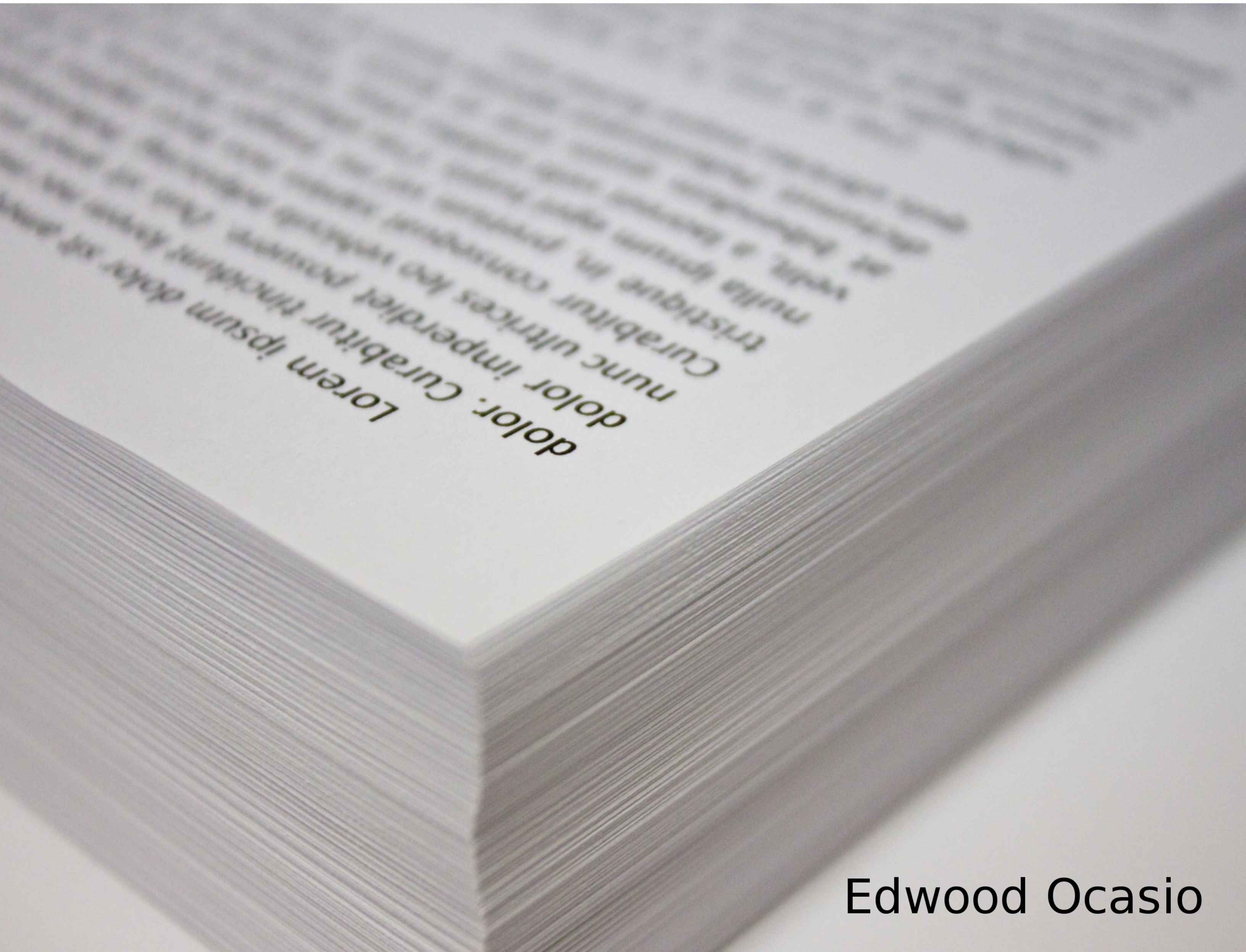
Python does PDF pyFPDF

code and unsolicited advice



Edwood Ocasio

Python does PDF: pyFPDF

Edwood Ocasio

This book is for sale at http://leanpub.com/pythondoespdfpyfpdf

This version was published on 2016-04-09

Contents

Introduction

Through this book I hope to share my hard earned knowledge about generating PDF documents dynamically using Python and libraries like `pyFPDF`, the main subject of this book.

The content will keep expanding with topics such as: tables, images, filling already digitized forms, use of document templates, tips and workarounds while using `pyFPDF`.

If you are a busy professional developer like I am you do not want to spend too much time reading through lots of explanations. You want to get to the solution to your problem fast. That is why this book is written in cookbook style, divided in some common themes. Most of the explaining will be done in the code's comments.

This book can help you add more value to client's projects that require generating PDF reports and, hopefully, that will mean more time and money for you.

All examples in this book will work for *Python 2.7x*, *pyfpdf 1.7.2* and *PIL 1.1.7* on Linux. The code was not tested on Windows or Mac OS. Feedback is welcome on code performance on those platforms.

> **IMPORTANT**: When you buy the book look here ("Introduction") for the link to download a ZIP file with all the code samples and resources.

Installation

PyFPDF[1] is a library for PDF document generation under Python, ported from PHP by Mariano Reingart[2].

To install this library you can use `pip`:

```
sudo pip install pyfpdf
```

The Python Imaging Library (PIL)[3] is only needed for GIF support. PNG and JPG support is built-in and does not require any external dependency. If you need to install it you can also use `pip`:

```
sudo pip install pil
```

If you have issues with the installation of any of these packages, please refer to their respective community forums.

[1] https://github.com/reingart/pyfpdf
[2] https://github.com/reingart
[3] http://www.pythonware.com/products/pil/

Writing text

This chapter presents examples dealing with writing, formatting and positioning text.

Every piece of text in *pyfpdf* is contained in a **cell**. You can think of a cell as a rectangular container or box. You can have every word on a sentence inside its own cell or have the whole sentence inside a cell.

Hello World!

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
pdf=FPDF()
# Add new page. Without this you cannot create the document.
pdf.add_page()
# Set font to Arial, 'B'old, 16 pts.  If you don't declare your font
# parameters you may get some font related errors.
pdf.set_font('Arial','B',16)
# Put a cell (box) 40 units wide and 10 units tall,
# beginning from the left margin, and write 'Hello World!' inside it.
pdf.cell(40,10,'Hello World!')
# Output content into a file ('F') named 'hello.pdf'
pdf.output('hello.pdf','F')
```

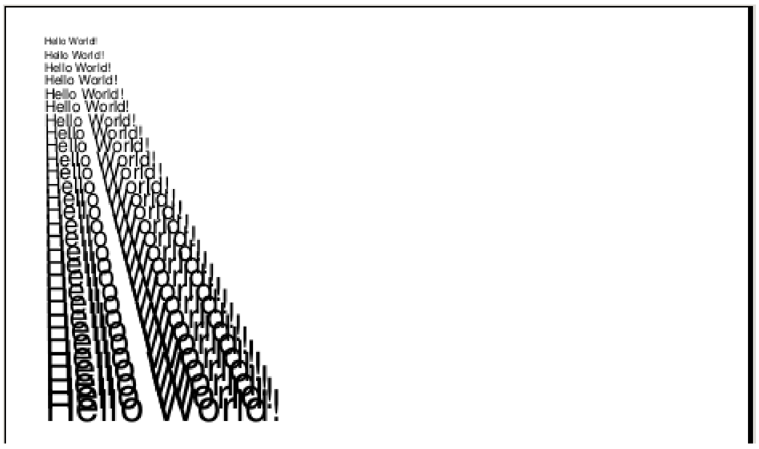
When you open ‘hello.pdf’ you will see this:

Hello World!

Change paper format to Letter and units to inches

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')
# Add new page. Without this you cannot create the document.
pdf.add_page()
# Loop through some font sizes
for fs in range(8, 37):
    # Regular font, notice the empty string ''
    pdf.set_font('Arial','',fs)

    # Create a cell 1.0 inches wide, automatic height
    # and write some text
    pdf.cell(1.0,0.0,'Hello World!')

    # Insert break 0.15 inches in height
    # Just like a carriage return
    pdf.ln(0.15)

# output content into a file ('F') named 'hello2.pdf'
pdf.output('hello2.pdf','F')
```

When you open 'hello2.pdf' you will see this:



Text color

The method `set_text_color()` allows us to color text if we provide the color in an RGB triplet (RED, GREEN, BLUE), like pure green (0, 255, 0).

