EXPORT HTML WEB PAGE TO PDF USING JSPDF

Using jsPDF in Django templates to export as PDF

jsPDF is used to generate pdf files in client-side Javascript.

You can find the links for jsPDF here and also you can find the link to project homepage.

You've to include the scripting files/links in head section to work properly.

Tip: We have to download the newest version of the library and include it in the HEAD or at the end of the BODY section.

Example to run the script from local storage:

In the HEAD section:

(or)

In the BODY section:

After loading the scripts in HEAD/BODY section, now we can start using the jsPDF library.

Lets start with some of basics of jsPDF to get the idea of using it in our applications:

First let us discuss how to create a new document?

It's simple as below mentioned:

```
var doc = new jsPDF(orientation, unit, format, compress);
```

The constructor can take several parameters.

- **orientation** - The default value of orientation is **"portrait"**. We can set it to **"landscape"** if we want a different page orientation.
- **unit** - We can tell jsPDF in which units we want to work. Use on of the following: "**pt**" (points), "**mm**" (default), "**cm**", "**in**".
- **format** - It's default page format. We can change it we want other formats also like: "**a3**", "**a4**" (default), "**a5**", "**letter**", "**legal**".

As an order to understand, here is an example:

```
var doc = new jsPDF('landscape');
doc.text(20, 20, 'Hello landscape world!');
```

We can add new page using the following code:

```
doc.addPage(width, height);
```

As parameters we can pass the page width and height in the units defined in the docuent constructor. Adding pages moves us to this page, so many operations will be executed on that page. If we want to go to another page we can use the **setPage** function.

```
doc.setPage(pageNumber);
```

You can also get the actual number of pages by using the below code:

```
doc.internal.getNumberOfPages();
```

In an order to understand, here is an example:

```
var doc = new jsPDF();
doc.text(20, 20, 'Hello world!');
doc.text(20, 30, 'This is client-side Javascript, pumping out a PDF.');
doc.addPage('a6','l');
doc.text(20, 20, 'Do you like that?');
```

Now it's time to work with text:

1. The most important thing is displaying text, we do it using the funtion named **doc.text** which takes 3 parameters. like below:

```
doc.text(X, Y, "Text to be displayed");
```

- The first two are X and Y positions of the text in units defined in the document constructor.

* Here we have to notice a point: Y position, it is the position of the text baseline, so printing something with the Y position set to 0 will actually print it over the top edge of the document.

- The third argument is the text to be displayed.

2. The second thing is the font name used to draw the text. We can choose one of the following: **courier**, **times**, **arial**. We can change the font family and font style by running the **doc.setFont** function.

```
doc.setFont("arial", "bold");
```

By executing the **doc.getFontList** function we can find out what fonts are available and what font styles we can set for given font.

```
doc.getFontList();
/*
{
    "courier": ["normal", "bold", "italic", "bolditalic"],
    "Courier": ["", "Bold", "Oblique", "BoldOblique"],
    "times": ["normal", "bold", "italic", "bolditalic"],
    "Times": ["Roman", "Bold", "Italic", "BoldItalic"],
    "helvetica": ["normal", "bold", "italic", "bolditalic"],
    "Helvetica": ["", "Bold", "", ""]
}
*/
```

We can also change the font style individually by using the **doc.setFontStyle** or the **doc.setFontType** function, which is alias to the first one.

```
doc.setFontType("bolditalic");
// is the same as calling
doc.setFontStyle("bolditalic");
```

3. Next things is the font size. Now, for that we use **doc.setFontSize** function.

```
doc.setFontSize(40);
```

Example:

```
var doc = new jsPDF();
doc.setFontSize(22);
doc.text(20, 20, 'This is a title');

doc.setFontSize(16);
doc.text(20, 30, 'This is some normal sized text underneath.');
```

4. Last but not the least, the text color. We can change the text color using the **doc.setTextColor** function and passing three parameters which are RGB (Red, Green, Blue) color values.

```
var doc = new jsPDF();
// I know the proper spelling is colour ;)
doc.setTextColor(100);
doc.text(20, 20, 'This is gray.');

doc.setTextColor(150);
doc.text(20, 30, 'This is light gray.');

doc.setTextColor(255, 0, 0);
doc.text(20, 40, 'This is red.');

doc.setTextColor(0, 255, 0);
doc.text(20, 50, 'This is green.');

doc.setTextColor(0, 0, 255);
doc.text(20, 60, 'This is blue.');
```

