**CSV**

**CSV stands for comma separated values is a text file**

**CSV is standard import and export format used by many database application and programming languages.**

**CSV response/out is generated in two ways**

**1. csv module**

**2. Djagno template system**

**content-type --> text/csv --> MIME**

**How to create CSV output**

How to output CSV (Comma Separated Values) dynamically using Django views. To do this, you can either use the Python CSV library or the Django template system.

**Using the Python CSV library**

Python comes with a CSV library, [**csv**](https://docs.python.org/3/library/csv.html#module-csv). The key to using it with Django is that the [**csv**](https://docs.python.org/3/library/csv.html#module-csv) module’s CSV-creation capability acts on file-like objects, and Django’s [**HttpResponse**](https://docs.djangoproject.com/en/4.0/ref/request-response/#django.http.HttpResponse) objects are file-like objects.

Here’s an example:

**import** **csv**

**from** **django.http** **import** HttpResponse

**def** some\_view(request):

*# Create the HttpResponse object with the appropriate CSV header.*

response = HttpResponse(

content\_type='text/csv',

headers={'Content-Disposition': 'attachment; filename="somefilename.csv"'},

)

writer = csv.writer(response)

writer.writerow(['First row', 'Foo', 'Bar', 'Baz'])

writer.writerow(['Second row', 'A', 'B', 'C', '"Testing"', "Here's a quote"])

return response

The response gets a special MIME type, *text/csv*. This tells browsers that the document is a CSV file, rather than an HTML file. If you leave this off, browsers will probably interpret the output as HTML, which will result in junk output in the browser window.

The response gets an additional **Content-Disposition** header, which contains the name of the CSV file. This filename is arbitrary; call it whatever you want. It’ll be used by browsers in the “Save as…” dialog, etc.

from django.shortcuts import render

# Create your views here.

import csv

from django.http import HttpResponse

def some\_view(request):

    # Create the HttpResponse object with the appropriate CSV header.

    response = HttpResponse(

        content\_type='text/csv',

        headers={'Content-Disposition': 'attachment; filename="somefilename.csv"'},

    )

    writer = csv.writer(response)

    writer.writerow([101,'naresh','python',4000.0])

    writer.writerow([102,'suresh','java',2000.0])

    return response

**Using the template system**

Alternatively, you can use the [Django template system](https://docs.djangoproject.com/en/4.0/topics/templates/) to generate CSV. This is lower-level than using the convenient Python [**csv**](https://docs.python.org/3/library/csv.html#module-csv) module, but the solution is presented here for completeness.

The idea here is to pass a list of items to your template, and have the template output the commas in a [**for**](https://docs.djangoproject.com/en/4.0/ref/templates/builtins/#std-templatetag-for) loop.

Here’s an example, which generates the same CSV file as above:

**from** **django.http** **import** HttpResponse

**from** **django.template** **import** loader

**def** some\_view(request):

*# Create the HttpResponse object with the appropriate CSV header.*

response = HttpResponse(

content\_type='text/csv',

headers={'Content-Disposition': 'attachment; filename="somefilename.csv"'},

)

csv\_data = (

('First row', 'Foo', 'Bar', 'Baz'),

('Second row', 'A', 'B', 'C', '"Testing"', "Here's a quote"),

)

t = loader.get\_template('my\_template\_name.txt')

c = {'data': csv\_data}

response.write(t.render(c))

**return** response

my\_template\_name.txt

{% for row in data %} "{{ row.0|addslashes }}", "{{ row.1|addslashes }}", "{{ row.2|addslashes }}", "{{ row.3|addslashes }}", "{{ row.4|addslashes }}"

{% endfor %}

**How to create PDF files**

This document explains how to output PDF files dynamically using Django views. This is made possible by the excellent, open-source [ReportLab](https://www.reportlab.com/opensource/) Python PDF library.

The advantage of generating PDF files dynamically is that you can create customized PDFs for different purposes – say, for different users or different pieces of content.

**Install ReportLab**

The ReportLab library is [available on PyPI](https://pypi.org/project/reportlab/).

python -m pip install reportlab

Test your installation by importing it in the Python interactive interpreter:

**>>> import** **reportlab**

If that command doesn’t raise any errors, the installation worked.