PYTHON

Filling PDF Forms with Python

Fillable forms have been a part of Adobe's PDF format for years. One of the most famous examples of fillable forms in the United States are documents from the Internal Revenue Service. There are lots of government forms that use fillable forms. There are many different approaches for filling in these forms programmatically. The most time consuming method I have heard about is to just recreate the form in ReportLab by hand and then fill it in. Frankly I think this is probably the worst idea, except when your company is in charge of creating the PDFs itself. Then that might be a viable option because you then have complete control over the PDF creation and the inputs that need to go into it.

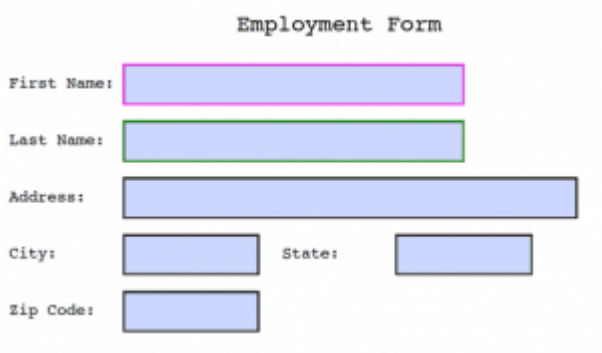
Creating a Simple Form

We need a simple form to use for our first example. ReportLab has built-in support for creating interactive forms, so let's use ReportLab to create a simple form. Here is the code:

```
# simple_form.py

from reportlab.pdfgen import canvas
from reportlab.pdfbase import pdfform
from reportlab.lib.colors import magenta, pink, blue, green

def create_simple_form():
    c = canvas.Canvas('simple_form.pdf')

    c.setFont("Courier", 20)
    c.drawCentredString(300, 700, 'Employment Form')
    c.setFont("Courier", 14)
    form = c.acroForm

    c.drawString(10, 650, 'First Name:')
    form.textfield(name='fname', tooltip='First Name',
                   x=110, y=635, borderStyle='inset',
                   borderColor=magenta, fillColor=pink,
                   width=300,
                   textColor=blue, forceBorder=True)

    c.drawString(10, 600, 'Last Name:')
    form.textfield(name='lname', tooltip='Last Name',
                   x=110, y=585, borderStyle='inset',
```

```
                    borderColor=green, fillColor=magenta,
                    width=300,
                    textColor=blue, forceBorder=True)

    c.drawString(10, 550, 'Address:')
    form.textfield(name='address', tooltip='Address',
                    x=110, y=535, borderStyle='inset',
                    width=400, forceBorder=True)

    c.drawString(10, 500, 'City:')
    form.textfield(name='city', tooltip='City',
                    x=110, y=485, borderStyle='inset',
                    forceBorder=True)

    c.drawString(250, 500, 'State:')
    form.textfield(name='state', tooltip='State',
                    x=350, y=485, borderStyle='inset',
<strong>                    forceBorder=True)

    c.drawString(10, 450, 'Zip Code:')
    form.textfield(name='zip_code', tooltip='Zip Code',
                    x=110, y=435, borderStyle='inset',
                    forceBorder=True)

    c.save()

if __name__ == '__main__':
    create_simple_form()
</strong>
```

When you run this example, the interactive PDF form looks like this:



Now we are ready to learn one of the ways that we can fill in this form!

Merging Overlays

Jan ChÄ™Ä‡ wrote an article on Medium that contained several different approaches to this problem of filling in forms in PDFs. The first solution proposed was to take an unfilled form in a PDF and create a separate PDF using ReportLab that has the data we want to us to "fill" this form. The author then used pdfrw to merge the two PDFs together. You could theoretically use PyPDF2 for the merging process too. Let's go ahead and take a look at how this approach might work using the pdfrw package.

