Django uses request and response objects to pass state through the system.

When a page is requested, Django creates an **[HttpRequest](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** object that contains metadata about the request. Then Django loads the appropriate view, passing the **[HttpRequest](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** as the first argument to the view function. Each view is responsible for returning an **[HttpResponse](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** object.

This document explains the APIs for **[HttpRequest](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** and **[HttpResponse](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** objects, which are defined in the **[django.http](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "module-django.http" \o "django.http: Classes dealing with HTTP requests and responses.)** module.

**HttpRequest** objects[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#httprequest-objects)

***class*HttpRequest[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest" \o "Permalink to this definition)**

**Attributes**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#attributes)

All attributes should be considered read-only, unless stated otherwise.

**HttpRequest.scheme[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.scheme" \o "Permalink to this definition)**

A string representing the scheme of the request (**http** or **https** usually).

**HttpRequest.body[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.body" \o "Permalink to this definition)**

The raw HTTP request body as a bytestring. This is useful for processing data in different ways than conventional HTML forms: binary images, XML payload etc. For processing conventional form data, use **[HttpRequest.POST](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.POST" \o "django.http.HttpRequest.POST)**.

You can also read from an **HttpRequest** using a file-like interface with **[HttpRequest.read()](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.read" \o "django.http.HttpRequest.read)** or **[HttpRequest.readline()](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.readline" \o "django.http.HttpRequest.readline)**. Accessing the **body** attribute *after* reading the request with either of these I/O stream methods will produce a **RawPostDataException**.

**HttpRequest.path[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.path" \o "Permalink to this definition)**

A string representing the full path to the requested page, not including the scheme, domain, or query string.

Example: **"/music/bands/the\_beatles/"**

**HttpRequest.path\_info[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.path_info" \o "Permalink to this definition)**

Under some web server configurations, the portion of the URL after the host name is split up into a script prefix portion and a path info portion. The **path\_info** attribute always contains the path info portion of the path, no matter what web server is being used. Using this instead of [**path**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.path) can make your code easier to move between test and deployment servers.

For example, if the **WSGIScriptAlias** for your application is set to **"/minfo"**, then **path** might be **"/minfo/music/bands/the\_beatles/"** and **path\_info** would be **"/music/bands/the\_beatles/"**.

**HttpRequest.method[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.method" \o "Permalink to this definition)**

A string representing the HTTP method used in the request. This is guaranteed to be uppercase. For example:

**if** request.method == 'GET':

do\_something()

**elif** request.method == 'POST':

do\_something\_else()

**HttpRequest.encoding[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.encoding" \o "Permalink to this definition)**

A string representing the current encoding used to decode form submission data (or **None**, which means the [**DEFAULT\_CHARSET**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-DEFAULT_CHARSET) setting is used). You can write to this attribute to change the encoding used when accessing the form data. Any subsequent attribute accesses (such as reading from [**GET**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.GET) or [**POST**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.POST)) will use the new **encoding** value. Useful if you know the form data is not in the [**DEFAULT\_CHARSET**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-DEFAULT_CHARSET) encoding.

**HttpRequest.content\_type[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.content_type" \o "Permalink to this definition)**

A string representing the MIME type of the request, parsed from the **CONTENT\_TYPE** header.

**HttpRequest.content\_params[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.content_params" \o "Permalink to this definition)**

A dictionary of key/value parameters included in the **CONTENT\_TYPE** header.

**HttpRequest.GET[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.GET" \o "Permalink to this definition)**

A dictionary-like object containing all given HTTP GET parameters. See the **[QueryDict](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.QueryDict" \o "django.http.QueryDict)** documentation below.

**HttpRequest.POST[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.POST" \o "Permalink to this definition)**

A dictionary-like object containing all given HTTP POST parameters, providing that the request contains form data. See the **[QueryDict](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.QueryDict" \o "django.http.QueryDict)** documentation below. If you need to access raw or non-form data posted in the request, access this through the **[HttpRequest.body](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.body" \o "django.http.HttpRequest.body)** attribute instead.

It’s possible that a request can come in via POST with an empty **POST** dictionary – if, say, a form is requested via the POST HTTP method but does not include form data. Therefore, you shouldn’t use **if request.POST** to check for use of the POST method; instead, use **if request.method == "POST"** (see **[HttpRequest.method](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.method" \o "django.http.HttpRequest.method)**).

**POST** does *not* include file-upload information. See [**FILES**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.FILES).

**HttpRequest.COOKIES[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.COOKIES" \o "Permalink to this definition)**

A dictionary containing all cookies. Keys and values are strings.

**HttpRequest.FILES[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.FILES" \o "Permalink to this definition)**

A dictionary-like object containing all uploaded files. Each key in **FILES** is the **name** from the **<input type="file" name="">**. Each value in **FILES** is an **[UploadedFile](https://docs.djangoproject.com/en/4.1/ref/files/uploads/" \l "django.core.files.uploadedfile.UploadedFile" \o "django.core.files.uploadedfile.UploadedFile)**.

