

Examples

0. Getting started

0.1 Installation using pip

Getting started with pText is easy.

1. Create a virtual environment (if you have not done so already)

```
python3 -m venv venv
```

2. Activate your virtual environment

```
source venv/bin/activate
```

3. Install pText using pip

```
pip install ptext-joris-schellekens
```

4. Done 🎉 You are all ready to go.

Try out some of the examples to get to know pText .

0.2 About AGPLv3

The AGPL license differs from the other GNU licenses in that it was built for network software. You can distribute modified versions if you keep track of the changes and the date you made them. As per usual with GNU licenses, you must license derivatives under AGPL.

It provides the same restrictions and freedoms as the GPLv3 but with an additional clause which makes it so that source code must be distributed along with web publication.

Since web sites and services are never distributed in the traditional sense, the AGPL is the GPL of the web.

CAN	CAN NOT	MUST
Commercial Use	Sublicense	Include Copyright
Modify	Hold Liable	Include License
Distribute		State changes
Place Warranty		Disclose Source
		Include Install Instructions

1. Working with existing PDFs

1.1 Extracting text from a Document using SimpleTextExtraction

Let's start by reading the PDF Document .

```
with open("input.pdf", "rb") as pdf_file_handle:  
    l = SimpleTextExtraction()  
    doc = PDF.loads(pdf_file_handle, [l])
```

Notice that we are passing an `EventListener` instance to the `PDF.loads` method. This `EventListener` will be notified every time a rendering instruction takes place. `SimpleTextExtraction` processes those rendering instructions related to displaying text, and attempts to build the resulting text on the `Page` using some (simple) heuristics.

Now that we've processed the `Page`, we can get the resulting text and store it.

```
# export txt
with open("output.txt", "w") as txt_file_handle:
    txt_file_handle.write(l.get_text(0))
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.2 Working with ligatures using `SimpleNonLigatureTextExtraction`

In writing and typography, a ligature occurs where two or more graphemes or letters are joined as a single glyph. An example is the character æ as used in English, in which the letters a and e are joined. The common ampersand (&) developed from a ligature in which the handwritten Latin letters e and t (spelling et, Latin for and) were combined.

Dealing with ligatures can make text-parsing challenging. You never know whether your PDF `Document` is going to contain "fi" (ligature) or "f i" (two separate characters).

And although these characters might look the same, a regular expression that matches "fi" (two separate characters) will not match "fi" (ligature).

Hence `SimpleNonLigatureTextExtraction`, it works much like `SimpleTextExtraction`, replacing every ligature in the resultant text with its separate characters, ensuring text that is easy to process afterwards.

Let's start by reading the PDF `Document`.

```
with open("input.pdf", "rb") as pdf_file_handle:  
    l = SimpleNonLigatureTextExtraction()  
    doc = PDF.loads(pdf_file_handle, [l])
```

Once the `Document` is done processing, we can easily obtain and store the text:

```
# export txt  
with open("output.txt", "w") as txt_file_handle:  
    txt_file_handle.write(l.get_text(0))
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.3 Looking for a regular expression in a `Document` using `RegularExpressionTextExtraction`

We start by reading the PDF:

```
doc = None  
l = RegularExpressionTextExtraction("[sS]orbitol")  
with open("input.pdf", "rb") as in_file_handle:  
    doc = PDF.loads(in_file_handle, [l])
```

Notice that we are passing an `EventListener` instance to the `PDF.loads` method. This `EventListener` will be notified every time a rendering instruction takes place. The `RegularExpressionTextExtraction` implementation will use these instructions to determine whether a given regular expression has been matched.

We can access this information in the following manner:

```

# export matches
with open("sorbitol_matches.json", "w") as json_file_handle:
    obj = [
        {
            "text": x.get_text(),
            "x": int(x.get_baseline().x),
            "y": int(x.get_baseline().y),
            "width": int(x.get_baseline().width),
            "height": int(x.get_baseline().height),
        }
        for x in l.get_matched_chunk_of_text_render_events_per_page(0)
    ]
    json_file_handle.write(json.dumps(obj, indent=4))

```

This should store the coordinates of the individual letters that matched the regular expression.

In the example `Document`, this was the output:

```
[
{
    "text": "S",
    "x0": 73,
    "y0": 265,
    "width": 5,
    "height": 9
},
{
    "text": "o",
    "x0": 78,
    "y0": 265,
    "width": 5,
    "height": 9
}
```

```
},
{
    "text": "r",
    "x0": 84,
    "y0": 265,
    "width": 3,
    "height": 9
},
...
}
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.4 Extracting keywords from a Document using `TFIDFKeywordExtraction`

We can easily extract all likely keywords from the `Document` using `TFIDFKeywordExtraction`. This class acts like a regular `EventListener` and will keep track of all text being parsed. Optionally, you can give this class a `List` of stop words (which it will then ignore).

```
with open("input.pdf", "rb") as pdf_file_handle:
    l = TFIDFKeywordExtraction(ENGLISH_STOP_WORDS)
    doc = PDF.loads(pdf_file_handle, [l])
```

Now let's export the keywords in json format:

```
# export txt
with open("output.json", "w") as json_file_handle:
    json_file_handle.write(
        json.dumps(
            [x.__dict__ for x in l.get_keywords_per_page(0, 5)], indent=4
```

```
)  
)
```

For the document I picked, this gives me the following output:

```
[  
{  
    "text": "CONSTIPATION",  
    "page_number": 0,  
    "words_on_page": 120,  
    "term_frequency": 5,  
    "occurs_on_pages": [  
        0,  
        1  
    ],  
    "number_of_pages": 2  
},  
...  
]
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.5 Meta-Information

1.5.1 Using the `\Info` dictionary in a Document

1.5.1.1 Getting the author of an existing PDF

A PDF Document can have an `\Info` dictionary entry, containing meta-information. Because this entry is optional, we need to check at every step of the way whether the path we attempt to navigate exists.

We start by opening the `Document`:

```
with open("input.pdf", "rb") as pdf_file_handle:  
    doc = PDF.loads(pdf_file_handle)
```

Then we check whether the `Document` has an `XRef` table (it should, unless the `Document` is corrupt)

```
if "XRef" not in doc:  
    return False
```

Next we check whether the `XRef` table has a `\Trailer` (it should).

```
if "Trailer" not in doc["XRef"]:  
    return False
```

In the `\Trailer` dictionary, we may find an `\Info` dictionary. This dictionary could contain an entry for `\Author`.

```
if (  
    "Info" in doc["XRef"]["Trailer"]  
    and "Author" in doc["XRef"]["Trailer"]["Info"]  
):  
    author = doc["XRef"]["Trailer"]["Info"]["Author"]  
    print("The author of this PDF is %s" % author)
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.5.1.2 Getting all meta-information of an existing PDF using `DocumentInfo`

`DocumentInfo` represents a convenience class to easily extract all meta-information in the `Document` catalog's `\Info` dictionary. You can use it to quickly query the meta-information.

```
with open("input.pdf", "rb") as pdf_file_handle:  
    doc = PDF.loads(pdf_file_handle)  
    doc_info = doc.get_document_info()  
    print("title : %s" % doc_info.get_title())  
    print("author : %s" % doc_info.get_author())  
    print("creator : %s" % doc_info.get_creator())  
    print("producer : %s" % doc_info.get_producer())  
    print("ids : %s" % doc_info.get_ids())  
    print("language : %s" % doc_info.get_language())
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.5.1.3 Changing the author of an existing PDF

Let's start by reading the PDF Document .

```
doc = None  
with open("input.pdf", "rb") as pdf_file_handle:  
    doc = PDF.loads(pdf_file_handle)
```

Now we check whether the PDF has an XRef, containing a \Trailer

```
if "XRef" not in doc:  
    return False  
if "Trailer" not in doc["XRef"]:  
    return False
```

If there is no \Info dictionary in the \Trailer , we create it

```
if "Info" not in doc["XRef"]["Trailer"]:
    doc["XRef"]["Trailer"][Name("Info")] = Dictionary()
```

Let's set the \Author entry in the \Info dictionary

```
# change author
doc["XRef"]["Trailer"]["Info"]["Author"] = String("Joris Schellekens")
```

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

Check out the tests directory to find more tests like this one, and discover what you can do with pText .

1.5.1.4 Changing the producer of an existing PDF

Let's start by reading the PDF Document .

```
doc = None
with open("input.pdf", "rb") as pdf_file_handle:
    doc = PDF.loads(pdf_file_handle)
```

Now we check whether the PDF has an XRef, containing a \Trailer

```
if "XRef" not in doc:  
    return False  
if "Trailer" not in doc["XRef"]:  
    return False
```

If there is no \Info dictionary in the \Trailer , we create it

```
if "Info" not in doc["XRef"]["Trailer"]:  
    doc["XRef"]["Trailer"][Name("Info")] = Dictionary()
```

Let's set the \Producer entry in the \Info dictionary

```
# change author  
doc["XRef"]["Trailer"]["Info"]["Producer"] = String("pText")
```

Now we can store the PDF document again:

```
with open("output.pdf", "wb") as out_file_handle:  
    PDF.dumps(out_file_handle, doc)
```

Check out the tests directory to find more tests like this one, and discover what you can do with pText .

1.5.2 Using the XMP metadata in a Document

1.5.2.1 Reading the XMP metadata of an existing PDF

This example is similar to the earlier example involving `DocumentInfo`. But instead, we will use `XMPDocumentInfo`. This class offers even more methods to get information from a PDF `Document`. Keep in mind that XMP is not a requirement for a PDF `Document` to be valid. So you may find these methods return `None` when you test them on a `Document` that does not have embedded XMP data.

```
with open(file, "rb") as pdf_file_handle:  
    doc = PDF.loads(pdf_file_handle)  
    doc_info = doc.get_xmp_document_info()  
    print("title : %s" % doc_info.get_title())  
    print("author : %s" % doc_info.get_author())  
    print("creator : %s" % doc_info.get_creator())  
    print("producer : %s" % doc_info.get_producer())  
    print("ids : %s" % doc_info.get_ids())  
    print("language : %s" % doc_info.get_language())  
    print("document-ID : %s" % doc_info.get_document_id())  
    print("original document-ID : %s" % doc_info.get_original_document_id())  
    print("creation date : %s" % doc_info.get_creation_date())  
    print("modification date : %s" % doc_info.get_modification_date())  
    print("metadata date : %s" % doc_info.get_metadata_date())  
    print("")
```

I tried this on a `Document` with XMP meta-data, and it printed the following:

title	: None
author	: None
creator	: None
producer	: Adobe PDF Library 15.0
ids	: ['0952B683A7F340E48FD2F5409F3E6D08', 'AF7A23737C7A664D93DF2CBE97397150']
language	: en-GB
document-ID	: xmp.id:54e5adca-494c-4c10-983a-daa03cdae65a

```
original document-ID : xmp.did:b857e947-9e0d-4cd3-aff9-40a81c991e7a
creation date       : 2017-12-15T15:38:40+01:00
modification date   : 2017-12-15T16:23:53+01:00
metadata date       : 2017-12-15T16:23:53+01:00
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.6 Images

1.6.1 Extracting Images from a Document using SimpleImageExtraction

Like in the previous examples, we'll use an implementation of `EventListener`:

```
with open(file, "rb") as pdf_file_handle:
    l = SimpleImageExtraction()
    doc = PDF.loads(pdf_file_handle, [l])
```

In this case `SimpleImageExtraction` will listen to all PDF parser commands that tell the reader to display an `Image`.

Once the `Document` is parsed, we can extract all `Image` objects from the `SimpleImageExtraction`

```
for i, img in enumerate(l.get_images_per_page(0)):
    output_file = self.output_dir / (file.stem + str(i) + ".jpg")
    with open(output_file, "wb") as image_file_handle:
        img.save(image_file_handle)
```

1.7 Annotations

An annotation associates an object such as a note, sound, or movie with a location on a page of a PDF document, or provides a way to interact with the user by means of the mouse and keyboard. PDF includes a wide variety of standard annotation types, described in detail in 12.5.6, “Annotation Types.”

1.7.1 Adding a rubber stamp annotation to an existing PDF

A rubber stamp annotation (PDF 1.3) displays text or graphics intended to look as if they were stamped on the page with a rubber stamp. When opened, it shall display a pop-up window containing the text of the associated note. Table 181 shows the annotation dictionary entries specific to this type of annotation.

We start by reading the Document :

```
# attempt to read PDF
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Then we add the annotation:

```
# add annotation
doc.get_page(0).append_stamp_annotation(
    name=RubberStampAnnotationIconType.CONFIDENTIAL,
    contents="Approved by Joris Schellekens",
    color=X11Color("White"),
    rectangle=(Decimal(128), Decimal(128), Decimal(32), Decimal(64)),
)
```

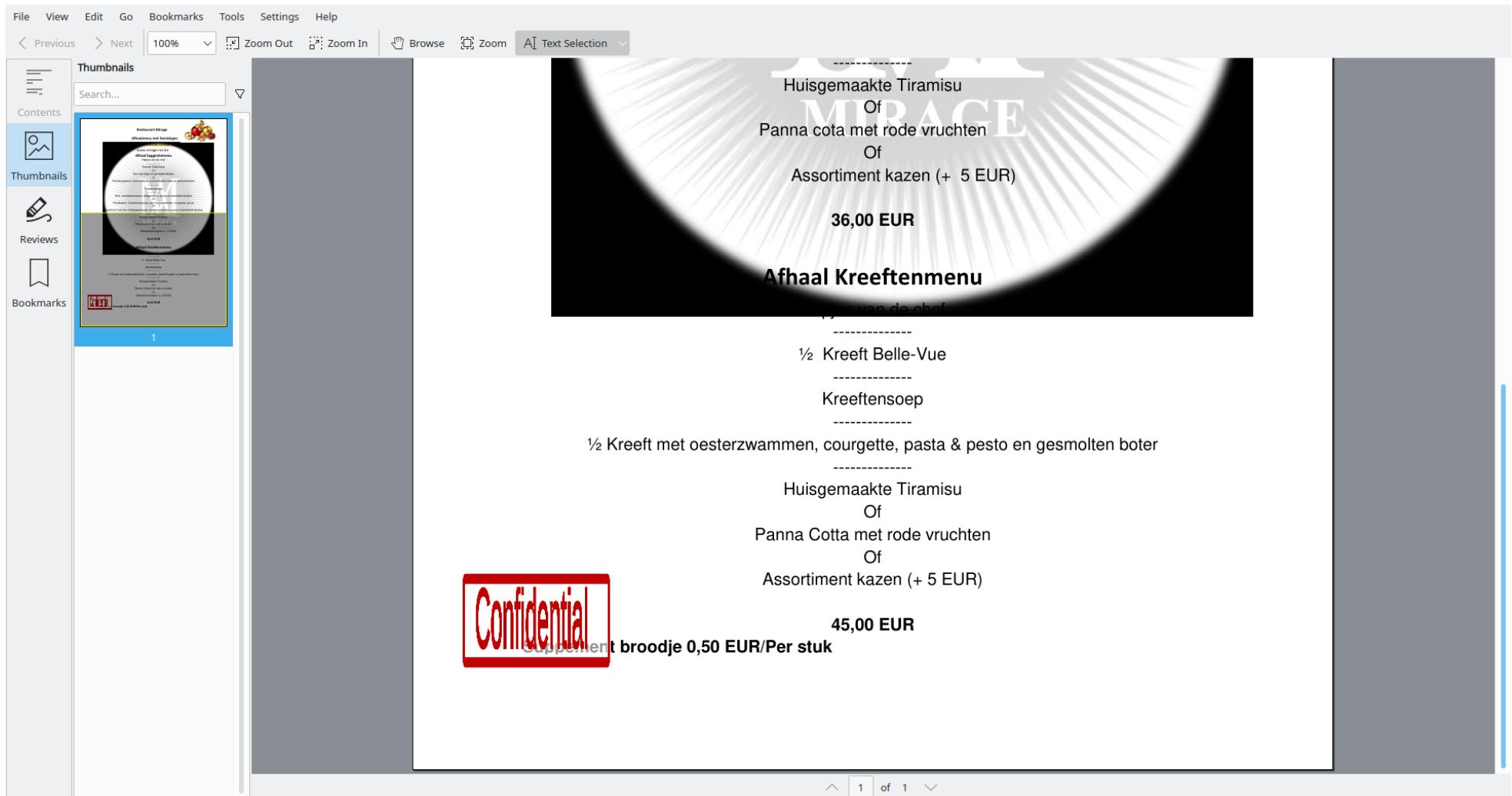
There are various parameters we can set here;

- `name` : conforming readers should support at least the following values for the `name` parameter: Approved, Experimental, NotApproved, AsIs, Expired, NotForPublicRelease, Confidential, Final, Sold, Departmental, ForComment, TopSecret, Draft, ForPublicRelease
- `contents` : is the text that should appear in the pop-up window when the stamp annotation is clicked
- `color` : is the `color` of the pop-up window
- `rectangle` : denotes the coordinates of the rubber stamp annotation

Finally, we store the output:

```
# attempt to store PDF
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The result should be something like this (keep in mind the rendering of the rubber stamp is the responsibility of the PDF reader you happen to be using. Your result may differ accordingly.):



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.2 Adding all possible rubber stamp annotations to an existing PDF

A rubber stamp annotation (PDF 1.3) displays text or graphics intended to look as if they were stamped on the page with a rubber stamp. When opened, it shall display a pop-up window containing the text of the associated note. Table 181 shows the annotation dictionary entries specific to this type of annotation.

Let's have a look how our PDF reader renders all possible rubber stamp annotations.

We start by reading an existing PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

We now define a `List[str]` to hold all valid types of rubber stamp annotations, we iterate over it, and add them to the `Document` one at a time:

```
# add annotation
for index, name in enumerate(RubberStampAnnotationIconType):
    doc.get_page(0).append_stamp_annotation(
        name=name,
        contents="Approved by Joris Schellekens",
        color=X11Color("White"),
        rectangle=Rectangle(
            Decimal(128), Decimal(128 + index * 34), Decimal(64), Decimal(32)
        ),
    )
```

There are some parameters we can set here:

- `name` : indicates the kind of stamp (e.g. 'Approved' or 'Draft' etc)

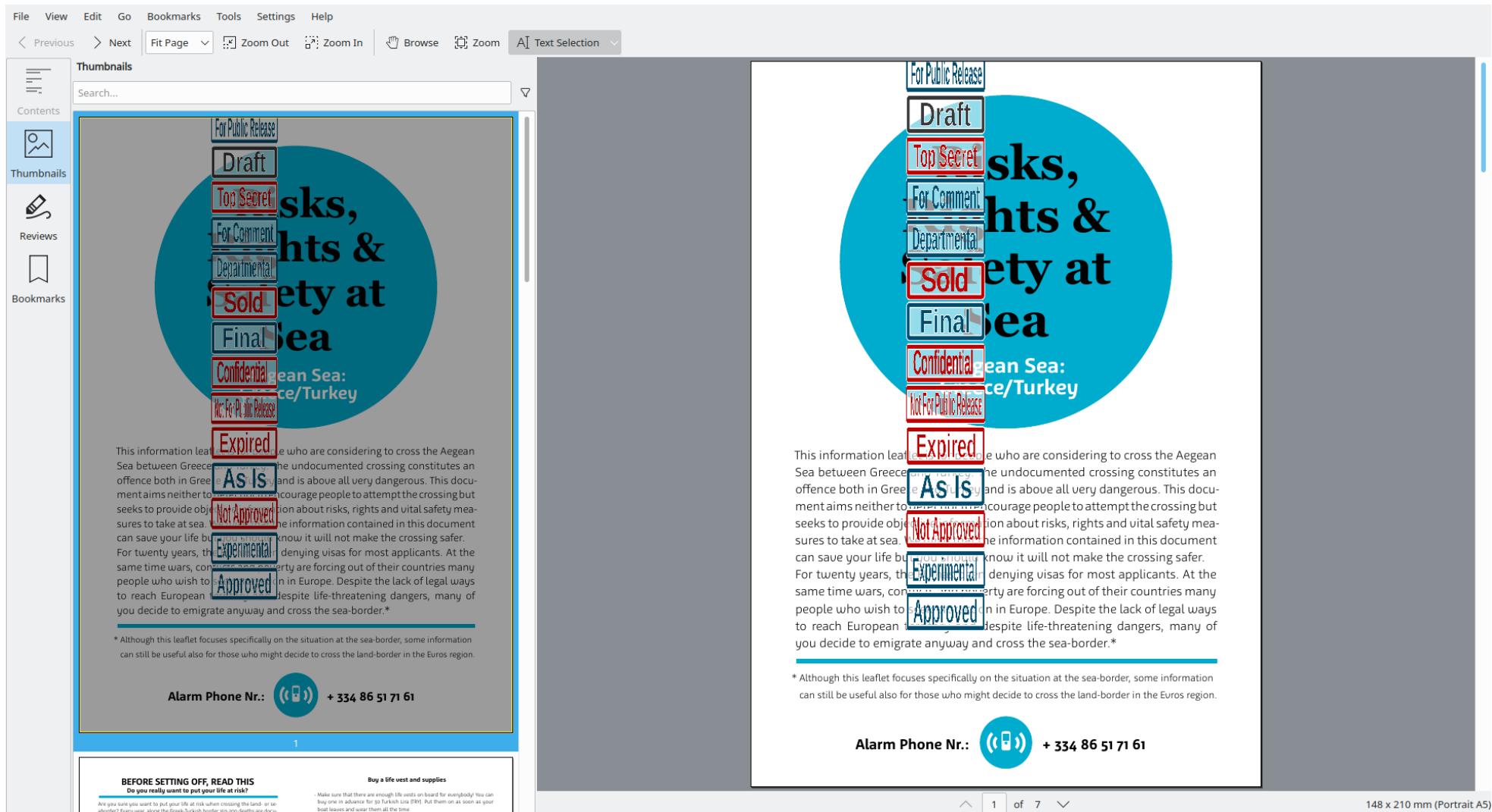
- `contents` : this is the text shown when the annotation is clicked in a PDF reader
- `color` : this is the `color` of the pop-up that displays the aforementioned text
- `rectangle` : this is where the annotation is to be placed

Note that you do not have control over the appearance of this particular annotation. The specific appearance is down to the implementation of the PDF reader (e.g. Adobe Acrobat Reader).

Now we can store the PDF document again:

```
with open("output.pdf", "wb") as out_file_handle:  
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.3 Adding a circle annotation to an existing PDF

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Now we can add the annotation:

```
# add annotation
doc.get_page(0).append_circle_annotation(
    rectangle=Rectangle(Decimal(128), Decimal(128), Decimal(64), Decimal(64)),
    stroke_color=X11Color("Plum"),
    fill_color=X11Color("Crimson"),
)
```

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next 75% Zoom Out Zoom In Browse Zoom Text Selection

Thumbnails
Search...
Contents
Reviews
Bookmarks

Verantwoord werken



Verantwoord werken met machines die op een duurzame wijze geproduceerd worden. Machines met een lange levensduur, die weinig onderhoud nodig hebben en daardoor garant staan voor een lang en probleemloos leven. Bij Conver zijn de maaiverzamelboten speciaal ontwikkeld om waterplanten en/of drijfvuil uit watergangen en meren te verwijderen. Niet alleen drijvende planten als Lemna (kroos) en Waterhyacinth (Eichhornia Crassipes), maar ook wortelende planten als Waterpest (Elodea) en vele andere soorten kunnen worden geogst.

[Ontdek de nieuwe MC101 maaiverzamelboot](#)

CONVER HET ZIT IN ONZE NATUUR

1

Compacte maaiverzamelboot
De MC101 is een zeer compacte, lichte en handzame maaiverzamelboot voor het verwijderen van drijvende waterplanten uit kleine meren, plassen en grote vijvers. Een constructie met vlakke laadvloer. Eigen, beproefde, voortstuwing aan de achterzijde en een dubbele messenbalk aan de



Verantwoord werken met machines die op een duurzame wijze geproduceerd worden. Machines met een lange levensduur, die weinig onderhoud nodig hebben en daardoor garant staan voor een lang en probleemloos leven. Bij Conver zijn de maaiverzamelboten speciaal ontwikkeld om waterplanten en/of drijfvuil uit watergangen en meren te verwijderen. Niet alleen drijvende planten als Lemna (kroos) en Waterhyacinth (Eichhornia Crassipes), maar ook wortelende planten als Waterpest (Elodea) en vele andere soorten kunnen worden geogst.

Ontdek de nieuwe MC101 maaiverzamelboot

CONVER **HET ZIT IN ONZE NATUUR**

Compacte maaiverzamelboot
De MC101 is een zeer compacte, lichte en handzame maaiverzamelboot voor het verwijderen van drijvende waterplanten uit kleine meren, plassen en grote vijvers. Een constructie met vlakke laadvloer. Eigen, beproefde, voortstuwing aan de achterzijde en een dubbele messenbalk aan de

Check out the tests directory to find more tests like this one, and discover what you can do with pText .

1.7.4 Adding a square annotation to an existing PDF

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Now we can add the annotation:

```
# add annotation
doc.get_page(0).append_square_annotation(
    rectangle=Rectangle(Decimal(128), Decimal(128), Decimal(64), Decimal(64)),
    stroke_color=X11Color("Plum"),
    fill_color=X11Color("Crimson"),
)
```

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next 75% Zoom In Zoom Out Browse Zoom Text Selection

Contents

Search...

Contents

Rules of Procedure of the European Council 9

Article 7

Written procedure

Decisions of the European Council on an urgent matter may be adopted by a written vote where the President of the European Council proposes to use that procedure. Written votes may be used where all members of the European Council having the right to vote agree to that procedure.

