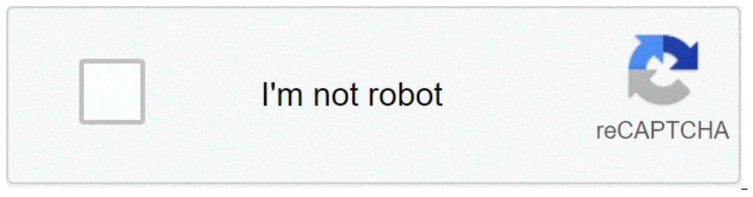

Continue

Jspdf fromhtml line break

I recently worked on the website where I would allow users to pull the text from an API. However, sometimes the text would be up to 15,000 characters, and sometimes it would be only 1,500 characters. However, without line breaks or new paragraphs, such large text blocks are difficult to read for a user. I'm sure there's something that's built in a place to do it more elegantly, but after a short search, I couldn't find it if it exists. So instead, I created a small script that automatically breaks the text in paragraphs. My initial thought was something like:1) Count the total number of words in the text.2) Split the text into the desired number of paragraphs. As an example, if I want 8 total paragraphs, and the total text is 10,000 characters, each paragraph will be 1250 characters long. so I would need to select the text between [0] and [1250], and then the text from [1250] and [2500], and so on. However, going along this path leads to paragraphs that can start or end in the middle of a word. So I altered it a little bit. the variable periods here gives me the total number of periods in the text. From I want 8 paragraphs, the variable index1 gives me the total number of periods between each paragraph. If there were 8 total periods, then there would be a new break in every 1 period. I have 16 total periods, then index1 occurs in every 2 periods, and so on. Next, I want to get an index of where each period occurs in the text. So, I get it.the text, and if the current tetration, speakingText[v] is a period, ".", then I push (v) into the array indices. Now, indexes include an index of each text period. I will then use the index1 variable as an index of my index series. So here, index 11 is the location of the first paragraph break. Now, I'll do the same for the other eight paragraphs. Now, each paragraph will end with index 11, index 22, index 33, and so on. Afterwards, I will get an original text substrange between [0] and index11, and then another substrand between index11 and index22, etc. Finally, each str variable contains the string I need for each paragraph, ending with a period instead of medium word. Now, using a model expression, I can only popularize an HTML element with new data. This may seem a little round to do it, but it works. Let me know if you have any feedback. Thank you! You can't do this right now. You signed up with another tab or window. Reload to update the session. You signed in another tab or window. Reload to update the session. jsPDF is used to generate pdf file in Javascript client side. You can find links for jsPDF here and also you can find the link to the project homepage. You must include scripting/link files in the head section to work properly. Tip: We need to download the new version of the library and include it in HEAD or at the end of the BODY section. Example to run the local storage script: In the heart(or) in the body section: after uploading scripts in the HEAD/BODY section, we can now start using the jspdf library. we start with some of the jspdf bases to get the idea of using it in our applications: First of all, we talk about how to create a new document? is simple as below indicated: var doc = new jspdf (orientation, unit, format, compression;) the manufacturer can take several parameters. orientation - the default orientation value 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 the following: pt (points,) mm (default,) cm, in. format - is default page format. we can change it we want other formats also like: a3, a4 (default,) a5, letter, letter. to understand, here is an example: var doc = new jspdf(panorama';) doc.text(20, 20, 'hello landscape world'!;) we can add a new page using the following code: doc.addPage (width, height;) as parameters we can pass the width and the page builder defined 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 oate the setpage function. doc.setPage(pageNumber); you can also get the actual number of pages using the following code: doc.internal.getNumberOfPages(); to understand, here is an example: var doc = new jspdf;)(doc.text(20, 20, 'hello world'!;) doc.text(20, 30, 'this isJavascript, pumping a PDF.'); doc.addPage('a6','l'); doc.text(20, 20, 'Do you like this?'); Now is time to work with the text: 1. The most important thing is the display of the text, we do it using the fun called doc.text which requires 3 parameters. as below: doc.text(X, Y, "Text to display"); - The first two are X and Y positions of the text in units defined in the document manufacturer. Here we have to see a point: The Y position, is the base line position of the text, then printing something with the Y position set to 0 actually printing on the top edge of the document. - No. The third argument is the text to be displayed. 