Test of preformatted text wrapping

Our Preformatted class can be used for printing simple blocks of code. It respects whitespace and newlines, and will not normally attempt to wrap your code. However, if your individual lines are too long, this can overflow the width of the column and even run off the page. Three optional attributes - maximumLineLength, splitCharacters and newLineCharacter - can be used to do simple wrapping. maximumLineLength will force the text to wrap. Note that this simply counts characters - it takes no account of actual width on the page. The examples below wrap lines above a certain length and add a '>' to the start of the following line.

```
#Copyright ReportLab Europe Ltd. 2000-2017
#see license.txt for license details
#history https://hg.reportlab.com/hg-public/reportlab/log/
> tip/src/reportlab/platypus/xpreformatted.py
__version__='3.3.0'
__doc__='''A 'rich preformatted text' widget allowing
> internal markup'''

from reportlab.lib import PyFontify
from paragraph import Paragraph, cleanBlockQuotedText,
> _handleBulletWidth, ParaLines, _getFragWords, stringWidth,
>  _sameFrag, getAscentDescent, imgVRange, imgNormV
from flowables import _dedenter

class XPreformatted(Paragraph):
    def __init__(self, text, style, bulletText = None,
> frags=None, caseSensitive=1, dedent=0):
        self.caseSensitive = caseSensitive
        cleaner = lambda text, dedent=dedent: ,'\n'.join(
> _dedenter(text or '',dedent))
        self._setup(text, style, bulletText, frags, cleaner)

    def breakLines(self, width):
        if isinstance(width,list): maxWidths = [width]
        else: maxWidths = width
        lines = []
        lineno = 0
        maxWidth = maxWidths[lineno]
        style = self.style
        fFontSize = float(style.fontSize)
        requiredWidth = 0

        #for bullets, work out width and ensure we wrap the
> right amount onto line one
        _handleBulletWidth(self.bulletText,style,maxWidths)

        self.height = 0
        autoLeading = getattr(self,'autoLeading',getattr(
> style,'autoLeading',''))
        calcBounds = autoLeading not in ('','off')
        frags = self.frags
        nFrags= len(frags)
        if nFrags==1:
            f = frags[0]
            if hasattr(f,'text'):
                fontSize = f.fontSize
                fontName = f.fontName
                ascent, descent = getAscentDescent(fontName,
> fontSize)
                kind = 0
                L=f.text.split('\n')
                for l in L:
                    currentWidth = stringWidth(l,fontName,
> fontSize)
                    requiredWidth = max(currentWidth,
> requiredWidth)
                    extraSpace = maxWidth-currentWidth
                    lines.append((extraSpace,l.split(' '),
> currentWidth))
                    lineno = lineno+1
                    maxWidth = lineno&lt;len(maxWidths) and
> maxWidths[lineno] or maxWidths[-1]
                blPara = f.clone(kind=kind, lines=lines,
> ascent=ascent,descent=descent,fontSize=fontSize)
            else:
                kind = f.kind
                lines = f.lines
                for L in lines:
                    if kind==0:
                        currentWidth = L[2]
                    else:
                        currentWidth = L.currentWidth
                    requiredWidth = max(currentWidth,
> requiredWidth)
                blPara = f.clone(kind=kind, lines=lines)

            self.width = max(self.width,requiredWidth)
            return blPara
        elif nFrags&lt;=0:
            return ParaLines(kind=0, fontSize=style.
> fontSize, fontName=style.fontName,
                            textColor=style.textColor,
> ascent=style.fontSize,descent=-0.2*style.fontSize,
                            lines=[])
        else:
            for L in _getFragLines(frags):
                currentWidth, n, w = _getFragWord(L,
> maxWidth)
                f = w[0][0]
                maxSize = f.fontSize
                maxAscent, minDescent = getAscentDescent(f.
> fontName,maxSize)
                words = [f.clone()]
                words[-1].text = w[0][1]
                for i in w[1:]:
                    f = i[0].clone()
                    f.text=i[1]
                    words.append(f)
                    fontSize = f.fontSize
                    fontName = f.fontName
                    if calcBounds:
                        cbDefn = getattr(f,'cbDefn',None)
                        if getattr(cbDefn,'width',0):
                            descent,ascent = imgVRange(
> imgNormV(cbDefn.height,fontSize),cbDefn.valign,fontSize)
                        else:
                            ascent, descent =
> getAscentDescent(fontName,fontSize)
                    else:
                        ascent, descent = getAscentDescent(
> fontName,fontSize)
                    maxSize = max(maxSize,fontSize)
                    maxAscent = max(maxAscent,ascent)
                    minDescent = min(minDescent,descent)

                lineno += 1
                maxWidth = lineno&lt;len(maxWidths) and
> maxWidths[lineno] or maxWidths[-1]
                requiredWidth = max(currentWidth,
> requiredWidth)
                extraSpace = maxWidth - currentWidth
                lines.append(ParaLines(
> extraSpace=extraSpace,wordCount=n, words=words,
> fontSize=maxSize, ascent=maxAscent,descent=minDescent,
> currentWidth=currentWidth))

            self.width = max(self.width,requiredWidth)
            return ParaLines(kind=1, lines=lines)

        return lines
```