A summary of acts adopted by the written procedure shall be drawn up periodically by the General Secretariat of the Council.

Article 8

Minutes

Minutes of each meeting shall be drawn up; a draft of those minutes shall be prepared by the General Secretariat of the Council within 15 days. The draft shall be submitted to the European Council for approval, and then signed by the Secretary-General of the Council.

The minutes shall contain:

- a reference to the documents submitted to the European Council,
- a reference to the conclusions approved,
- the decisions taken,
- the statements made by the European Council and those whose entry has been requested by a member of the European Council.

Article 9

Deliberations and decisions on the basis of documents and drafts drawn up in the languages provided for by the language rules in force

1. Except as otherwise decided unanimously by the European Council on grounds of urgency, the European Council shall deliberate and take decisions only on the basis of documents and drafts drawn up in the languages specified in the rules in force governing languages.
2. Any member of the European Council may oppose discussion where the texts of any proposed amendments are not drawn up in one of the languages referred to in paragraph 1 as he or she may specify.

176 x 250 mm (Portrait B5)

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.5 Adding a polygon annotation to an existing PDF

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Next we add the annotation:

```
doc.get_page(0).append_polygon_annotation(
    points=[
        (Decimal(72), Decimal(390)),
        (Decimal(242), Decimal(500)),
        (Decimal(156), Decimal(390)),
    ],
    color=X11Color("Crimson"),
)
```

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next Fit Width Zoom Out Zoom In Browse Text Selection

Thumbnails

Contents

Reviews

Bookmarks

Search...

Lifestyle advice continued...
Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools. You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

Sorbitol is a sugar, which soften the stools and acts like a natural laxative. Sorbitol is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of sorbitol is about 5-10 times higher in dried fruit.

- * Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS.

More information available at:
www.patient.co.uk/health/constipation-in-adults-leaflet
www.bladderandbowiefoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS Rotherham Clinical Commissioning Group

Patient Information leaflet Constipation

Your life, Your health

What is constipation?
Constipation is common. Usual symptoms include stools being hard or motionless becoming hard, and difficult or painful to pass. It may be that bowel movements increase compared with your usual pattern, but may also feel bloated and feel sick if you have constipation.

Bristol Stool Chart

Type 1	Segments of faeces. No water
Type 2	Seagreen-shaped hard stool
Type 3	Like a log with cracks on the surface
Type 4	Like a sausage with crust
Type 5	Soft blobs with edges
Type 6	Fluffy pieces with no edges - easily流动性
Type 7	Watery, runny stools

Note: Type 1, 2 or 3 on the Bristol stool chart show some level of constipation, with Type 7 the most normal.

Note: It is normal to have a large range of normal bowel habit, from 2-3 times per day to 2-3 times per week. It is a change from your normal pattern, or the hardness and pain passing the stools that defines constipation.

What causes constipation?

- * Not eating enough fibre (meat/pasta)
- * It is a common cause. See below
- * Not drinking enough, as stools require water to keep them soft and move along the gut
- * Some medicines can cause constipation as a side effect. For example, tablets like co-codamol, codine and morphine slow down the movement of the gut. You may need a laxative to start it moving again. You may wish to check the patient information leaflet or with your pharmacist.
- * Underactive bowel muscles can cause constipation. For example, an underactive thyroid, irritable bowel syndrome, or a muscle spasm that reduce your mobility and wastes.
- * Pregnancy. Hormonal changes during pregnancy can affect gut movements, and in later pregnancy, the baby's position can make it more difficult for the stools to move.
- * Unknown cause (idiopathic) Some people have constipation that has no clear cause. They may have constipation, but still become constipated.

Their bowel motions are said to be underactive, and they may have constipation, and mostly occurs in women. It may start in childhood or early adulthood, and persists throughout life.

What can I do to reduce my constipation?

- * Eat more fibre. Aim to drink about 8-10 cups (2 litres) of fluid per day. This helps water to stay in the gut and soften the stools. Most drinks will do, but alcoholic drinks can dehydrate you.
- * Exercise regularly. Keeping your body active helps to keep your gut moving. If you have constipation, try walking, low mobility or bed-bound lives. If you are already, you are more likely to get constipated.
- * Taking routines. Do not ignore the feelings of needing to go to the toilet. Some people suppress the feeling of the need to go to the toilet, and end up with stools which is difficult to pass later.
- * As a rule, it is best to try going to the toilet after a meal, and after about 30 minutes after a meal. This is because the movement of stools through the gut is increased in the morning and after meals.

How you sit on the toilet is also important. If you sit upright and eat your food, do not have a disease or talk any more, it will help the passage of stools. Relax, sit upright and put your feet on your thighs. You can lean forward and hold your breath to pass stools.

NHS Rotherham Clinical Commissioning Group

Your life, Your health

Lifestyle advice continued...
Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools. You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

Sorbitol is a sugar, which soften the stools and acts like a natural laxative. Sorbitol is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of sorbitol is about 5-10 times higher in dried fruit.

- * Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS.

More information available at:
www.patient.co.uk/health/constipation-in-adults-leaflet
www.nhs.uk/Conditions/Constipation
www.bladderandbowiefoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS Rotherham Clinical Commissioning Group

Patient Information leaflet Constipation



NHS
Rotherham
Clinical Commissioning Group

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.6 Adding a polyline annotation to an existing PDF

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Next we add the annotation:

```
doc.get_page(0).append_polyline_annotation(
    points=[
        (Decimal(72), Decimal(390)),
        (Decimal(242), Decimal(500)),
        (Decimal(156), Decimal(390)),
    ],
    color=X11Color("Crimson"),
)
```

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next Fit Width Zoom Out Zoom In Text Selection

Thumbnails

Search...

Contents

Reviews

Bookmarks

Lifestyle advice continued...

Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools.

You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

Sorbitol is a sugar, which softens the stools and acts like a natural laxative. Sorbitol is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of sorbitol is about 5-10 times higher in dried fruit.

* Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS

More information available at:

www.patient.co.uk/health/constipation-in-adults-leaflet
www.nhs.uk/Conditions/Constipation
www.bladderandbowelfoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS Rotherham Clinical Commissioning Group

Patient Information leaflet Constipation

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.7 Adding an annotation using a shape from the `LineArtFactory` to an existing PDF

The `LineArtFactory` class allows you to easily create shapes (defined as `List[Tuple[Decimal, Decimal]]`), it contains everything you need to render:

- triangles (right sided triangle, isoceles triangles)
- stars (with convenience methods for 4-sided stars, 5-sided stars, 6-sided stars)
- 4-gons (parallelogram, trapezoid, diamond)
- regular n-gons (with convenience methods for pentagon, hexagon, heptagon, octagon)
- fractions of circles (with convenience methods for half a circle and three quarters of a circle)
- arrows (left, right, up, down)
- misc. (droplet, sticky note, etc)

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Now we can add the annotation:

```
# get the shape
shape = LineArtFactory.droplet(
    Rectangle(Decimal(100), Decimal(100), Decimal(100), Decimal(100))
)

# add annotation
doc.get_page(0).append_polyline_annotation(
    points=shape,
```

```
    stroke_color=X11Color("Salmon"),  
)  
  
Now we can store the PDF Document again:
```

```
with open("output.pdf", "wb") as out_file_handle:  
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next Fit Width Zoom Out Zoom In Browse Zoom Text Selection

Thumbnails

Search...

Contents

Reviews

Bookmarks

Lifestyle advice continued...

Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools.

You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

Sorbitol is a sugar, which softens the stools and acts like a natural laxative. Sorbitol is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of sorbitol is about 5-10 times higher in dried fruit.

* Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS

More information available at:

www.patient.co.uk/health/constipation-in-adults-leaflet
www.nhs.uk/Conditions/Constipation
www.bladderandbowelfoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS Rotherham Clinical Commissioning Group

Patient Information leaflet Constipation

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.8 Adding a highlight annotation to an existing PDF

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Next we add the annotation:

```
# add annotation
doc.get_page(0).append_highlight_annotation(
    rectangle=Rectangle(
        Decimal(72.86), Decimal(486.82), Decimal(129), Decimal(13)
    ),
    contents="Lorem Ipsum Dolor Sit Amet",
    color=X11Color("Yellow"),
)
```

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next Fit Width Zoom Out Zoom In Browse Zoom Text Selection

Thumbnails

Search...

Contents

Reviews

Bookmarks

Lifestyle advice continued...
Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools.
You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

Sorbitol is a sugar which softens the stools and acts like a natural laxative. Sorbitol is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of sorbitol is about 5-10 times higher in dried fruit.

* Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS.

More information available at:
www.patient.co.uk/health/constipation-in-adults-leaflet
www.nhs.uk/Conditions/Constipation
www.bladderandbowelfoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS Rotherham Clinical Commissioning Group

Patient Information leaflet Constipation

Lifestyle advice continued...
Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools.

You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

Sorbitol is a sugar which softens the stools and acts like a natural laxative. Sorbitol is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of sorbitol is about 5-10 times higher in dried fruit.

* Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS.

More information available at:
www.patient.co.uk/health/constipation-in-adults-leaflet
www.nhs.uk/Conditions/Constipation
www.bladderandbowelfoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS Rotherham Clinical Commissioning Group

Patient Information leaflet Constipation

What is constipation?
Constipation is common. Usual symptoms include stools (faeces) or motions becoming hard, lumpy or pellet-like. The time between toilet trips increases compared with your usual pattern. If you are worried about constipation and feel sick if you have severe constipation.

Bristol Stool Chart

Type 1	1	2	3	4	5	6	7	8
Very dry, hard, lumpy stool	Small amount of stool	Stools stick together but pass easily	Stools pass easily	Normal soft stool	Soft stool	Very soft stool	Watery stool	Runny stool

What causes constipation?
• Not eating enough fibre (roughage) is a common cause (See below)

• Not drinking enough fluid or water can lead to constipation. It is important to drink at least 1.5 litres of fluid daily.

• Some medicines can cause constipation. For example, painkillers like codeine, codamide and morphine slow down the movement of the gut. You may need a laxative to treat it if you are worried. Ask your pharmacist or your doctor.

• Some medical conditions can cause constipation. For example, an underactive thyroid, irritable bowel syndrome, and other conditions that reduce your mobility and muscle.

• Pregnancy. Hormones change in pregnancy and slow down the gut movements, and in late pregnancy, the baby puts pressure on the bowel, making it more difficult for the stools to move.

• Unknown cause (Diarrhoea) Some people have a normal range of bowel habit, from 2-3 times per week. It is a change from your normal pattern of bowel habit and pain passing the stools that defines constipation.

How many bowel movements is a normal bowel habit, from 2-3 times per week. It is a change from your normal pattern of bowel habit and pain passing the stools that defines constipation.

What can I do to reduce my constipation? (Lifestyle advice)

Have plenty to drink. Aim to drink about 8-10 cups (2 litres) of fluid daily. This includes water, juice, tea and coffee. Milk, soft drinks, beer, wine, alcohol, drinks can be dehydrating.

Exercise regularly. Keeping your body active helps to move the bowels. It is well known that people with low mobility or bed-bound lives if just going to the toilet is easier to get constipated.

Toileting advice: Do not ignore the feeling of needing the toilet. Some people suppress this feeling if they are busy. This can lead to a backlog of stools which is difficult to pass later. As a rule, it is best to go to the toilet first thing in the morning and about 30 minutes after a meal. This is because the movement of stools through the bowel is stimulated in the mornings and after meals.

When you sit on the toilet, it is also important to relax. Relaxing your feet is a good way to help the passage of stools. Relax, lean forward and keep your feet flat on the floor and hold your breath to pass stools.

NHS Rotherham Clinical Commissioning Group

Your life, Your health

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.9 Adding a link annotation to an existing PDF

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Next we add the annotation:

```
doc.get_page(0).append_link_annotation(
    page=Decimal(0),
    destination_type=DestinationType.FIT,
    color=X11Color("Red"),
    rectangle=Rectangle(Decimal(128), Decimal(128), Decimal(64), Decimal(64)),
)
```

There are some parameters we can set here:

- `page` : indicates the page number of the `Page` you would like to link to
- `destination_type` : In this case 'Fit' means 'show the entire `Page` , and force the viewer to zoom until it fits'

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next Fit Width Zoom Out Zoom In Browse Zoom Text Selection

Thumbnails

Search...

Contents

Reviews

Bookmarks

Lifestyle advice continued...

Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools.

You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

Sorbitol is a sugar, which softens the stools and acts like a natural laxative. Sorbitol is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of sorbitol is about 5-10 times Higher in dried fruit.

* Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS

More information available at:

www.patient.co.uk/health/constipation-in-adults-leaflet
www.nhs.uk/Conditions/Constipation
www.bladderandbowelfoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS Rotherham Clinical Commissioning Group

Patient Information leaflet Constipation

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.10 Adding a text annotation to an existing PDF

We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Now we can add the annotation:

```
# add annotation
doc.get_page(0).append_text_annotation(
    contents="The quick brown fox ate the lazy mouse",
    rectangle=Rectangle(Decimal(128), Decimal(128), Decimal(64), Decimal(64)),
    name_of_icon="Key",
    open=True,
    color=X11Color("Orange"),
)
```

Finally, we need to store the resulting PDF Document .

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

Check out the tests directory to find more tests like this one, and discover what you can do with pText .

1.7.11 Adding a square annotation around a regular expression match to an existing PDF

Let's combine what we saw earlier, about finding the coordinates of a regular expression with our new understanding of annotations.

In this example, we'll just draw a rectangle around each letter of a matching text-snippet. But we would easily expand this example to include a text annotation. Imagine automatically tagging documents such that hard-to-understand terms have an annotation the end-user can click open for more explanation.

We start by reading the PDF:

```
doc = None
l = RegularExpressionTextExtraction("[sS]orbitol")
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle, [l])
```

Notice that we are passing an `EventListener` instance to the `PDF.loads` method. This `EventListener` will be notified every time a rendering instruction takes place. The `RegularExpressionTextExtraction` implementation will use these instructions to determine whether a given regular expression has been matched.

Next we are going to add annotations (in this case square annotations) around every `ChunkOfTextRenderEvent` that belongs to a regular expression match.

```
for e in l.get_matched_chunk_of_text_render_events_per_page(0):
    doc.get_page(0).append_square_annotation(
        rectangle=e.get_baseline(),
        stroke_color=X11Color("Firebrick"),
    )
```

Now we can store the PDF document again:

```
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this:

The screenshot shows a patient information leaflet for constipation. The page is titled "Patient Information Leaflet Constipation". It features a header with the NHS Rotherham Clinical Commissioning Group logo. The main content includes sections on "Lifestyle advice continued...", "More information available at:", "Produced by:", "Supported by:", and a copyright notice for February 2014. The page is annotated with several green boxes highlighting key text. A sidebar on the left shows thumbnails of the document's pages. The PDF viewer interface includes a toolbar at the top and a navigation bar at the bottom.

Lifestyle advice continued...

Eat foods containing plenty of fibre. Fibre from food stays in your gut and adds bulk and softness to the stools.

You may have some bloating and wind at first, and it can take up to four weeks to help your constipation. So it is best to increase your fibre slowly and make it a long term change. You will also need to drink lots of water with your high fibre foods.

High-fibre foods include:

- Fruit and vegetables. Aim to eat at least five portions of different fruit and vegetables each day
- Oats, nuts and seeds
- Wholegrain cereals, bran and wholemeal pasta, bread etc *

is a sugar, which softens the stools and acts like a natural laxative. is found in fruits (and juices) such as apples, apricots, gooseberries, grapes (and raisins), peaches, pears, plums, prunes, raspberries and strawberries. The amount of is about 5-10 times higher in dried fruit.

* Sometimes bran and wholemeal may cause more bloating and cramps and worsen constipation in patients with IBS

More information available at:

www.patient.co.uk/health/constipation-in-adults-leaflet
www.nhs.uk/Conditions/Constipation
www.bladderandbowelfoundation.org

Produced by:
Medicines Management Team
Rotherham CCG
Oak House
Bramley
Rotherham
S66 1YY

Supported by:
Rotherham bladder and bowel patient group

February 2014

NHS
Rotherham
Clinical Commissioning Group

Your life, Your health

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.12 Adding a square annotation in the free space of a page to an existing PDF

Sometimes the position of the annotation does not matter that much, as long as it does not block any other visible content.

Finding the available free space on a `Page` can be tricky, it would involve re-parsing all the content to figure out where existing content intersects with the desired location of the annotation. That is why `pText` comes with `FreeSpaceFinder`, this class searches for an `Rectangle` of a given size, nearest to a given point (in Euclidean space).

Let's see it in action. We start by reading the PDF:

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Next we instantiate the `FreeSpaceFinder` with a given `Page` as argument.

```
# determine free space
space_finder = FreeSpaceFinder(doc.get_page(0))
```

Now we can attempt to add the annotation. We call the method `find_free_space` passing it the ideal `Rectangle` where we would like to place the annotation (or any other object really). `find_free_space` returns an `Optional[Rectangle]` (sometimes the `Page` is full).

```
# add annotation
w, h = doc.get_page(0).get_page_info().get_size()
free_rect = space_finder.find_free_space(
    Rectangle(
        Decimal(w / Decimal(2)),
        Decimal(h * Decimal(0.1)),
        Decimal(64),
```

```
        Decimal(64),  
    )  
)
```

If there is room on the `Page` for the annotation, we can now add it. Notice that we wanted to add the annotation to the bottom center of the `Page`.

```
if free_rect is not None:  
    doc.get_page(0).append_square_annotation(  
        rectangle=free_rect,  
        stroke_color=HexColor("#F75C03"),  
        fill_color=HexColor("#04A777"),  
    )
```

Now we can store the PDF Document again:

```
with open("output.pdf", "wb") as out_file_handle:  
    PDF.dumps(out_file_handle, doc)
```

The end result (at least the annotations) should look something like this: Notice how our use of `FreeSpaceFinder` meant that the annotation did not collide with the existing page-number on the bottom of the `Page`.

Even at its most creative, Felix's imagination sometimes struggles and becomes simply incapable of overcoming the horror and brutality with which it is continually confronted. At times like this, his stories cannot help him - yet he always comes back to them. As the novel draws to a close, Felix and the reader are left wondering what the future holds, '*I don't know what the rest of my story will be*' (p.150). But the love of stories and incredible resilience we have witnessed throughout Felix's journey create a sense of hope and optimism for us - enough to believe he may one day share his ultimate story, one of survival.

Morris Gleitzman has created a wonderful character and vehicle in Felix. Through the eyes of this central character the reader will witness humanity's capacity for inhumanity. The events and experiences endured by Felix (and many other characters) are extremely difficult and complex to comprehend, if comprehensible at all. The reader for whom this is an introduction to the Holocaust may experience the same terrible dawning realization of what was truly happening at that time. A study of Felix (his history, personality, actions & impact) should encourage this reader, and those with a greater depth of knowledge about the Holocaust, to spend time thinking, reflecting and sharing questions or views in an attempt to find some understanding.

10

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.7.13 Getting all annotations from a PDF

Getting all annotations from a PDF is easy, if you know where to look. Let's start by opening the PDF Document :

```
with open("input.pdf", "rb") as pdf_file_handle:  
    doc = PDF.loads(pdf_file_handle)  
    page = doc.get_page(0)
```

Annotations are defined in the \Page dictionary of whatever page the annotation appears at. Let's check the first Page .

```
if "Annots" in page:  
    print("%s has %d annotations" % ("input.pdf", len(page["Annots"])))
```

Check out the tests directory to find more tests like this one, and discover what you can do with pText .

1.7.14 Showcase : Adding a collection of annotations shaped like super mario to an existing PDF

From the spec:

An annotation associates an object such as a note, sound, or movie with a location on a page of a PDF document, or provides a way to interact with the user by means of the mouse and keyboard. PDF includes a wide variety of standard annotation types, described in detail in 12.5.6, “Annotation Types.”

[...]

A link annotation represents either a hypertext link to a destination elsewhere in the document (see 12.3.2, “Destinations”) or an action to be performed (12.6, “Actions”). Table 173 shows the annotation dictionary entries specific to this type of annotation.

Let's add a few annotations to an existing PDF, shaped like super-mario.

First we start by defining the pixel-art grid, and the colors:

```
m = [
    [0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
    [0, 0, 0, 2, 2, 2, 3, 3, 2, 3, 0, 0, 0, 0],
    [0, 0, 2, 3, 2, 3, 3, 2, 3, 3, 0, 0, 0, 0],
    [0, 0, 2, 3, 2, 2, 3, 3, 2, 3, 3, 0, 0, 0],
    [0, 0, 2, 2, 2, 3, 3, 3, 2, 3, 3, 0, 0, 0],
    [0, 0, 2, 2, 3, 3, 3, 2, 2, 2, 2, 0, 0, 0],
    [0, 0, 0, 0, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 1, 4, 1, 1, 1, 1, 0, 0, 0, 0],
    [0, 0, 1, 1, 1, 4, 1, 1, 4, 1, 1, 0, 0, 0],
    [0, 1, 1, 1, 1, 4, 4, 4, 1, 1, 1, 1, 0, 0],
    [0, 3, 3, 1, 4, 5, 4, 4, 5, 4, 1, 3, 3, 0],
    [0, 3, 3, 3, 4, 4, 4, 4, 4, 3, 3, 3, 0],
    [0, 3, 3, 4, 4, 4, 4, 4, 4, 4, 3, 3, 0],
    [0, 0, 0, 4, 4, 4, 0, 0, 4, 4, 4, 0, 0, 0],
    [0, 0, 2, 2, 2, 0, 0, 0, 2, 2, 2, 0, 0],
    [0, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 0, 0],
]
c = [
    None,
    X11Color("Red"),
    X11Color("Black"),
    X11Color("Tan"),
    X11Color("Blue"),
    X11Color("White"),
]
```

Next we need to read an existing PDF Document :

```
doc = None
with open('input.pdf', "rb") as in_file_handle:
    doc = PDF.loads(in_file_handle)
```

Now we can simply add all the annotations by calling the appropriate method on the `Page` object

```
# add annotation
pixel_size = 2
for i in range(0, len(m)):
    for j in range(0, len(m[i])):
        if m[i][j] == 0:
            continue
        x = pixel_size * j
        y = pixel_size * (len(m) - i)
        doc.get_page(0).append_link_annotation(
            page=Decimal(0),
            color=c[m[i][j]],
            destination_type=DestinationType.FIT,
            rectangle=(
                Decimal(x),
                Decimal(y),
                Decimal(x + pixel_size),
                Decimal(y + pixel_size),
            ),
        )
```

When adding a link annotation, we need to specify some arguments related to *what* we are linking to. In this case we specify that we want the annotation to link to `Page 0`, and to force the pdf-viewer (e.g. Adobe Reader) to fit the `Page` (potentially zooming in/out).

We also specify a `Rectangle` (this is where the user would click to activate the link), and a `color` (this is the color of the aforementioned rectangle).

In our case, we calculate the `color` and position based on our earlier grid of super-mario.

As a final step we need to store the resulting PDF Document .

```
with open("output.pdf", "wb") as out_file_handle:  
    PDF.dumps(out_file_handle, doc)
```

The result should be something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next 66% Zoom Out Zoom In Browse Zoom A Text Selection

Thumbnails

Search...

Subject to rule 7, a tax invoice referred to in section 31 shall be issued by the registered person containing the following particulars—

- name, address and GSTIN of the supplier;
- a consecutive serial number not exceeding sixteen characters, in one or multiple units, containing alphabets or numerals or special characters hyphen or dash and slash separated as “-” and “/”, respectively, and any combination thereof, unique for a financial year;
- date of the issue;
- name, address and GSTIN or UIN, if registered, of the recipient;
- name and address of the recipient and the address of delivery, along with the name of State and its code, if each recipient is unregistered and where the value of taxable supply is fifty thousand rupees or more;
- HSN code of goods or Accounting Code of services;
- description of goods or services;
- quantity in case of goods and unit or Unique Quantity Code thereof;
- total value of supply of goods or services or both;
- variable value of supply of goods or services or both taking into account discount or allowance, if any;
- rate of tax (central tax, State tax, integrated tax, Union territory tax or cess);
- amount of tax charged in respect of taxable goods or services referred to in clause (a) above (central tax, State tax, integrated tax, Union territory tax or cess);
- place of supply along with the name of State, in case of a supply in the course of inter-State trade or commerce;
- address of delivery where the same is different from the place of supply;
- whether the tax is payable on reverse charge basis; and
- signature or digital signature of the supplier or his authorized representative:

Provided that the Commissioner may, on the recommendations of the Council, by notification, specify—

- the number of digits of HSN code for goods or the Accounting Code for services, that a class of registered persons shall be required to mention, for such period as may be specified in the said notification, and
- the class of registered persons that would not be required to mention the HSN code for goods or the Accounting Code for services, for such period as may be specified in the said notification:

Provided further that where an invoice is required to be issued under clause (f) of sub-section (3) of section 31, it shall bear the signature or digital signature of the recipient or his authorized representative:

1

1

2. **Time limit for issuing tax invoices**
The invoice referred to in rule 7, in case of taxable supply of services, shall be issued within a period of forty days from the date of supply of service.

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.8 Exporting a PDF

1.8.1 Exporting a PDF as JSON

This scenario is particularly useful when debugging. It enables you to see the PDF Document in the same way pText sees it.