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')
# Add new page. Without this you cannot create the document.
pdf.add_page()
# Set font face to Times, size 10.0 pt
pdf.set_font('Times','',10.0)

# Set color red
pdf.set_text_color(255,0,0)
pdf.cell(1.0,0.0,'Hello World!')
# Line break 0.15 inches height
pdf.ln(0.15)

# Set color green
pdf.set_text_color(0,255,0)
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.15)

# Set color blue
pdf.set_text_color(0,0,255)
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.15)

# output content into a file ('F') named 'hello3.pdf'
pdf.output('hello3.pdf','F')
```

When you open 'hello3.pdf' you will see this:

Hello World!
Hello World!
Hello World!

For a reference of RGB triplets and their color names (and a color picker) see Rapidtables RGB Color Codes Chart[4]

Text color by color hex values using `hex2dec` function

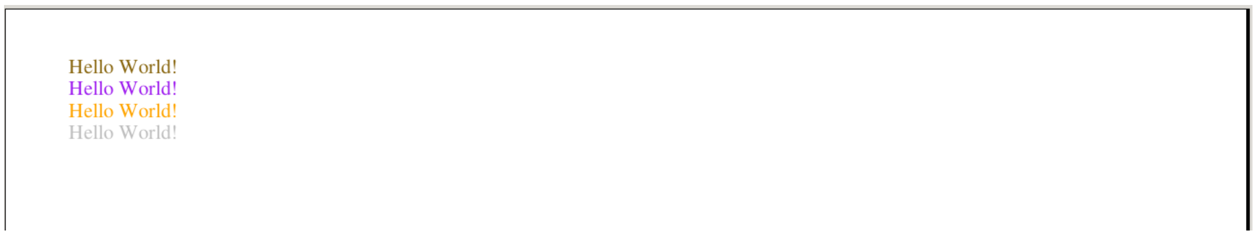
`pyFPDF` comes with an HTML module that helps turning HTML to PDF. One of its utility functions is `hex2dec()` which converts colors hex values to the RGB triplets we need for `set_text_color()`. The following code shows how to leverage that function.

```python
# Import FPDF class
from fpdf import FPDF

# IMPORTANT: Import html module inside fpdf package and import hex2dec
from fpdf.html import hex2dec

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# IMPORTANT: Notice the use of the operator '*' to unpack the triplet
pdf.set_text_color(*hex2dec('#8B6914'))
pdf.cell(0.5,0.0, u'Hello World!')
pdf.ln(0.15)

```

[4]http://www.rapidtables.com/web/color/RGB_Color.htm

```
pdf.set_text_color(*hex2dec('#A020F0'))
pdf.cell(0.5,0.0, u'Hello World!')
pdf.ln(0.15)

pdf.set_text_color(*hex2dec('#FFA500'))
pdf.cell(0.5,0.0, u'Hello World!')
pdf.ln(0.15)

pdf.set_text_color(*hex2dec('#BFBFBF'))
pdf.cell(0.5,0.0, u'Hello World!')
pdf.ln(0.15)

pdf.output('colors-hex2dec.pdf','F')
```

When you open ‘colors-hex2dec.pdf’ you will see this:



```
Hello World!
Hello World!
Hello World!
Hello World!
```

For a reference of color hex values (and a color picker) see Rapidtables Web Color Codes[5]

Text color by color names using `webcolors` module

For a Python module that can change color names to RGB triplets see webcolors[6]

`webcolors` can be installed by issuing the command:

```
pip install webcolors
```

The following code uses the module to facilitate declaring colors by their name, like ‘navy’.

[5]http://www.rapidtables.com/web/color/Web_Color.htm
[6]https://pypi.python.org/pypi/webcolors/1.3

```python
# Import FPDF class
from fpdf import FPDF
from webcolors.webcolors import name_to_rgb

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')
# Add new page. Without this you cannot create the document.
pdf.add_page()
# Set font face to Times, size 10.0 pt
pdf.set_font('Times','',10.0)

# Set color navy
pdf.set_text_color(*name_to_rgb('navy'))
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.15)

# Set color dark olive green
pdf.set_text_color(*name_to_rgb('darkolivegreen'))
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.15)

# Set color salmon
pdf.set_text_color(*name_to_rgb('salmon'))
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.15)

pdf.output('hello4.pdf','F')
```

Notice the operator '*' used before `name_to_rgb` to unpack the triplet.

When you open 'hello4.pdf' you will see this:

```
Hello World!
Hello World!
Hello World!
```

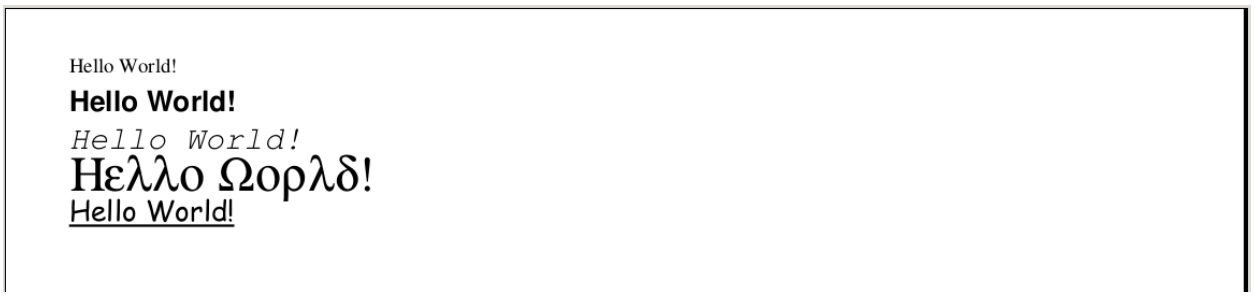
Change font family and style

Using the method `pdf.set_font(font_family,font_style,font_size)` we can change font attributes at any moment through out the document.

```
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')
# Add new page. Without this you cannot create the document.
pdf.add_page()

# Set font family to Times, regular style, size 10.0 pt
pdf.set_font('Times','',10.0)
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.25)

# Set font family to Arial, 'B'old, size 14.0 pt
pdf.set_font('Arial','B',14.0)
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.25)

# Set font family to Courier, 'I'talic, size 16.0 pt
pdf.set_font('Courier','I',16.0)
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.25)

# Set font family to Symbol, regular text, size 24.0 pt
pdf.set_font('Symbol','',24.0)
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.25)

# Add font from system. Font file path must be specified if it is not
# in an accesible path to pyFPDF.
# Second parameter is always empty for backward compatibility.
# uni=True enables Unicode
pdf.add_font('Comic Sans','','/usr/share/fonts/truetype/msttcorefonts/Comic_Sans\
_MS.ttf', uni=True)

# Set font family to Comic Sans, 'U'nderlined, size 14.0 pt
pdf.set_font('Comic Sans','U',14.0)
pdf.cell(1.0,0.0,'Hello World!')
pdf.ln(0.25)

