[smtplib](https://docs.python.org/3/library/smtplib.html" \l "module-smtplib" \o "smtplib: SMTP protocol client (requires sockets).) — SMTP protocol client

The [smtplib](https://docs.python.org/3/library/smtplib.html" \l "module-smtplib" \o "smtplib: SMTP protocol client (requires sockets).) module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. For details of SMTP and ESMTP operation, consult [**RFC 821**](https://tools.ietf.org/html/rfc821.html) (Simple Mail Transfer Protocol) and [**RFC 1869**](https://tools.ietf.org/html/rfc1869.html) (SMTP Service Extensions).

An [SMTP](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP) instance encapsulates an SMTP connection. It has methods that support a full repertoire of SMTP and ESMTP operations. If the optional host and port parameters are given, the SMTP [connect()](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP.connect) method is called with those parameters during initialization. If specified, local\_hostname is used as the FQDN of the local host in the HELO/EHLO command. Otherwise, the local hostname is found using [socket.getfqdn()](https://docs.python.org/3/library/socket.html" \l "socket.getfqdn" \o "socket.getfqdn). If the [connect()](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP.connect) call returns anything other than a success code, an [SMTPConnectError](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTPConnectError" \o "smtplib.SMTPConnectError) is raised. The optional timeout parameter specifies a timeout in seconds for blocking operations like the connection attempt

For normal use, you should only require the initialization/connect, [sendmail()](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTP.sendmail" \o "smtplib.SMTP.sendmail), and [SMTP.quit()](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTP.quit" \o "smtplib.SMTP.quit) methods. An example is included below.

The [SMTP](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP) class supports the [with](https://docs.python.org/3/reference/compound_stmts.html#with) statement. When used like this, the SMTP QUIT command is issued automatically when the with statement exits. E.g.:

>>>

**>>> from** **smtplib** **import** SMTP

**>>> with** SMTP("domain.org") **as** smtp:

**...**  smtp.noop()

**...**

(250, b'Ok')

>>>

## **SMTP Objects**

An [SMTP](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP) instance has the following methods:

SMTP.**set\_debuglevel**(level)

Set the debug output level. A value of 1 or True for level results in debug messages for connection and for all messages sent to and received from the server. A value of 2 for level results in these messages being timestamped.

*Changed in version 3.5:*Added debuglevel 2.

SMTP.**docmd**(cmd, args='')

Send a command cmd to the server. The optional argument args is simply concatenated to the command, separated by a space.

This returns a 2-tuple composed of a numeric response code and the actual response line (multiline responses are joined into one long line.)

In normal operation it should not be necessary to call this method explicitly. It is used to implement other methods and may be useful for testing private extensions.

If the connection to the server is lost while waiting for the reply, [SMTPServerDisconnected](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTPServerDisconnected" \o "smtplib.SMTPServerDisconnected) will be raised.

SMTP.**connect**(host='localhost', port=0)

Connect to a host on a given port. The defaults are to connect to the local host at the standard SMTP port (25). If the hostname ends with a colon (':') followed by a number, that suffix will be stripped off and the number interpreted as the port number to use. This method is automatically invoked by the constructor if a host is specified during instantiation. Returns a 2-tuple of the response code and message sent by the server in its connection response.

Raises an [auditing event](https://docs.python.org/3/library/sys.html#auditing) smtplib.connect with arguments self, host, port.

SMTP.**helo**(name='')

Identify yourself to the SMTP server using HELO. The hostname argument defaults to the fully qualified domain name of the local host. The message returned by the server is stored as the helo\_resp attribute of the object.

In normal operation it should not be necessary to call this method explicitly. It will be implicitly called by the [sendmail()](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTP.sendmail" \o "smtplib.SMTP.sendmail) when necessary.

SMTP.**ehlo**(name='')

Identify yourself to an ESMTP server using EHLO. The hostname argument defaults to the fully qualified domain name of the local host. Examine the response for ESMTP option and store them for use by [has\_extn()](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTP.has_extn" \o "smtplib.SMTP.has_extn). Also sets several informational attributes: the message returned by the server is stored as the ehlo\_resp attribute, does\_esmtp is set to true or false depending on whether the server supports ESMTP, and esmtp\_features will be a dictionary containing the names of the SMTP service extensions this server supports, and their parameters (if any).