We'll start by opening and reading the Document :

```
with open("input.pdf", "rb") as pdf_file_handle:  
    doc = PDF.loads(pdf_file_handle)  
    output_file = self.output_dir / (file.stem + ".json")
```

And that's all there is to it. Now we can call the method `to_json_serializable` on Document which will give you access to a json like structure.

```
# export to json  
with open("output.json", "w") as json_file_handle:  
    json_file_handle.write(  
        json.dumps(doc.to_json_serializable(doc), indent=4)  
    )
```

On my example input document, this yielded the following output:

```
{  
"null": {  
    "Trailer": {  
        "ID": [  
            "5e670a36ab70bb047b6c9eed6ee3892",  
            "5e670a36ab70bb047b6c9eed6ee3892"  
        ],  
        "Info": {
```

```

    "CreationDate": "D:20190409213301+02'00'",
    "ModDate": "D:20190409213301+02'00'",
    "Producer": "iText\u00ae 7.1.5 \u00a92000-2019 iText Group NV \\(AGPL-version\\)"
},
"Root": {
    "Pages": {
        "Count": 1.0,
        "Kids": [
            {
                "Type": "Page",
                "MediaBox": [
                    0.0,
                    0.0,
                    878.221,
                    637.276
                ],
...

```

Here we can clearly see the xref table being persisted. This table acts as the starting point of the document, it contains references to other data-structures that contain meta-information, information about each page, etc.

1.8.2 Exporting a PDF as SVG

Sometimes, all you need is an image. With `pText` you can easily convert any `Page` of a `Document` into an SVG image.

As usual, we start by reading the `Document`:

```

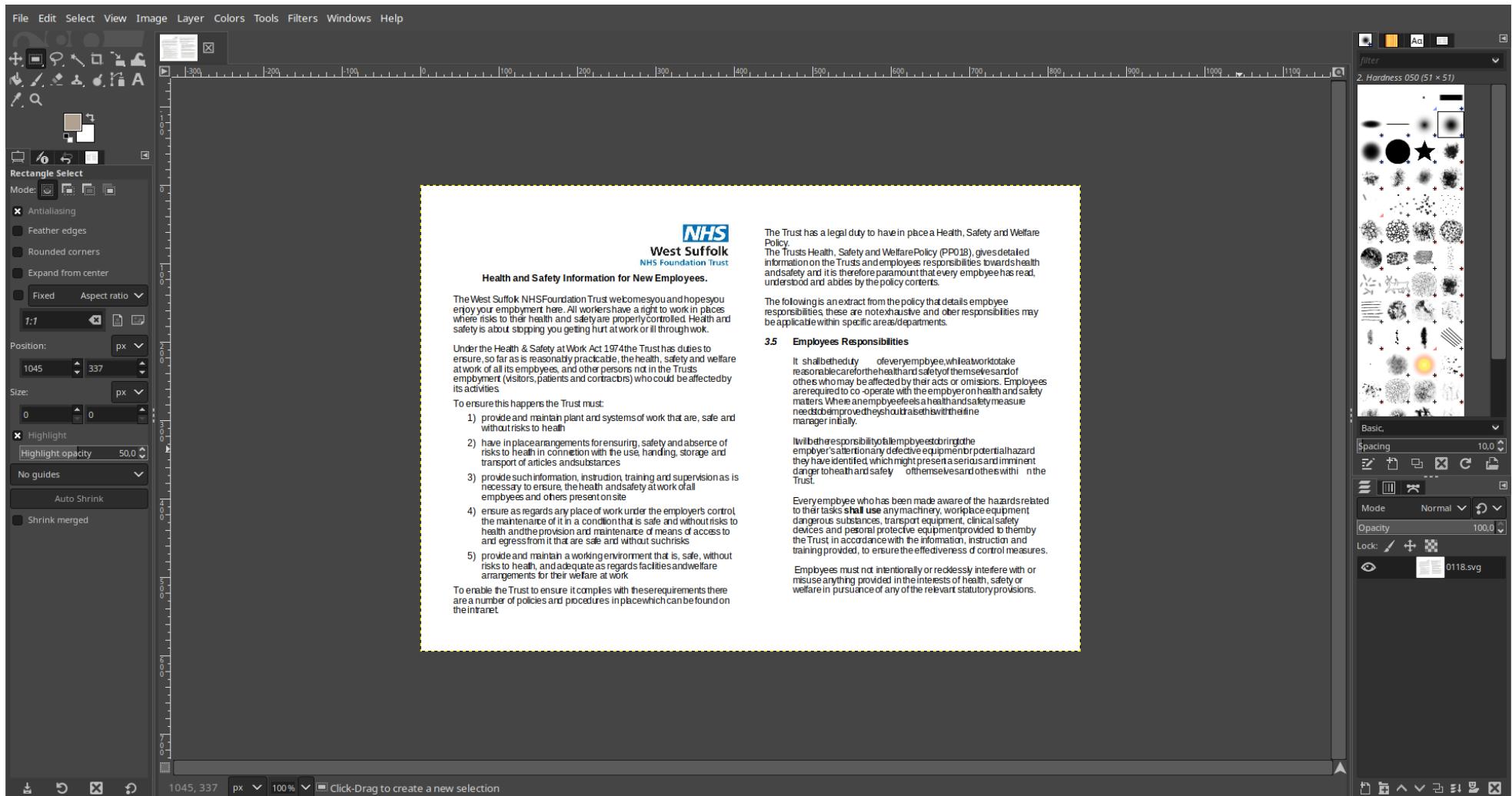
with open("input.pdf", "rb") as pdf_file_handle:
    l = PDFToSVG()
    doc = PDF.loads(pdf_file_handle, [l])

```

Here we are using `PDFToSVG` which acts like an `EventListener`. `EventListener` implementations are notified every time a rendering instruction is parsed. `PDFToSVG` uses that knowledge to convert the pdf-syntax rendering instructions to svg-syntax.

```
with open("output.svg", "wb") as svg_file_handle:  
    svg_file_handle.write(ET.tostring(l.get_svg_per_page(0)))
```

The result turned something like this:



This was the input document:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next Fit Width Zoom Out Zoom In Browse Text Selection

Thumbnails Contents

Search...

NHS
West Suffolk
NHS Foundation Trust

Health and Safety Information for New Employees.

The West Suffolk NHS Foundation Trust welcomes you and hopes you enjoy your employment here. All workers have a right to work in places where risks to their health and safety are properly controlled. Health and safety is about stopping you getting hurt at work or ill through work.

Under the Health & Safety at Work Act 1974 the Trust has duties to ensure, so far as is reasonably practicable, the health, safety and welfare at work of all its employees, and other persons not in the Trust's employment (visitors, patients and contractors) who could be affected by its activities.

To ensure this happens the Trust must:

- 1) provide and maintain plant and systems of work that are, safe and without risks to health
- 2) have in place arrangements for ensuring, safety and absence of risks to health in connection with the use, handling, storage and transport of articles and substances
- 3) provide such information, instruction, training and supervision as is necessary to ensure, the health and safety at work of all employees and others present on site
- 4) ensure as regards any place of work under the employer's control, the maintenance of it in a condition that is safe and without risks to health and the provision and maintenance of means of access to and egress from it that are safe and without such risks
- 5) provide and maintain a working environment that is, safe, without risks to health, and adequate as regards facilities and welfare arrangements for their welfare at work

To enable the Trust to ensure it complies with these requirements there are a number of policies and procedures in place which can be found on the intranet.

The Trust has a legal duty to have in place a Health, Safety and Welfare Policy.

The Trust's Health, Safety and Welfare Policy (PP018), gives detailed information on the Trusts and employees responsibilities towards health and safety and it is therefore paramount that every employee has read, understood and abides by the policy contents.

The following is an extract from the policy that details employee responsibilities, these are not exhaustive and other responsibilities may be applicable within specific areas/departments.

3.5 Employees Responsibilities

It shall be the duty of every employee, while at work to take reasonable care for the health and safety of themselves and of others who may be affected by their acts or omissions. Employees are required to co-operate with the employer on health and safety matters. Where an employee feels a health and safety measure needs to be improved they should raise this with their line manager initially.

It will be the responsibility of all employees to bring to the employer's attention any defective equipment or potential hazard they have identified, which might present a serious and imminent danger to health and safety of themselves and others within the Trust.

Every employee who has been made aware of the hazards related to their tasks **shall use** any machinery, workplace equipment, dangerous substances, transport equipment, clinical safety devices and personal protective equipment provided to them by the Trust, in accordance with the information, instruction and training provided, to ensure the effectiveness of control measures.

Employees must not intentionally or recklessly interfere with or misuse anything provided in the interests of health, safety or welfare in pursuance of any of the relevant statutory provisions.

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

1.8.2 Exporting a PDF as MP3

For those with hearing-impairments, it can be very useful to be able to convert a PDF Document to an MP3 file. This is perfectly possible with `pText`.

```
with open("input.pdf", "rb") as pdf_file_handle:  
    l = PDFToMP3()  
    doc = PDF.loads(pdf_file_handle, [l])
```

`PDFToMP3` then allows you to store an mp3 file for each page.

For this, you can use the `get_audio_file_per_page` method. You need to provide it with a `page_number` and `path`.

```
l.get_audio_file_per_page(0, "output.mp3")
```

The constructor of `PDFToMP3` has some arguments that allow us to tweak the export.

- `include_position` : This should be set to `True` if you want the position of each `Paragraph` to be spoken as well. This results in output such as "page 1, paragraph 1, top left; once upon a time"
- `language` : This is the 2-letter abbreviation of the language you expect the text to be in. Default is `en`
- `slow` : This indicates whether you want the speaking-voice to go (extra) slow, or not

1.9 Concatenating PDFs, and other page-manipulations

A common scenario, when working with existing PDF Document objects is concatenation. Let's look at how you can concatenate two or more existing Document objects:

1.9.1 Concatenating entire PDF Documents

```
# attempt to read PDF
doc_a = None
with open("input_a.pdf", "rb") as in_file_handle:
    doc_a = PDF.loads(in_file_handle)

# attempt to read PDF
doc_b = None
with open("input_b.pdf", "rb") as in_file_handle_b:
    doc_b = PDF.loads(in_file_handle_b)
```

Now we can simply call `append_document` on a new Document

```
# concat all pages to same document
doc_c = Document()
doc_c.append_document(doc_a)
doc_c.append_document(doc_b)
```

And finally store the merged PDF:

```
# attempt to store PDF
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc_c)
```

1.9.2 Concatenating parts of a Document

```
# attempt to read PDF
doc_a = None
with open("input_a.pdf", "rb") as in_file_handle:
```

```
doc_a = PDF.loads(in_file_handle)

# attempt to read PDF
doc_b = None
with open("input_b.pdf", "rb") as in_file_handle_b:
    doc_b = PDF.loads(in_file_handle_b)
```

In stead of calling `append_document` , we can select `Page` objects, and call `append_page` . In fact `append_document` is just a shortcut for repeatedly calling `append_page`

```
# concat all pages to same document
doc_c = Document()
for i in range(0, int(doc_a.get_document_info().get_number_of_pages())):
    doc_c.append_page(doc_a.get_page(i))
for i in range(0, int(doc_b.get_document_info().get_number_of_pages())):
    doc_c.append_page(doc_b.get_page(i))
```

And finally we can store the merged PDF:

```
# attempt to store PDF
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc_c)
```

1.9.3 Removing a Page from a Document

First, we open the Document

```
doc = None
with open("input.pdf", "rb") as in_file_handle:
    print("\treading (1) ..")
    doc = PDF.loads(in_file_handle)
```

Let's check the number of pages

```
number_of_pages = int(doc.get_document_info().get_number_of_pages())
print(number_of_pages)
```

Now we can remove the first Page

```
# remove page
doc.pop_page(0)
```

And finally we store the modified Document

```
# attempt to store PDF
with open("output.pdf", "wb") as out_file_handle:
    PDF.dumps(out_file_handle, doc)
```

2. PDF Creation

2.0 Creating an empty PDF

This basic example gives you an idea of how to create a `Document` using `pText`. Other examples will show you how to add rich content to it.

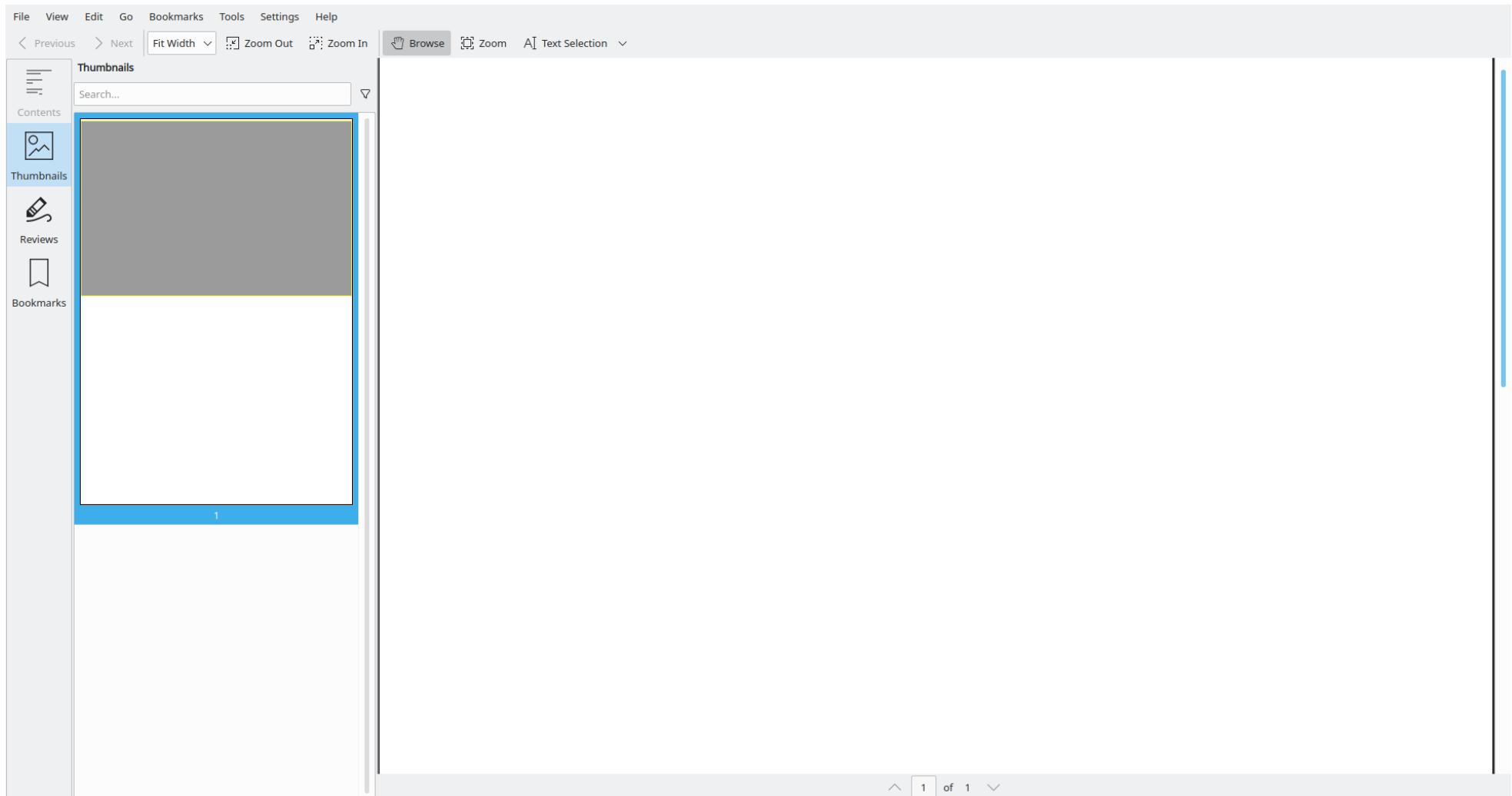
```
# create empty document
pdf: Document = Document()

# create empty page
page: Page = Page()

# add page to document
pdf.append_page(page)

# write
with open("output.pdf", "wb") as pdf_file_handle:
    PDF.dumps(pdf_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1 Adding text to a Document

2.1.1 Adding text to a Document using low-level commands

This example describes how to create a PDF from scratch. This example focuses on giving the reader an understanding of the underlying PDF syntax. This is definitely not the easiest way to write text in a PDF.

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

This is where the actual content generation begins. To get content on a `Page` we need to alter its content-stream. First we'll create a content stream, and then we'll set its bytes to the appropriate operators to write 'Hello World!'

```
# create content stream
content_stream = Stream()
content_stream[
    Name("DecodedBytes")
] = b"""
    q
    BT
    /F1 24 Tf
    100 742 Td
    (Hello World!) Tj
    ET
    Q
"""
....
```

The `q` and `Q` operator define a context in which we can work. These operators respectively push and pop the entire graphics state unto/from a stack. By doing so, we can ensure our content will not interfere with other content that may exist on the page.

Next we have the `BT` (begin text) and `ET` (end text) operators. They set up everything to enable us to write text. `Tf` sets the font (in this case `F1`) and font-size.

`Td` determines the position at which we will draw text. `Tj` writes a string (enclosed in round brackets) to the PDF.

Next we need to set the properties of the content-stream to match its content. In this example we'll encode the bytes using `FlateDecode`. Thus we need to provide a `Filter` property (so the reader knows which decompression algorithm to use), and provide a `Length` (so the reader knows how long our encoded byte-stream is).

```
content_stream[Name("Bytes")] = zlib.compress(content_stream["DecodedBytes"], 9)
content_stream[Name("Filter")] = Name("FlateDecode")
content_stream[Name("Length")] = Decimal(len(content_stream["Bytes"]))
```

Next we can set this `Stream` to be the `Contents` of the `Page`

```
# set content of page
page[Name("Contents")] = content_stream
```

In the following code-snippet, we set every property related to the font we used. We need to specify the font used by the `Tj` operator in the `Resources` dictionary of the `Page`.

```
# set Font
page[Name("Resources")] = Dictionary()
page["Resources"][Name("Font")] = Dictionary()
page["Resources"]["Font"][Name("F1")] = Dictionary()
```

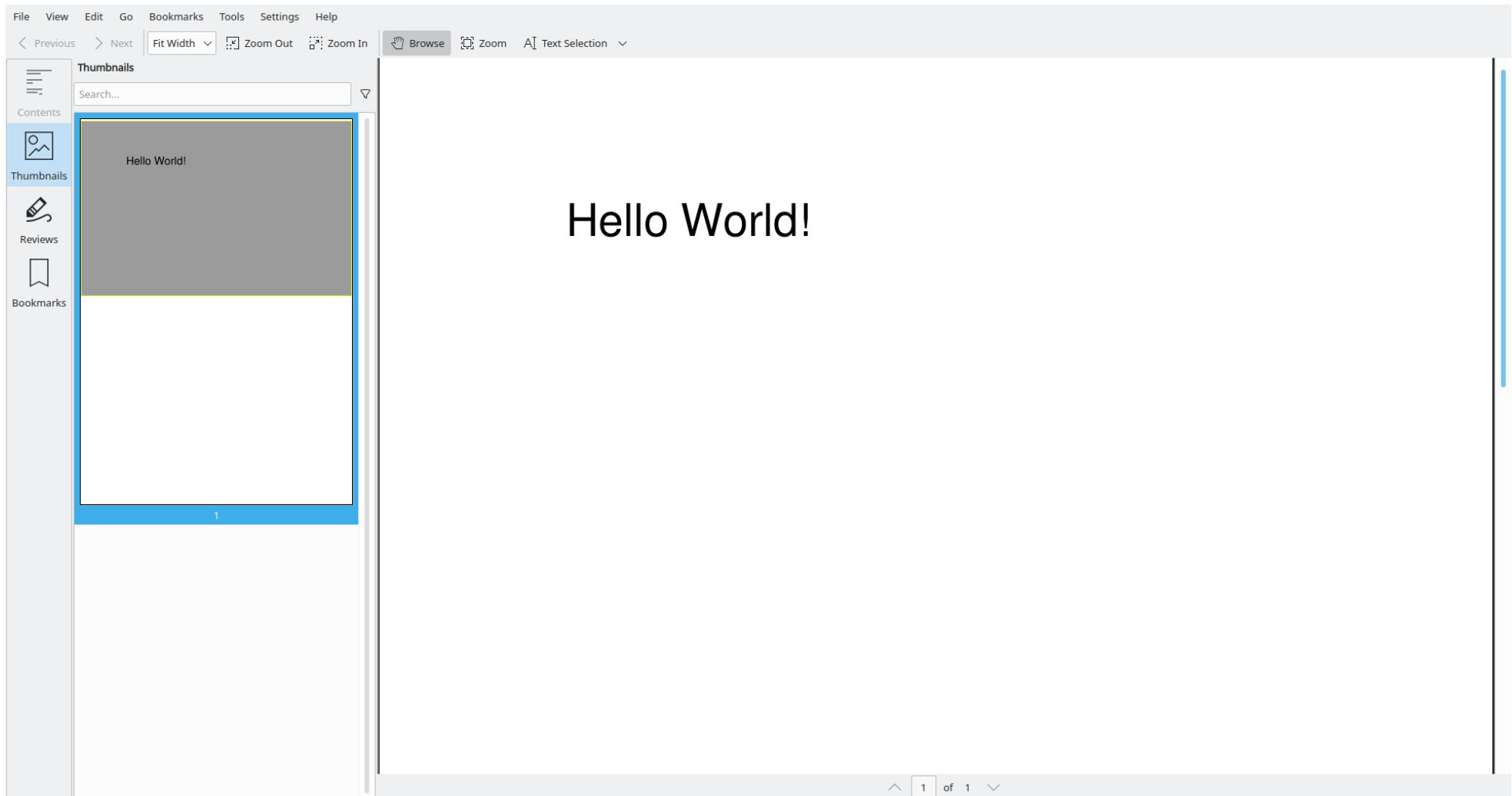
```
page["Resources"]["Font"]["F1"][Name("Type")] = Name("Font")
page["Resources"]["Font"]["F1"][Name("Subtype")] = Name("Type1")
page["Resources"]["Font"]["F1"][Name("Name")] = Name("F1")
page["Resources"]["Font"]["F1"][Name("BaseFont")] = Name("Helvetica")
page["Resources"]["Font"]["F1"][Name("Encoding")] = Name("MacRomanEncoding")
```

In this example I chose Helvetica, because the reader is supposed to know all the details of this font (width of every glyph, bounding box, etc). That means we don't have to specify all the details. In the above code-snippet, we only really mentioned the name and character encoding.

Next we store the PDF.

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



2.1.2 Adding text to a Document using ChunkOfText

Luckily, there is an easier way to get content on a PDF. Let's look at the convenience classes `pText` provides.

We'll start similar to our previous example, by creating an empty `Document` and `Page`.

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

Now instead of having to figure out all these instructions ourselves, we can let `pText` do the heavy lifting. Here we add a `ChunkOfText` to the `Page`, but other classes allow you to add lines of text, paragraphs, tables, etc.

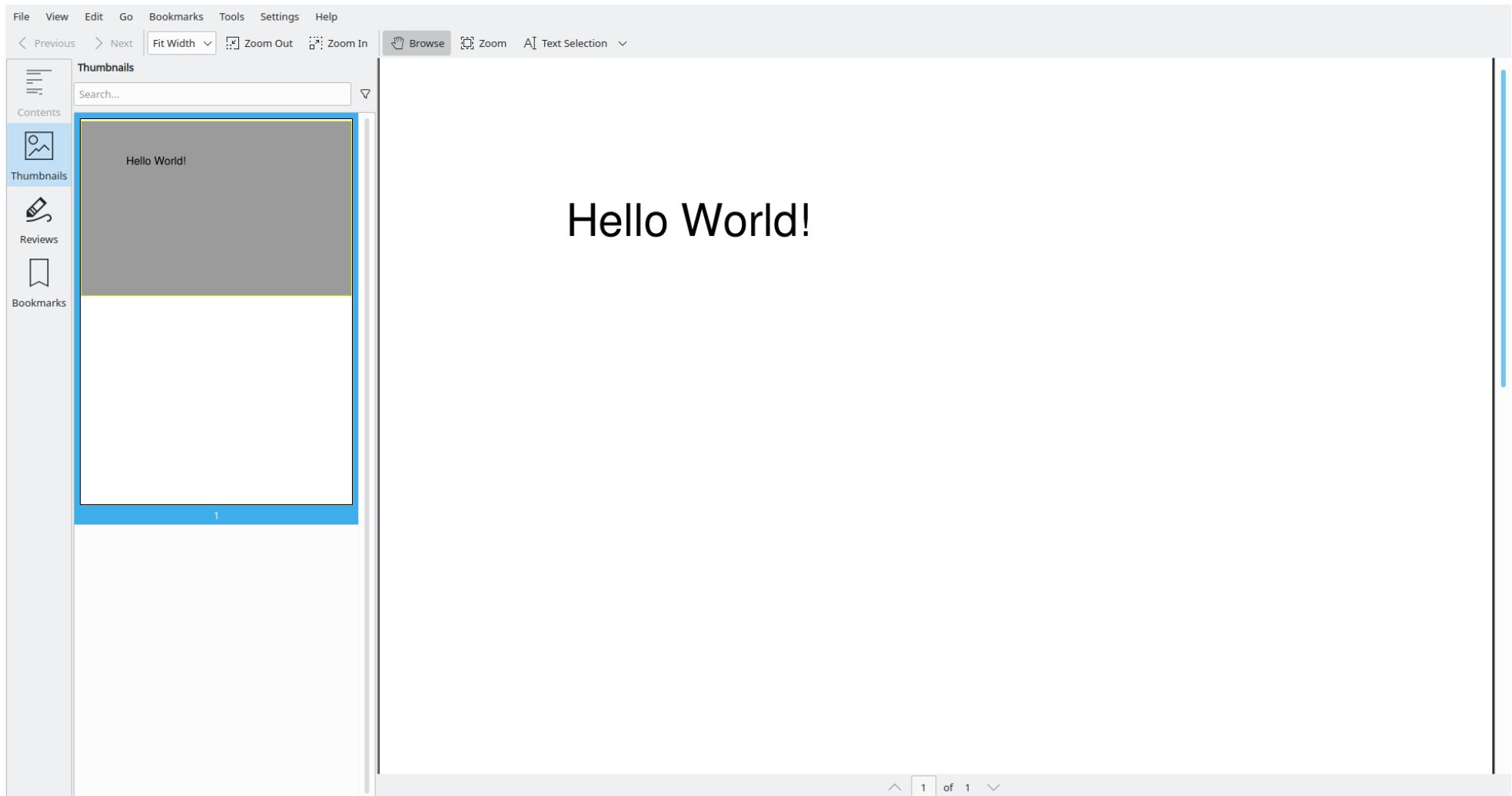
```
ChunkOfText(
    "Hello World!", font_size=Decimal(24)
).layout(
    page, Rectangle(Decimal(100), Decimal(724), Decimal(100), Decimal(100))
)
```

`ChunkOfText` allows us to specify the `font_size`, `Color` and `font`. If not provided, `ChunkOfText` defaults to black Helvetica, size 12. We then call `layout` on this object to have it put on the `Page`.

Finally, we can store the PDF.

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.2 (ctd) Adding text to a Document using ChunkOfText

Let's add some color to the `Document` this time:

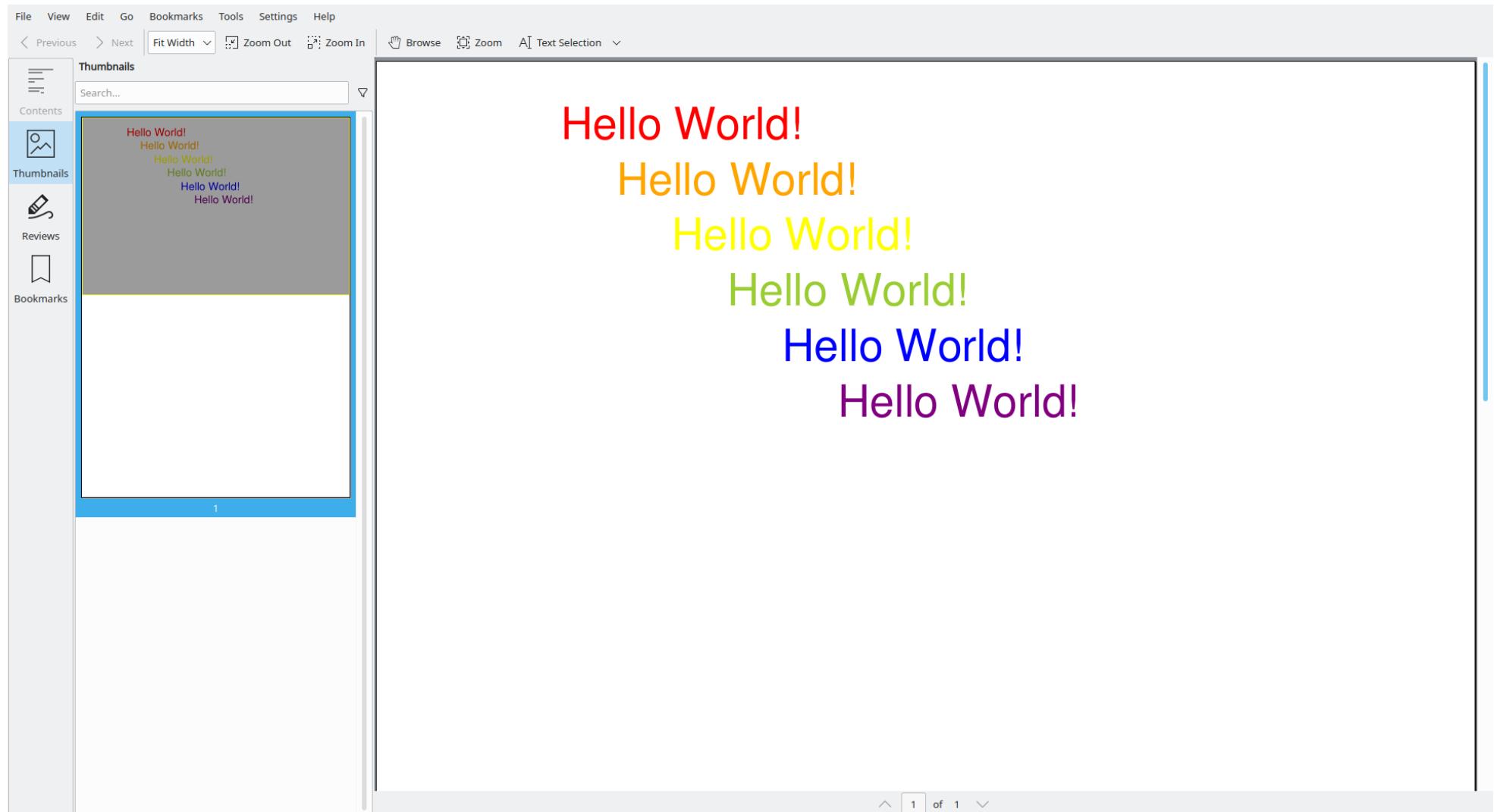
```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)

