

# What Are Cookies?

Cookies are **small pieces of data stored in the user's browser**, sent by the server, and automatically included in future HTTP requests to the same server.

- Size limit: **~4 KB per cookie**
- Stored per **domain + path**
- Used to maintain **state** in stateless HTTP (e.g., login sessions, preferences, carts)

## ✓ What Kind of Data Is Stored in Cookies?

Use Case	Example
Authentication	sessionid, JWT, auth_token
Preferences	theme=dark, language=en
Tracking / Analytics	Google Analytics cookie <code>_ga</code>
Shopping	cart items, last visited product

✓ Only cookies **matching the domain + path + protocol rules** are sent.

Example:

Cookie	Domain	Path	Sent To
theme=dark	example.com	/	✓ example.com/home ✓ example.com/cart
cart_id=22	shop.example.com	/	✓ shop.example.com
debug=true	example.com	/admin	✓ example.com/admin ✗ example.com/home

So the browser decides which cookies to send, not the server.

## Important Cookie Attributes

Attribute	Purpose	Example
<code>expires</code>	Expiry date	<code>expires=Wed, 09 Jun 2027</code>
<code>max-age</code>	Expiry in seconds	<code>max-age=3600</code>
<code>path</code>	URL scope	<code>/admin</code>
<code>domain</code>	Domain scope	<code>.example.com</code>
<code>secure</code>	HTTPS only	<code>secure</code>
<code>httponly</code>	JS can't access	<code>httponly</code>
<code>SameSite</code>	CSRF protection	<code>SameSite=Lax</code>

If you set **both** `expires` and `max_age` in a cookie, the browser will prioritize `max-age` and ignore `expires` (according to the modern cookie spec).

**Cookies Usage in Django:**

## ► Setting a Cookie in Response

python

```
from django.http import HttpResponseRedirect

def set_cookie_view(request):
    response = HttpResponseRedirect("Cookie Set")
    response.set_cookie(
        key="theme",
        value="dark",
        max_age=3600,          # 1 hour
        secure=True,          # only on https
        httponly=True,        # not accessible via JS
        samesite='Lax'        # prevents CSRF
    )
    return response
```

## ► Reading a Cookie

python

```
def get_cookie_view(request):
    theme = request.COOKIES.get('theme')
    return HttpResponseRedirect(f"Theme: {theme}")
```

## ► Deleting a Cookie

python

```
def delete_cookie_view(request):
    response = HttpResponseRedirect("Cookie deleted")
    response.delete_cookie("theme")
    return response
```

## 1 Session Cookie (Deleted when browser closes)

A session cookie is simply a cookie **without** `max_age` or `expires`.

```
def session_cookie(request):
    response = HttpResponseRedirect("Session cookie set ✅")
    response.set_cookie("session_demo", "hello_session")    # no expiry
    return response
```

- ✓ Browser deletes it when closed
  - ✓ Default behaviour of Django session cookie (`sessionid`)
- 

## 2 Persistent Cookie (Expires after given time)

You set `max_age` (seconds) or `expires` (date).

```
def persistent_cookie(request):
    response = HttpResponseRedirect("Persistent cookie set ✅")
    response.set_cookie("remember_me", "true", max_age=7 * 24 * 60 *
60)    # 7 days
    return response
```

- ✓ Stored even after browser restart
  - ✓ Good for `remember_me` or `language=en`
- 

## 3 Secure Cookie (Sent only over HTTPS)

```
def secure_cookie(request):
    response = HttpResponseRedirect("Secure cookie set ✅")
    response.set_cookie(
        "secure_token",
        "12345",
```

```
        secure=True    # only sent via HTTPS
    )
    return response
```

⚠ Browsers treat `localhost` as a "secure context" even without HTTPS

✓ But only `localhost` and `127.0.0.1` get this exception

✖ Any other HTTP domain/IP will reject secure cookies

✓ Prevents cookie leak on insecure HTTP

---

## 4 HttpOnly Cookie (JavaScript cannot access)

Prevents XSS attacks by blocking `document.cookie` access.

```
def httponly_cookie(request):
    response = HttpResponse("HttpOnly cookie set ✓")
    response.set_cookie(
        "auth_user",
        "ajay",
        httponly=True
    )
    return response
```

✓ JS cannot read it

✓ Browser still sends it in requests

✓ Used for sessionid by Django by default

---


## 5 SameSite Cookie (CSRF protection)

```
def samesite_cookie(request):
    response = HttpResponse("SameSite cookie set ✓")
    response.set_cookie(
        "csrftoken_demo",
        "abc123",
        samesite="Strict"    # Strict | Lax | None
```

```
)  
return response
```

SameSite Value	Behaviour
Strict	Never sent in cross-site request (most secure)
Lax	Sent for GET/navigation links (Django default)
None	Allows cross-site requests but <b>must also use secure=True</b>

### Cookies vs LocalStorage (Browser Memory)

Feature	Cookies	LocalStorage	
Where stored?	Browser (per domain, auto-managed by browser)	Browser (key-value storage, per domain)	
Max size	~4 KB per cookie	~5–10 MB per domain	
Sent automatically with every request?	✓ Yes (HTTP headers)	✗ No (manual access only)	
Good for authentication?	✓ Yes (sessionid, JWT if HttpOnly)	✗ Not recommended (exposed to JS → XSS risk)	
Accessible via JavaScript?	✓ Yes (unless <code>HttpOnly=True</code> )	✓ Yes	
Can be <code>HttpOnly</code> ?	✓ Yes (extra security)	✗ No, always readable by JS	
Can be <code>Secure</code> (HTTPS only)?	✓ Yes	✗ No	
CSRF risk?	✓ Yes (auto-sent)	✗ No (not auto-sent)	
Expires automatically?	✓ Yes (session or expiry time)	✗ No (persists until cleared manually)	
Use cases	login session, CSRF token, remember-me, tracking	app state, form drafts, theme, cached API data	
Storage type	String only	String only (but you can store JSON via <code>JSON.stringify()</code> )	