# output content into a file ('F') named 'hello5.pdf'
```

```
pdf.output('hello5.pdf','F')
```

When you open 'hello5.pdf' you will see this:

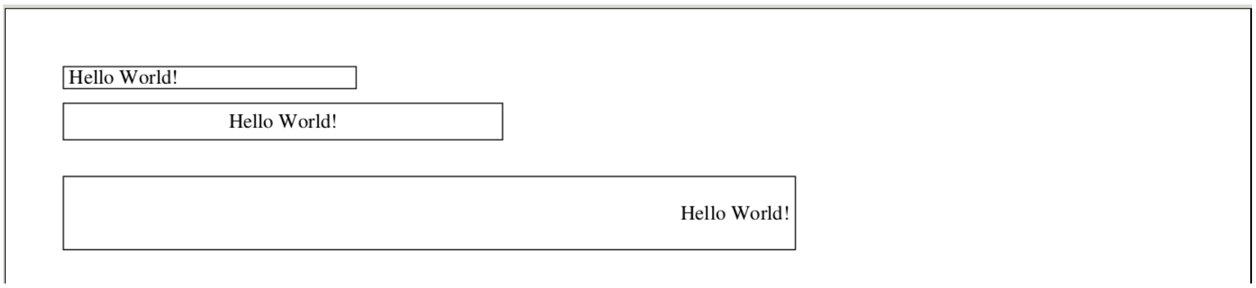


Hello World!
Hello World!
Hello World!
Ηελλο Ωορλδ!
Hello World!

Text alignment and borders

```
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# Text will be left aligned in cell. This is the default alignment.
# Cell is 2.0 inches wide, will have a border and that is why we must now
# define a height of 0.15 inches
pdf.cell(2.0,0.15,'Hello World!', border=1)
pdf.ln(0.25)

# Text will be centered in cell.
# Cell is 3.0 inches wide, will have a border and that is why we must now
# define a height of 0.25 inches
pdf.cell(3.0,0.25,'Hello World!', border=1, align='C')
pdf.ln(0.50)

# Text will be right aligned in cell.
# Cell is 5.0 inches wide, will have a border and that is why we must now
# define a height of 0.50 inches
pdf.cell(5.0,0.50,'Hello World!', border=1, align='R')
pdf.ln(0.25)

# output content into a file ('F') named 'hello6.pdf'
pdf.output('hello6.pdf','F')
```

When you open 'hello6.pdf' you will see this:



```
Hello World!
Hello World!
Hello World!
```

Positioning text cells (boxes)

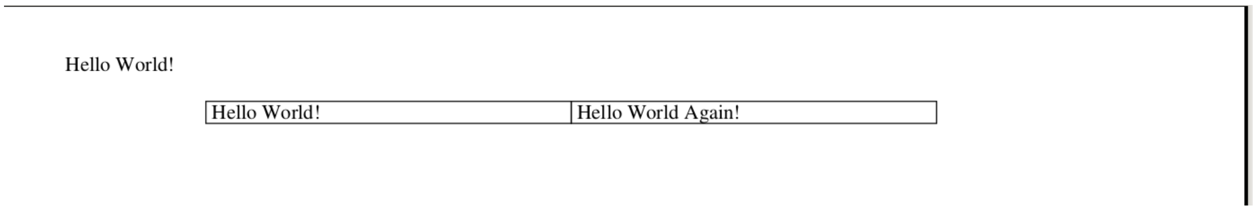
To get the hang of pyFPDF text layout you must keep in mind that everything is a cell or box that occupies an area of the document. You can put those boxes at precise coordinates or let pyFPDF automatically place everything for you. For a simple one column document, like an essay or sales letter, the automatic placement will be enough. For multi-column documents or tables, you have to take charge.

Using empty text cells to place other text cells

```
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# Text in cell 2.5 inches wide, automatic height (0.0)
```

```python
# starting at left margin
pdf.cell(2.5,0.0,'Hello World!')

# Remember this? 0.25 inches line break
pdf.ln(0.25)

# Move 1.0 inch to the right
pdf.cell(1.0)
# Now write text in cell 2.5 inches wide, 0.15 in height.
# Notice we did not put any line break, so this text should follow
# immediately. Will leave borders to emphasize the cell concept.
pdf.cell(2.5,0.15,'Hello World!', border=1)
# Keep in the same line and write again.
pdf.cell(2.5,0.15,'Hello World Again!', border=1)

# Now a line break
pdf.ln(0.15)

# output content into a file ('F') named 'hello7.pdf'
pdf.output('hello7.pdf','F')
```

When you open ‘hello7.pdf’ you will see this:


Hello World!

Hello World! | Hello World Again!

Using the effective page width to distribute content evenly

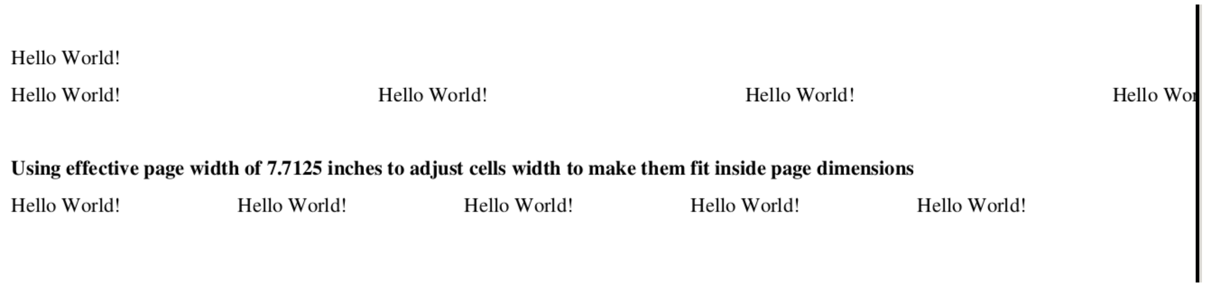
The effective page width is the area between the left and right margin of the document. If those margins are 1.0 inch wide and the paper format is letter (8.5x11 in) with portrait orientation then the effective page width (or epw) is 8.5 - 1.0 - 1.0 = 6.5 inches. That is the area available to write or draw.

Of course we have the entire page surface at our disposition, but it is within that region that users expect the content.

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# Text in cell 2.5 inches wide, automatic height (0.0)
# starting at left margin
pdf.cell(2.5,0.0,'Hello World!')
pdf.ln(0.25)

# Let's put 5 more of this so text goes beyond page margins. Ugly result.
# To avoid this you must always be aware of your page dimensions.
for i in range(5):
    pdf.cell(2.5,0.0,'Hello World!')

pdf.ln(0.50)

# Let us use the page dimensions to better position those cells.
# The full page width is stored in pdf.w and the full height in pdf.h
# We will substract from pdf.w both left and right margins. Here we are
# infering both are the same dimension.

effective_page_width = pdf.w - 2*pdf.l_margin

pdf.set_font('Times','B',10.0)
pdf.cell(1.0, 0.0, 'Using effective page width of %s inches to adjust cells widt\
h to make them fit inside page dimensions' % effective_page_width, 'U')
pdf.ln(0.25)

pdf.set_font('Times','',10.0)

# Using page width we will adjust cell width to make them all fit
# inside the page. The length of the text will determine if this
# will work or not
```

```python

for i in range(5):
    pdf.cell(effective_page_width/5.0,0.0,'Hello World!')


# output content into a file ('F') named 'hello08.pdf'
pdf.output('hello08.pdf','F')
```

When you open 'hello08.pdf' you will see this:



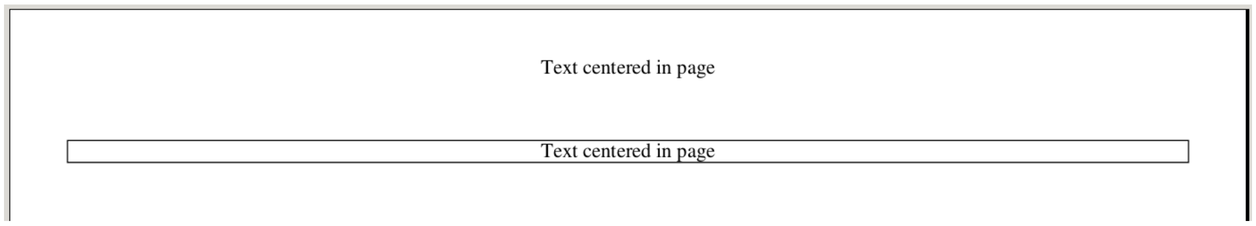
```
Hello World!
Hello World!                 Hello World!                 Hello World!                 Hello Wor

Using effective page width of 7.7125 inches to adjust cells width to make them fit inside page dimensions
Hello World!        Hello World!        Hello World!        Hello World!        Hello World!
```

Using the effective page width to center text across a page

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# One easy way to center text in a page, like a title, is to use a cell as wide
# as the page available width and center the text it contains.
# To obtain the maximum widht available to write we substract the margins
# from the page_width 'pdf.w'. We are assuming left and right margins are of
# the same dimension
effective_page_width = pdf.w - 2*pdf.l_margin

pdf.cell(effective_page_width,0.0, 'Text centered in page', align='C')
pdf.ln(0.50)
```

```

# Now with cell borders to visualize what we are doing.
# Remember to declare the cell height when doing borders.
pdf.cell(effective_page_width,0.15, 'Text centered in page', border=1, align='C')
pdf.ln(0.15)

# output content into a file ('F') named 'hello10.pdf'
pdf.output('hello10.pdf','F')
```

When you open ‘hello10.pdf’ you will see this:


Text centered in page

Text centered in page

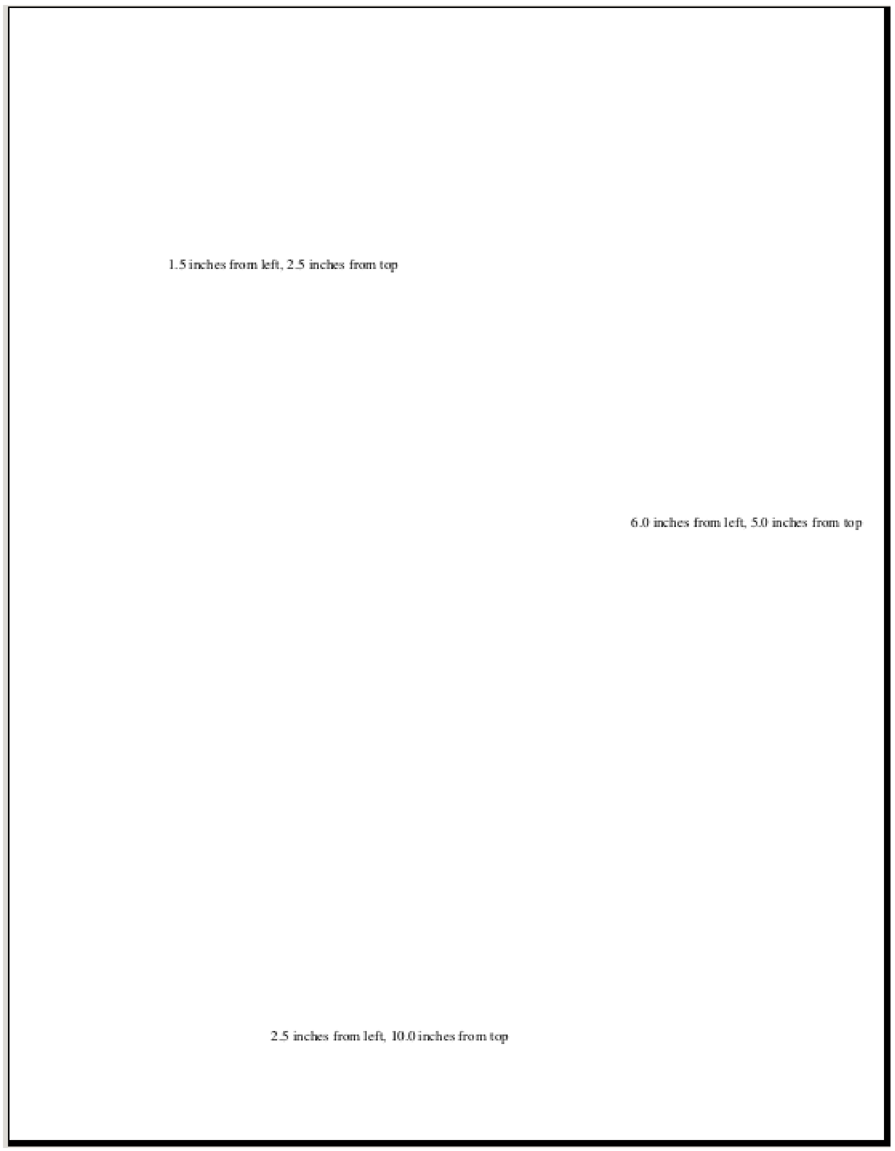
Using method set_xy() to exactly position a text cell

We will also use methods `pdf.get_x()` and `pdf.get_y()` to get current `x` and `y` coordinates respectively.

```
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# Text will be drawn 1.5 inches from the left, 2.5 inches from the top
pdf.set_xy(1.5, 2.5)
pdf.cell(2.5,0.0,'%s inches from left, %s inches from top' % (pdf.get_x(), pdf.g\
et_y()))
pdf.ln(0.25)

```

```
# Text will be drawn 6 inches from the left, 4.0 inches from top
pdf.set_xy(6.0, 5.0)
pdf.cell(2.5,0.0,'%s inches from left, %s inches from top' % (pdf.get_x(), pdf.g\
et_y()))
pdf.ln(0.25)

# Text will be drawn 6.0 inches from the right, 1.0 inch from bottom.
# Notice the negative coordinate values.  They are relative to the right and
# bottom of the page respectively.

pdf.set_xy(-6.0, -1.0)
pdf.cell(2.5,0.0,'%s inches from left, %s inches from top using negative values'\
 % (pdf.get_x(), pdf.get_y()))
pdf.ln(0.25)