for i, c in enumerate(
    [
        X11Color("Red"),
        X11Color("Orange"),
        X11Color("Yellow"),
        X11Color("YellowGreen"),
        X11Color("Blue"),
        X11Color("Purple"),
    ]
):
    ChunkOfText("Hello World!", font_size=Decimal(24), color=c).layout(
        page,
        Rectangle(
            Decimal(100 + i * 30), Decimal(724 - i * 30), Decimal(100), Decimal(100)
        ),
    )

# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.3 Adding text to a Document using `LineOfText`

By using `LineOfText` we can add `Alignment.(left, center, right, full)` to our text. We start by creating an empty `Document` (just like the other examples).

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

Here we're going to add 4 lines of text, all of them will be justified `RIGHT`. That means we're going to give them all the same bounding box (apart from the y-coordinate), and have `pText` work out where to start the text to achieve the correct Alignment.

```
for i, s in enumerate(
    [
        "Once upon a midnight dreary,",
        "while I pondered weak and weary,",
        "over many a quaint and curious",
        "volume of forgotten lore",
    ]
):
    LineOfText(
        s,
        font_size=Decimal(20),
        horizontal_alignment=Alignment.RIGHT,
    ).layout(
        page,
        Rectangle(
            Decimal(20), Decimal(724 - 24 * i), Decimal(500), Decimal(124)
        ),
    )
```

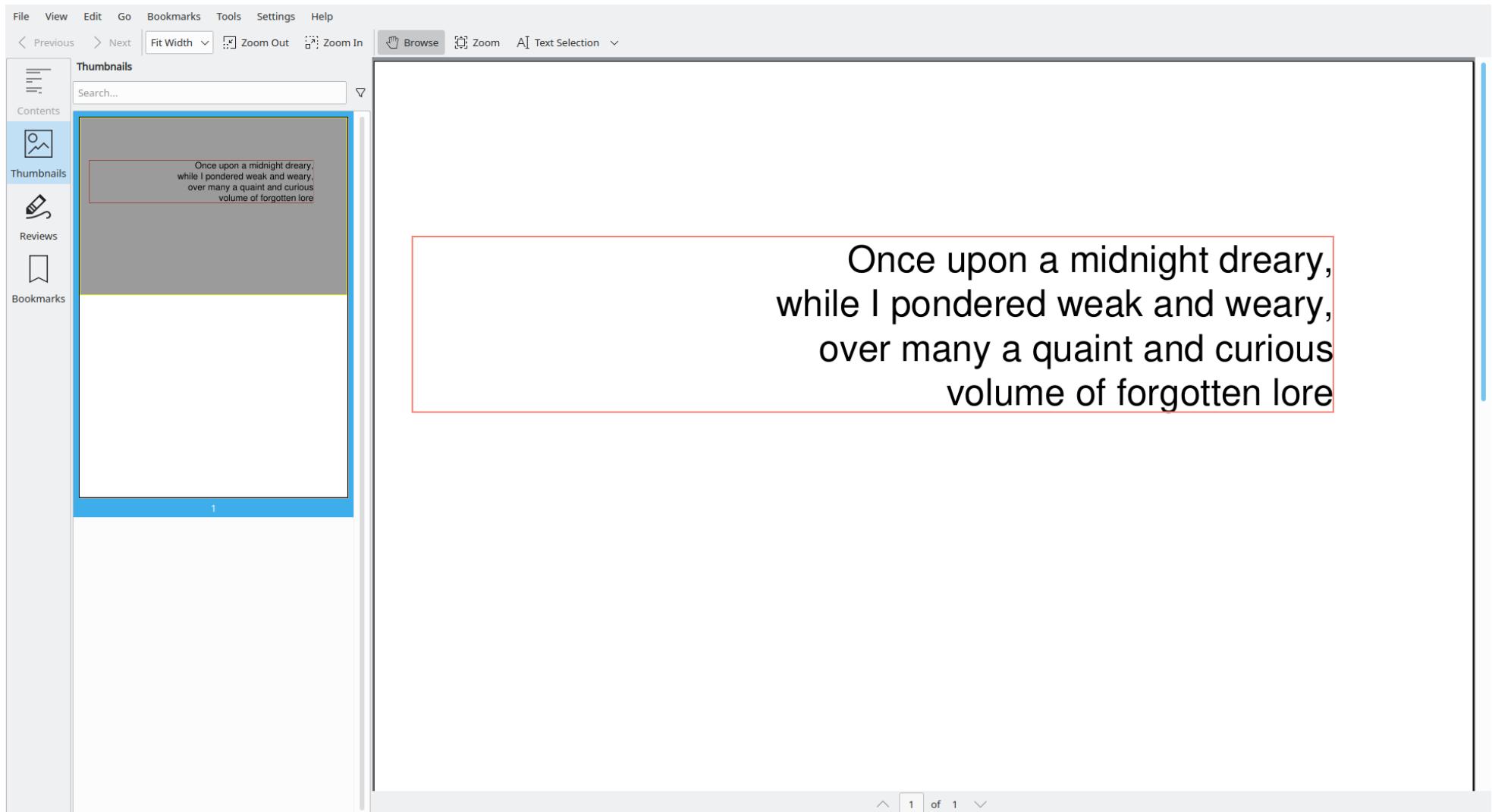
We are also going to add a rectangle annotation, to give us a rough idea of the bounding box of the text.

```
# add rectangle annotation
page.append_square_annotation(
    stroke_color=X11Color("Red"),
    rectangle=Rectangle(
        Decimal(20), Decimal(724 - 24 * 4), Decimal(500), Decimal(24 * 4)
    ),
)
```

Finally, we can store the Document

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.3 Adding text to a Document using Paragraph

2.1.3.1 Basic example

This is by far the easiest way to add text to a page. Let's start by creating an empty `Document`.

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

Next we define the text we want to add.

```
s = "Once upon a midnight dreary, while I pondered weak and weary, over many a quaint and curious volume of
forgotten lore"
```

And now we construct a `Paragraph` object from that text, we are also going to set its `color`, `Alignment`. and `font_size``.

```
Paragraph(
    s,
    font_size=Decimal(20),
).layout(
    page,
    Rectangle(Decimal(20), Decimal(724), Decimal(400), Decimal(124)),
)
```

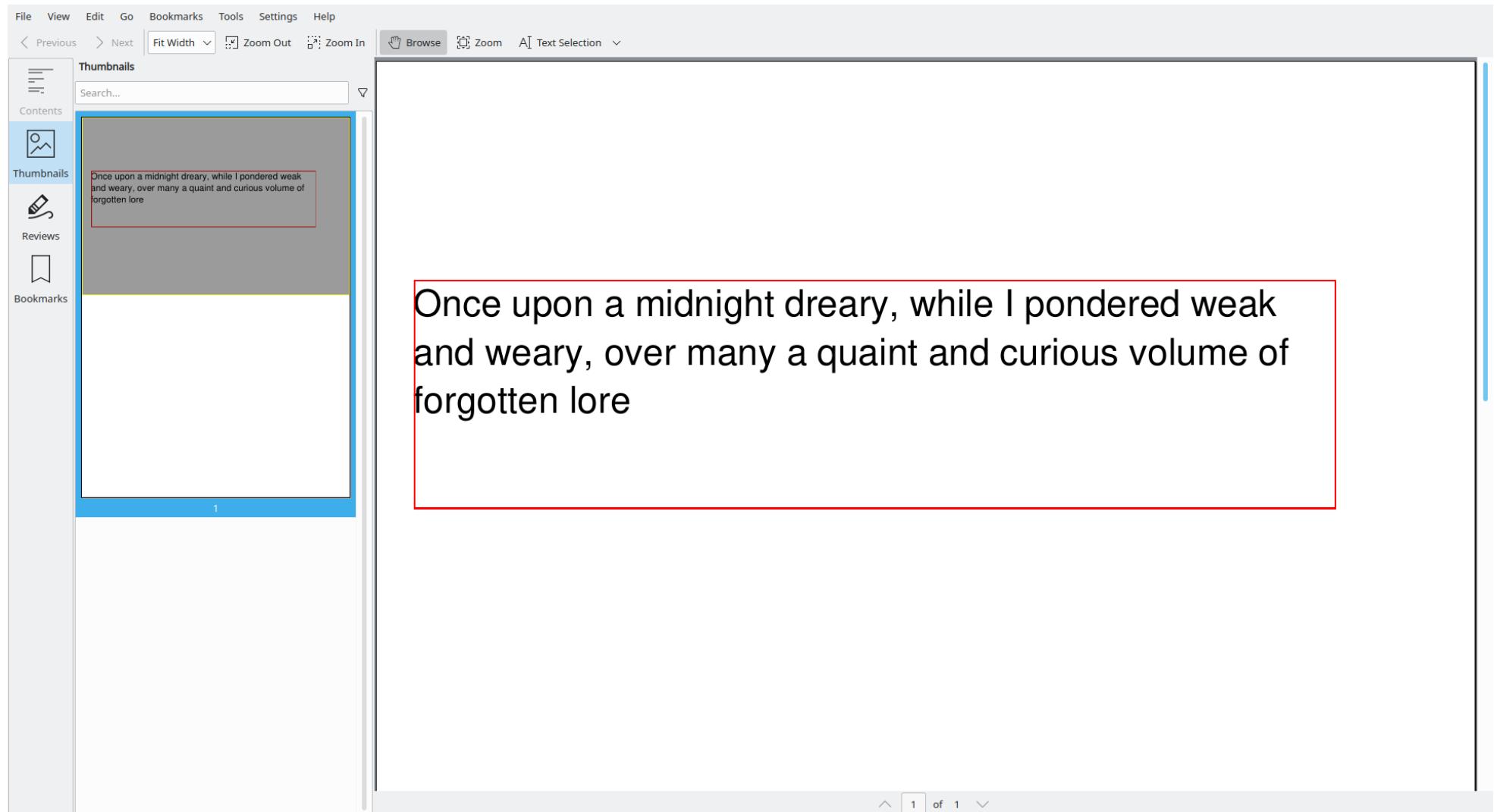
We are also going to add a rectangle annotation, to visually mark the boundaries of the box that we want our paragraph to be in.

```
# add rectangle annotation
page.append_square_annotation(
    stroke_color=X11Color("Red"),
    rectangle=Rectangle(
        Decimal(20), Decimal(724 - 124), Decimal(400), Decimal(124)
    ),
)
```

Lastly, we write the Document .

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



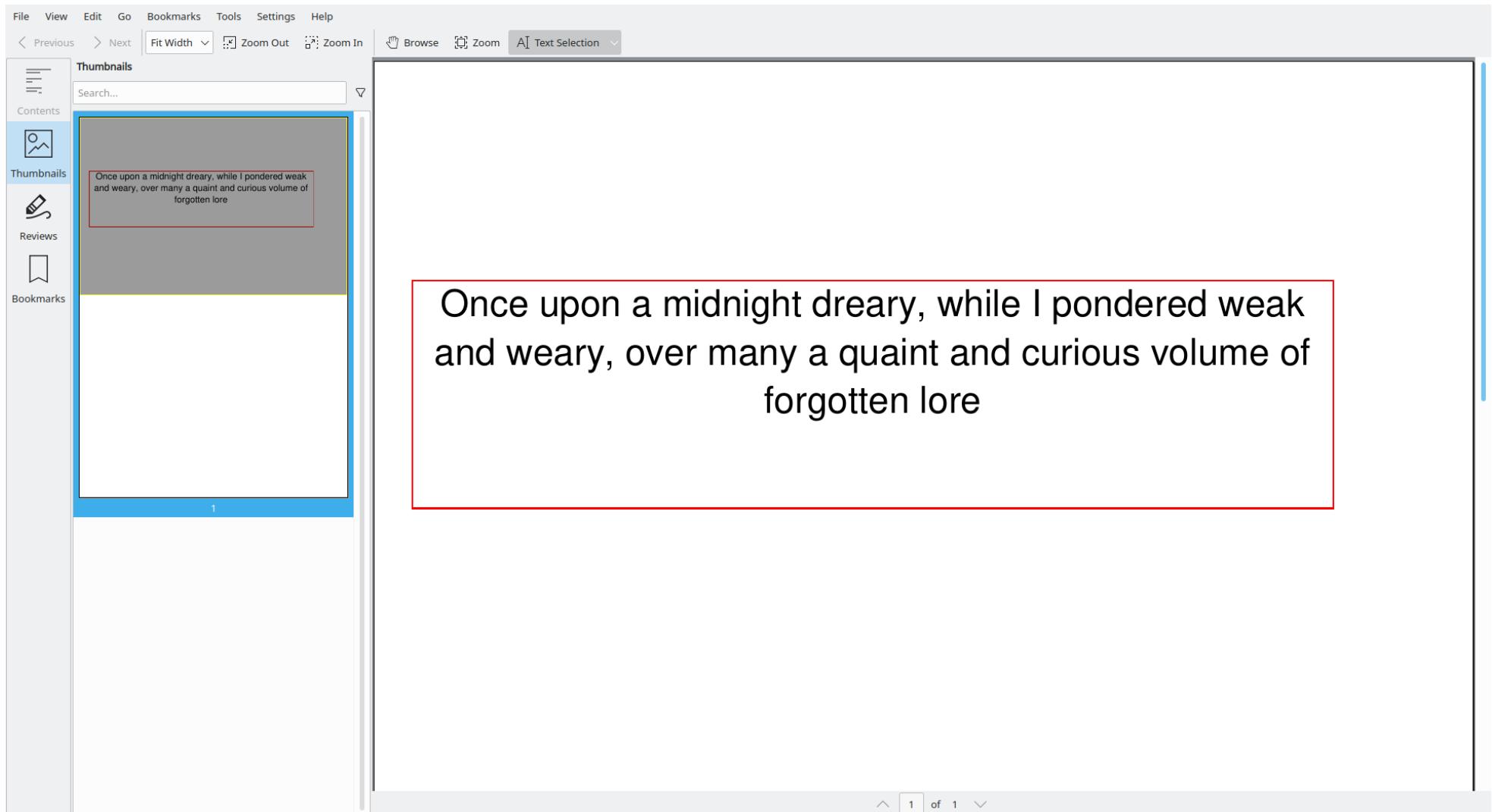
Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.3.2 Setting justification

Let's change the code we wrote earlier to have the Paragraph alignment CENTERED

```
Paragraph(  
    "Once upon a midnight dreary, while I pondered weak and weary, over many a quaint and curious volume of  
forgotten lore",  
    font_size=Decimal(20),  
    font_color=X11Color("YellowGreen"),  
    horizontal_alignment=Alignment.CENTERED,  
) . layout(  
    page,  
    Rectangle(Decimal(20), Decimal(600), Decimal(500), Decimal(124)),  
)
```

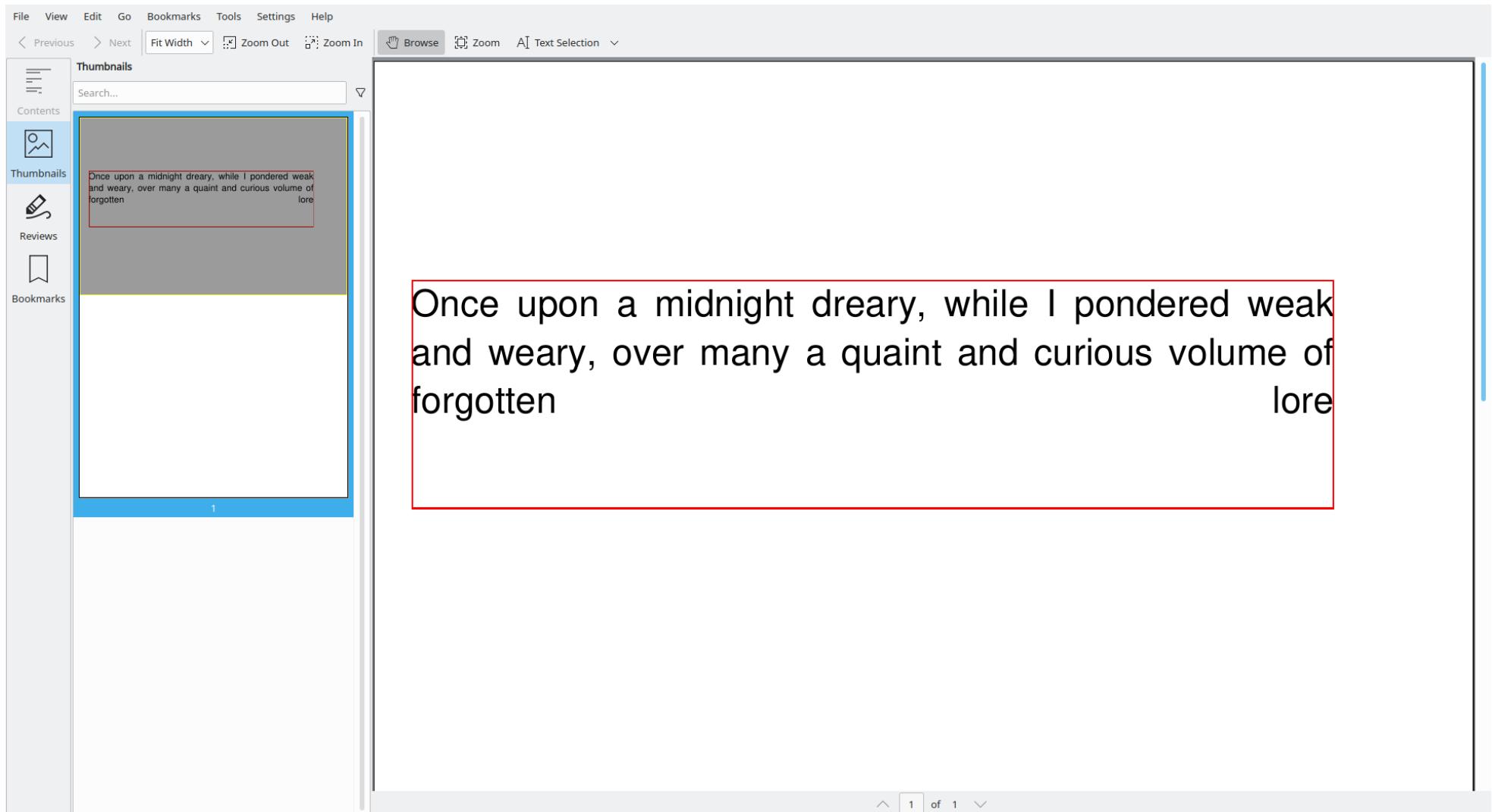
The result should be something like this:



We can do the same for alignment FULL

```
Paragraph(  
    "Once upon a midnight dreary, while I pondered weak and weary, over many a quaint and curious volume of  
forgotten lore",  
    font_size=Decimal(20),  
    font_color=X11Color("YellowGreen"),  
    horizontal_alignment=Alignment.JUSTIFIED,  
).layout(  
    page,  
    Rectangle(Decimal(20), Decimal(600), Decimal(500), Decimal(124)),  
)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.3.3 Setting padding

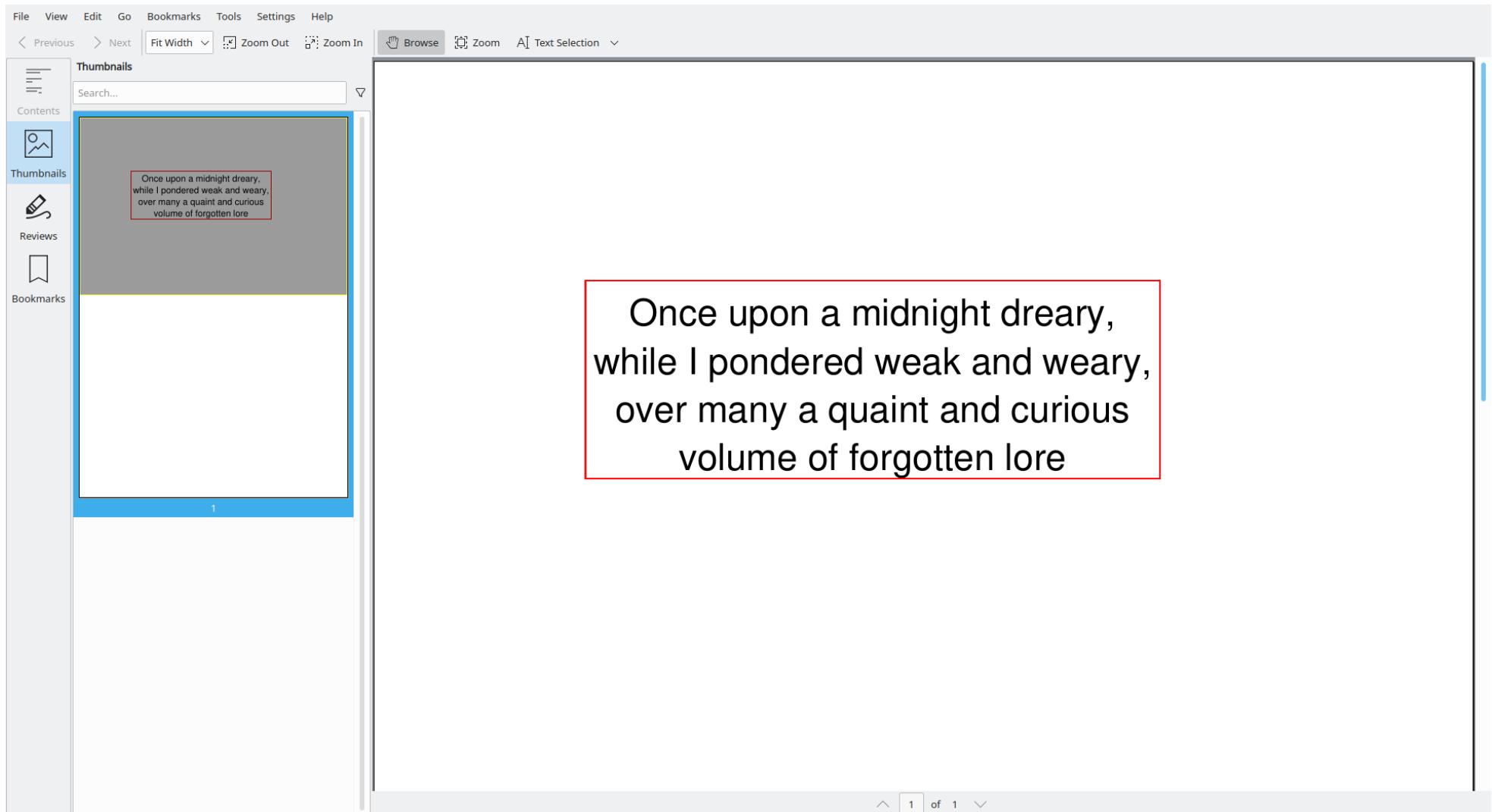
Let's change the code we wrote earlier to have the `Paragraph` alignment `CENTERED`. This time, we're also going to set some padding. This will ensure the `Paragraph` stays away from its bounding box.