Let's get started by installing pdfrw:

Let's get started by installing pdfrw:

```
python -m pip install pdfrw
```

Now that we have those installed, let's create a file called **fill_by_overlay.py**. We will add two functions to this file. The first function will create our overlay. Let's check that out:

```
# fill_by_overlay.py

import pdfrw
from reportlab.pdfgen import canvas


def create_overlay():
    """
    Create the data that will be overlayed on top
    of the form that we want to fill
    """
    c = canvas.Canvas('simple_form_overlay.pdf')

    c.drawString(115, 650, 'Mike')
    c.drawString(115, 600, 'Driscoll')
    c.drawString(115, 550, '123 Greenway Road')
    c.drawString(115, 500, 'Everytown')
    c.drawString(355, 500, 'IA')
    c.drawString(115, 450, '55555')

    c.save()
```

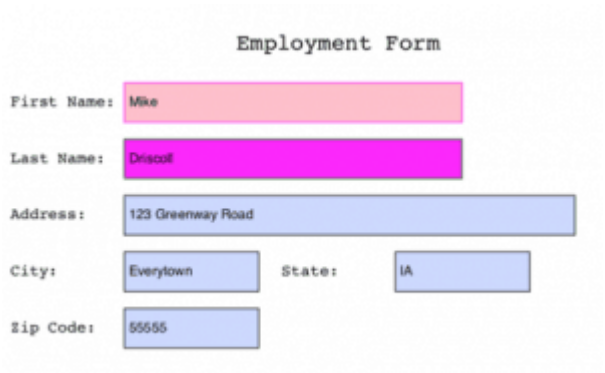
Here we import the **pdfrw** package and we also import the **canvas** sub-module from ReportLab. Then we create a function called **create_overlay** that creates a simple PDF using ReportLab's **Canvas** class. We just use the **drawString** canvas method. This will take some trial-and-error. Fortunately on Linux and Mac, there are decent PDF Previewer applications that you can use to just keep the PDF open and they will automatically refresh with each change. This is very helpful in figuring out the exact coordinates you need to draw your strings to. Since we created the original form, figuring out the offset for the overlay is actually pretty easy. We already knew where on the page the form elements were, so we can make a good educated guess of where to draw the strings to.

The next piece of the puzzle is actually merging the overlay we created above with the form we created in the previous section. Let's write that function next:

```
def merge_pdfs(form_pdf, overlay_pdf, output):
    """
    Merge the specified fillable form PDF with the
    overlay PDF and save the output
    """
    form = pdfrw.PdfReader(form_pdf)
    olay = pdfrw.PdfReader(overlay_pdf)

    for form page, overlay page in zip(form.pages, olay.pages):
```

```
            merge_obj = pdfrw.PageMerge()
            overlay = merge_obj.add(overlay_page)[0]
            pdfrw.PageMerge(form_page).add(overlay).render()

        writer = pdfrw.PdfWriter()
        writer.write(output, form)


if __name__ == '__main__':
    create_overlay()
    merge_pdfs('simple_form.pdf',
               'simple_form_overlay.pdf',
               'merged_form.pdf')
```

Here we open up both the form and the overlay PDFs using pdfrw's **PdfReader** classes. Then we loop over the pages of both PDFs and merge them together using **PageMerge**. At the end of the code, we create an instance of **PdfWriter** that we use to write the newly merged PDF out. The end result should look like this:



Note: When I ran this code, I did receive some errors on stdout. Here's an example:

```
[ERROR] tokens.py:226 stream /Length attribute (171) appears to be too
small (size 470) -- adjusting (line=192, col=1)
```

As I mentioned, this doesn't actually prevent the merged PDF from being created. But you might want to keep an eye on these as they might hint at a problem should you have any issues.

Other Ways to Fill Forms

I have read about several other ways to "fill" the fields in these kinds of PDFs. One of them was to take a PDF and save the pages as a series of images. Then draw rectangles at the locations you want to add text and then use your new image as a config file for filling out the PDF. Seems kind of wacky and frankly I don't want to go to all that work.

A better method would be to open a PDF in a PDF editor where you can add invisible read-only fields. You can label the fields with unique names

add invisible read-only fields. You can label the fields with unique names and then access them via the PDF's metadata. Loop over the metadata and use ReportLab's canvas methods to create an overlay again and then merge it in much the same way as before.