# output content into a file ('F') named 'hello9.pdf'
pdf.output('hello9.pdf','F')
```

When you open ‘hello9.pdf’ you will see this:



1.5 inches from left, 2.5 inches from top

6.0 inches from left, 5.0 inches from top

2.5 inches from left, 10.0 inches from top

Tip

Instead of pdf.get_x() and pdf.get_y() we can also use `pdf.x` and `pdf.y`.

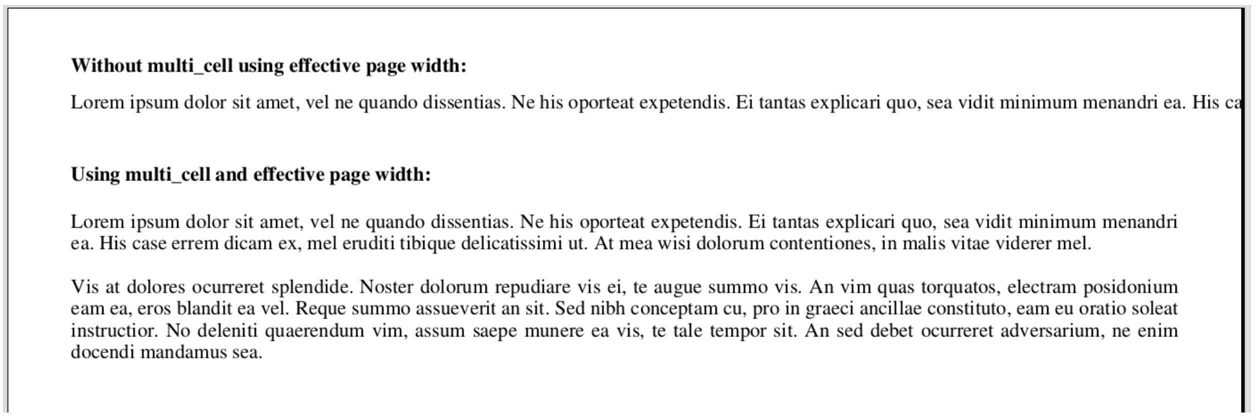
Wrapping text automatically (multi_cell)

The method `pdf.multi_cell()` automatically breaks long lines of text, a paragraph for example, within the available effective page width. We do not have to worry about word wrapping.

Simple use of multi_cell

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# Long meaningless piece of text
loremipsum = """Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his opo\
rteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His ca\
se errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum con\
tentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue su\
mmo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel.\
 Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae co\
nstituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum sa\
epe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne en\
im docendi mandamus sea.
"""

effective_page_width = pdf.w - 2*pdf.l_margin

pdf.set_font('Times','B',10.0)
pdf.cell(1.0,0.0, 'Without multi_cell using effective page width:')
pdf.ln(0.25)
pdf.set_font('Times','',10.0)
# Cell is as wide as the effective page width
pdf.cell(effective_page_width, 0.0, loremipsum)
pdf.ln(0.5)
pdf.set_font('Times','B',10.0)
pdf.cell(1.0,0.0, 'Using multi_cell and effective page width:')
pdf.ln(0.25)

pdf.set_font('Times','',10.0)
# Cell is as wide as the effective page width
# and multi_cell requires declaring the height of the cell.
pdf.multi_cell(effective_page_width, 0.15, loremipsum)
pdf.ln(0.5)

# Cell half as wide as the effective page width
# and multi_cell requires declaring the height of the cell.
pdf.set_font('Times','B',10.0)
pdf.cell(1.0,0.0, 'Using multi_cell and half the effective page width:')
pdf.ln(0.25)

pdf.set_font('Times','',10.0)
pdf.multi_cell(effective_page_width/2, 0.15, loremipsum)
pdf.ln(0.5)

pdf.output('multi_cell.pdf','F')
```

When you open 'multi_cell.pdf' you will see this:



Without multi_cell using effective page width:

Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His ca

Using multi_cell and effective page width:

Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum contentiones, in malis vitae viderer mel.

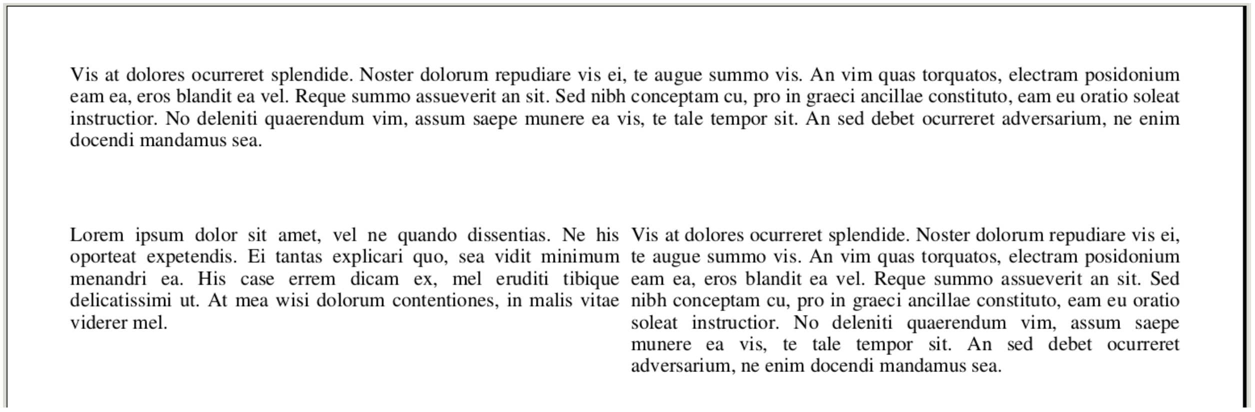
Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

Putting two adjacent multi_cell blocks

There are times when we need to render a multi-column document with multi-line content. Here is an idea on how to do that with multi_cell, although it may require lots of tweaking to make it work for each use case. This example shows again the importance of take into account the dimensions of the page, including its margins.

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# Long meaningless piece of text
loremipsum_1 = """Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his o\
porteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His \
case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum c\
ontentiones, in malis vitae viderer mel.
"""

# Even more meaningless text
loremipsum_2 = """Vis at dolores ocurreret splendide. Noster dolorum repudiare v\
is ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, er\
os blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in \
graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaere\
ndum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret \
adversarium, ne enim docendi mandamus sea.
"""

effective_page_width = pdf.w - 2*pdf.l_margin

# Some text full page width for position reference

pdf.multi_cell(effective_page_width, 0.15, loremipsum_2)
pdf.ln(0.5)

# First save the y coordinate just before rendering the first multi_cell
ybefore = pdf.get_y()
pdf.multi_cell(effective_page_width/2, 0.15, loremipsum_1)

# Now use ybefore to position the cursor at the same level of the first
# multi_cell.
# Notice the use of "effective_page_width/2 + pdf.l_margin" as x to position
# the cursor horizontally just beyond the first multi_cell

pdf.set_xy(effective_page_width/2 + pdf.l_margin, ybefore)
pdf.multi_cell(effective_page_width/2, 0.15, loremipsum_2)
pdf.ln(0.5)

pdf.output('multi_cell_adjacent.pdf','F')
```

When you open 'multi_cell_adjacent.pdf' you will see this:



Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum contentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

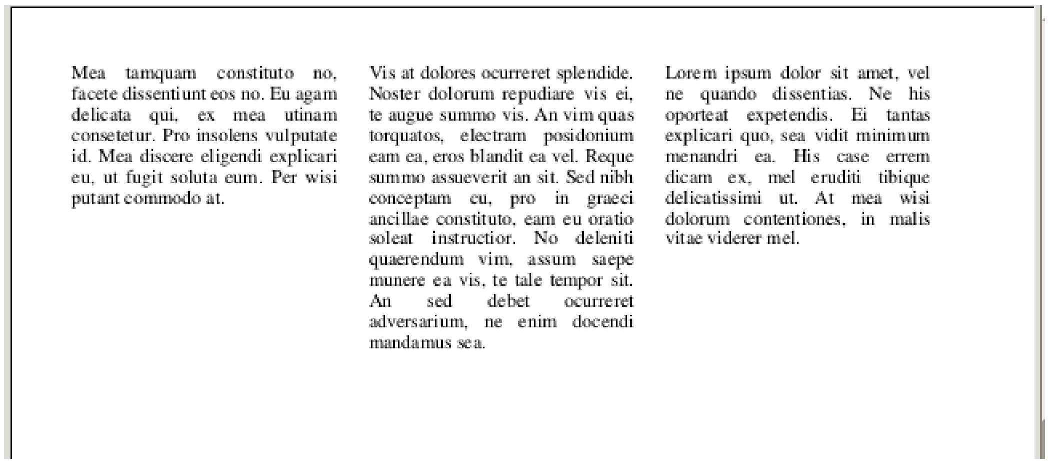
Three adjacent multi cell columns

I am including this snippet here to answer what seems to be one of the most asked question about `multi_cell` in the Internet: *How to put 3 adjacent columns with multi cell.* Here is the code:

```
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

column_width = 2.0
```

```
column_spacing = 0.15

# Here we save what will be the top of each columns
ybefore = pdf.get_y()

pdf.multi_cell(column_width, 0.15, "Mea tamquam constituto no, facete dissentiun\
t eos no. Eu agam delicata qui, ex mea utinam consetetur. Pro insolens vulputate\
 id. Mea discere eligendi explicari eu, ut fugit soluta eum. Per wisi putant com\
modo at.")