```
padding: Decimal = Decimal(5)
Paragraph(
    "Once upon a midnight dreary, while I pondered weak and weary, over many a quaint and curious volume of
forgotten lore",
    font_size=Decimal(20),
    font_color=X11Color("YellowGreen"),
    horizontal_alignment=Alignment.CENTERED,
    padding_top=padding,
    padding_right=padding,
    padding_bottom=padding,
    padding_left=padding,
).layout(
    page,
    Rectangle(Decimal(20), Decimal(600), Decimal(500), Decimal(124)),
)
```

We're going to add a rectangle annotation around the result of the `layout` method. Typically the `layout` method returns the bounding box that was occupied after having performed layout of a given `LayoutElement`.

```
# add rectangle annotation
page.append_square_annotation(
    stroke_color=X11Color("Red"), rectangle=layout_rect
)
```

The result should be something like this:



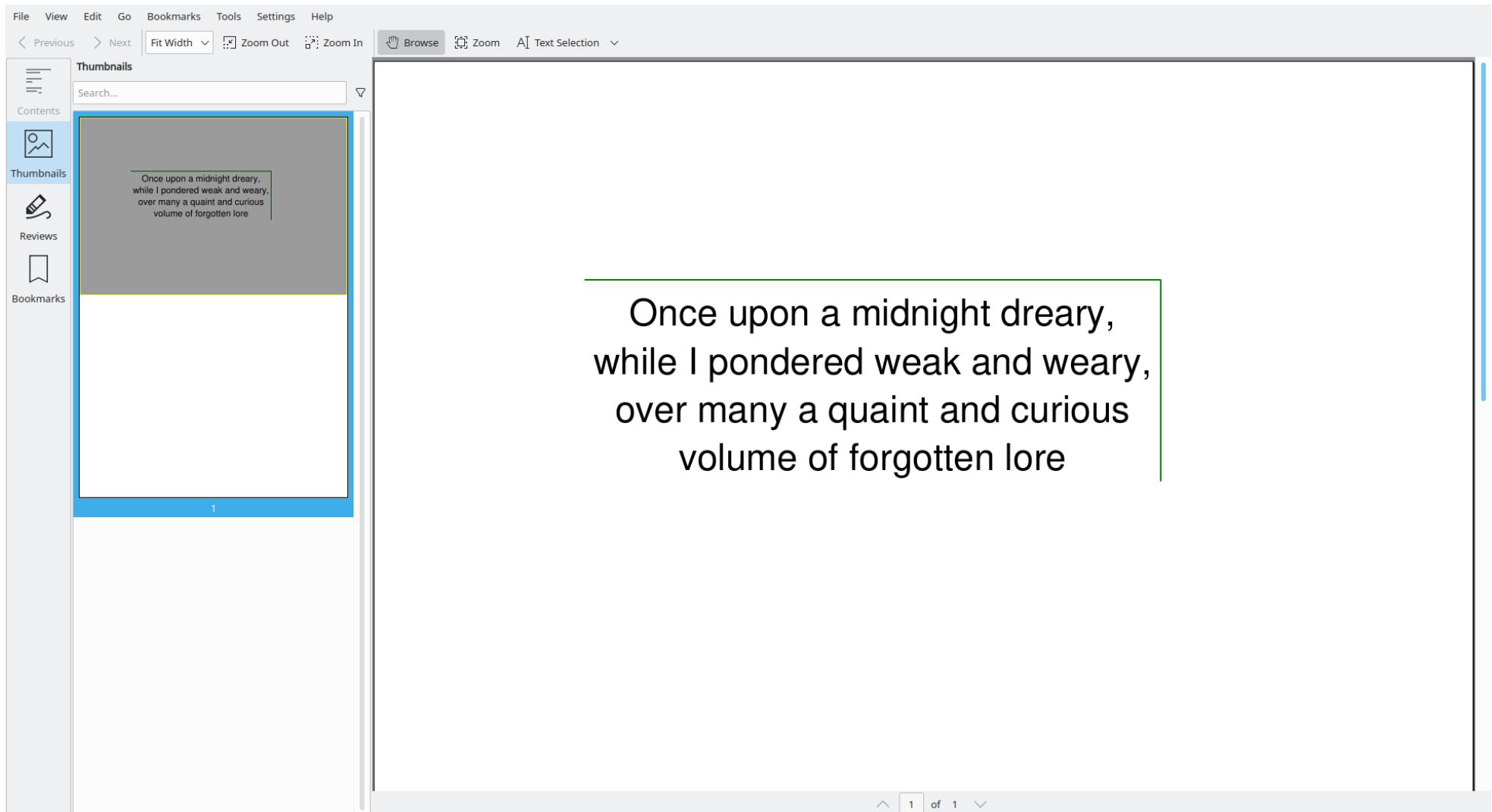
Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.3.4 Setting borders

pText also allows you to set borders on any LayoutElement Let's try that:

```
padding = Decimal(5)
layout_rect = Paragraph(
    "Once upon a midnight dreary,\nwhile I pondered weak and weary,\nover many a quaint and curious\nvolume of
forgotten lore",
    font_size=Decimal(20),
    horizontal_alignment=Alignment.CENTERED,
    respect_newlines_in_text=True,
    padding_top=padding,
    padding_right=padding,
    padding_bottom=padding,
    padding_left=padding,
    border_right=True,
    border_top=True,
    border_color=X11Color("Green")
).layout(
    page,
    Rectangle(Decimal(20), Decimal(600), Decimal(500), Decimal(124)),
)
```

The result should be something like this:



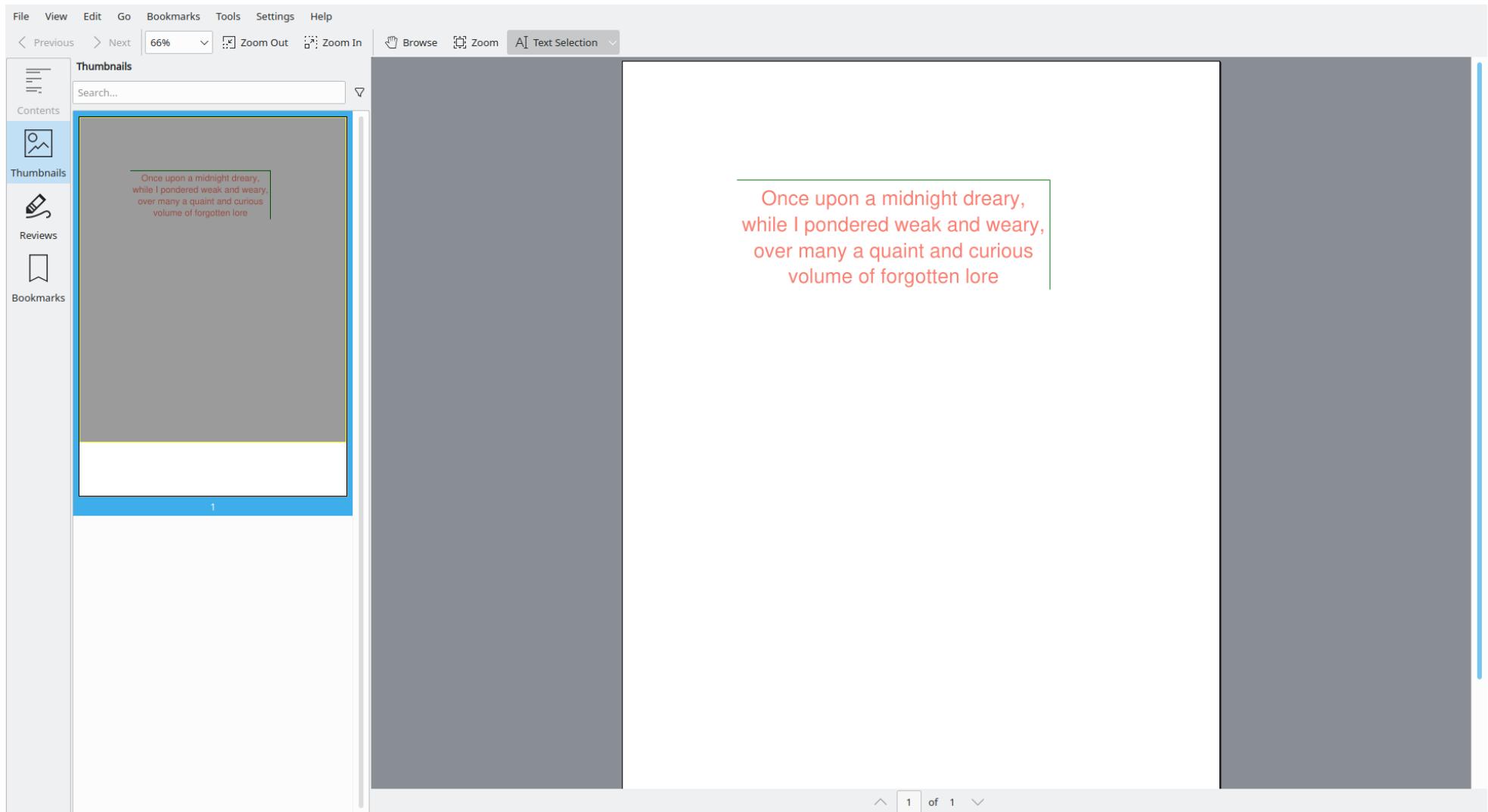
Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.3.5 Setting color

Black is boring. Let's set the `font_color` to `salmon` for a change:

```
padding = Decimal(5)
layout_rect = Paragraph(
    "Once upon a midnight dreary,\\nwhile I pondered weak and weary,\\nover many a quaint and curious\\nvolume of
forgotten lore",
    font_size=Decimal(20),
    horizontal_alignment=Alignment.CENTERED,
    respect_newlines_in_text=True,
    padding_top=padding,
    padding_right=padding,
    padding_bottom=padding,
    padding_left=padding,
    border_right=True,
    border_top=True,
    border_color=X11Color("Green"),
    font_color=X11Color("Salmon")
).layout(
    page,
    Rectangle(Decimal(20), Decimal(600), Decimal(500), Decimal(124)),
)
```

The result should be something like this (maybe salmon was not the greatest colour in the world):



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.3.6 Forcing a split

Sometimes you'd like to force a certain split on a `Paragraph`. The default behaviour for `Paragraph` is to ignore whitespaces, and decide (based on the bounding box of the layout) where to start a new line.

But, by tweaking the setting `respect_newlines_in_text` we can tell the `Paragraph` to respect newlines.

We'll start by creating a new `Document`:

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

Now we can add the title `Paragraph`

```
layout = MultiColumnLayout(page, number_of_columns=2)
layout.add(Paragraph("The Raven", font_size=Decimal(20), font="Helvetica-Oblique",
font_color=HexColor("708090")))
```

Finally, we add the `Paragraph`

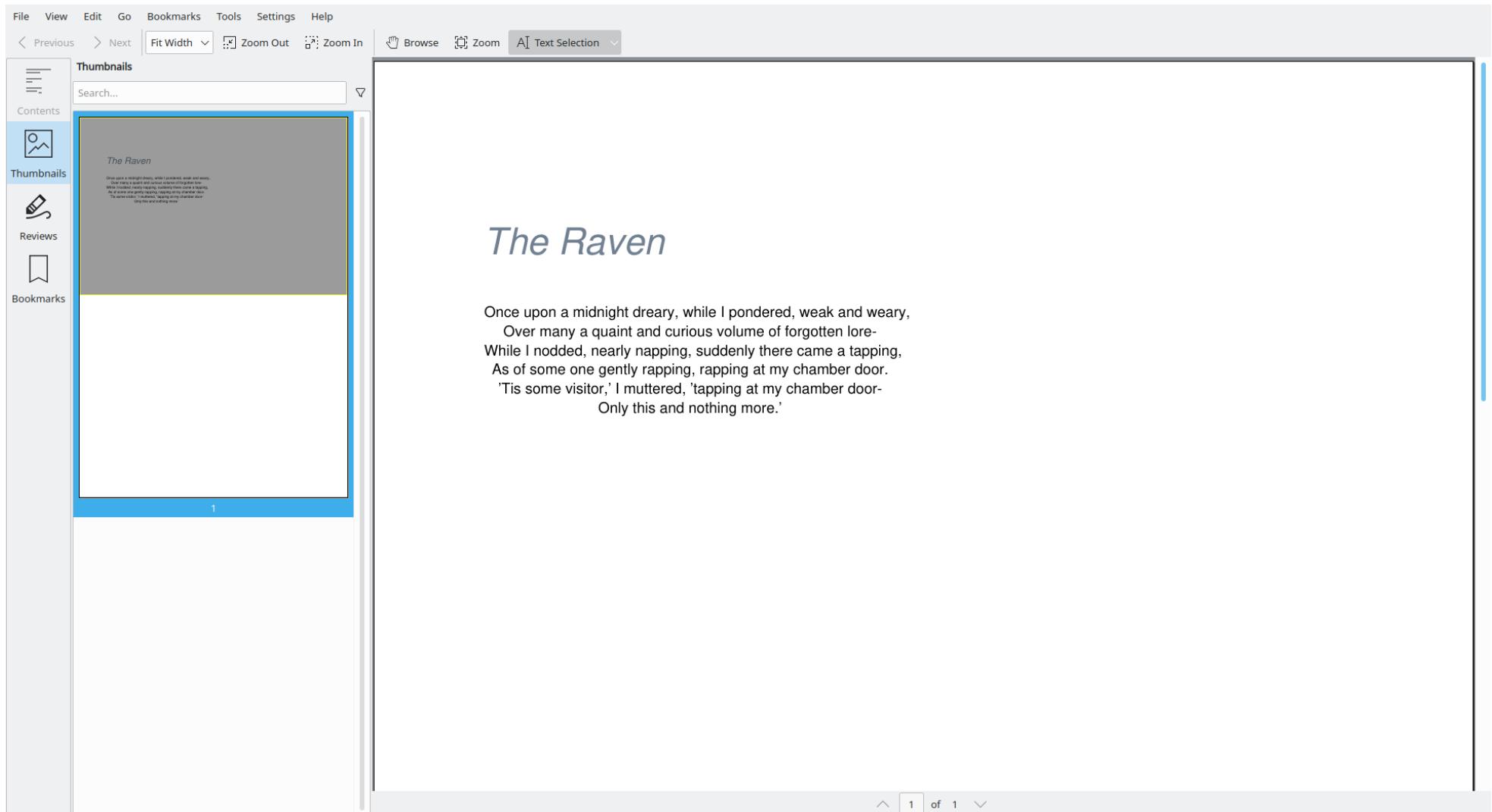
```
layout.add(Paragraph("""Once upon a midnight dreary, while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore-
While I nodded, nearly napping, suddenly there came a tapping,
As of some one gently rapping, rapping at my chamber door.
'Tis some visitor,' I muttered, 'tapping at my chamber door-
Only this and nothing more.'""",
horizontal_alignment=Alignment.CENTERED,
```

```
font_size=Decimal(8),  
respect_newlines_in_text=True))
```

Now we can store the Document

```
# attempt to store PDF  
with open("output.pdf", "wb") as in_file_handle:  
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.1.4 Adding text to a Document using Heading

A `Heading` acts like any other `Paragraph` object, at least visually it does.

So what's the big deal then?

`Heading` objects also modify the `\outlines` dictionary of the `Document`. This dictionary is responsible for the side-menu you might see in Adobe Reader, displaying titles, and allowing you to quickly navigate a `Document`.

Let's create an example. We'll start by creating an empty `Document`.

```
pdf = Document()
page = Page()
pdf.append_page(page)
layout = MultiColumnLayout(page, number_of_columns=2)
```

We're going to add our first `Heading`. `Heading` accepts the same arguments as `Paragraph`, and its layout works exactly the same.

`Heading` also takes a few (optional) extra arguments. `outline_text` allows you to set a text to be used in the side-menu. If you don't specify anything, the text in the `Paragraph` will be used. `outline_level` allows you to go deeper in the tree-hierarchy (or return to a parent). The default (`Document`) level is 0.

```
layout.add(Heading("The Raven", font_size=Decimal(20)))
layout.add(
    Paragraph(
        "Edgar Allen Poe",
        font="Helvetica-Oblique",
        font_size=Decimal(8),
        font_color=X11Color("SteelBlue"),
    )
)
```

Next we're going to add 100 `Heading` objects, followed by a random number of `Paragraph` objects.

```
for i in range(0, 100):
    layout.add(Heading("Heading %d" % i, font_size=Decimal(20), outline_level=1))
```

Notice that I have set the `outline_level` to 1 here.

```
for _ in range(0, random.choice([10,20,3])):
    layout.add(
        Paragraph(
            "Once upon a midnight dreary, while I pondered, weak and weary, Over many a quaint and curious
volume of forgotten lore- While I nodded, nearly napping, suddenly there came a tapping, As of some one gently
rapping, rapping at my chamber door. Tis some visitor, I muttered, tapping at my chamber door- Only this and
nothing more.",
            font_size=Decimal(12),
            font_color=X11Color("SlateGray"),
            horizontal_alignment=Alignment.LEFT,
        )
    )
```

Finally we're going to store the Document

```
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:

The screenshot shows a document editor interface with a toolbar at the top and a sidebar on the left. The sidebar contains icons for Contents, Thumbnails, Reviews, and Bookmarks, with 'Contents' being the active tab. A search bar is also present in the sidebar.

The main area displays a large block of text repeated four times across the page. The first instance is associated with 'Heading 45' in the table of contents. The text is a famous poem by Edgar Allan Poe:

Once upon a midnight dreary, while I
pondered, weak and weary, Over many a
quaint and curious volume of forgotten
lore- While I nodded, nearly napping,
suddenly there came a tapping, As of
some one gently rapping, rapping at my
chamber door. Tis some visitor, I
muttered, tapping at my chamber door-
Only this and nothing more.

The heading 'Heading 45' is highlighted in the table of contents, indicating it is the current section being viewed. The page number 59 is visible at the bottom center of the screen.

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.2 Using a `PageLayout`

So far we've used absolute positioning whenever we wanted to add something to a `Page`. Although this gives us precise control over where the content needs to go, it makes it harder to add multiple `LayoutElement` objects.

Luckily, `pText` comes with various `PageLayout` classes. These keep track of what parts of a `Page` are free, and where to flow content to.

2.2.1 Using `SingleColumnLayout`

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

We'll use `SingleColumnLayout` which takes into account top-, bottom-, left- and right-margins and lays out the content of the `Page` by adding each `LayoutElement` from top to bottom. It adds leading between every 2 elements, based on the `LayoutElement`. For text-based elements, this is typically a multiplied leading of 1.3 (meaning the leading after a `Paragraph` with `font_size` 10 will be 13).

```
layout = SingleColumnLayout(page)
```

Now that we've created a `PageLayout` we can simply call its `add` method. It will keep track of where each `LayoutElement` is, and will calculate the next available `Rectangle` whenever a new `LayoutElement` is added.

```
layout.add(Paragraph(
    "Once upon a midnight dreary, while I pondered weak and weary, over many a quaint and curious volume of
forgotten lore.",
    font_size=Decimal(20),
```

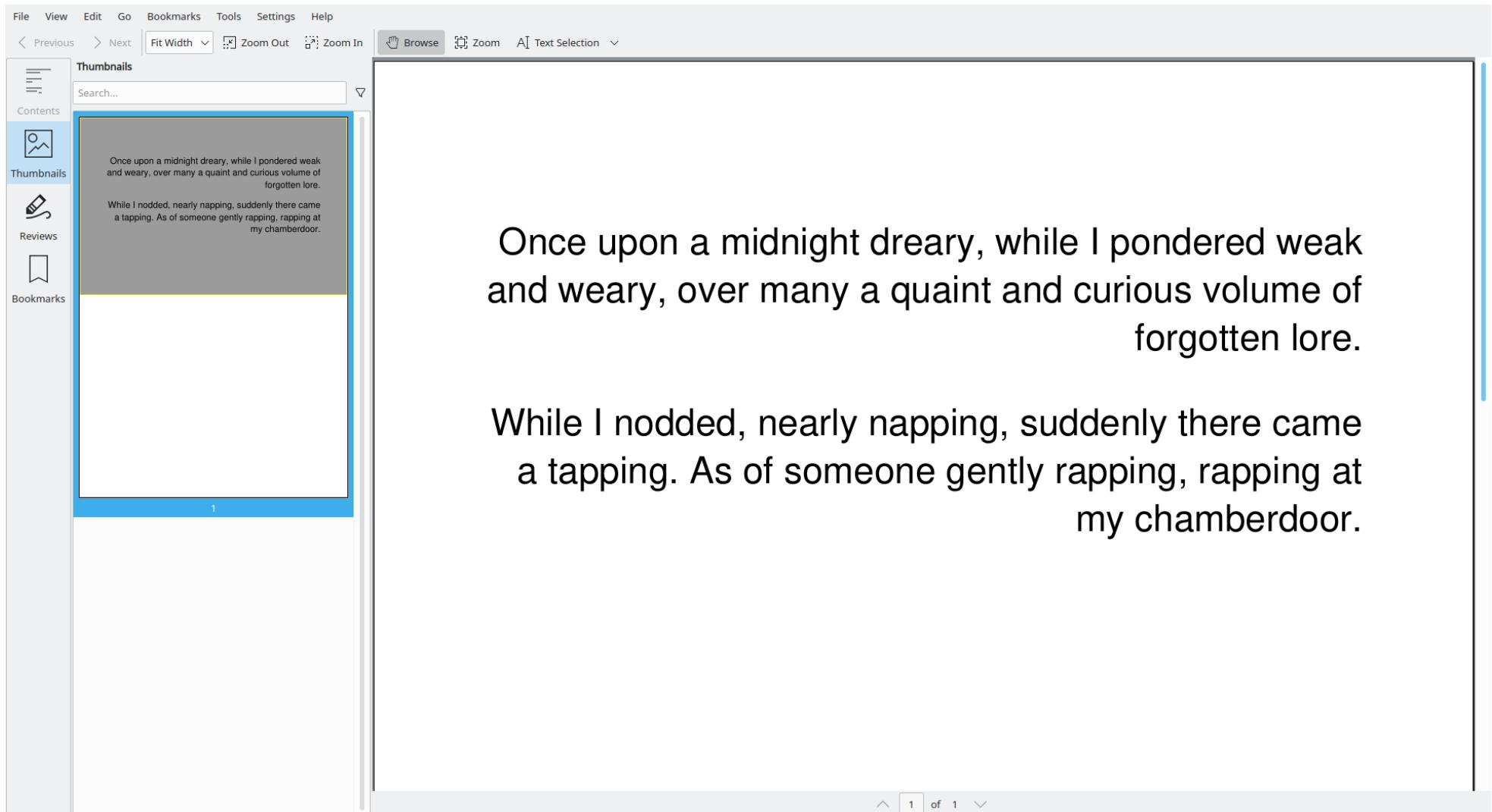
```
        horizontal_alignment=Alignment.RIGHT,
    )))
layout.add(Paragraph(
    "While I nodded, nearly napping, suddenly there came a tapping. As of someone gently rapping, rapping at my
chamberdoor.",
    font_size=Decimal(20),
    horizontal_alignment=Alignment.RIGHT,
))

```

Let's store the PDF

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.2.2 Using MultiColumnLayout

`pText` also comes with `MultiColumnLayout`, which enables you to create a `Document` with multiple columns on each page.

Most of our previous code will stay the same. We will need to change the `PageLayout` we used. Now we're using `MultiColumnLayout`.