I have also seen a lot of people talking about using Forms Data Format or FDF. This is the format that PDFs are supposed to use to hold that data that is to be filled in a PDF. You can use **PyPDFtk** and **PdfJinja** to do the form filling. Interestingly, **PyPDFtk** doesn't work with image fields, such as where you might want to paste a signature image. You can use **PdfJinja** for this purpose. However **PdfJinja** seems to have some limitations when working with checkboxes and radioboxes.

You can read more about these topics at the following links:

- https://yoongkang.com/blog/pdf-forms-with-python/
- https://medium.com/@zwinny/filling-pdf-forms-in-python-the-right-way-eb9592e03dba

---

Using the pdfforms Package

The package that I think holds the most promise in regards to simplicity to use is the new **pdfforms** package. It requires that you install a cross-platform application called **pdftk** though. Fortunately pdftk is free so that's not really a problem.

You can install pdfforms using pip like this:

```
1. python -m pip install pdfforms
```

To use pdfforms, you must first have it **inspect** the PDF that contains a form so it knows how to fill it out. You can do the inspection like this:

```
1. pdfforms inspect simple_form.pdf
```

If pdfforms works correctly, it will create a "filled" PDF in its "test" sub-folder. This sub-folder appears next to where pdfforms itself is, not where you run it from. It will fill the form with numbers in a sequential order. These are the **field numbers**.

The next thing you do is create a CSV file where the first column and row contains the name of the PDF. The other rows in the first column correspond to the field numbers. You enter the numbers of the fields that you want to fill here. Then you enter the data you want to fill use in the form in the third column of your CSV file. The second column is ignored, so

you can put a description here. All columns after the third column are also ignored, so these can be used for whatever you want.

For this example, your CSV file might look something like this:

```
simple_form.pdf,,,
1,first name,Mike
2,last name,Driscoll
```

Once you have the CSV filled out, you can run the following command to actually fill your form out with your custom data:

```
pdfforms fill data.csv
```

The filled PDF will appear in a sub-folder called **filled** by default.

Now on to the bad news. I wasn't able to get this to work correctly on Windows or Mac. I got the **inspect** step to work on Windows, but on Mac it just hangs. On Windows, when I run the **fill** command it just fails with an error about not finding the PDF to fill.

I think when this package becomes less error-prone, it will be really amazing. The only major downside other than it having issues running is that you need to install a 3rd party tool that isn't written in Python at all.

Wrapping Up

After looking at the many different options available to the Python developer for filling PDF forms, I think the most straight-forward method is creating the overlay and then merging it to the fillable form PDF using a tool like pdfrw. While this feels a bit like a hack, the other methods that I have seen seem just as hacky and just as time consuming. Once you have the position of one of the cells in the form, you can reasonably calculate the majority of the others on the page.

Additional Reading

- The pdfforms package on PyPI
- The pdfrw package on Github
- The PdfJinja package on Github
- PDF Forms with Python blog post

Similar Posts

Similar Posts

What's New in Python 3.11 (Video)

In this video, Mike Driscoll talks about what to expect [...]

Data Science Packages in Python (Video)

In this tutorial, I will talk about some of the [...]

PyDev of the Week: Allen Downey

This week we welcome Allen Downey (@AllenDowney) as the PyDev [...]

An Intro to Python Web Frameworks (Video)

Learn about some of the many Python web frameworks that [...]

An Intro to the contextlib Module in Python (Video)

Learn how to create different types of context managers using [...]

An Intro to Context Managers in Python (Video)

Context managers are a handy way to open and close [...]

PyDev of the Week: Denny Perez

This week we welcome Denny Perez (@dennyperez18) as our PyDev [...]

Python Video Series: The builtins module

In this video tutorial, you will learn about Python's builtins [...]

Python's Calendar Module (video)

Learn the basics of Python's amazing calendar module in this [...]

Function Overloading with Python (Video)

In this tutorial, you will learn how to do function [...]

Recent Posts

Links

@driscollis

@driscolls

Buy me a Coffee

MouseVsPython on Twitter

MouseVsPython Youtube Channel