Unless you wish to use [has\_extn()](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTP.has_extn" \o "smtplib.SMTP.has_extn) before sending mail, it should not be necessary to call this method explicitly. It will be implicitly called by [sendmail()](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTP.sendmail" \o "smtplib.SMTP.sendmail) when necessary.

SMTP.**login**(*user*, *password*, *\**, *initial\_response\_ok=True*)

Log in on an SMTP server that requires authentication. The arguments are the username and the password to authenticate with. If there has been no previous EHLO or HELO command this session, this method tries ESMTP EHLO first. This method will return normally if the authentication was successful, or may raise the following exceptions:

SMTP.**starttls**(*keyfile=None*, *certfile=None*, *context=None*)

Put the SMTP connection in TLS (Transport Layer Security) mode. All SMTP commands that follow will be encrypted. You should then call [ehlo()](https://docs.python.org/3/library/smtplib.html" \l "smtplib.SMTP.ehlo" \o "smtplib.SMTP.ehlo) again.

If *keyfile* and *certfile* are provided, they are used to create an [ssl.SSLContext](https://docs.python.org/3/library/ssl.html" \l "ssl.SSLContext" \o "ssl.SSLContext).

Optional *context* parameter is an [ssl.SSLContext](https://docs.python.org/3/library/ssl.html" \l "ssl.SSLContext" \o "ssl.SSLContext) object; This is an alternative to using a keyfile and a certfile and if specified both *keyfile* and *certfile* should be None.

If there has been no previous EHLO or HELO command this session, this method tries ESMTP EHLO first.

Extended Simple Mail Transfer Protocol (**ESMTP**) is a protocol for sending and receiving emails over IP networks. **ESMTP** was created to allow additional functionality that Simple Mail Transfer Protocol (SMTP) can't support, such as Transport Layer Security (TLS).

*class*smtplib.**SMTP\_SSL**(*host=''*, *port=0*, *local\_hostname=None*, *keyfile=None*, *certfile=None*, [*timeout*, ]*context=None*, *source\_address=None*)

An [SMTP\_SSL](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP_SSL) instance behaves exactly the same as instances of [SMTP](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP). [SMTP\_SSL](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP_SSL) should be used for situations where SSL is required from the beginning of the connection and using starttls() is not appropriate. If *host* is not specified, the local host is used. If *port* is zero, the standard SMTP-over-SSL port (465) is used. The optional arguments *local\_hostname*, *timeout* and *source\_address* have the same meaning as they do in the [SMTP](https://docs.python.org/3/library/smtplib.html#smtplib.SMTP) class. *context*, also optional, can contain a [SSLContext](https://docs.python.org/3/library/ssl.html" \l "ssl.SSLContext" \o "ssl.SSLContext) and allows configuring various aspects of the secure connection. Please read [Security considerations](https://docs.python.org/3/library/ssl.html#ssl-security) for best practices.

*keyfile* and *certfile* are a legacy alternative to *context*, and can point to a PEM formatted private key and certificate chain file for the SSL connection.

# What is SSL?

Secure Sockets Layer (SSL) was the most widely deployed cryptographic protocol to provide security over internet communications before it was [preceded by TLS](https://www.globalsign.com/en/blog/ssl-vs-tls-difference/) (Transport Layer Security) in 1999. Despite the deprecation of the SSL protocol and the adoption of TLS in its place, most people still refer to this type of technology as ‘SSL’.

SSL provides a secure channel between two machines or devices operating over the internet or an internal network. One common example is when SSL is used to secure communication between a web browser and a web server. This turns a website's address from HTTP to HTTPS, the ‘S’ standing for ‘secure’.

HTTP is insecure and is subject to eavesdropping attacks because the data being transferred from the web browser to the web server or between other endpoints, is transmitted in plaintext. This means attackers can intercept and view sensitive data, such as credit card details and account logins. When data is sent or posted through a browser using HTTPS, SSL ensures that such information is encrypted and secure from interception.