```
layout = MultiColumnLayout(page, number_of_columns=2)
```

We're also going to add a lot more content, so you can really see the effect.

```
layout.add(Paragraph("The Raven", font_size=Decimal(20)))
layout.add(
    Paragraph(
        "Edgar Allen Poe",
        font="Helvetica-Oblique",
        font_size=Decimal(8),
        font_color=X11Color("SteelBlue"),
    )
)
for _ in range(0, 20):
    layout.add(
        Paragraph(
            "Once upon a midnight dreary, while I pondered, weak and weary, Over many a quaint and curious
volume of forgotten lore- While I nodded, nearly napping, suddenly there came a tapping, As of some one gently
rapping, rapping at my chamber door. Tis some visitor, I muttered, tapping at my chamber door- Only this and
nothing more.",
            font_size=Decimal(12),
            font_color=X11Color("SlateGray"),
            horizontal_alignment=Alignment.LEFT,
        )
)
```

The result should be something like this:

The screenshot shows a web-based document viewer or a digital library interface. At the top is a navigation bar with 'File', 'View', 'Edit', 'Go', 'Bookmarks', 'Tools', 'Settings', and 'Help' menus. Below the menu is a toolbar with 'Previous', 'Next', 'Fit Width', 'Zoom Out', 'Zoom In', 'Browse', 'Zoom', and 'Text Selection' buttons.

The left sidebar contains a 'Contents' section with a tree icon, a 'Reviews' section with a pencil icon, and a 'Bookmarks' section with a bookmark icon. The main content area features a large title 'The Raven' in bold, followed by the author's name 'Edgar Allan Poe'. The poem is presented in three separate blocks, each containing the full text of the poem. The text is in a serif font, with the title and author in a larger size. The sidebar also contains a 'Thumbnails' section with a search bar and a list of thumbnail previews for other documents.

The poem text is as follows:

```
Once upon a midnight dreary, while I
pondered, weak and weary, Over many a
quaint and curious volume of forgotten
lore- While I nodded, nearly napping,
suddenly there came a tapping, As of
some one gently rapping, rapping at my
chamber door. Tis some visitor, I
muttered, tapping at my chamber door-
Only this and nothing more.

Once upon a midnight dreary, while I
pondered, weak and weary, Over many a
quaint and curious volume of forgotten
lore- While I nodded, nearly napping,
suddenly there came a tapping, As of
some one gently rapping, rapping at my
chamber door. Tis some visitor, I
muttered, tapping at my chamber door-
Only this and nothing more.

Once upon a midnight dreary, while I
pondered, weak and weary, Over many a
quaint and curious volume of forgotten
lore- While I nodded, nearly napping,
suddenly there came a tapping, As of
some one gently rapping, rapping at my
chamber door. Tis some visitor, I
muttered, tapping at my chamber door-
Only this and nothing more.
```

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.2.3 Mixing various PageLayout instances

Each `PageLayout` acts independently of any other `PageLayout` objects that may be defined for a given `Page`. This allows us to do a layout in layers. For instance render the background using a `SingleColumnLayout`, and then rendering the foreground using `MultiColumnLayout`. We can even define some components at absolute positions, to mix and match.

2.2.5 Showcase : Making a copy of The Raven by Edgar Allen Poe

This example showcases using multiple layout-layers. We'll start by creating an empty `Document`:

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

On the first layer (the background), we'll just put an `Image`

```
# first layer, displaying a raven
layout = SingleColumnLayout(page)
for _ in range(0, 12):
    layout.add(Paragraph(" "))
layout.add(Image("https://cdn3.vectorstock.com/i/1000x1000/03/47/black-raven-on-white-background-vector-4780347.jpg"))
```

Then we'll create a new `PageLayout`, and start layout on `Paragraph` objects

```
# second layer, displaying the poem
layout = MultiColumnLayout(page, number_of_columns=2)
layout.add(
    Paragraph(
        "The Raven",
        font_size=Decimal(20),
        font="Helvetica-Oblique",
        font_color=HexColor("708090"),
    )
)
layout.add(
    Paragraph(
        """Once upon a midnight dreary, while I pondered, weak and weary,
        Over many a quaint and curious volume of forgotten lore-
        While I nodded, nearly napping, suddenly there came a tapping,
        As of some one gently rapping, rapping at my chamber door.
        'Tis some visitor,' I muttered, 'tapping at my chamber door-
        Only this and nothing more.'""",
        horizontal_alignment=Alignment.CENTERED,
        font_size=Decimal(8),
        respect_newlines_in_text=True,
    )
)
layout.add(
    Paragraph(
        """Ah, distinctly I remember it was in the bleak December;
        And each separate dying ember wrought its ghost upon the floor.
        Eagerly I wished the morrow;-vainly I had sought to borrow
        From my books surcease of sorrow-sorrow for the lost Lenore-
        For the rare and radiant maiden whom the angels name Lenore-
        Nameless here for evermore."""",
        horizontal_alignment=Alignment.CENTERED,
        font_size=Decimal(8),
```

```
    respect_newlines_in_text=True,
)
)
layout.add(
    Paragraph(
        """And the silken, sad, uncertain rustling of each purple curtain
            Thrilled me-filled me with fantastic terrors never felt before;
            So that now, to still the beating of my heart, I stood repeating
            'Tis some visitor entreating entrance at my chamber door-
            Some late visitor entreating entrance at my chamber door;-
            This it is and nothing more.'""",
        horizontal_alignment=Alignment.CENTERED,
        font_size=Decimal(8),
        respect_newlines_in_text=True,
    )
)
layout.add(
    Paragraph(
        """Presently my soul grew stronger; hesitating then no longer,
            'Sir,' said I, 'or Madam, truly your forgiveness I implore;
            But the fact is I was napping, and so gently you came rapping,
            And so faintly you came tapping, tapping at my chamber door,
            That I scarce was sure I heard you'-here I opened wide the door;-
            Darkness there and nothing more.'""",
        horizontal_alignment=Alignment.CENTERED,
        font_size=Decimal(8),
        respect_newlines_in_text=True,
    )
)
layout.switch_to_next_column()
layout.add(
    Paragraph(
        """Deep into that darkness peering, long I stood there wondering, fearing,
            Doubting, dreaming dreams no mortal ever dared to dream before;
```

```
        But the silence was unbroken, and the stillness gave no token,  
        And the only word there spoken was the whispered word, 'Lenore?'  
        This I whispered, and an echo murmured back the word, 'Lenore!'-  
        Merely this and nothing more."",  
        horizontal_alignment=Alignment.CENTERED,  
        font_size=Decimal(8),  
        respect_newlines_in_text=True,  
    )  
)  
layout.add(  
    Paragraph(  
        """Back into the chamber turning, all my soul within me burning,  
            Soon again I heard a tapping somewhat louder than before.  
            'Surely,' said I, 'surely that is something at my window lattice;  
            Let me see, then, what thereat is, and this mystery explore-  
            Let my heart be still a moment and this mystery explore;-  
            'Tis the wind and nothing more!""",  
        horizontal_alignment=Alignment.CENTERED,  
        font_size=Decimal(8),  
        respect_newlines_in_text=True,  
    )  
)  
layout.add(  
    Paragraph(  
        """Open here I flung the shutter, when, with many a flirt and flutter,  
            In there stepped a stately Raven of the saintly days of yore;  
            Not the least obeisance made he; not a minute stopped or stayed he;  
            But, with mien of lord or lady, perched above my chamber door-  
            Perched upon a bust of Pallas just above my chamber door-  
            Perched, and sat, and nothing more."",  
        horizontal_alignment=Alignment.CENTERED,  
        font_size=Decimal(8),  
        respect_newlines_in_text=True,  
    )
```

```
)  
layout.add(  
    Paragraph(  
        """Then this ebony bird beguiling my sad fancy into smiling,  
            By the grave and stern decorum of the countenance it wore,  
            'Though thy crest be shorn and shaven, thou,' I said, 'art sure no craven,  
            Ghastly grim and ancient Raven wandering from the Nightly shore-  
            Tell me what thy lordly name is on the Night's Plutonian shore!'  
                Quoth the Raven 'Nevermore.'"""",  
        horizontal_alignment=Alignment.CENTERED,  
        font_size=Decimal(8),  
        respect_newlines_in_text=True,  
    )  
)  
layout.add(  
    Paragraph(  
        """Much I marvelled this ungainly fowl to hear discourse so plainly,  
            Though its answer little meaning-little relevancy bore;  
            For we cannot help agreeing that no living human being  
            Ever yet was blessed with seeing bird above his chamber door-  
            Bird or beast upon the sculptured bust above his chamber door,  
            With such name as 'Nevermore.'"""",  
        horizontal_alignment=Alignment.CENTERED,  
        font_size=Decimal(8),  
        respect_newlines_in_text=True,  
    )  
)  
layout.add(  
    Paragraph(  
        """But the Raven, sitting lonely on the placid bust, spoke only  
            That one word, as if his soul in that one word he did outpour.  
            Nothing farther then he uttered-not a feather then he fluttered-  
            Till I scarcely more than muttered 'Other friends have flown before-  
            On the morrow he will leave me, as my Hopes have flown before.'  
    )
```

```
    Then the bird said 'Nevermore.'"",
horizontal_alignment=Alignment.CENTERED,
font_size=Decimal(8),
respect_newlines_in_text=True,
)
)
```

Finally, we can store the Document .

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next 75% Zoom In Zoom Out Browse Zoom Text Selection

Thumbnails

Search...

Contents

Reviews

Bookmarks

Thumbnails

The Raven

Deep into that darkness peering, long I stood there wondering, fearing,
Doubting, dreaming dreams no mortal ever dared to dream before;
But the silence was unbroken, and the stillness gave no token,
And the only word there spoken was the whispered word, 'Lenore!'
This I whispered, as an echo murmured back the word, 'Lenore!'-

Once upon a midnight dreary, while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore,
While I nodded, nearly napping, suddenly there came a tapping,
At my chamber window, as of some one gently rapping, rapping at my chamber door.
'Tis some visitor,' I muttered, 'tapping at my chamber door—
Only this and nothing more.'

Ah, distinctly I remember it was in the bleak December;
And each separate dying ember wrought its ghost upon the floor.
Eagerly I wished the morrow;—vainly I had sought to borrow
From my books surcease of sorrow-sorrow for the lost Lenore—
For the rare and radiant maiden whom the angels name Lenore—
Nameless here for evermore.

And the silken, sad, uncertain rustling of each purple curtain
Thrilled me—filled me with fantastic terrors never felt before;
So that now, to still the beating of my heart, I stood repeating
'Tis some visitor entreating entrance at my chamber door—
Some late visitor entreating entrance at my chamber door—
This it is and nothing more.'

Presently my soul grew stronger; hesitating then no longer,
'Sir,' said I, 'or Madam, truly your forgiveness I implore;
But the fact is I was napping, and so faintly you came rapping,
And so faintly I came tapping, tapping at my chamber door,
That I scarce was sure I heard you—here I opened wide the door—
Darkness there and nothing more.'

Deep into that darkness peering, long I stood there wondering, fearing,
Doubting, dreaming dreams no mortal ever dared to dream before;
But the silence was unbroken, and the stillness gave no token,
And the only word there spoken was the whispered word, 'Lenore!'
This I whispered, as an echo murmured back the word, 'Lenore!'-

Merely this and nothing more.

Back into the chamber turning, all my soul within me burning,
Soon again I heard a tapping somewhat louder than before.
Surely,' said I, 'surely that is something at my window lattice;
Let me see, then, what there is, and this mystery explore—
Let my heart be still a moment and this mystery explore—
'Tis the wind and nothing more!'

Open here I flung the shutter, when, with many a furt and flutter,
In there stepped a stately Raven of the saintly days of yore;
Not the least obeisance made he; not a minute stopped or stayed he;
But, with mien of lord or lady, perched above my chamber door—
Perched upon a bust of Pallas just above my chamber door—
Perched, and sat, and nothing more.

Then this ebony bird beguiling my sad fancy into smiling,
By the grave and stern decorum of the countenance it wore,
Though thy crest be shorn and shaven, thou,' I said, 'art sure no craven,
Ghastly grim and ancient Raven wandering from the Nightly shore—
Tell me what thy lordly name is on the Night's Plutonian shore!'
Quoth the Raven 'Nevermore.'

Much I marvelled this ungainly fowl to hear discourse so plainly,
Though its answer little meaning—little relevancy bore;
For we cannot help agreeing that no living human being
Ever yet was blessed with seeing bird above his chamber door—
Bird or beast upon the sculptured bust above his chamber door,
With such name as 'Nevermore.'

But the Raven, sitting lonely on the placid bust, spoke only
That one word, as if his soul in that one word he did outpour.
Nothing farther then he uttered—not a feather then he fluttered—
Till I scarcely more than muttered 'Other friends have flown before—
On the morrow he will leave me, as my Hopes have flown before.'
Then the bird said 'Nevermore.'



1 of 1

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.3 Using Table

2.3.1 Basic Example

Let's start by creating an empty Document

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

Now we're going to create a simple Table . By simple I mean; no row-span, no col-span.

```
t = Table(number_of_rows=5, number_of_columns=2)
t.add(Paragraph("Language", color=X11Color("SteelBlue"), font_size=Decimal(20),
horizontal_alignment=Alignment.CENTERED))
t.add(Paragraph("Nof. Questions", color=X11Color("SteelBlue"), font_size=Decimal(20),
horizontal_alignment=Alignment.CENTERED))

t.add(Paragraph("Javascript"))
t.add(Paragraph("2,167,178"))

t.add(Paragraph("Php"))
t.add(Paragraph("1,391,524"))

t.add(Paragraph("C++"))
t.add(Paragraph("711,944"))

t.add(Paragraph("Java"))
t.add(Paragraph("1,752,877"))
t.set_border_width_on_all_cells(Decimal(0.2))
```

```
t.set_padding_on_all_cells(Decimal(5), Decimal(5), Decimal(5), Decimal(5))

table_rect = t.layout(
    page,
    bounding_box=Rectangle(
        Decimal(20), Decimal(600), Decimal(500), Decimal(200)
    ),
)
```

We're also going to add a Paragraph underneath the Table .

```
Paragraph(text="**Data gathered from Stackoverflow.com on 10th of february 2021", font_size=Decimal(8),
color=X11Color("Gray"))\
    .layout(page, bounding_box=Rectangle(Decimal(20), table_rect.y - 40, table_rect.width, Decimal(20)))
```

Finally, we can store the Document .

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:

The screenshot shows a software application window with a menu bar at the top. The menu items include File, View, Edit, Go, Bookmarks, Tools, Settings, and Help. Below the menu is a toolbar with buttons for Previous, Next, Fit Width, Zoom Out, Zoom In, Browse, and Text Selection.

The main area of the window contains a table titled "Language" and "Nof. Questions". The table has four rows, each representing a programming language and its corresponding number of questions. The data is as follows:

Language	Nof. Questions
Javascript	2,167,178
PHP	1,391,524
C++	711,944
Java	1,752,877

A small note at the bottom left of the table area states: "Data gathered from Stackoverflow.com on 10th of february 2021".

On the left side of the window, there is a sidebar with several icons and labels: "Contents" (document icon), "Thumbnails" (camera icon), "Reviews" (pencil icon), and "Bookmarks" (bookmark icon). The "Thumbnails" icon is highlighted with a blue border. Below the sidebar is a large, mostly empty white space.

At the bottom of the window, there is a navigation bar with a page number "1" and "of 1" followed by up and down arrows.

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.3.2 Using `row_span`

Let's start by creating an empty Document

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

Like in the other Table examples, we'll start by building a Table object.

```
t = Table(number_of_rows=5, number_of_columns=3)
t.add(Paragraph(" "))
t.add(Paragraph("Language", color=X11Color("SteelBlue"), font_size=Decimal(20)))
t.add(Paragraph("Nof. Questions", color=X11Color("SteelBlue"), font_size=Decimal(20)))
```

Table allows us to add LayoutElement implementations directly (such as Paragraph) but also supports adding TableCell elements, which optionally allow you to define row_span and col_span .

```
t.add(TableCell(Paragraph("front-end", color=X11Color("SteelBlue")), row_span=2))
t.add(Paragraph("Javascipt"))
t.add(Paragraph("2,167,178"))

t.add(Paragraph("Php"))
t.add(Paragraph("1,391,524"))

t.add(TableCell(Paragraph("back-end", color=X11Color("SteelBlue")), row_span=2))
t.add(Paragraph("C++"))
t.add(Paragraph("711,944"))
```

```
t.add(Paragraph("Java"))
t.add(Paragraph("1,752,877"))
t.set_border_width_on_all_cells(Decimal(0.2))
t.set_padding_on_all_cells(Decimal(5), Decimal(5), Decimal(5), Decimal(5))

table_rect = t.layout(
    page,
    bounding_box=Rectangle(
        Decimal(20), Decimal(600), Decimal(500), Decimal(200)
    ),
)

Paragraph(text="**Data gathered from Stackoverflow.com on 10th of february 2021", font_size=Decimal(8),
color=X11Color("Gray"))\
    .layout(page, bounding_box=Rectangle(Decimal(20), table_rect.y - 40, table_rect.width, Decimal(20)))
```

Finally, we can store the Document .

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:

The screenshot shows a presentation slide with a navigation bar at the top. The menu items include File, View, Edit, Go, Bookmarks, Tools, Settings, and Help. Below the menu are buttons for Previous, Next, Fit Width, Zoom Out, Zoom In, Browse, and Text Selection.

The left sidebar contains icons for Contents, Thumbnails, Reviews, and Bookmarks. The 'Thumbnails' icon is selected and highlighted in blue.

The main content area displays a table titled "Language" and "Nof. Questions". The table has four rows:

	Language	Nof. Questions
front-end	Javascript	2,167,178
	Php	1,391,524
back-end	C++	711,944
	Java	1,752,877

A small note at the bottom left of the slide states: "Data gathered from Stackoverflow.com on 10th of february 2021".

At the bottom of the slide, there is a navigation bar with a page number indicator: "1 of 1".

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.3.3 Using `col_span`

We're going to change the previous example a bit, to include some `col_span`

We'll start by re-defining the `Table` to include one extra row (we'll show the total in that row):

```
t = Table(number_of_rows=6, number_of_columns=3)
```

And lastly, we add this last row:

```
t.add(Paragraph("Total"))
t.add(TableCell(Paragraph("6,023,523"), col_span=2))
```

The result should be something like this:

The screenshot shows a presentation slide with a navigation bar at the top. The menu includes File, View, Edit, Go, Bookmarks, Tools, Settings, and Help. Below the menu are buttons for Previous, Next, Fit Width, Zoom In, Zoom Out, and Text Selection. On the left side, there is a sidebar with icons for Thumbnails, Contents, Reviews, and Bookmarks. The main content area displays a table titled "Language" and "Nof. Questions". The table has four rows: front-end (Javascript: 2,167,178, Php: 1,391,524), back-end (C++: 711,944, Java: 1,752,877), and a total row (Total: 6,023,523). A note at the bottom states: "Data gathered from Stackoverflow.com on 10th of february 2021".

	Language	Nof. Questions
front-end	Javascript	2,167,178
	Php	1,391,524
back-end	C++	711,944
	Java	1,752,877
Total		6,023,523

**Data gathered from Stackoverflow.com on 10th of february 2021

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.3.4 Using other `LayoutElement` objects in a `Table`

Let's start by defining a convenience method for adding an `Image` to a `Table`. This method will accept the URL of the `Image`, and a `Table` as arguments:

```
def _add_image_to_table(self, url: str, table: Table):
    im = PILImage.open(
        requests.get(
            url,
            stream=True,
        ).raw
    )
    table.add(Image(im, width=Decimal(128), height=Decimal(128)))
```

In order to keep `pText` `Image` separate from `PIL` `Image` I use the following import statement:

```
from PIL import Image as PILImage
```

Now we can get to work. We'll begin by creating an empty `Document` with an empty `Page`

```
pdf = Document()
page = Page()
pdf.append_page(page)
```

I want to add a `Table` with 3 rows (2 rows for data, 1 header):

```
t = Table(number_of_rows=3, number_of_columns=3)
```

I'm going to start the `Table` by writing the header

```

t.add(Paragraph(" "))
t.add(
    Paragraph(
        "Close-up",
        font_color=X11Color("SteelBlue"),
        font_size=Decimal(20),
        horizontal_alignment=Alignment.CENTERED,
    )
)
t.add(
    Paragraph(
        "Panoramic",
        font_color=X11Color("SteelBlue"),
        font_size=Decimal(20),
        horizontal_alignment=Alignment.CENTERED,
    )
)

```

The first entry in this row is the row header, followed by two `Image` objects, which we'll add using our utility method

```

t.add(Paragraph("Nature"))
self._add_image_to_table("https://images.unsplash.com/photo-1520860560195-0f14c411476e?
ixid=MXwxMjA3fDB8MHxwaG90by1wYwdfHx8fGVufDB8fHw", t)
self._add_image_to_table("https://images.unsplash.com/photo-1613480123595-c5582aa551b9?
ixid=MXwxMjA3fDB8MHxwaG90by1wYwdfHx8fGVufDB8fHw", t)

```

Same for the second row:

```

t.add(Paragraph("Architecture"))
self._add_image_to_table("https://images.unsplash.com/photo-1611321569296-1305a38ebd74?

```

```
ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw", t)
    self._add_image_to_table("https://images.unsplash.com/photo-1613262666714-acebcc37f11e?
ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw", t)
```

Finally, I want to set padding and borders:

```
t.set_border_width_on_all_cells(Decimal(0.2))
t.set_padding_on_all_cells(Decimal(5), Decimal(5), Decimal(5), Decimal(5))
```

And use a `PageLayout` to add everything to a `Page`

```
layout = SingleColumnLayout(page)
layout.add(t)
```

And now we can store the `Document`

```
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:

Screenshot of a software interface showing a sidebar with navigation links (File, View, Edit, Go, Bookmarks, Tools, Settings, Help) and a toolbar with zoom controls (Zoom Out, Zoom In, Text Selection). The main area displays a 'Thumbnails' view with a search bar and a table structure. The table has two rows: 'Nature' and 'Architecture'. Each row contains three columns: 'Close-up' (with an image of a cactus), 'Panoramic' (with an image of a waterfall), and a blank column. The 'Nature' row is highlighted with a blue border.

	Close-up	Panoramic
Nature		
Architecture		

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.3.5 Showcase : displaying a `Table` that doubles as a heatmap-plot

We'll start by creating an empty Document :

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)
```

We'll use a layout manager to make things easy on ourselves:

```
# set layout
layout = SingleColumnLayout(page)
```

This data comes from a StackOverflow question. The author of the question wanted to display this data in a Table and use colors on each TableCell depending on the value.

