



Janja Programmers

1. Request Processing:

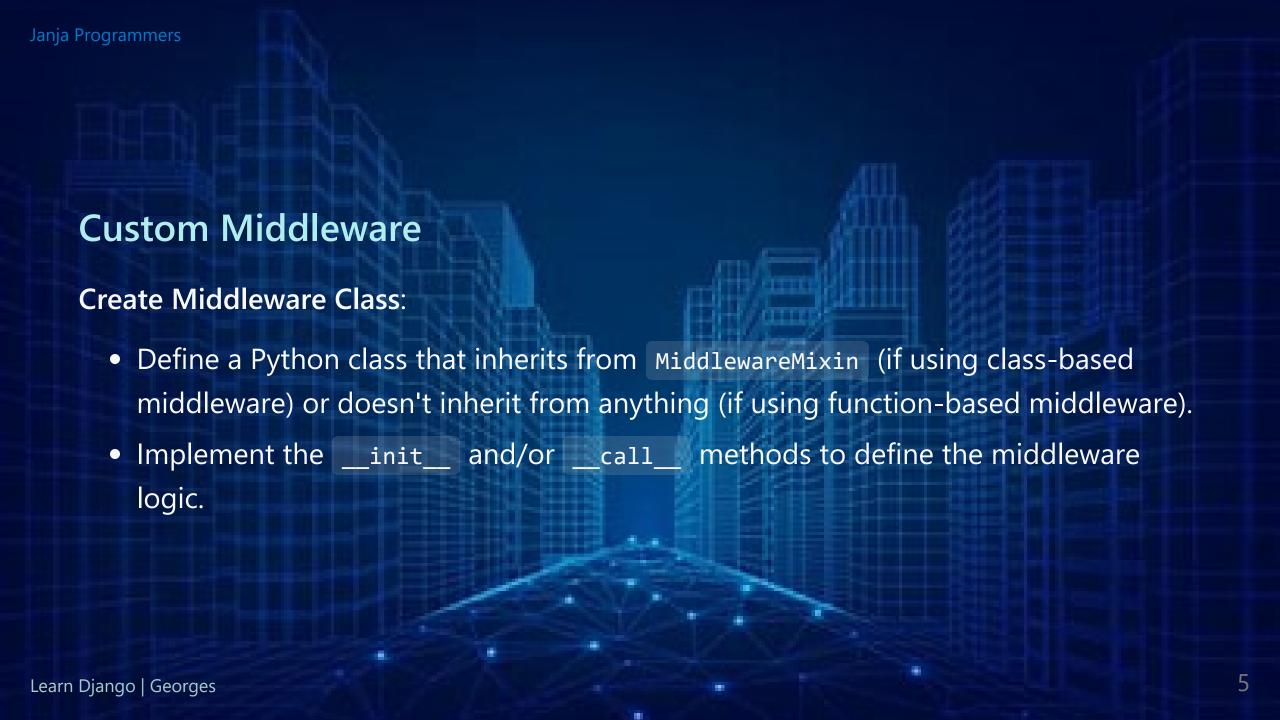
- Middleware executes during the request phase before reaching the view layer.
- It can intercept incoming requests, modify request headers, perform authentication, etc.

2. Response Processing:

- Middleware executes during the response phase after leaving the view layer.
- It can intercept outgoing responses, modify response headers, set cookies, etc.

3





Modify Requests and Responses:

- Inside the middleware class, implement logic to modify incoming requests, outgoing responses, or both.
- You can access and modify request and response objects as needed.

Example: Custom Middleware Adding Custom Header

```
# middleware.py
class CustomHeaderMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

def __call__(self, request):
    # Add custom header to the response
    response = self.get_response(request)
    response['X-Custom-Header'] = 'Hello from Custom Middleware'
    return response
```



Steps to Register Middleware

- 1. Add Middleware to Settings:
 - In your project's settings module (settings.py), add the path to your custom
 middleware class or function to the MIDDLEWARE list.
 - Ensure that the order of middleware in the list reflects the desired execution order.

Snippet

```
# settings.py
MIDDLEWARE = [
    # Other middleware...
    'myapp.middleware.CustomHeaderMiddleware',
]
```

Learn Django | Georges



Summary

Middleware plays a vital role in Django's request/response processing pipeline, allowing you to intercept requests and responses, perform additional tasks, and modify data as needed. By writing custom middleware, you can extend Django's functionality to meet the specific requirements of your application. Understanding middleware is essential for building robust and flexible Django applications.