In a order of full example:

```
var doc = new jsPDF();

doc.text(20, 20, 'This is the default font.');

doc.setFont("courier");
doc.setFontType("normal");
doc.text(20, 30, 'This is courier normal.');

doc.setFont("times");
doc.setFontType("italic");
doc.text(20, 40, 'This is times italic.');

doc.setFont("helvetica");
doc.setFontType("bold");
doc.text(20, 50, 'This is helvetica bold.');

doc.setFont("courier");
doc.setFontType("bolditalic");
doc.text(20, 60, 'This is courier bolditalic.');
```

Now it's time to work with Images:

We have only function for the images is the **doc.addImage**. It takes image as a first parameter, image format/type as a second and X,Y positions of the image as a third and fourth arguments respectively. Here we can also optionally pass new image size as a fifth and sixth arguments.

```
var img = new Image();
img.addEventListener('load', function() {
    var doc = new jsPDF();
    doc.addImage(img, 'png', 10, 50);
});
img.src = 'image_path/image_name.png';
```

In the above example, we passed an Image HTML DOM element as a first argument of the **addImage** function, however it can also be a base64 encoded image string.

```
var imgData =
'data:image/jpeg;base64,/.......base64code.....iiigAoooo//2Q==';
var doc = new jsPDF();
doc.setFontSize(40);
doc.text(35, 25, "Octonyan loves jsPDF");
doc.addImage(imgData, 'JPEG', 15, 40, 180, 180);
```

Working With Graphics:

First of all, we have to set the drawn shapes fill and stroke with colors. We do it using the **doc.setFillColor** and the **doc.setDrawColor** accordingly, passing RGB color values as parameters.

```
doc.setFillColor(100, 100, 240);
doc.setDrawColor(100, 100, 0);
```

We can also set the stroke width. The stroke width unit is the same as defined in the document constructor.

```
doc.setLineWidth(1);
```

Every shape drawing function takes the center point co-ordinates (triangle is the only exception) as two first parameters. They also take the last parameter for drawing style. It can be "**S**", "**F**", "**DF**", "**FD**" string and the meanings are: "stroke", "fill", "stroke and fill", "fill and stroke". The last two of course differ in the order of the drawing operations.

For example, we can draw an ellipse, by passing two radiuses :

```
// Empty ellipse
doc.ellipse(50, 50, 10, 5);
// Filled ellipse
doc.ellipse(100, 50, 10, 5, 'F');
// Filled circle with borders
```

or a circle, by passing only one radius as parameter :

```
doc.circle(150, 50, 5, 'FD');
```

or a rectangle, by passing it's width and height as parameters :

```
var doc = new jsPDF();

// Empty square
doc.rect(20, 20, 10, 10);

// Filled square
doc.rect(40, 20, 10, 10, 'F');

// Empty red square
doc.setDrawColor(255,0,0);
doc.rect(60, 20, 10, 10);

// Filled square with red borders
doc.setDrawColor(255,0,0);
doc.rect(80, 20, 10, 10, 'FD');

// Filled red square
doc.setDrawColor(0);
doc.setFillColor(255,0,0);
doc.rect(100, 20, 10, 10, 'F');

 // Filled red square with black borders
doc.setDrawColor(0);
doc.setFillColor(255,0,0);
doc.rect(120, 20, 10, 10, 'FD');
```

or a rounded rectangle, by passing it's width, height and border radius as parameters

:

```
// Filled sqaure with rounded corners
doc.roundedRect(50, 150, 10, 10, 3, 3, 'FD');
// Black sqaure with rounded corners
doc.setDrawColor(0);
doc.setFillColor(255, 255, 255);
doc.roundedRect(140, 20, 10, 10, 3, 3, 'FD');
```

and a triangle, by passing each corners co-ordinates.

```
// Filled triangle with borders
var doc = new jsPDF();
doc.triangle(60, 100, 60, 120, 80, 110, 'FD');
doc.setLineWidth(1);
doc.setDrawColor(255,0,0);
doc.setFillColor(0,0,255);
doc.triangle(100, 100, 110, 100, 120, 130, 'FD');
```

Last but not least, we can also draw lines passing through the co-ordinates of two points.

```
// Line
doc.line(50, 250, 100, 250);
```

Here is an example and some of functionalities we can apply on lines:

```
var doc = new jsPDF();

doc.line(20, 20, 60, 20); // horizontal line

doc.setLineWidth(0.5);
doc.line(20, 25, 60, 25);

doc.setLineWidth(1);
doc.line(20, 30, 60, 30);

doc.setLineWidth(1.5);
doc.line(20, 35, 60, 35);

doc.setDrawColor(255,0,0); // draw red lines

doc.setLineWidth(0.1);
doc.line(100, 20, 100, 60); // vertical line

doc.setLineWidth(0.5);
doc.line(105, 20, 105, 60);

doc.setLineWidth(1);
doc.line(110, 20, 110, 60);

doc.setLineWidth(1.5);
doc.line(115, 20, 115, 60);
```

