements the Ascii85 variant used by PDF files.

```
$ python3 base64_base85.py

Original    : 31 bytes b'This is the data, in the clear.'
b64 Encoded : 44 bytes b'VGhpcyBpcyB0aGUgZGF0YSwgaW4gdGhlIGNsZWF
yLg=='
```

```
b85 Encoded : 39 bytes b'RA^~)AZc?TbZBKDWMOn+EFfuaAarPDAY*K0VR9}
'
a85 Encoded : 39 bytes b'<+oue+DGm>FD,5.A79Rg/0JYE+EV:.+Cf5!@<*t
'
```

See also

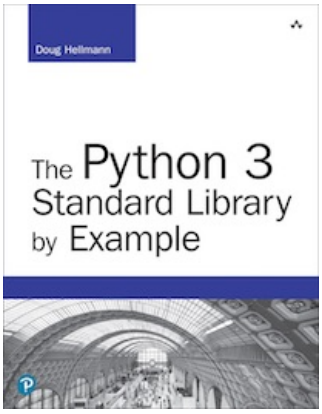
- Standard library documentation for base64
- **RFC 3548** – The Base16, Base32, and Base64 Data Encodings
- **RFC 1924** – A Compact Representation of IPv6 Addresses (suggests base-85 encoding for IPv6 network addresses)
- Ascii85
- Python 2 to 3 porting notes for base64

urllib.robotparser — Internet Spider Access Control
http.server — Base Classes for Implementing Web Servers

Quick Links
Base 64 Encoding
Base 64 Decoding
URL-safe Variations
Other Encodings

This page was last updated 2018-03-18.

Navigation
urllib.robotparser — Internet Spider Access Control
http.server — Base Classes for Implementing Web Servers



Get the book

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