# [email.message](https://docs.python.org/3/library/email.message.html#module-email.message): Representing an email message

**Source code:** [Lib/email/message.py](https://github.com/python/cpython/tree/3.8/Lib/email/message.py)

*New in version 3.6:*[1](https://docs.python.org/3/library/email.message.html#id2)

The central class in the [email](https://docs.python.org/3/library/email.html#module-email) package is the [EmailMessage](https://docs.python.org/3/library/email.message.html" \l "email.message.EmailMessage" \o "email.message.EmailMessage) class, imported from the [email.message](https://docs.python.org/3/library/email.message.html" \l "module-email.message" \o "email.message: The base class representing email messages.) module. It is the base class for the [email](https://docs.python.org/3/library/email.html#module-email) object model. [EmailMessage](https://docs.python.org/3/library/email.message.html" \l "email.message.EmailMessage" \o "email.message.EmailMessage) provides the core functionality for setting and querying header fields, for accessing message bodies, and for creating or modifying structured messages.

An email message consists of headers and a payload (which is also referred to as the content). Headers are [RFC 5322](https://tools.ietf.org/html/rfc5322.html) or [RFC 6532](https://tools.ietf.org/html/rfc6532.html) style field names and values, where the field name and value are separated by a colon. The colon is not part of either the field name or the field value. The payload may be a simple text message, or a binary object, or a structured sequence of sub-messages each with their own set of headers and their own payload. The latter type of payload is indicated by the message having a MIME type such as multipart/\* or message/rfc822.

**add\_attachment**(\*args, content\_manager=None, \*\*kw)

If the message is a multipart/mixed, create a new message object, pass all of the arguments to its [set\_content()](https://docs.python.org/3/library/email.message.html" \l "email.message.EmailMessage.set_content" \o "email.message.EmailMessage.set_content) method, and [attach()](https://docs.python.org/3/library/email.compat32-message.html#email.message.Message.attach) it to the multipart. If the message is a non-multipart, multipart/related, or multipart/alternative, call [make\_mixed()](https://docs.python.org/3/library/email.message.html" \l "email.message.EmailMessage.make_mixed" \o "email.message.EmailMessage.make_mixed) and then proceed as above. If content\_manager is not specified, use the content\_manager specified by the current [policy](https://docs.python.org/3/library/email.policy.html#module-email.policy). If the added part has no Content-Disposition header, add one with the value attachment. This method can be used both for explicit attachments (Content-Disposition: attachment) and inline attachments (Content-Disposition: inline), by passing appropriate options to the content\_manager.

# [imghdr](https://docs.python.org/2/library/imghdr.html#module-imghdr) — Determine the type of an image

**Source code:** [Lib/imghdr.py](https://github.com/python/cpython/tree/2.7/Lib/imghdr.py)

The **[imghdr](https://docs.python.org/2/library/imghdr.html" \l "module-imghdr" \o "imghdr: Determine the type of image contained in a file or byte stream.)** module determines the type of image contained in a file or byte stream.

The **[imghdr](https://docs.python.org/2/library/imghdr.html" \l "module-imghdr" \o "imghdr: Determine the type of image contained in a file or byte stream.)** module defines the following function:

imghdr.**what**(filename[, h])

Tests the image data contained in the file named by filename, and returns a string describing the image type. If optional h is provided, the filename is ignored and h is assumed to contain the byte stream to test.

The following image types are recognized, as listed below with the return value from [**what()**](https://docs.python.org/2/library/imghdr.html#imghdr.what):

| **Value** | **Image format** |
| --- | --- |
| 'rgb' | SGI ImgLib Files |
| 'gif' | GIF 87a and 89a Files |
| 'pbm' | Portable Bitmap Files |
| 'pgm' | Portable Graymap Files |
| 'ppm' | Portable Pixmap Files |
| 'tiff' | TIFF Files |
| 'rast' | Sun Raster Files |
| 'xbm' | X Bitmap Files |
| 'jpeg' | JPEG data in JFIF or Exif formats |
| 'bmp' | BMP files |
| 'png' | Portable Network Graphics |

## **File Handling**

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

## **Syntax**

To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt")

Open the file "demofile2.txt" and append content to the file:

f = open("demofile2.txt", "a")  
f.write("Now the file has more content!")  
f.close()  
  
#open and read the file after the appending:  
f = open("demofile2.txt", "r")  
print(f.read())