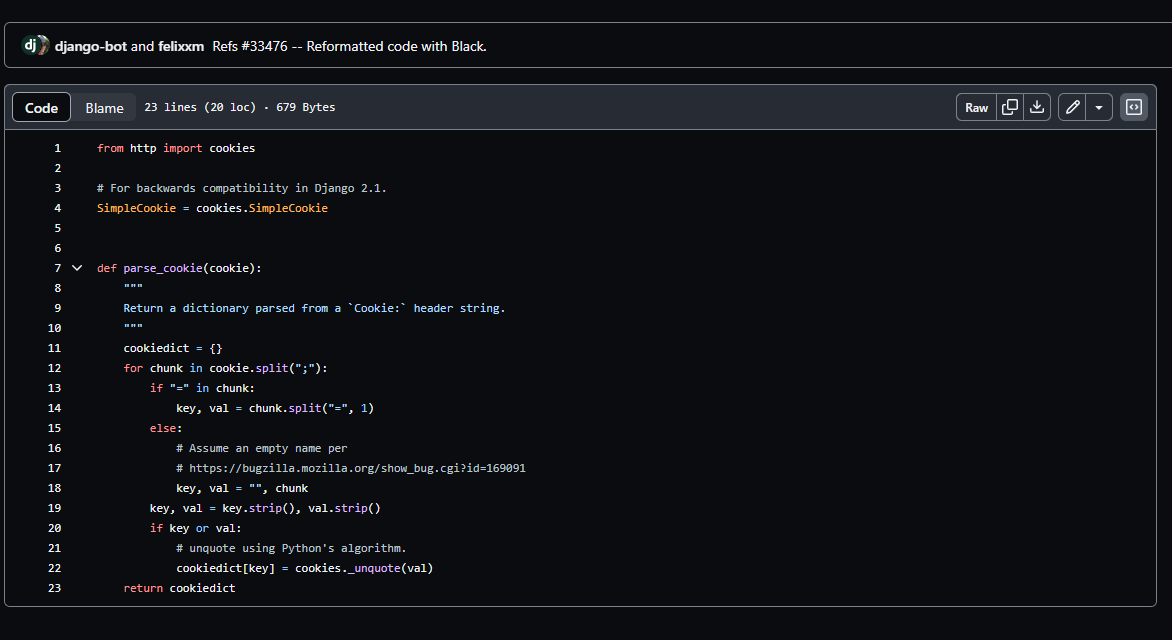
Open Source Contributions

<https://github.com/django/django/blob/main/django/http/cookie.py>



**Changed code suggestion:-**

from http import cookies

def parse\_cookie(cookie):

"""

Return a dictionary parsed from a `Cookie:` header string.

"""

cookiedict = {}

for chunk in cookie.split(";"):

key, \_, val = map(str.strip, chunk.partition("="))

if key or val:

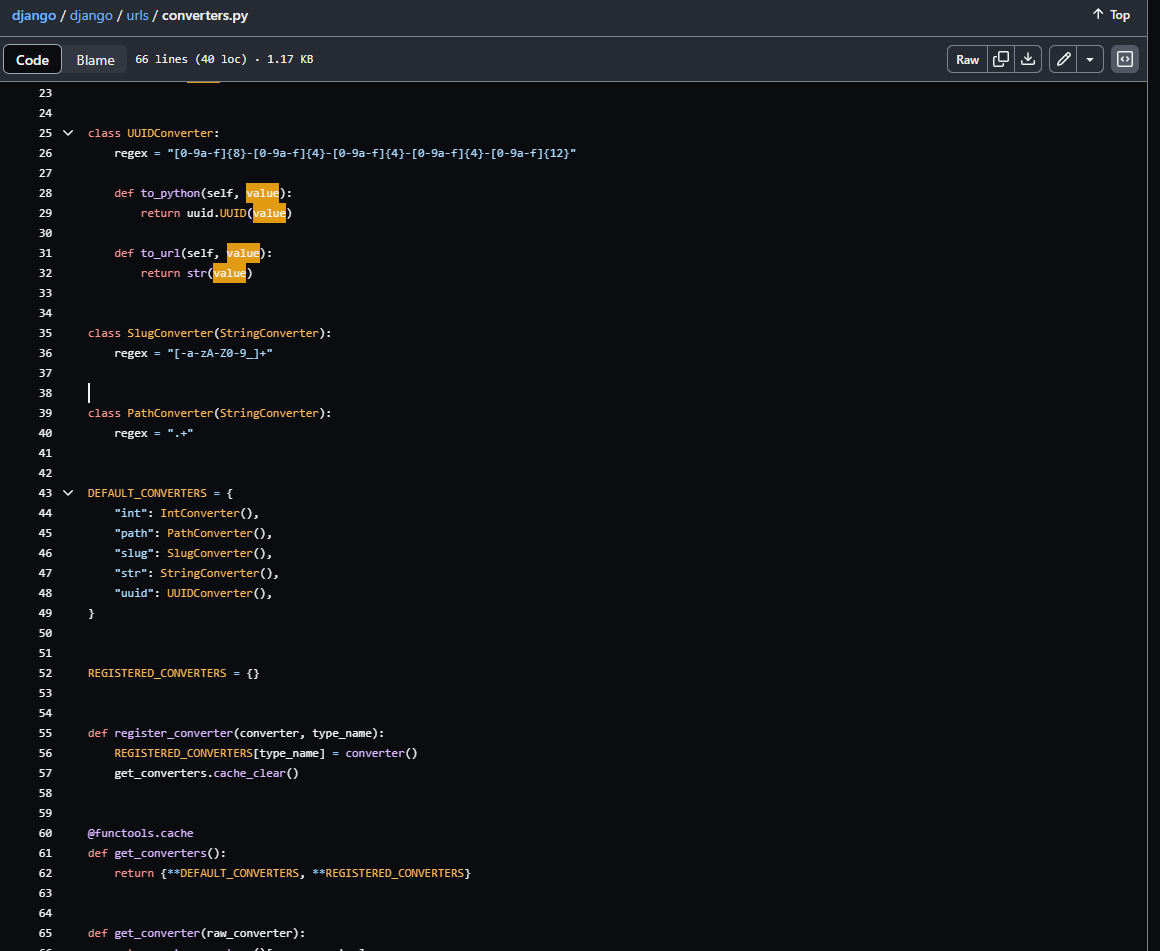
cookiedict[key] = cookies.\_unquote(val)