See [Managing files](https://docs.djangoproject.com/en/4.1/topics/files/) for more information.

**FILES** will only contain data if the request method was POST and the **<form>** that posted to the request had **enctype="multipart/form-data"**. Otherwise, **FILES** will be a blank dictionary-like object.

**HttpRequest.META[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.META" \o "Permalink to this definition)**

A dictionary containing all available HTTP headers. Available headers depend on the client and server, but here are some examples:

* **CONTENT\_LENGTH** – The length of the request body (as a string).
* **CONTENT\_TYPE** – The MIME type of the request body.
* **HTTP\_ACCEPT** – Acceptable content types for the response.
* **HTTP\_ACCEPT\_ENCODING** – Acceptable encodings for the response.
* **HTTP\_ACCEPT\_LANGUAGE** – Acceptable languages for the response.
* **HTTP\_HOST** – The HTTP Host header sent by the client.
* **HTTP\_REFERER** – The referring page, if any.
* **HTTP\_USER\_AGENT** – The client’s user-agent string.
* **QUERY\_STRING** – The query string, as a single (unparsed) string.
* **REMOTE\_ADDR** – The IP address of the client.
* **REMOTE\_HOST** – The hostname of the client.
* **REMOTE\_USER** – The user authenticated by the web server, if any.
* **REQUEST\_METHOD** – A string such as **"GET"** or **"POST"**.
* **SERVER\_NAME** – The hostname of the server.
* **SERVER\_PORT** – The port of the server (as a string).

With the exception of **CONTENT\_LENGTH** and **CONTENT\_TYPE**, as given above, any HTTP headers in the request are converted to **META** keys by converting all characters to uppercase, replacing any hyphens with underscores and adding an **HTTP\_** prefix to the name. So, for example, a header called **X-Bender** would be mapped to the **META** key **HTTP\_X\_BENDER**.

Note that **[runserver](https://docs.djangoproject.com/en/4.1/ref/django-admin/" \l "django-admin-runserver)** strips all headers with underscores in the name, so you won’t see them in **META**. This prevents header-spoofing based on ambiguity between underscores and dashes both being normalizing to underscores in WSGI environment variables. It matches the behavior of web servers like Nginx and Apache 2.4+.

[**HttpRequest.headers**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.headers) is a simpler way to access all HTTP-prefixed headers, plus **CONTENT\_LENGTH** and **CONTENT\_TYPE**.

**HttpRequest.headers[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.headers" \o "Permalink to this definition)**

A case insensitive, dict-like object that provides access to all HTTP-prefixed headers (plus **Content-Length** and **Content-Type**) from the request.

The name of each header is stylized with title-casing (e.g. **User-Agent**) when it’s displayed. You can access headers case-insensitively:

**>>>** request.headers

{'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_6', ...}

**>>>** 'User-Agent' **in** request.headers

True

**>>>** 'user-agent' **in** request.headers

True

**>>>** request.headers['User-Agent']

Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_6)

**>>>** request.headers['user-agent']

Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_6)

**>>>** request.headers.get('User-Agent')

Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_6)

**>>>** request.headers.get('user-agent')

Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_6)

For use in, for example, Django templates, headers can also be looked up using underscores in place of hyphens:

{{ request.headers.user\_agent }}

**HttpRequest.resolver\_match[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.resolver_match" \o "Permalink to this definition)**

An instance of **[ResolverMatch](https://docs.djangoproject.com/en/4.1/ref/urlresolvers/" \l "django.urls.ResolverMatch" \o "django.urls.ResolverMatch)** representing the resolved URL. This attribute is only set after URL resolving took place, which means it’s available in all views but not in middleware which are executed before URL resolving takes place (you can use it in **[process\_view()](https://docs.djangoproject.com/en/4.1/topics/http/middleware/" \l "process_view" \o "process_view)** though).

**Attributes set by application code**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#attributes-set-by-application-code)

Django doesn’t set these attributes itself but makes use of them if set by your application.

**HttpRequest.current\_app[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.current_app" \o "Permalink to this definition)**

The **[url](https://docs.djangoproject.com/en/4.1/ref/templates/builtins/" \l "std-templatetag-url)** template tag will use its value as the **current\_app** argument to [**reverse()**](https://docs.djangoproject.com/en/4.1/ref/urlresolvers/#django.urls.reverse).

**HttpRequest.urlconf[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.urlconf" \o "Permalink to this definition)**

This will be used as the root URLconf for the current request, overriding the [**ROOT\_URLCONF**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-ROOT_URLCONF) setting. See [How Django processes a request](https://docs.djangoproject.com/en/4.1/topics/http/urls/#how-django-processes-a-request) for details.