2. The second thing is the name of the character used to draw the text. We can choose one of the following: courier, schedule, arial. We can change the character family and character style by performing the Doc.set Character function. Doc.setFont("arial", "bold"); Doc.get execution The FontList function can find out which fonts are available and which font styles we can set for a given character. Doc.getFontList(); /* { "courier": ["normal", "bold", "italic", "bolditalic"], "Courier": ["", "Bold", "Oblique", "BoldOblique", "time": ["normal", "bold", "italic", "bolditalic"], "Times": ["Romanta", "Bold". We can also change the font style individually using the Doc.set FontStyle or the Doc.set FontType function, which is alias at the first.// is the same as to call doc.setFontStyle("bolditalic"); 3. Later things are the font size. Now, for this we use the 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 a normal size text below'); 4. The color of the text. We can change the text color using the doc.setTextColor function and passing three parameters that are RGB (red, green, blue) color values. > Now, for this we use doc.splitTextToSize function const splitText = doc.splitTextToSize('hey I am a veeeeeeery long string', 10) doc.text(splitText, 0, 0) 6.Text alignment within the document var doc = new jsPDF('p', 'pt', 'letter'); Doc. 'This text is normally\raligned', 140, 50); Doc. "This text is centred on\raround\rthis point.", 140, 120, 'center'); Doc. 'This text is rotated\rand centered around\rthis point.', 140, 300, 45, 'center'); doc.text(This text is reallined to the text\rright.', 140, 400, 'right'); Doc. "This text is reallined to \rright.550, 45, 'right'; Doc. 'This single line is centered,' 460, 50, center); Doc. 'This text lined right\r\r\r\rhas empty line,'. 460, 200, right); doc.save ('Test.pdf'); in a complete example order: var doc = new jspdf;)(doc.setFont (type;) doc.setFont("courier"); doc.setFontType("normal"); doc.text(20, 30, 'this is the normal consideration'.;) doc.setFont('time'); Doc.setFont20 is now time to work with images: we only have function for the images is the doc.addImage. takes the image as the first parameter, image/type format as the second and x, y positions of the image as the third and fourth argument respectively. here we can also optionally pass the new image size as a fifth and sixth arguments. var img = new image;)(img.addEventListener('load', function)(var doc = new jspdf;)(doc.add image (img, png, 10, 50;) }); img.src = 'image_path/image_name.png'; in the previous example, we passed an image html dom element as the first topic of the addimage function, however it can also be a base64 encoded image string. var imgdata = '....base64code....iiigAoooo//2 Q=='; var doc = new jspdf;)(doc.setFontSize(40); doc.text(35, 25, "octonyan loves jspdf";)JPEG, 15, 40, 180, 180); Work with graphics: First of all, we have to set the drawing shapes fill and strokes with colors. We do this using the Doc.setFillColor and the Doc.setDrawColor accordingly, passing the color values RGB as parameters. doc.setFillColor(100, 100, 240); doc.setDrawColor(100, 100, 0); We can also set the width of the stretch. The amplitude unit of the tract is the same as the document manufacturer. doc.setLineWidth(1); Each shape design function takes the central point coordinates (the triangle is the only exception) as two first parameters. They also take the last parameter for design style. It can be "S", "F", "DF", "FD" string and meanings are: "stroke", "fill", "stroke and fill", "fill and caress". The last two, of course, differ for the order of drawing operations. For example, we can draw an ellipse, passing two rays : // Empty ellipse doc.ellipse(50, 50, 10, 5); // Filled ellipse doc.ellipse(100, 50, 10, 5, F)); // Circle filled with edges or circle, passing only one ray as parameter: // square vacuum doc.rect(20, 20, 10, 10); // square filled doc.rect(40, 20, 10, 'F'); // square vacuum doc.setDrawColor(255,0,0); doc.rect(60, 20, 10, 10); /80 / square filled with red edges doc.setDrawColor(255,0,0); Doc.20, 10, 10, 'F'); // red square filled with black edges doc.setDrawColor(0); doc.setFillColor(255,0,0); doc.rect(120, 20, 10, 'FD'); or a rounded rectangle, passing is width, height and boundary radius as parameters: // sqm filled with rounded corners doc.roundedRect(50, 150, 10, 10, 3, 3, FD)); // Black well with rounded corners doc.setDrawColor(0); doc.setFillColor(255, 255, 255); doc.roundedRect(140, 20, 10, 10, 3, 'FD'); and a triangle, passing each corner coordinates. // Triangle filled with var doc edges = 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,F' 120, 130, Finally, but not least, we can also draw lines that cross the coordinates of two points. // Line doc.line(50, 250, 100, 250); > > >