return cookiedict

**Reason:-**

* Removed the unnecessary import of Simple Cookie since it is not used in the code.
* Simplified the extraction of key and val using str.partition.
* Combined the key and value stripping into a single line for cleaner code.

1. <https://github.com/django/django/blob/main/django/urls/converters.py>



**Changed code suggestion:-**

class ConverterRegistry:

def \_\_init\_\_(self):

self.converters = {

"int": IntConverter(),

"path": PathConverter(),

"slug": SlugConverter(),

"str": StringConverter(),

"uuid": UUIDConverter(),

}

def register\_converter(self, converter, type\_name):

self.converters[type\_name] = converter()

def get\_all\_converters(self):

return self.converters

def get\_converter(self, raw\_converter):

return self.converters.get(raw\_converter)

converter\_registry = ConverterRegistry()

**Reasons: -**

* Avoid Using Cache for get\_converters
* Use a Class for Converters

1. <https://github.com/django/django/blob/main/django/templatetags/l10n.py>

A screenshot of a computer program

Description automatically generated

**Changed code suggestion:-**

from django.template import Library, Node, TemplateSyntaxError

from django.utils import formats

register = Library()

@register.filter(is\_safe=False)

def localize(value):

"""

Force a value to be rendered as a localized value.

"""

return str(formats.localize(value, use\_l10n=True))

@register.filter(is\_safe=False)

def unlocalize(value):

"""

Force a value to be rendered as a non-localized value.

"""

return str(formats.localize(value, use\_l10n=False))

class LocalizeNode(Node):

def \_\_init\_\_(self, nodelist, use\_l10n):

self.nodelist = nodelist

self.use\_l10n = use\_l10n

def \_\_repr\_\_(self):

return f"<{self.\_\_class\_\_.\_\_name\_\_}>"

def render(self, context):

old\_setting = context.use\_l10n

context.use\_l10n = self.use\_l10n

output = self.nodelist.render(context)

context.use\_l10n = old\_setting

return output

@register.tag("localize")

def localize\_tag(parser, token):

"""

Force or prevents localization of values.

Sample usage::

{% localize off %}

var pi = {{ 3.1415 }};

{% endlocalize %}

"""

use\_l10n = None

bits = list(token.split\_contents())

if len(bits) == 1:

use\_l10n = True

elif len(bits) == 2 and bits[1] in ("on", "off"):

use\_l10n = bits[1] == "on"

else:

raise TemplateSyntaxError(f"{bits[0]} argument should be 'on' or 'off'")

nodelist = parser.parse(("endlocalize",))

parser.delete\_first\_token()

return LocalizeNode(nodelist, use\_l10n)

**Reasons:-**

* When raising a TemplateSyntaxError, consider using f-strings or format() for string formatting. It can make the code more readable and maintainable.
* Set is\_safe=False for the localize and unlocalize filters. This is appropriate since these filters may output HTML. And be ready for the setting is consistent with the actual behaviour.
* The LocalizeNode class and its methods follow the snake\_case naming convention, which is good. Ensure consistency in naming conventions throughout codebase.