```
my_dict= {' ': ['A Error', 'B Error', 'C Error', 'D Error'],
          'lab1': [0.34, 0.23, 0.80, 0.79],
          'lab2': [0.53, 0.38, 0.96, 1.25],
          'lab3': [0.40, 0.27, 0.68, 0.93]}

colors = {0: X11Color("Green"),
          0.25: X11Color("Yellow"),
          0.5: X11Color("Orange"),
          0.75: X11Color("Red")}
```

Now we can start building the Table :

```
table = Table(number_of_rows=4, number_of_columns=5)
```

First we'll add the header row:

```
table.add(Paragraph(" "))
for h in my_dict[" "]:
    table.add(Paragraph(text=h, font="Helvetica-Bold", font_size=Decimal(12)))
```

Now we can add the data-rows:

```
for name, row in [(k,v) for k,v in my_dict.items() if k != " "]:
    table.add(Paragraph(name))
    for v in row:
        c = X11Color("Green")
        for b,bc in colors.items():
            if v > b:
                c = bc
        table.add(Paragraph(str(v),
                            font_color=c,
                            horizontal_alignment=Alignment.CENTERED))
```

We're going to make the border on each cell a bit thinner than the default:

```
# set border
table.set_border_width_on_all_cells(Decimal(0.2))
```

Padding can make a `Table` a lot more legible. Let's have a look at how you'd set the padding on a `Table` in `pText`. Just like with borders, we could set them on each `TableCell` individually. But `Table` offers a convenience-method to set the padding on each of its `TableCell` objects:

```
# set padding
table.set_padding_on_all_cells(Decimal(5), Decimal(5), Decimal(5), Decimal(5))
```

Finally we add an explanatory `Paragraph` and the `Table`

```
# add to layout
layout.add(Paragraph("This table contains all measurands for 3 lab-sessions:"))
layout.add(table)
```

Now we can store the PDF

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:

The screenshot shows a PDF document open in a viewer. The left sidebar contains navigation buttons for File, View, Edit, Go, Bookmarks, Tools, Settings, and Help, along with buttons for Previous, Next, Fit Width, Zoom Out, Zoom In, Browse, and Text Selection. The main content area has a 'Thumbnails' tab selected, showing a preview of the document. A search bar is present above the thumbnails. The preview window displays a table with the following text and data:

This table contains all measurands for 3 lab-sessions:

	A Error	B Error	C Error	D Error
lab1	0.34	0.23	0.8	0.79
lab2	0.53	0.38	0.96	1.25
lab3	0.4	0.27	0.68	0.93

The table has four columns labeled A Error, B Error, C Error, and D Error. The rows represent three lab sessions: lab1, lab2, and lab3. The values in the table are color-coded: 0.34 (yellow), 0.23 (green), 0.8 (red), 0.79 (red), 0.53 (yellow), 0.38 (yellow), 0.96 (red), 1.25 (red), 0.4 (yellow), 0.27 (yellow), 0.68 (yellow), and 0.93 (red). A page number '1' is visible at the bottom of the preview window.

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.4 Using `OrderedList` and `UnorderedList`

2.4.1 Using `OrderedList`

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)

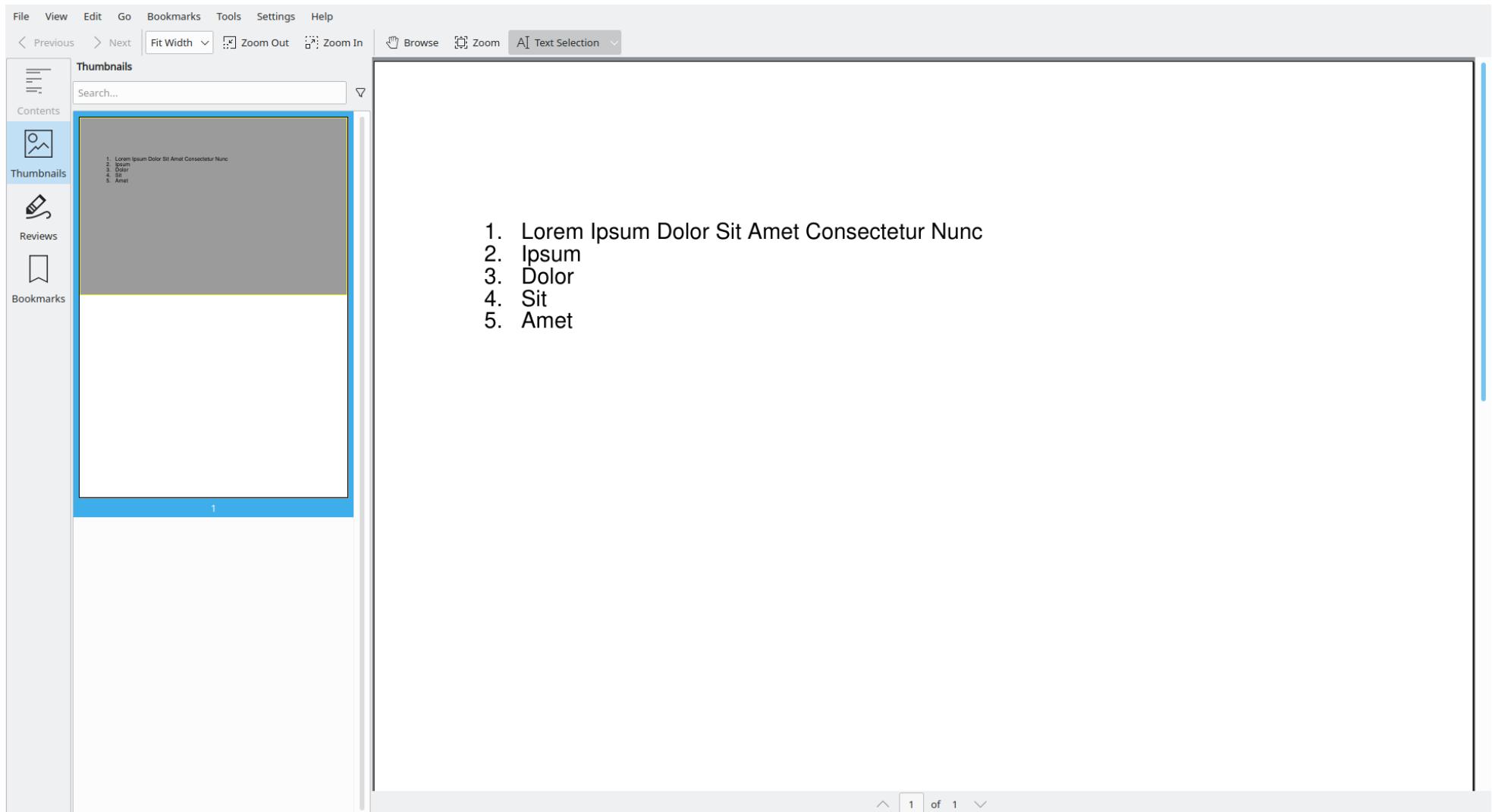
ul = OrderedList()
ul.add(Paragraph(text="Lorem Ipsum Dolor Sit Amet Consectetur Nunc"))
ul.add(Paragraph(text="Ipsum"))
ul.add(Paragraph(text="Dolor"))
ul.add(Paragraph(text="Sit"))
ul.add(Paragraph(text="Amet"))

layout = SingleColumnLayout(page)
layout.add(ul)
```

Finally, we can store the `Document`.

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.4.2 Using `UnorderedList`

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)

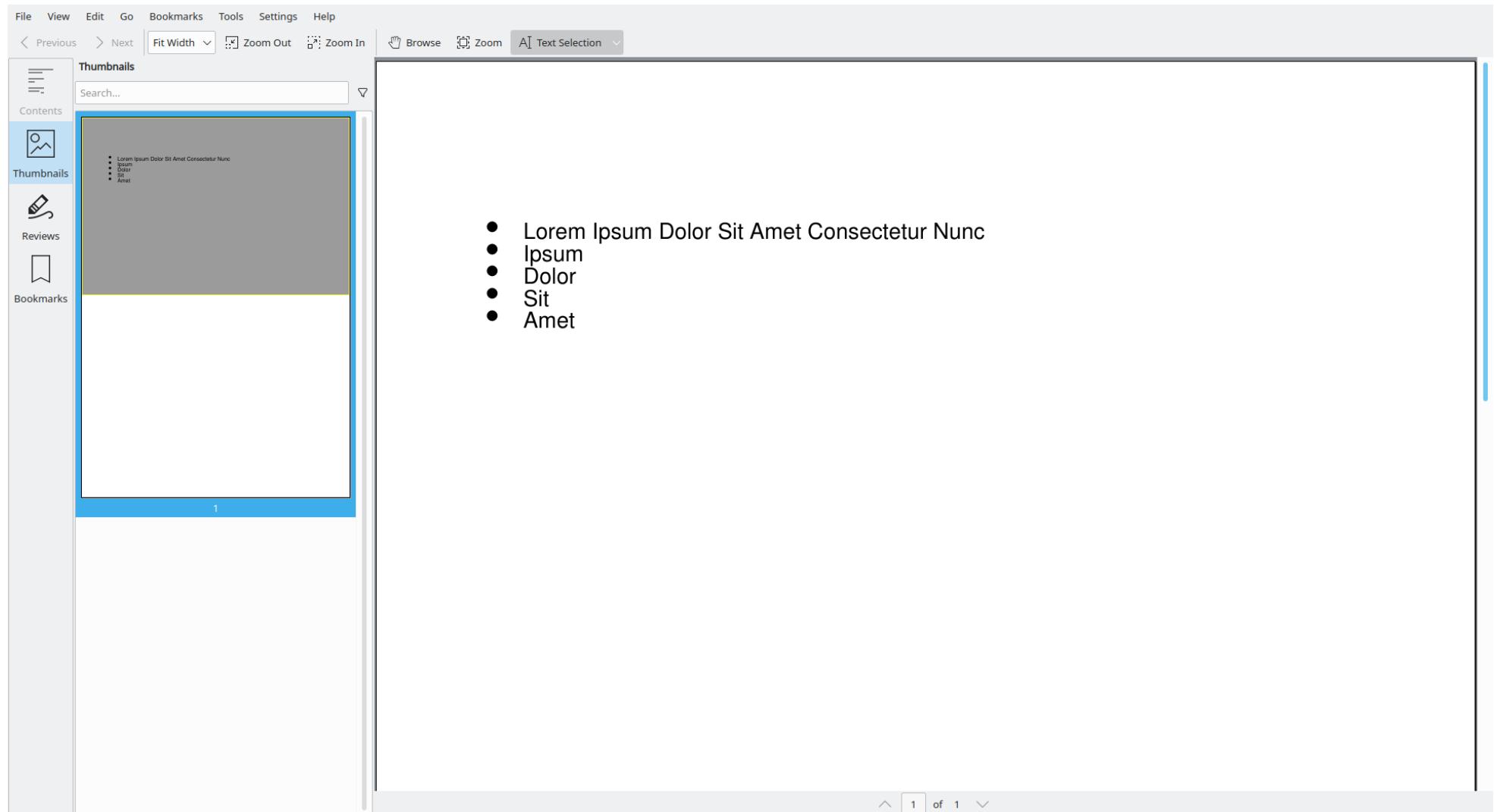
ul = UnorderedList()
ul.add(Paragraph(text="Lorem Ipsum Dolor Sit Amet Consectetur Nunc"))
ul.add(Paragraph(text="Ipsum"))
ul.add(Paragraph(text="Dolor"))
ul.add(Paragraph(text="Sit"))
ul.add(Paragraph(text="Amet"))

layout = SingleColumnLayout(page)
layout.add(ul)
```

Finally, we can store the Document .

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.4.3 Nested lists

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)

ul0 = UnorderedList()
ul0.add(Paragraph(text="Ipsum"))
ul0.add(Paragraph(text="Dolor"))

ul1 = UnorderedList()
ul1.add(Paragraph(text="Ipsum"))
ul1.add(Paragraph(text="Dolor"))
ul1.add(Paragraph(text="Sit"))
ul1.add(ul0)

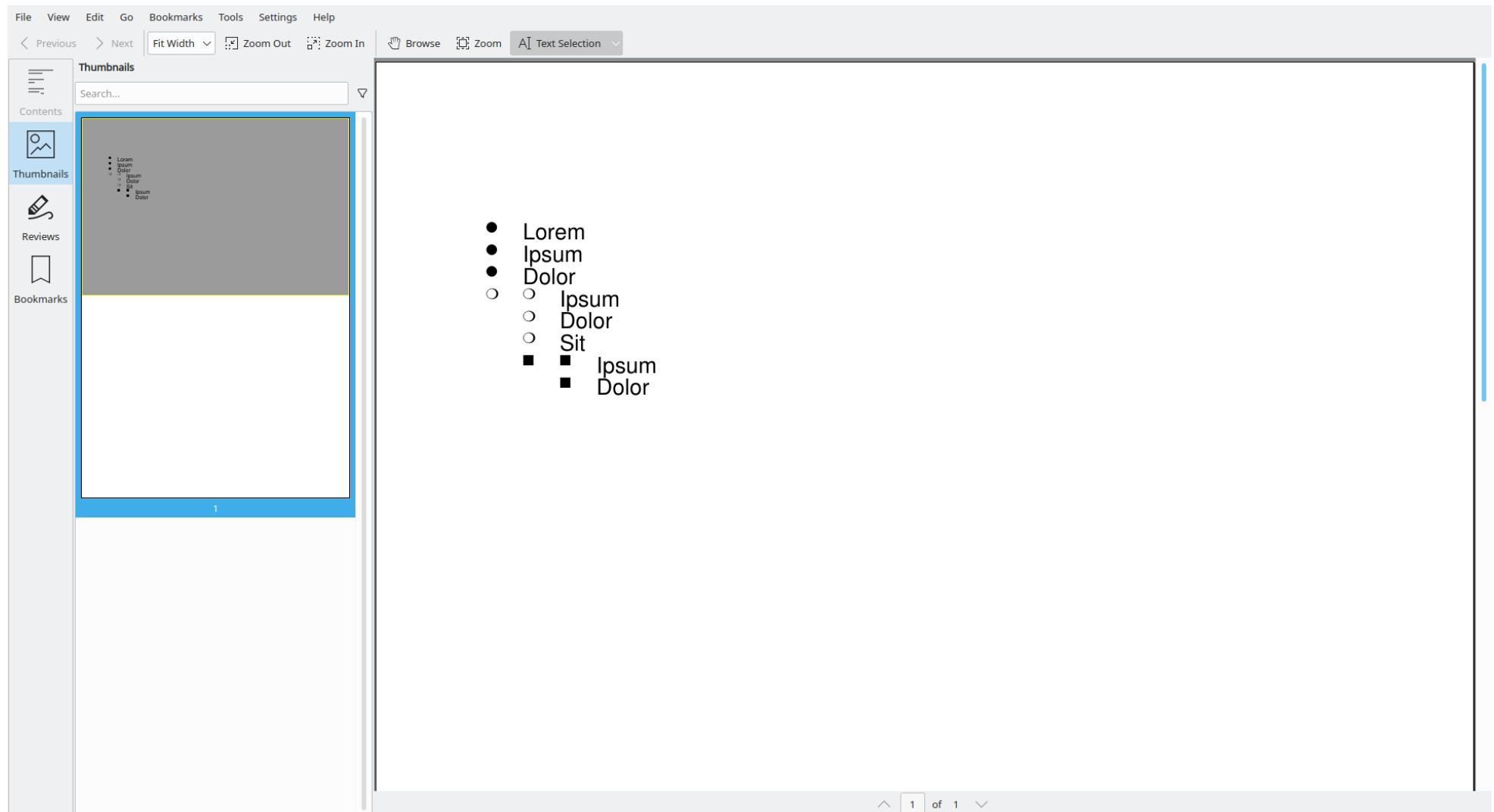
ul2 = UnorderedList()
ul2.add(Paragraph(text="Lorem"))
ul2.add(Paragraph(text="Ipsum"))
ul2.add(Paragraph(text="Dolor"))
ul2.add(ul1)

layout = SingleColumnLayout(page)
layout.add(ul2)
```

Finally, we can store the Document .

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.4.4 Showcase : Recreating a Wikipedia article

This example is going to re-create the basics of a Wikipedia article. It is not an attempt to copy the style and look/feel of Wikipedia. Merely a way of showcasing everything you can do with the examples and code in this section.

As usual, we start by creating an empty Document

```
# create empty document
pdf: Document = Document()

# create empty page
page: Page = Page()

# add page to document
pdf.append_page(page)
```

We set the PageLayout

```
# add Image
layout = MultiColumnLayout(page)
```

We're going to add a Paragraph to serve as the title:

```
layout.add(
    Paragraph(
        "Rose",
        font_color=X11Color("MistyRose"),
        font_size=Decimal(20),
        font="Helvetica-Bold",
```

```
)  
)
```

Now comes the actual content:

```
layout.add(  
    Paragraph(  
        "A rose is a woody perennial flowering plant of the genus Rosa, in the family Rosaceae, or the flower  
it bears."  
        "There are over three hundred species and tens of thousands of cultivars."  
    )  
)  
layout.add(  
    Paragraph(  
        "They form a group of plants that can be erect shrubs, climbing, or trailing, with stems that are often  
armed with sharp prickles."  
        "Flowers vary in size and shape and are usually large and showy,"  
        "in colours ranging from white through yellows and reds."  
    )  
)  
layout.add(  
    Paragraph(  
        "Most species are native to Asia, with smaller numbers native to Europe, North America, and  
northwestern Africa."  
        "Species, cultivars and hybrids are all widely grown for their beauty and often are fragrant."  
    )  
)  
layout.add(  
    Paragraph("Roses have acquired cultural significance in many societies."))  
)  
layout.add(  
    Paragraph(  
        "
```

```

        "Rose plants range in size from compact, miniature roses, to climbers that can reach seven meters in
height. "
        "Different species hybridize easily, and this has been used in the development of the wide range of
garden roses."
    )
)

```

Next we'll add an `Image` of a rose:

```

# add image
im = PILImage.open(
    requests.get(
        "https://images.unsplash.com/photo-1597826368522-9f4cb5a6ba48?
ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw",
        stream=True,
    ).raw
)
layout.add(Image(im, width=Decimal(256)))

```

And lastly some information on the genus Rosa, presented as a nested list:

```

# add UnorderedList
layout.add(Paragraph("The genus Rosa is subdivided into four subgenera:"))
layout.add(
    UnorderedList()
    .add(Paragraph("Hulthemia", padding_bottom=Decimal(2)))
    .add(Paragraph("Hesperrhodos", padding_bottom=Decimal(2)))
    .add(Paragraph("Platyrhodon", padding_bottom=Decimal(2)))
    .add(Paragraph("Rosa", padding_bottom=Decimal(2)))
    .add(

```

```
UnorderedList()  
    .add(Paragraph("Banksianae"))  
    .add(Paragraph("Bracteatae"))  
    .add(Paragraph("Caninae"))  
    .add(Paragraph("Carolinae"))  
    .add(Paragraph("Chinensis"))  
    .add(Paragraph("Gallicanae"))  
    .add(Paragraph("Gymnocarpae"))  
    .add(Paragraph("Laevigatae"))  
    .add(Paragraph("Pimpinellifoliae"))  
    .add(Paragraph("Rosa"))  
    .add(Paragraph("Synstylae"))  
)  
)
```

The result should be something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next 66% Zoom In Zoom Out Browse Zoom Text Selection

Thumbnails

Search...

Rose

The genus Rosa is subdivided into four subgenera:

- Hulthemia
- Hesperiros
- Platyrhodon
- Rosa
 - Banksianae
 - Caninae
 - Caroliniae
 - Chinensis
 - Gymnocarpae
 - Laevigatae
 - Pimpinellifoliae
 - Rosa
 - Synstyliae

A rose is a woody perennial flowering plant of the genus *Rosa*, in the family Rosaceae, or the flower it bears. There are over three hundred species and tens of thousands of cultivars.

They form a group of plants that can be erect shrubs, climbing, or trailing, with stems that are often armed with sharp prickles. Flowers vary in size and shape and are usually large and showy, in colours ranging from white through yellows and reds.

Most species are native to Asia, with smaller numbers native to Europe, North America, and northwestern Africa. Species, cultivars and hybrids are all widely grown for their beauty and often are fragrant.

Roses have acquired cultural significance in many societies.

Rose plants range in size from compact, miniature roses, to climbers that can reach seven meters in height. Different species hybridize easily, and this has been used in the development of the wide range of garden roses.

Rose

The genus Rosa is subdivided into four subgenera:

- Hulthemia
- Hesperiros
- Platyrhodon
- Rosa
 - Banksianae
 - Bracteatae
 - Caninae
 - Caroliniae
 - Chinensis
 - Gymnocarpae
 - Laevigatae
 - Pimpinellifoliae
 - Rosa
 - Synstyliae

A rose is a woody perennial flowering plant of the genus *Rosa*, in the family Rosaceae, or the flower it bears. There are over three hundred species and tens of thousands of cultivars.

They form a group of plants that can be erect shrubs, climbing, or trailing, with stems that are often armed with sharp prickles. Flowers vary in size and shape and are usually large and showy, in colours ranging from white through yellows and reds.

Most species are native to Asia, with smaller numbers native to Europe, North America, and northwestern Africa. Species, cultivars and hybrids are all widely grown for their beauty and often are fragrant.

Roses have acquired cultural significance in many societies.

Rose plants range in size from compact, miniature roses, to climbers that can reach seven meters in height. Different species hybridize easily, and this has been used in the development of the wide range of garden roses.



1 of 1

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.5 Using Image

2.5.1 Using a PIL Image

As you have seen in earlier examples, `pText` also handles `Image` objects. They act like any other `LayoutElement`. The most versatile way of constructing them is by passing a `PIL Image` to the constructor.

```
# add image
im = PILImage.open(
    requests.get(
        "https://images.unsplash.com/photo-1597826368522-9f4cb5a6ba48?
ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw",
        stream=True,
    ).raw
)
layout.add(Image(im, width=Decimal(256)))
```

You can specify a `width` and `height` for the `Image`. If you don't specify anything, `pText` will use the original width and height of the `Image`. If you specify only one, `pText` will derive the missing parameter by scaling the original width/height by the same ratio. If you specify both, `pText` will stick to the dimensions you've given.

2.5.2 Using a URL to create an Image

You can however, also pass a URL directly to the `Image` constructor, in which case it will use the `requests` library and `PIL` to fetch the bytes for you.

```
# add image
layout.add(Image("https://images.unsplash.com/photo-1597826368522-9f4cb5a6ba48?
ixid=MXwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHw", width=Decimal(256)))
```

2.5.3 Showcase : Making a PDF to thank the first 100 stars on GitHub

I'm going to start by defining two convenience methods:

```
def _write_footer(page: Page):
    rectangle_box = Rectangle(
        Decimal(0),
        Decimal(0),
        page.get_page_info().get_width(),
        page.get_page_info().get_height() * Decimal(0.1),
    )
    Shape(
        LineArtFactory.rectangle(rectangle_box),
        fill_color=self.ACCEPT_COLOR_1,
        stroke_color=self.ACCEPT_COLOR_1,
        line_width=Decimal(1),
    ).layout(page, rectangle_box)

    rectangle_box = Rectangle(
        Decimal(0),
        page.get_page_info().get_height() * Decimal(0.1),
        page.get_page_info().get_width(),
        Decimal(2),
    )
    Shape(
        LineArtFactory.rectangle(rectangle_box),
        fill_color=self.ACCEPT_COLOR_2,
        stroke_color=self.ACCEPT_COLOR_2,
        line_width=Decimal(1),
    ).layout(page, rectangle_box)
```

The second utility method will add a nice background with geometric stars:

```
def _write_background(self, page: Page):
    layout = SingleColumnLayout(page)
    t = Table(number_of_columns=10, number_of_rows=25)
    for i in range(0, 25):
        for j in range(0, 10):
            put_star = random.choice([x <= 3 for x in range(0, 10)])
            if i < 11 and j >= 5:
                t.add(Paragraph(" "))
                continue
            if put_star:
                c = random.choice(
                    [
                        self.ACCENT_COLOR_1,
                        self.ACCENT_COLOR_2,
                        self.ACCENT_COLOR_3,
                        self.ACCENT_COLOR_4,
                        self.ACCENT_COLOR_5,
                    ]
                )
            t.add(
                Shape(
                    LineArtFactory.n_pointed_star(
                        bounding_box=Rectangle(
                            Decimal(0), Decimal(0), Decimal(16), Decimal(16)
                        ),
                        n=random.choice([3, 5, 7, 12]),
                    ),
                    fill_color=c,
                    stroke_color=c,
                    line_width=Decimal(1),
                )
            )
    else:
```

```
        t.add(Paragraph(" "))
t.no_borders()
t.set_padding_on_all_cells(Decimal(5), Decimal(5), Decimal(5), Decimal(5))
layout.add(t)
```

Now we should be ready to define our main method:

```
# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)

layout = MultiColumnLayout(page)

# background
self._write_background(page)
```

To see the full table of GitHub ids I would recommend you check out this test. It seems a bit redundant to repeat that here.

```
# table
avatar_urls = [
    "https://avatars.githubusercontent.com/u/" + str(x)
    for x in self.FIRST_100_STARS
]
t = Table(number_of_columns=4, number_of_rows=25)
for s in avatar_urls[0 : (4 * 25)]:
    im = PILImage.open(requests.get(s, stream=True).raw)
    t.add(Image(im, width=Decimal(20), height=Decimal(20)))
```

```
t.set_padding_on_all_cells(Decimal(2), Decimal(2), Decimal(2), Decimal(2))
t.no_borders()
layout.add(t)

layout.add(
    Paragraph(
        "100 stars!",
        font="Helvetica-Bold",
        font_size=Decimal(20),
        font_color=self.ACCEPT_COLOR_1,
        horizontal_alignment=Alignment.CENTERED,
    )
)

# next column
layout.switch_to_next_column()

# paragraph
layout.add(
    Paragraph(
        "Thank you,",
        font="Helvetica-Bold",
        font_size=Decimal(20),
        font_color=self.ACCEPT_COLOR_1,
    )
)
layout.add(
    Paragraph(
        "Your support and encouragement have always been the driving factors in the development of pText. "
        "I want you to know that I value your appreciation immensely!"
    )
)
layout.add(
    Paragraph(
```