Here is a full example to understand in a better way:

```
var doc = new jsPDF();

doc.ellipse(40, 20, 10, 5);

doc.setFillColor(0,0,255);
doc.ellipse(80, 20, 10, 5, 'F');

doc.setLineWidth(1);
doc.setDrawColor(0);
doc.setFillColor(255,0,0);
doc.circle(120, 20, 5, 'FD');
```

Now, we are going to know about an interesting functionality over here - addHTML():

The below example will show you a preview of the html document that you've added this jsPDF script to generate pdf. This will generate the pdf as shown in the browsers.

```
var pdf = new jsPDF('p','pt','a4');

pdf.addHTML(document.body,function() {
 var string = pdf.output('datauristring');
 $('.preview-pane').attr('src', string);
});
```

The below example code will render the html and dispalys as plain text.

```
var doc = new jsPDF();

// We'll make our own renderer to skip this editor
var specialElementHandlers = {
 '#editor': function(element, renderer){
  return true;
 }
};

// All units are in the set measurement for the document
// This can be changed to "pt" (points), "mm" (Default), "cm", "in"
doc.fromHTML($('body').get(0), 15, 15, {
 'width': 170,
 'elementHandlers': specialElementHandlers
});
```

We can also add Metadata to our Generating pdf:

```
var doc = new jsPDF();
doc.text(20, 20, 'This PDF has a title, subject, author, keywords and a creator.');

// Optional - set properties on the document
doc.setProperties({
 title: 'Title',
 subject: 'This is the subject',
 author: 'Author Name',
 keywords: 'generated, javascript, web 2.0, ajax',
 creator: 'Creator Name'
});

// Output as Data URI
doc.save('Test.pdf');
```

For the people who want a full script with an example:

1. Add the following script to the HEAD:

```
<script src="http://mrrio.github.io/jsPDF/dist/jspdf.debug.js"></script>
```

or **Download Locally**

2. Add HTML script to execute jsPDF code:

Customize this to pass the identifier or just change the **#content** to be the identifier you need.

```
 <script>
    function demoFromHTML() {
        var pdf = new jsPDF('p', 'pt', 'letter');
        // source can be HTML-formatted string, or a reference
        // to an actual DOM element from which the text will be scraped.
        source = $('#content')[0];

        // we support special element handlers. Register them with
jQuery-style
        // ID selector for either ID or node name. ("#iAmID", "div",
"span" etc.)
        // There is no support for any other type of selectors
        // (class, of compound) at this time.
        specialElementHandlers = {
            // element with id of "bypass" - jQuery style selector
            '#bypassme': function (element, renderer) {
                // true = "handled elsewhere, bypass text extraction"
                return true
            }
        };
        margins = {
            top: 80,
            bottom: 60,
            left: 40,
            width: 522
        };
        // all coords and widths are in jsPDF instance's declared units
        // 'inches' in this case
        pdf.fromHTML(
        source, // HTML string or DOM elem ref.
        margins.left, // x coord
        margins.top, { // y coord
            'width': margins.width, // max width of content on PDF
            'elementHandlers': specialElementHandlers
        },

        function (dispose) {
            // dispose: object with X, Y of the last line add to the PDF
            //          this allow the insertion of new lines after html
            pdf.save('Test.pdf');
        }, margins);
    }
</script>
```

3. Add your body content to generate as PDF:

```
<a class="button" href="javascript:demoFromHTML()">Generate PDF</a>
```

Here is a complete example to generate a pdf using jsPDF.

To Know more about our Django CRM(Customer Relationship Management) Open Source Package. Check Code

Previous post

Next post

Posted On 15 October 2015 By MicroPyramid

Need any Help in your Project? Let's Talk

Latest Comments

Comments Community 1 Login

Recommend 3 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Name

N3PatHYA • 6 months ago

How do i manage to break pages so that my data will not cut when convert to pdf. As my web page will have multiple tables with dynamic number of rows. How do i able to figure out that and move to next page?

1 ^ | ⌄ • Reply • Share ›

dan s • 10 months ago

I am using jsPDF to generate the PDF, but one issue i saw is it is not reading some of the css styles when exported the data to the PDF. Please find the demo http://jsfiddle.net/ugD5L/859/ . Inside the table we see the some of the columns data with red,yellow and green colors, but when exported to the PDF it is showing black. Any inputs on how to make jsPDF support css styles. I have used html2Canvas with jsPDF to overcome this issue but issue with html2Canvas is only one page is shown ,if i have large data it is not showing from page2.Any inputs?