# Notice we have to account for the left margin to get the spacing between
# columns right.

pdf.set_xy(column_width + pdf.l_margin + column_spacing, ybefore)

pdf.multi_cell(column_width, 0.15, "Vis at dolores ocurreret splendide. Noster d\
olorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram pos\
idonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh con\
ceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. \
No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An se\
d debet ocurreret adversarium, ne enim docendi mandamus sea.")

pdf.set_xy(2*(column_width + column_spacing) + pdf.l_margin, ybefore)

pdf.multi_cell(column_width, 0.15, "Lorem ipsum dolor sit amet, vel ne quando di\
ssentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum\
 menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At \
mea wisi dolorum contentiones, in malis vitae viderer mel.")

pdf.output('multi_cell_3_cols.pdf','F')
```

When you open ‘multi_cell_3_cols.pdf’ you will see this:



Mea tamquam constituto no, facete dissentiunt eos no. Eu agam delicata qui, ex mea utinam consetetur. Pro insolens vulputate id. Mea discere eligendi explicari eu, ut fugit soluta eum. Per wisi putant commodo at.

Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

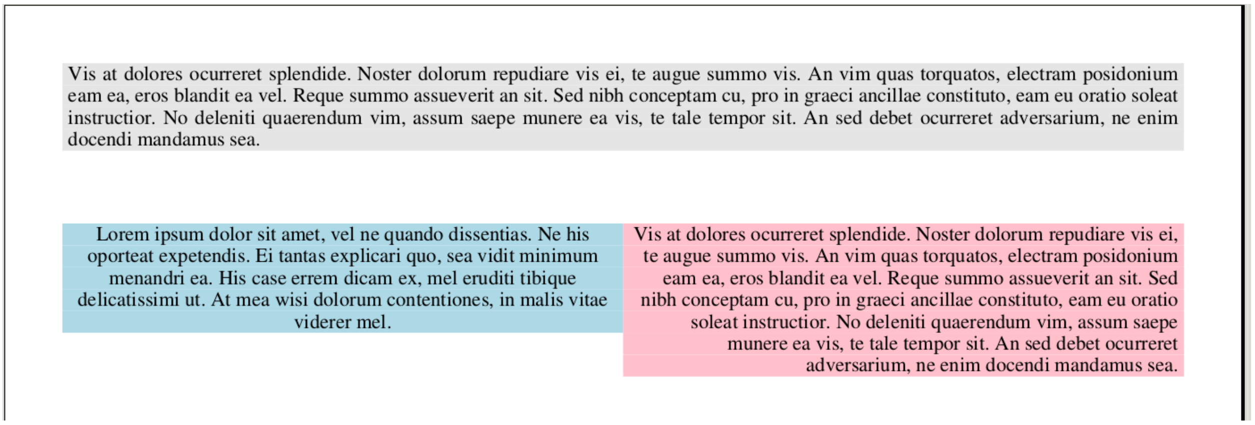
Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum contentiones, in malis vitae viderer mel.

Coloring multi_cell blocks

It is posible to paint the background of a multi_cell block by allowing filling each cell with a background color. However, there is an annoying padding space between cells that cannot (it seems) be removed easily. We will be using the same adjacent multi_cell code as before, but now with some `pdf.set_fill_color()` commands.

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')

# Add new page. Without this you cannot create the document.
pdf.add_page()

# Remember to always put one of these at least once.
pdf.set_font('Times','',10.0)

# Long meaningless piece of text
loremipsum_1 = """Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his o\
porteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His \
case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum c\
ontentiones, in malis vitae viderer mel.
"""

loremipsum_2 = """Vis at dolores ocurreret splendide. Noster dolorum repudiare v\
is ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, er\
os blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in \
graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaere\
ndum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret \
adversarium, ne enim docendi mandamus sea.
"""

effective_page_width = pdf.w - 2*pdf.l_margin

# Set background color light gray, text 'J'ustified and allow filling
# the cell (fill=1)
pdf.set_fill_color(229, 229, 229)
pdf.multi_cell(effective_page_width, 0.15, loremipsum_2, fill=1, align='J')
pdf.ln(0.5)

# First save the y coordinate just before rendering the first multi_cell
ybefore = pdf.get_y()

# Set background color some light blue, centered text and allow filling
# the cell (fill=1)
pdf.set_fill_color(173, 216, 230)
pdf.multi_cell(effective_page_width/2, 0.15, loremipsum_1, fill=1, align='C')

# Set background color some light red, right justified and allow filling
# the cell (fill=1)
pdf.set_fill_color(255, 192, 203)

pdf.set_xy(effective_page_width/2 + pdf.l_margin, ybefore)
pdf.multi_cell(effective_page_width/2, 0.15, loremipsum_2, fill=1, align='R')
pdf.ln(0.5)

pdf.output('multi_cell_adjacent_colored.pdf','F')
```

When you open ‘multi_cell_adjacent_colored.pdf’ you will see this:



Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum contentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

Adjust cell width and height exactly to the text

Up until now we have been guessing how wide or tall our cells should be to contain their text. About 0.15 inches tall seems to be enough for a 10 pt font size. But how to know for sure?

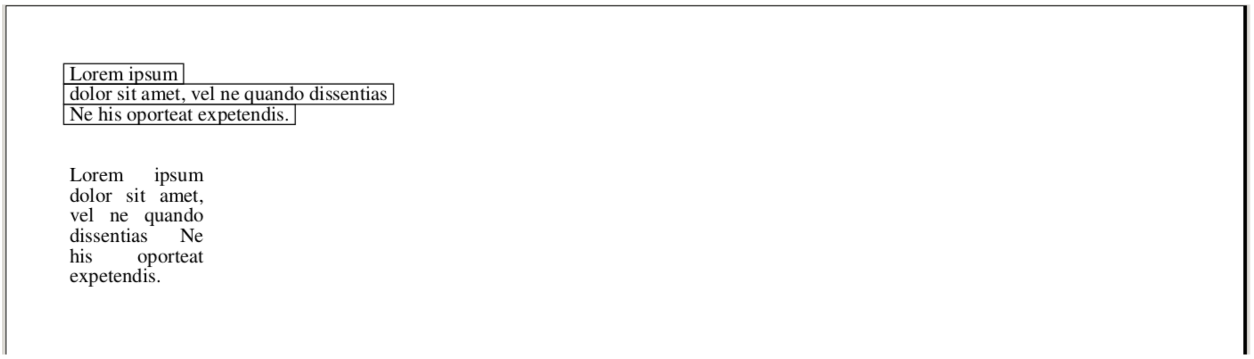
The text width can be computed exactly using the `get_string_width()` method. This method takes into account the font parameters to get the full width of the text when rendered.

However, there is not an equivalent function to compute the text height, because no computation is needed. Text height is the same as the current font size.