**urlconf** can be set to **None** to revert any changes made by previous middleware and return to using the [**ROOT\_URLCONF**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-ROOT_URLCONF).

**HttpRequest.exception\_reporter\_filter[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.exception_reporter_filter" \o "Permalink to this definition)**

This will be used instead of [**DEFAULT\_EXCEPTION\_REPORTER\_FILTER**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-DEFAULT_EXCEPTION_REPORTER_FILTER) for the current request. See [Custom error reports](https://docs.djangoproject.com/en/4.1/howto/error-reporting/#custom-error-reports) for details.

**HttpRequest.exception\_reporter\_class[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.exception_reporter_class" \o "Permalink to this definition)**

This will be used instead of [**DEFAULT\_EXCEPTION\_REPORTER**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-DEFAULT_EXCEPTION_REPORTER) for the current request. See [Custom error reports](https://docs.djangoproject.com/en/4.1/howto/error-reporting/#custom-error-reports) for details.

**Attributes set by middleware**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#attributes-set-by-middleware)

Some of the middleware included in Django’s contrib apps set attributes on the request. If you don’t see the attribute on a request, be sure the appropriate middleware class is listed in [**MIDDLEWARE**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-MIDDLEWARE).

**HttpRequest.session[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.session" \o "Permalink to this definition)**

From the **[SessionMiddleware](https://docs.djangoproject.com/en/4.1/ref/middleware/" \l "django.contrib.sessions.middleware.SessionMiddleware" \o "django.contrib.sessions.middleware.SessionMiddleware)**: A readable and writable, dictionary-like object that represents the current session.

**HttpRequest.site[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.site" \o "Permalink to this definition)**

From the **[CurrentSiteMiddleware](https://docs.djangoproject.com/en/4.1/ref/middleware/" \l "django.contrib.sites.middleware.CurrentSiteMiddleware" \o "django.contrib.sites.middleware.CurrentSiteMiddleware)**: An instance of [**Site**](https://docs.djangoproject.com/en/4.1/ref/contrib/sites/#django.contrib.sites.models.Site) or **[RequestSite](https://docs.djangoproject.com/en/4.1/ref/contrib/sites/" \l "django.contrib.sites.requests.RequestSite" \o "django.contrib.sites.requests.RequestSite)** as returned by **[get\_current\_site()](https://docs.djangoproject.com/en/4.1/ref/contrib/sites/" \l "django.contrib.sites.shortcuts.get_current_site" \o "django.contrib.sites.shortcuts.get_current_site)** representing the current site.

**HttpRequest.user[¶](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.user" \o "Permalink to this definition)**

From the **[AuthenticationMiddleware](https://docs.djangoproject.com/en/4.1/ref/middleware/" \l "django.contrib.auth.middleware.AuthenticationMiddleware" \o "django.contrib.auth.middleware.AuthenticationMiddleware)**: An instance of [**AUTH\_USER\_MODEL**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-AUTH_USER_MODEL) representing the currently logged-in user. If the user isn’t currently logged in, **user** will be set to an instance of **[AnonymousUser](https://docs.djangoproject.com/en/4.1/ref/contrib/auth/" \l "django.contrib.auth.models.AnonymousUser" \o "django.contrib.auth.models.AnonymousUser)**. You can tell them apart with **[is\_authenticated](https://docs.djangoproject.com/en/4.1/ref/contrib/auth/" \l "django.contrib.auth.models.User.is_authenticated" \o "django.contrib.auth.models.User.is_authenticated)**, like so:

**if** request.user.is\_authenticated:

... *# Do something for logged-in users.*

**else**:

... *# Do something for anonymous users.*

**Methods**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#methods)

**HttpRequest.get\_host()**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.get_host)

Returns the originating host of the request using information from the **HTTP\_X\_FORWARDED\_HOST** (if [**USE\_X\_FORWARDED\_HOST**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-USE_X_FORWARDED_HOST) is enabled) and **HTTP\_HOST** headers, in that order. If they don’t provide a value, the method uses a combination of **SERVER\_NAME** and **SERVER\_PORT** as detailed in [**PEP 3333**](https://peps.python.org/pep-3333/).

Example: **"127.0.0.1:8000"**

Raises **django.core.exceptions.DisallowedHost** if the host is not in [**ALLOWED\_HOSTS**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-ALLOWED_HOSTS) or the domain name is invalid according to [**RFC 1034**](https://datatracker.ietf.org/doc/html/rfc1034.html)/[**1035**](https://datatracker.ietf.org/doc/html/rfc1035.html).