^ | ⌄ • Reply • Share ›

Ceeyang • a year ago

My pdf document is showing Empty page. Need a help?

^ | ⌄ • Reply • Share ›

Geovanny Reyes • a year ago

I need Help

I have a error Uncaught ReferenceError: $ is not defined

here --- > source = $('#reportes')[0];

^ | ⌄ • Reply • Share ›

Vidyasagar Rudraram ➦ Geovanny Reyes
• a year ago

Hello Geovanny Reyes,

I think you've not included javascript/jquery file at first (before the script starts in the html file).
Can you verify whether the file included properly or not?

Example:
<html>
<body>
....

```
<-- include jquery (minified version) file before your
script starts -->
<script src="https://ajax.googleapis.com...">
</script>
<-- you can write your script below -->
<script type="text/javascript">
$(document).ready(function() {
// your script
});
</script>
</body>
</html>
```

^ | ⌄ • Reply • Share ›

romnick • a year ago

how about the if the img is so big and i want the other part of the browser in next page how can i make it? thanks

^ | ⌄ • Reply • Share ›

Christopher Val Tronco • 2 years ago

After you download locally... How do you add the local file to the header?

^ | ⌄ • Reply • Share ›

Vidyasagar Rudraram ➦ Christopher Val Tronco • a year ago

Hello Christopher (**@Christopher Val Tronco**),
Sorry for the late response, you can keep the downloaded jsPDF local file in the header section as mentioned below.

Example: If you have kept your downloaded "jspdf.debug.js" file like "jsPDF/dist/jspdf.debug.js" in a directory with the folder names as mentioned,

```
<script src="your_project_local_path_for_the
jsPDF_file/jsPDF/dist/jspdf.debug.js"></script>
```

If you are using staticfiles/static url and you kept your downloaded jsPDF file in static folder, then, you've to mention as below in your header to include the file

Services

Django Development

Python Development

ReactJS Development

AngularJS Development

Django Ecommerce Development

Django Rest Framework

Website Maintenance Services

Python Amazon Django Mongo DB Linux

RELATED ARTICLES

Creating Django App

Django is a high-level, free and open-source Python Web framework that encourages rapid development. Django follows the model–view–controller (MVC) architectural pattern. Django's primary goal is ...

Continue Reading...

Extract data from PDF and all Microsoft Office files in python

The quick way to get/extract text from PDFs in Python is with the Python library "slate". Slate is a Python package that simplifies the process ...

Continue Reading...

Understanding Checkout flow in Django Oscar.

Explaining Django Oscar checkout flow.

Continue Reading...

OPEN SOURCE PACKAGES

DJANGO-CRM : Customer relationship management based on Django

DJANGO-BLOG-IT : django blog with complete customization and ready to use with one click installer Edit

DJANGO-WEBPACKER : A django compressor tool

DJANGO-MFA : Multi Factor Authentication

DOCKER-BOX : Web Interface to manage full blown docker containers and images

More...

SUBSCRIBE TO OUR NEWS LETTER

Subscribe and Stay Updated about our Webinars, news and articles on Django, Python, Machine Learning, Amazon Web Services, DevOps, Salesforce, ReactJS, AngularJS, React Native.

*** We don't provide your email contact details to any third parties**

Email Id

SUBSCRIBE !

About Company

Blog

Our Experts

Careers

Cookie Policy

Contact Us

Sitemap

Services

Search Engine Optimization

Website Designing

Django Development

Salesforce CRM

Aws Consulting Services

React Native Development

E-Commerce SEO

Python Development

AMP Web Design

Simple Storage Services

Digital Marketing

Free IT Services Non Profit Organizations

Android Application Development

Docker

Bootstrap Web Development

SEO Resource Outsourcing

Simple Notification Services

ReactJs Development

AngularJs Development

Landing Page Optimization

Responsive Web Design

Get in Touch with us

India

Krishe Sapphire, 6th Floor, Madhapur, Hyderabad, India, 500081
+91-850 009 9499
hello@micropyramid.com

USA

3737 Mapleshade Ln, Suite 102,
Plano TX 75075
+1 510 230 0949
hello@micropyramid.com

UAE

Sharjah, UAE 341246
hello@micropyramid.com

UK

19 Ben tillet close, Barking, London, IG 11 9NT.
Reg. No: 11375687
(+44) 203 026 1643
(+44) 333 050 3851
hello@micropyramid.com

Connect With Social Networks

By continuing to navigate on this website, you accept the use of cookies to serve you more relevant services & content. For more information please read our Cookie Policy.