The following code shows how to take advantage of these facts to calculate exactly the cell dimensions needed to accomodate any text, including in multi-cell mode. Also you will see how we use text height for our line breaks.

```python
# Import FPDF class
from fpdf import FPDF

# Letter size paper, use inches as unit of measure
pdf=FPDF(format='letter', unit='in')
pdf.add_page()
pdf.set_font('Times','',10.0)

# Text to be rendered
phrases = ['Lorem ipsum', 'dolor sit amet, vel ne quando dissentias', \
'Ne his oporteat expetendis.' ]

paragraph = " ".join(phrases)

# The height of the rendered text is just the current font size
# which is stored internally as the ratio (font in points)/72.
# In this case 10/72 or about 0.14 in.
th = pdf.font_size

for phrase in phrases:
    # The width of the rendered text has yo account for the internal
    # cell left and right margins (each 1 mm by default)
    tw = pdf.get_string_width(phrase) + 2*pdf.c_margin

    # Will draw cell borders to show how good the fit is.
    pdf.cell(tw, th, phrase, border=1)

    # Let's use the font height as the line break height.
    pdf.ln(th)

# 2 line breaks as tall as the fon size
pdf.ln(2*th)

# Now in multi-cell mode using same height as before

pdf.multi_cell(1.0, th, paragraph)

pdf.output('cell-fits-text.pdf','F')
```

When you open 'cell-fits-text.pdf' you will see this:



```
Lorem ipsum
dolor sit amet, vel ne quando dissentias
Ne his oporteat expetendis.


Lorem     ipsum
dolor  sit  amet,
vel   ne   quando
dissentias      Ne
his        oporteat
expetendis.
```

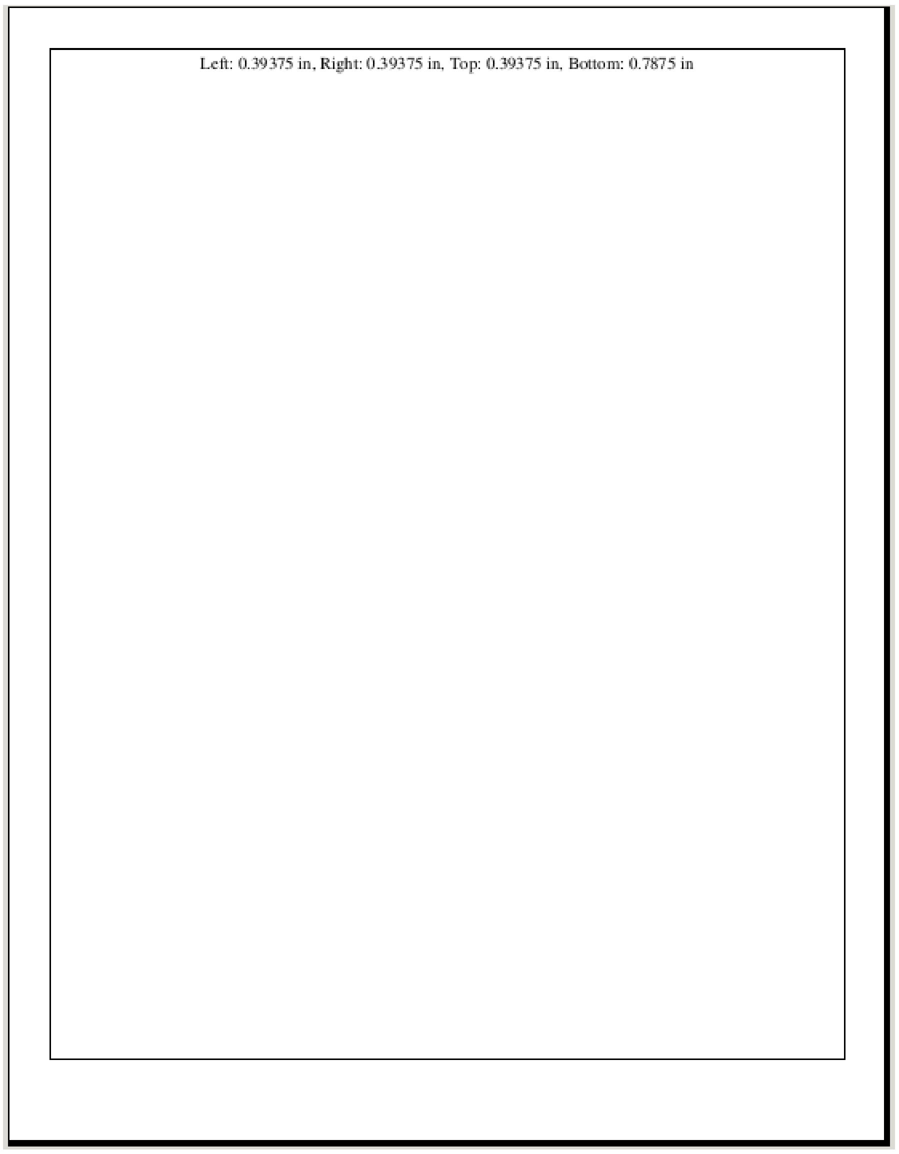
Margins

Default margins

When you create an empty PDF document with `pyFPDF` it has already default margins. The following code demonstrates how to get those margins and how to draw them.

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
pdf=FPDF(format='letter',unit='in')
# Add new page. Without this you cannot create the document.
pdf.add_page()

# Get default margins
left = pdf.l_margin
right = pdf.r_margin
top = pdf.t_margin
bottom = pdf.b_margin

# Effective page width and height
epw = pdf.w - left - right
eph = pdf.h - top - bottom

# Draw margins for our viewing pleasure
pdf.rect(left, top, w=epw, h=eph)

# Draw margin sizes
# Need this line break, otherwise top line crosses text. Have not figure
# out why yet.
pdf.ln(0.15)

# Remember to put at least one of this font declarations when you are going
# to render text.  Not needed if not drawing text.
pdf.set_font('Times','',12)
pdf.cell(0,0, 'Left: %s in, Right: %s in, Top: %s in, Bottom: %s in' % (left, \
right, top, bottom), align= 'C')

pdf.output('margins.pdf','F')