**Note**

The **[get\_host()](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.get_host" \o "django.http.HttpRequest.get_host)** method fails when the host is behind multiple proxies. One solution is to use middleware to rewrite the proxy headers, as in the following example:

**class** **MultipleProxyMiddleware**:

FORWARDED\_FOR\_FIELDS = [

'HTTP\_X\_FORWARDED\_FOR',

'HTTP\_X\_FORWARDED\_HOST',

'HTTP\_X\_FORWARDED\_SERVER',

]

**def** \_\_init\_\_(self, get\_response):

self.get\_response = get\_response

**def** \_\_call\_\_(self, request):

*"""*

*Rewrites the proxy headers so that only the most*

*recent proxy is used.*

*"""*

**for** field **in** self.FORWARDED\_FOR\_FIELDS:

**if** field **in** request.META:

**if** ',' **in** request.META[field]:

parts = request.META[field].split(',')

request.META[field] = parts[-1].strip()

**return** self.get\_response(request)

This middleware should be positioned before any other middleware that relies on the value of **[get\_host()](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.get_host" \o "django.http.HttpRequest.get_host)** – for instance, **[CommonMiddleware](https://docs.djangoproject.com/en/4.1/ref/middleware/" \l "django.middleware.common.CommonMiddleware" \o "django.middleware.common.CommonMiddleware)** or **[CsrfViewMiddleware](https://docs.djangoproject.com/en/4.1/ref/middleware/" \l "django.middleware.csrf.CsrfViewMiddleware" \o "django.middleware.csrf.CsrfViewMiddleware)**.

**HttpRequest.get\_port()**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.get_port)

Returns the originating port of the request using information from the **HTTP\_X\_FORWARDED\_PORT** (if [**USE\_X\_FORWARDED\_PORT**](https://docs.djangoproject.com/en/4.1/ref/settings/#std-setting-USE_X_FORWARDED_PORT) is enabled) and **SERVER\_PORT** **META** variables, in that order.

**HttpRequest.get\_full\_path()**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.get_full_path)

Returns the **path**, plus an appended query string, if applicable.

Example: **"/music/bands/the\_beatles/?print=true"**

**HttpRequest.get\_full\_path\_info()**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.get_full_path_info)

Like **[get\_full\_path()](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.get_full_path" \o "django.http.HttpRequest.get_full_path)**, but uses **[path\_info](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.path_info" \o "django.http.HttpRequest.path_info)** instead of [**path**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.path).

Example: **"/minfo/music/bands/the\_beatles/?print=true"**

**HttpRequest.build\_absolute\_uri(*location=None*)**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.build_absolute_uri)

Returns the absolute URI form of **location**. If no location is provided, the location will be set to **request.get\_full\_path()**.

If the location is already an absolute URI, it will not be altered. Otherwise the absolute URI is built using the server variables available in this request. For example:

**>>>** request.build\_absolute\_uri()

'https://example.com/music/bands/the\_beatles/?print=true'

**>>>** request.build\_absolute\_uri('/bands/')

'https://example.com/bands/'

**>>>** request.build\_absolute\_uri('https://example2.com/bands/')

'https://example2.com/bands/'

**Note**

Mixing HTTP and HTTPS on the same site is discouraged, therefore **[build\_absolute\_uri()](https://docs.djangoproject.com/en/4.1/ref/request-response/" \l "django.http.HttpRequest.build_absolute_uri" \o "django.http.HttpRequest.build_absolute_uri)** will always generate an absolute URI with the same scheme the current request has. If you need to redirect users to HTTPS, it’s best to let your web server redirect all HTTP traffic to HTTPS.

**HttpRequest.get\_signed\_cookie(*key*, *default=RAISE\_ERROR*, *salt=''*, *max\_age=None*)**[**¶**](https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.get_signed_cookie)

Returns a cookie value for a signed cookie, or raises a **django.core.signing.BadSignature** exception if the signature is no longer valid. If you provide the **default** argument the exception will be suppressed and that default value will be returned instead.

The optional **salt** argument can be used to provide extra protection against brute force attacks on your secret key. If supplied, the **max\_age** argument will be checked against the signed timestamp attached to the cookie value to ensure the cookie is not older than **max\_age** seconds.

For example:

**>>>** request.get\_signed\_cookie('name')

'Tony'

**>>>** request.get\_signed\_cookie('name', salt='name-salt')

'Tony' # assuming cookie was set using the same salt

**>>>** request.get\_signed\_cookie('nonexistent-cookie')

**...**

KeyError: 'nonexistent-cookie'

**>>>** request.get\_signed\_cookie('nonexistent-cookie', **False**)

False

**>>>** request.get\_signed\_cookie('cookie-that-was-tampered-with')

**...**

BadSignature: ...

**>>>** request.get\_signed\_cookie('name', max\_age=60)

**...**

SignatureExpired: Signature age 1677.3839159 > 60 seconds

**>>>** request.get\_signed\_cookie('name', **False**, max\_age=60)

False

See [cryptographic signing](https://docs.djangoproject.com/en/4.1/topics/signing/) for more information.