#~ import os
#~ os.system('evince margins.pdf')
```

When you open ‘margins.pdf’ you will see this:


Left: 0.39375 in, Right: 0.39375 in, Top: 0.39375 in, Bottom: 0.7875 in

Changing margins

To change the margins of a page you need to set them **before** the `add_page` method or you won't see the changes.

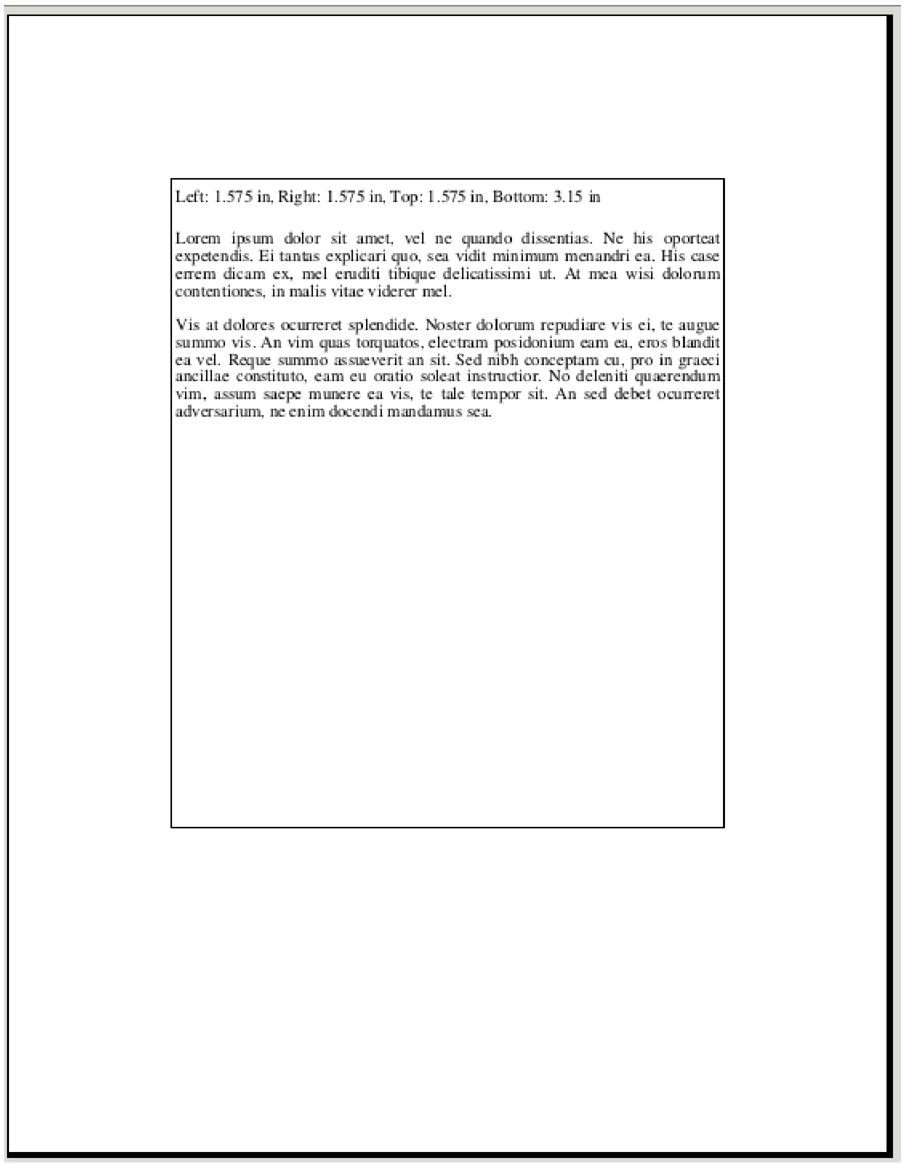
Tip

You can set the margins using different methods: `set_margins()`, `set_left_margin()`, `set_top_margin()`, `set_right_margin()` but I prefer to use the margin variables directly: `l_margin`, `r_margin`, `t_margin`, `b_margin`. Is less verbose, concise and gives me more control.

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
pdf=FPDF(format='letter',unit='in')

# Set margins BEFORE add_page!
# Margin sizes multiples of default size
pdf.l_margin = pdf.l_margin*4.0
pdf.r_margin = pdf.r_margin*4.0
pdf.t_margin = pdf.t_margin*4.0
pdf.b_margin = pdf.b_margin*4.0

# Now we add a page. It is in this step that margins settings take effect.
pdf.add_page()

# Effective page width and height
epw = pdf.w - pdf.l_margin - pdf.r_margin
eph = pdf.h - pdf.t_margin - pdf.b_margin

# Draw new margins.
pdf.rect(pdf.l_margin, pdf.t_margin, w=epw, h=eph)


# Remember to put at least one of this font declarations when you are going
# to render text.  Not needed if not drawing text.
pdf.set_font('Times','',12)

# Text height
th = pdf.font_size

# Draw margin sizes.
# Need this line break, otherwise top line crosses text. Still have not
# figure that one out but it has to do with how text is positioned with
# respect x and y coordinates.

pdf.ln(th)

pdf.cell(epw,0, 'Left: %s in, Right: %s in, Top: %s in, Bottom: %s in' % \
(pdf.l_margin, pdf.r_margin, pdf.t_margin, pdf.b_margin))

pdf.ln(2*th)
```

```

loremipsum = """Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his opo\
rteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His ca\
se errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum con\
tentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue su\
mmo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel.\
 Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae co\
nstituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum sa\
epe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne en\
im docendi mandamus sea.
"""

pdf.multi_cell(epw,th, loremipsum)

pdf.output('margins-change.pdf','F')

#~ import os
#~ os.system('evince margins-change.pdf')
```

When you open ‘margins-change.pdf’ you will see this:



Left: 1.575 in, Right: 1.575 in, Top: 1.575 in, Bottom: 3.15 in

Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum contentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

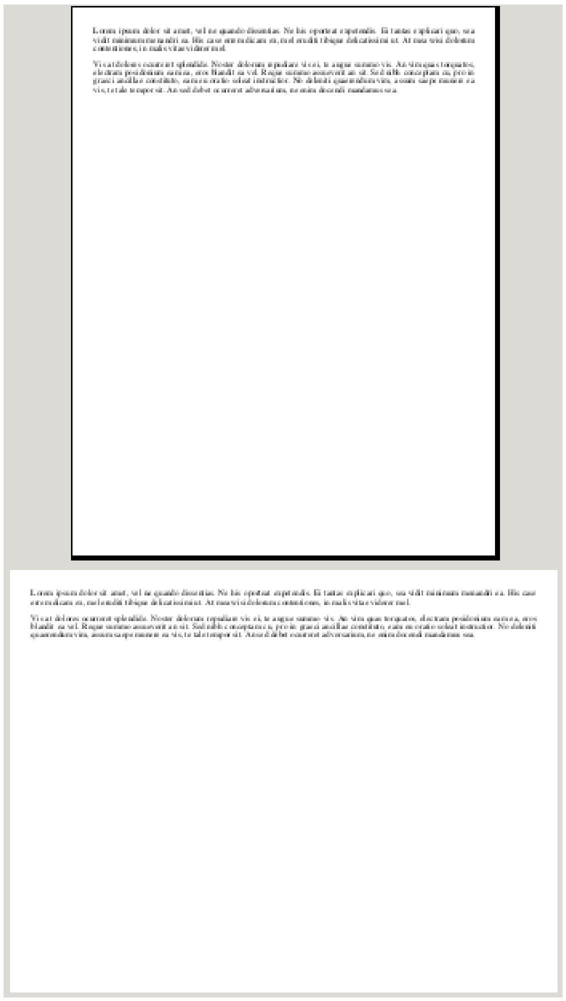
Mix page orientations

Each page in a PDF document can have its own orientation. The trick lies in adding the parameter `orientation` when using the `add_page` method.

```python
# Import FPDF class
from fpdf import FPDF

# Create instance of FPDF class
# You can set the orientation here also. 'P'ortait by default.
pdf=FPDF(format='letter',unit='in')

# Now we add a page. Since no orientation has been set, it is 'P'ortrait
# by default
pdf.add_page()

# Remember to put at least one of this font declarations when you are going
# to render text.  Not needed if not drawing text.
pdf.set_font('Times','',12)

# Text height
th = pdf.font_size

loremipsum = """Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his opo\
rteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His ca\
se errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum con\
tentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorum repudiare vis ei, te augue su\
mmo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel.\
 Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae co\
nstituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum sa\
epe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne en\
im docendi mandamus sea.
"""
# Since w=0 it will go all the way to the right margin.
pdf.multi_cell(0,th, loremipsum)

# Next page will be landscape.

pdf.add_page(orientation='L')

# Same text as before.
# Since w=0 it will go all the way to the right margin.
pdf.multi_cell(0,th, loremipsum)

pdf.output('mix-orientations.pdf','F')
```

```

#~ import os
#~ os.system('evince mix-orientations.pdf')
```

When you open ‘mix-orientations.pdf’ you will see this:



Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum contentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorem repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.

Lorem ipsum dolor sit amet, vel ne quando dissentias. Ne his oporteat expetendis. Ei tantas explicari quo, sea vidit minimum menandri ea. His case errem dicam ex, mel eruditi tibique delicatissimi ut. At mea wisi dolorum contentiones, in malis vitae viderer mel.

Vis at dolores ocurreret splendide. Noster dolorem repudiare vis ei, te augue summo vis. An vim quas torquatos, electram posidonium eam ea, eros blandit ea vel. Reque summo assueverit an sit. Sed nibh conceptam cu, pro in graeci ancillae constituto, eam eu oratio soleat instructior. No deleniti quaerendum vim, assum saepe munere ea vis, te tale tempor sit. An sed debet ocurreret adversarium, ne enim docendi mandamus sea.