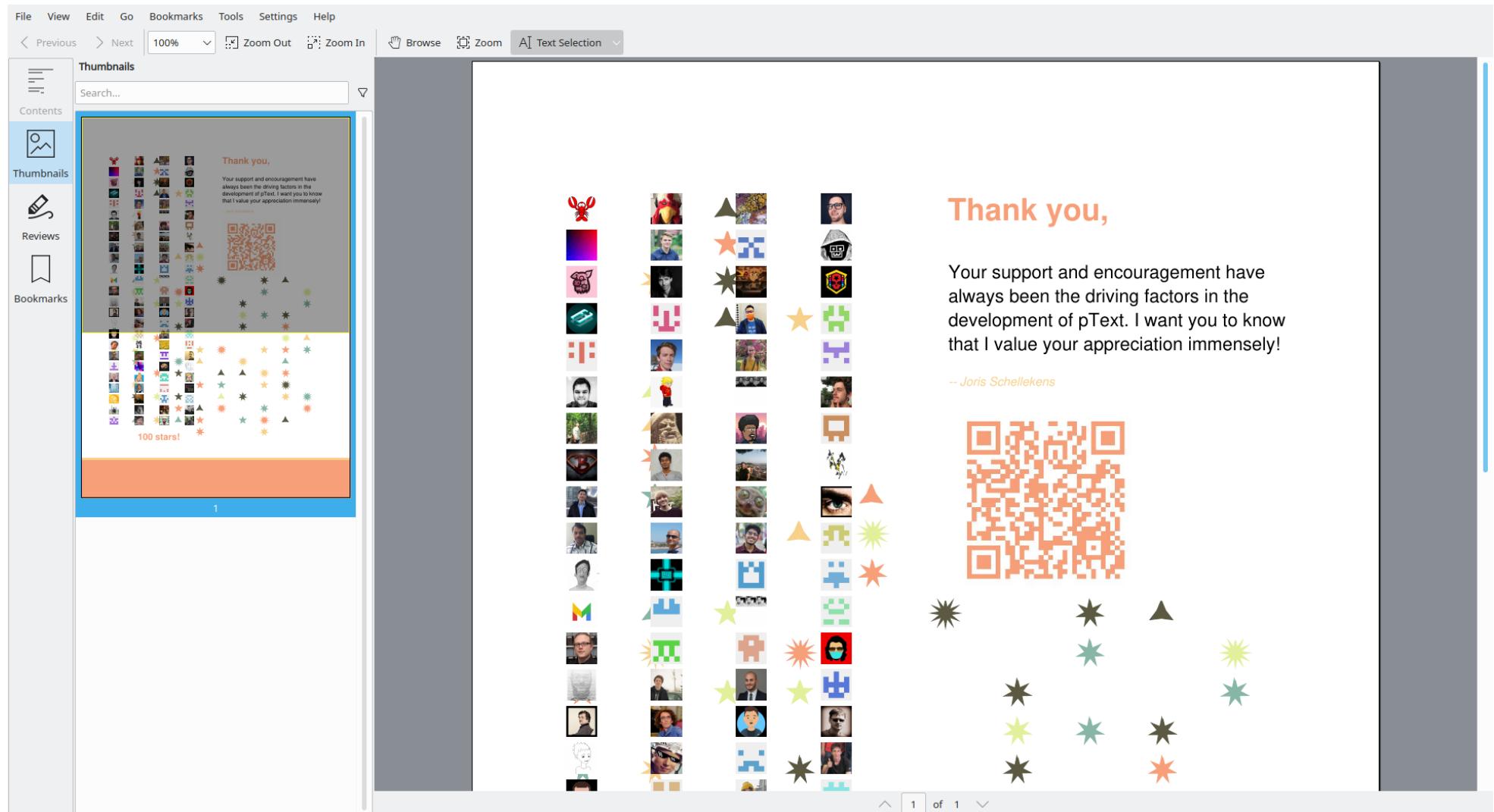
```
    " -- Joris Schellekens",
    font="Helvetica-Oblique",
    font_size=Decimal(8),
    font_color=self.ACCEPT_COLOR_2,
)
)

layout.add(
Barcode(
    data="https://github.com/jorisschellekens/ptext-release/stargazers",
    type=BarcodeType.QR,
    width=Decimal(128),
    stroke_color=self.ACCEPT_COLOR_1,
)
)
```

Lastly, we can store the Document

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.6 Using Barcode

2.6.1 Basic Example

pText also supports most barcode formats. Let's create an example Document :

```
pdf: Document = Document()
page: Page = Page()
pdf.append_page(page)

# set layout
layout = SingleColumnLayout(page)
```

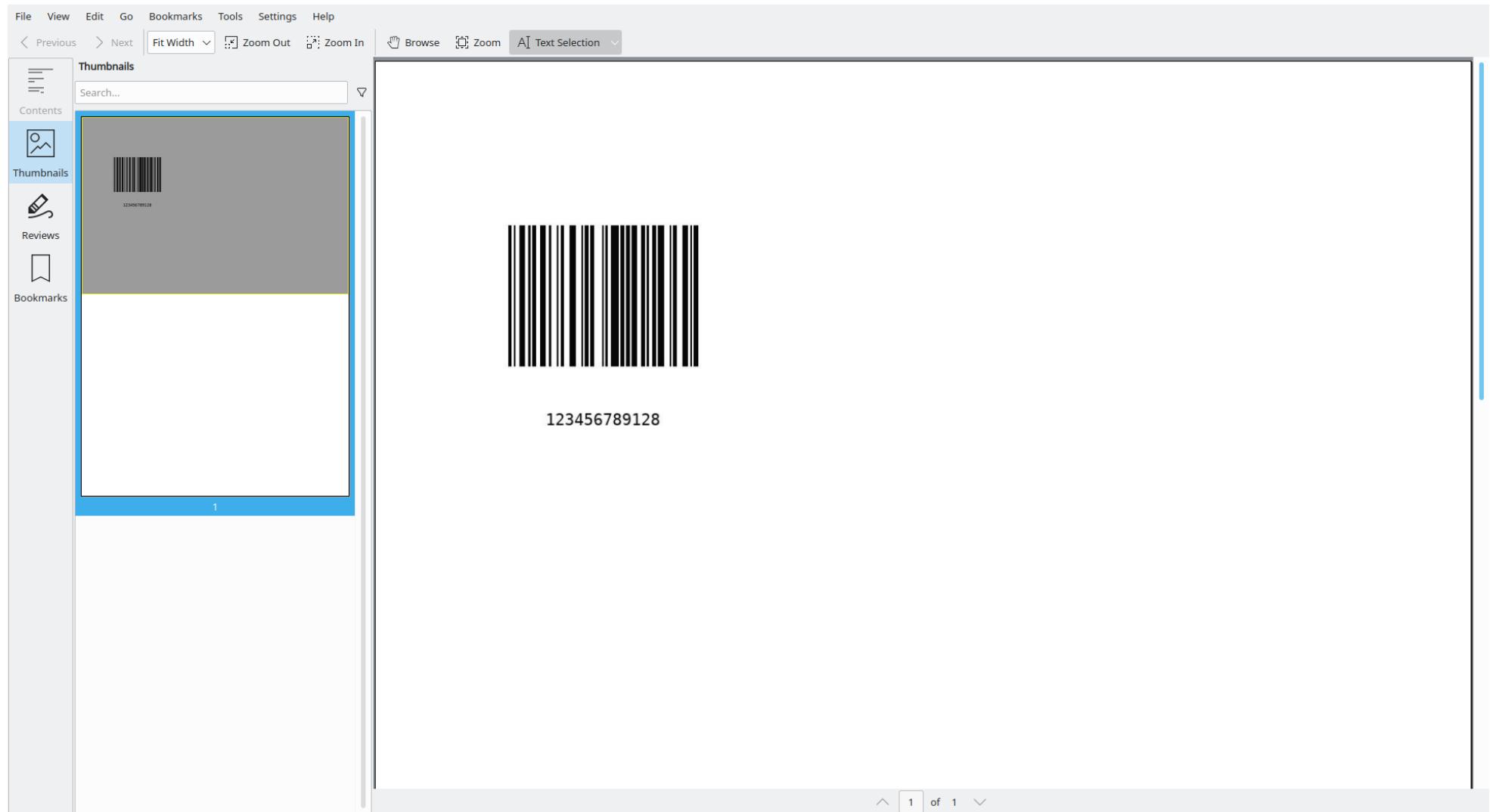
Next we'll add a single Barcode

```
# add barcode
layout.add(Barcode(
    data="123456789128",
    type=BarcodeType.CODE_128,
    width=Decimal(128),
    stroke_color=HexColor("#080708"),
)
)
```

Finally, we can write the Document :

```
with open("output.pdf", "wb") as pdf_file_handle:
    PDF.dumps(pdf_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.6.2 Using Color on Barcode objects

Like most `LayoutElement` implementations, `Barcode` objects can be colored. Use the `stroke_color` to set the foreground color of a `Barcode`. Use `fill_color` to set the background color.

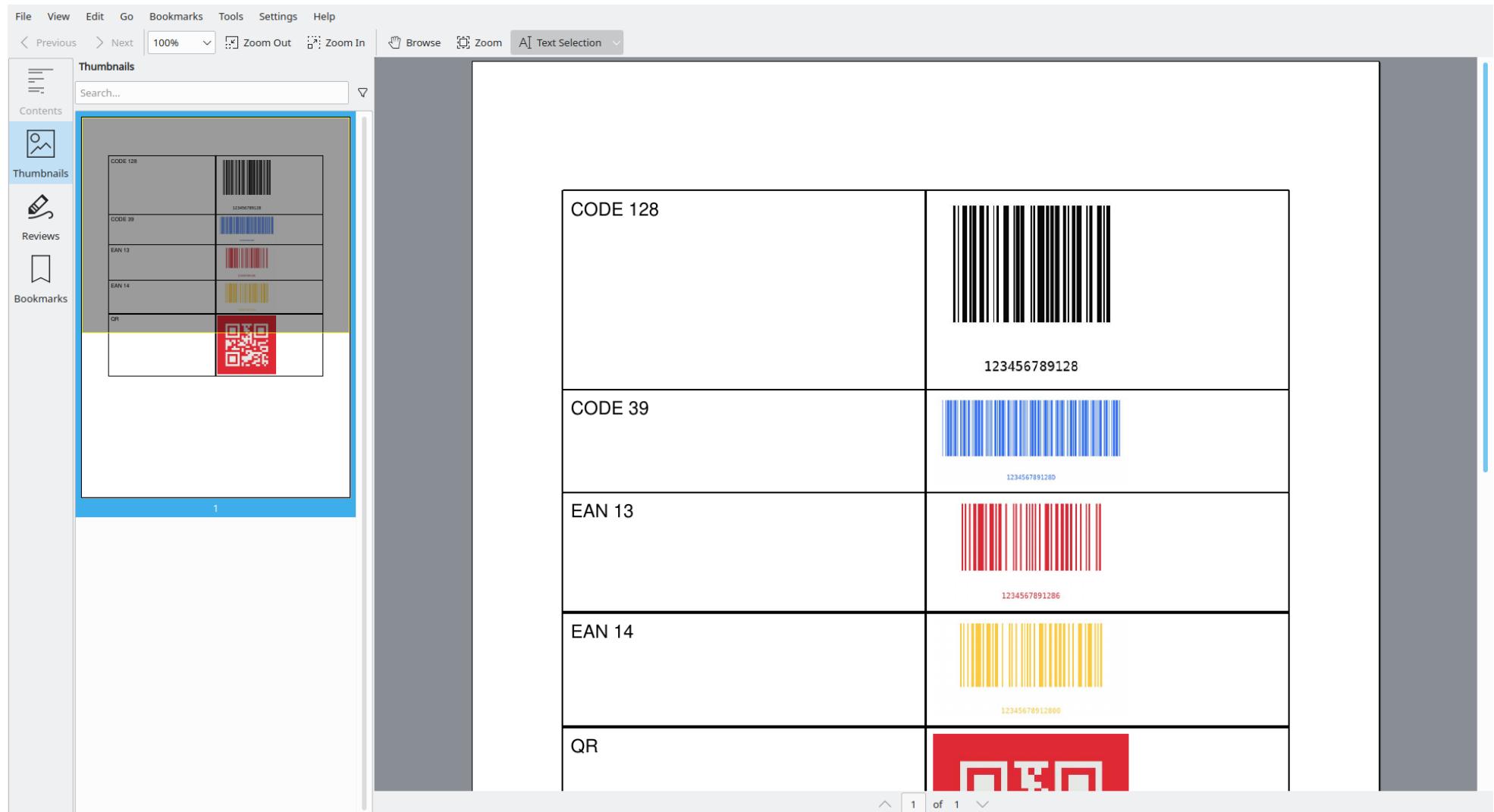
In this example, we're going to show some more `BarcodeTypes`, and give each one some color:

```
# set layout
layout = SingleColumnLayout(page)

# add barcode
layout.add(
    Table(number_of_rows=5, number_of_columns=2)
    .add(Paragraph("CODE 128"))
    .add(
        Barcode(
            data="123456789128",
            type=BarcodeType.CODE_128,
            width=Decimal(128),
            stroke_color=HexColor("#080708"),
        )
    )
    .add(Paragraph("CODE 39"))
    .add(
        Barcode(
            data="123456789128",
            type=BarcodeType.CODE_39,
            width=Decimal(128),
            stroke_color=HexColor("#3772FF"),
        )
    )
    .add(Paragraph("EAN 13"))
    .add(
        Barcode(
```

```
        data="123456789128",
        type=BarcodeType.EAN_13,
        width=Decimal(128),
        stroke_color=HexColor("#DF2935"),
    )
)
.add(Paragraph("EAN 14"))
.add(
    Barcode(
        data="1234567891280",
        type=BarcodeType.EAN_14,
        width=Decimal(128),
        stroke_color=HexColor("#FDCA40"),
    )
)
.add(Paragraph("QR"))
.add(
    Barcode(
        data="1234567891280",
        type=BarcodeType.QR,
        width=Decimal(128),
        stroke_color=HexColor("#E6E8E6"),
        fill_color=HexColor("#DF2935"),
    )
)
.set_padding_on_all_cells(Decimal(5), Decimal(5), Decimal(5), Decimal(5))
)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.6.3 Showcase : Outputting the results of a Jenkins run

In this showcase, we're going to convert jUnit output results to a PDF. That should make them a lot more easy to digest.

We'll start by defining two convenience methods: The first method draws the company logo at the corner of a given `Page`

```
def _write_logo(self, page: Page):
    image_url = "https://icons.iconarchive.com/icons/thesquid.ink/free-flat-sample/256/rubber-duck-icon.png"
    im = PILImage.open(requests.get(image_url, stream=True).raw)
    Image(im).layout(
        page,
        bounding_box=Rectangle(Decimal(20), Decimal(800), Decimal(49), Decimal(18)),
    )
```

The second method adds a colorful footer to a given `Page`

```
def _write_footer(self, page: Page):
    # footer
    rectangle_box = Rectangle(
        Decimal(0),
        Decimal(0),
        page.get_page_info().get_width(),
        page.get_page_info().get_height() * Decimal(0.05),
    )
    Shape(
        LineArtFactory.rectangle(rectangle_box),
        fill_color=HexColor("5dbb46"),
        stroke_color=HexColor("5dbb46"),
        line_width=Decimal(1),
    ).layout(page, rectangle_box)

    rectangle_box = Rectangle(
        Decimal(0),
```

```

        page.get_page_info().get_height() * Decimal(0.05),
        page.get_page_info().get_width(),
        Decimal(2),
    )
Shape(
    LineArtFactory.rectangle(rectangle_box),
    fill_color=X11Color("SlateGray"),
    stroke_color=X11Color("SlateGray"),
    line_width=Decimal(1),
).layout(page, rectangle_box)

```

Next we're going to create a method that writes 1 page of results:

```

def _write_page(self, doc: Document, results, from_index: int, to_index: int):
    page = Page()
    doc.append_page(page)

    # set layout manager
    layout = SingleColumnLayout(page)

    # create Table
    N = to_index - from_index
    table = Table(
        number_of_rows=N + 1,
        number_of_columns=5,
        column_widths=[Decimal(1), Decimal(2), Decimal(3), Decimal(2), Decimal(2)],
    )

    # logo
    self._write_logo(page)

    # header

```

```
table.add(
    Paragraph("Nr.", font_color=HexColor("#5dbb46"), font_size=Decimal(20))
)
table.add(
    Paragraph("Category", font_color=HexColor("#5dbb46"), font_size=Decimal(20))
)
table.add(
    Paragraph("Name", font_color=HexColor("#5dbb46"), font_size=Decimal(20))
)
table.add(
    Paragraph("Time", font_color=HexColor("#5dbb46"), font_size=Decimal(20))
)
table.add(
    Paragraph("Status", font_color=HexColor("#5dbb46"), font_size=Decimal(20))
)

# iterate over results
annotation_positions: typing.Dict[Paragraph, str] = {}
for i, testcase in enumerate(results[from_index:to_index]):
    class_name = testcase.attrib.get("classname", "")
    name = testcase.attrib.get("name", "")
    time = round(Decimal(testcase.attrib.get("time", "0")), 2)
    fail = any([x.tag == "failure" for x in testcase])
    fail_message = next(
        iter(
            [
                x.attrib.get("message", "")
                for x in testcase
                if x.tag == "failure"
            ]
        ),
        "",
    )
)
```

```
table.add(Paragraph(str(i + from_index), font_size=Decimal(8)))
table.add(Paragraph(class_name, font_size=Decimal(8)))
table.add(Paragraph(name, font_size=Decimal(8)))
table.add(
    Paragraph(
        str(time),
        font_size=Decimal(8),
        horizontal_alignment=Alignment.CENTERED,
    )
)
```

If the test passes, we'll print a green 'V', else a red 'X'

```
status_para = None
if fail:
    status_para = Paragraph(
        "X",
        font_color=X11Color("Red"),
        horizontal_alignment=Alignment.CENTERED,
    )
else:
    status_para = Paragraph(
        "V",
        font_color=X11Color("Green"),
        horizontal_alignment=Alignment.CENTERED,
    )
table.add(status_para)
annotation_positions[status_para] = fail_message

table.set_padding_on_all_cells(Decimal(5), Decimal(5), Decimal(5), Decimal(5))
table.set_border_width_on_all_cells(Decimal(0.5))
layout.add(table)
```

During the building of `Table` we also kept track of each status `Paragraph`. Now that layout is completed, all those `LayoutElement` objects should have a bounding box. We can now add annotations right next to them wherever more details (for instance reasons why the test failed) are needed.

```
# add text annotations for failed tests
for p, s in annotation_positions.items():
    if s == "":
        continue
    page.append_text_annotation(
        rectangle=Rectangle(
            p.get_bounding_box().x + Decimal(64),
            p.get_bounding_box().y,
            Decimal(16),
            Decimal(16),
        ),
        contents=s,
    )
```

Lastly, we add the footer on the `Page`

```
# add footer
self._write_footer(page)
```

Now we can work on the main program:

```
# create empty Document
doc = Document()
```

```
# add page
tree = ET.parse("/home/joris/Downloads/testsuite.xml")
testsuite = tree.getroot()
for i in range(0, len(testsuite), 30):
    self._write_page(doc, [x for x in testsuite], i, i + 30)
```

Finally, we'll add a QR code to the last page:

```
# last page, containing QR code
page: Page = Page()
doc.append_page(page)
layout = MultiColumnLayout(page)

# add paragraph
layout.add(
    Paragraph(
        "For more information go to jenkins.com, or scan the following qr code:",
        font="Helvetica-Bold",
    )
)

# add qr code
layout.add(
    Barcode(
        data="https://jenkins.com",
        type=BarcodeType.QR,
        stroke_color=HexColor("#5dbb46"),
        width=Decimal(128),
    )
)
```

```
# footer  
self._write_footer(page)
```

And finally, we can store the PDF

```
# attempt to store PDF  
with open("output.pdf", "wb") as in_file_handle:  
    PDF.dumps(in_file_handle, doc)
```

The result should be something like this:

File View Edit Go Bookmarks Tools Settings Help

< Previous > Next Fit Width Zoom Out Zoom In Browse Text Selection

Thumbnails

Search...

Reviews

Bookmarks

Thumbnails

Nr.	Category	Name	Time	Status
0	actions	GetLock	1.01	✓
1	actions	LoadConnect	25.15	✓
2	gprs	GoToEth	147.69	✓
3	actions	SetConfigProfile	1872.64	✓
4	performance	TestFactorLargePrime	1.46	✓
5	actions	TestFactorLargePrime	101.10	✓
6	actions	KeysPerMinute	6.32	✓
7	actions	CheckEVCCVersion	6.53	✓
8	smoke	CheckJavaVersion	2.05	✓
9	smoke	CheckOverloaded	1.98	✓
10	smoke	CheckLinuxVersion	1.42	✓
11	gprs	SignalQuality	6.24	✓
12	smoke	TestConfigProfile	11.29	✓
13	gprs	GetSimOperator	6.32	✓
14	actions	CheckConnectivity	11.29	✓
15	actions	TestTags	12.29	✓
16	actions	TestLock	11.00	✓
17	actions	TestLockPartial	6.00	✓
18	actions	TestLockWithReset	6.00	✓
19	actions	TestLockWithWait	6.00	✓
20	actions	TestLockWithWaitPartial	6.00	✓
21	actions	TestLockWithWaitPartialWithReset	6.00	✓
22	actions	TestLockWithWaitPartialWithResetPartial	6.00	✓
23	actions	TestLockWithWaitPartialWithResetPartialWithReset	6.00	✓
24	actions	TestLockWithWaitPartialWithResetPartialWithResetPartial	6.00	✓
25	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithReset	6.00	✓
26	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithResetPartial	6.00	✓
27	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithResetPartialWithReset	6.00	✓
28	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithResetPartialWithReset	6.00	✓
29	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithResetPartialWithReset	6.00	✓
30	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithResetPartialWithReset	6.00	✓
31	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithResetPartialWithReset	6.00	✓
32	actions	TestLockWithWaitPartialWithResetPartialWithResetPartialWithResetPartialWithReset	6.00	✓

1

Nr.	Category	Name	Time	Status
0	actions	GetLock	1.01	✓
1	actions	LoadConnect	25.15	✓
2	gprs	GoToEth	147.69	✓
3	actions	SetConfigProfile	1872.64	✓
4	performance	TestFactorLargePrime	1.46	✓
5	actions	KeysPerMinute	101.10	✓
6	smoke	CheckEVCCVersion	6.53	✓
7	smoke	CheckJavaVersion	2.05	✓
8	performance	CheckOverloaded	1.98	✓
9	smoke	CheckLinuxVersion	1.42	✓
10	gprs	SignalQuality	6.24	✓
11	smoke	TestConfigProfile	11.29	✓
12	gprs	GetSimOperator	6.32	✓

1 of 4

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.7 Using Chart

Chart objects can be used to integrate Matplotlib graphics into a PDF Document .

First we'll define a utility method that generates the chart:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as MatplotlibPlot

def _create_plot(self) -> None:
    # Dataset
    df = pd.DataFrame(
        {
            "X": range(1, 101),
            "Y": np.random.randn(100) * 15 + range(1, 101),
            "Z": (np.random.randn(100) * 15 + range(1, 101)) * 2,
        }
    )

    # plot
    fig = MatplotlibPlot.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.scatter(df["X"], df["Y"], df["Z"], c="skyblue", s=60)
    ax.view_init(30, 185)

    return MatplotlibPlot.gcf()
```

Next we'll go about creating a Document , as we usually do, starting with the empty Document

```
# create empty document
pdf: Document = Document()
page: Page = Page()
```

```
pdf.append_page(page)

# set layout
layout = MultiColumnLayout(page)
```

Next we're going to add the `chart` itself. The constructor of `Chart` takes a `matplotlib.pyplot` as input. Optionally, you can specify a `width` and `height`. These work the same as when specifying `width` and `height` on `Image` objects.

```
# add chart
layout.add(Chart(self._create_plot()))
```

We're going to force the `MultiColumnLayout` to go to the next column:

```
layout.switch_to_next_column()
```

We'll add some title and text, giving the reader some details about the plot:

```
# add Heading
layout.add(
    Heading(
        "3D Density Chart",
        font_color=X11Color("YellowGreen"),
        font_size=Decimal(20),
    )
)
layout.add(
    Paragraph(
        "The mplot3D toolkit of Matplotlib allows to easily create 3D scatterplots. "
    )
)
```

```
        "Note that most of the customisations presented in the Scatterplot section will work in 3D as well. "
        "The result can be a bit disappointing since each marker is represented as a dot, not as a sphere.."
    )
)
```

Finally, because I did re-use an existing example, I am going to add some acknowledgements, and links:

```
layout.add(Paragraph("Check out https://python-graph-gallery.com/ for more wonderful examples of plots in Python."))
layout.add(Barcode(data="https://python-graph-gallery.com/", type=BarcodeType.QR,
stroke_color=X11Color("YellowGreen")))
```

Now we're ready to write the Document

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:

File View Edit Go Bookmarks Tools Settings Help

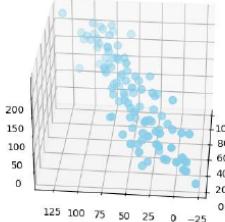
< Previous > Next 100% Zoom Out Zoom In Browse Text Selection

Contents Search... 3D Density Chart 1

3D Density Chart

The matplotlib toolkit of Matplotlib allows to easily create 3D scatterplots. Note that most of the customisations presented in the Scatterplot section will work in 3D as well. The result can be a bit disappointing since each marker is represented as a dot, not as a sphere..

Check out <https://python-graph-gallery.com/> for more wonderful examples of plots in Python.



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.8 Using Shape

`Shape` and `DisjointShape` allow you to draw geometric objects on a `Document`. Use `Shape` if you want to display a continuous figure (a path representing a figure that can be closed by connecting the first and last point), use `DisjointShape` if you have a collection of lines (that are not connected).

2.8.1 Using `Shape` to display a figure from `LineArtFactory`

We'll start by creating an empty `Document`

```
pdf = Document()
page = Page()
pdf.append_page(page)
layout = SingleColumnLayout(page)
```

Next we get `width` and `height` to be able to scale our figure properly

```
w = page.get_page_info().get_width()
h = page.get_page_info().get_height()
assert w is not None
assert h is not None
```

Note that the asserts aren't strictly needed. But the method `get_width` and `get_height` return a `typing.Optional[Decimal]` so, to appease my static typechecker I included the asserts.

```
layout.add(
    Shape(
        LineArtFactory.dragon_curve(
            bounding_box=Rectangle(Decimal(0), Decimal(0), w, h),
            number_of_iterations=10,
```

```
        ),
        stroke_color=HexColor("64B6AC"),
        line_width=Decimal(1),
        fill_color=None,
    )
)
)
```

So, what's happening here?

1. First we construct a `typing.List[typing.Tuple[Decimal, Decimal]]` (a list of points). In this case we're using the `LineArtFactory` to generate them. `LineArtFactory` has a number of methods that generate all kinds of figures (geometric, arrows, stars, etc). For this example, I picked the dragon curve. You can find more information about this curve here:
https://en.wikipedia.org/wiki/Dragon_curve.
2. Next we are feeding these points into the constructor of `Shape`. `Shape` takes a few other arguments as well, including `stroke_color` (the color in which to draw, default= `x11color('black')`), `fill_color`, and `line_width`.

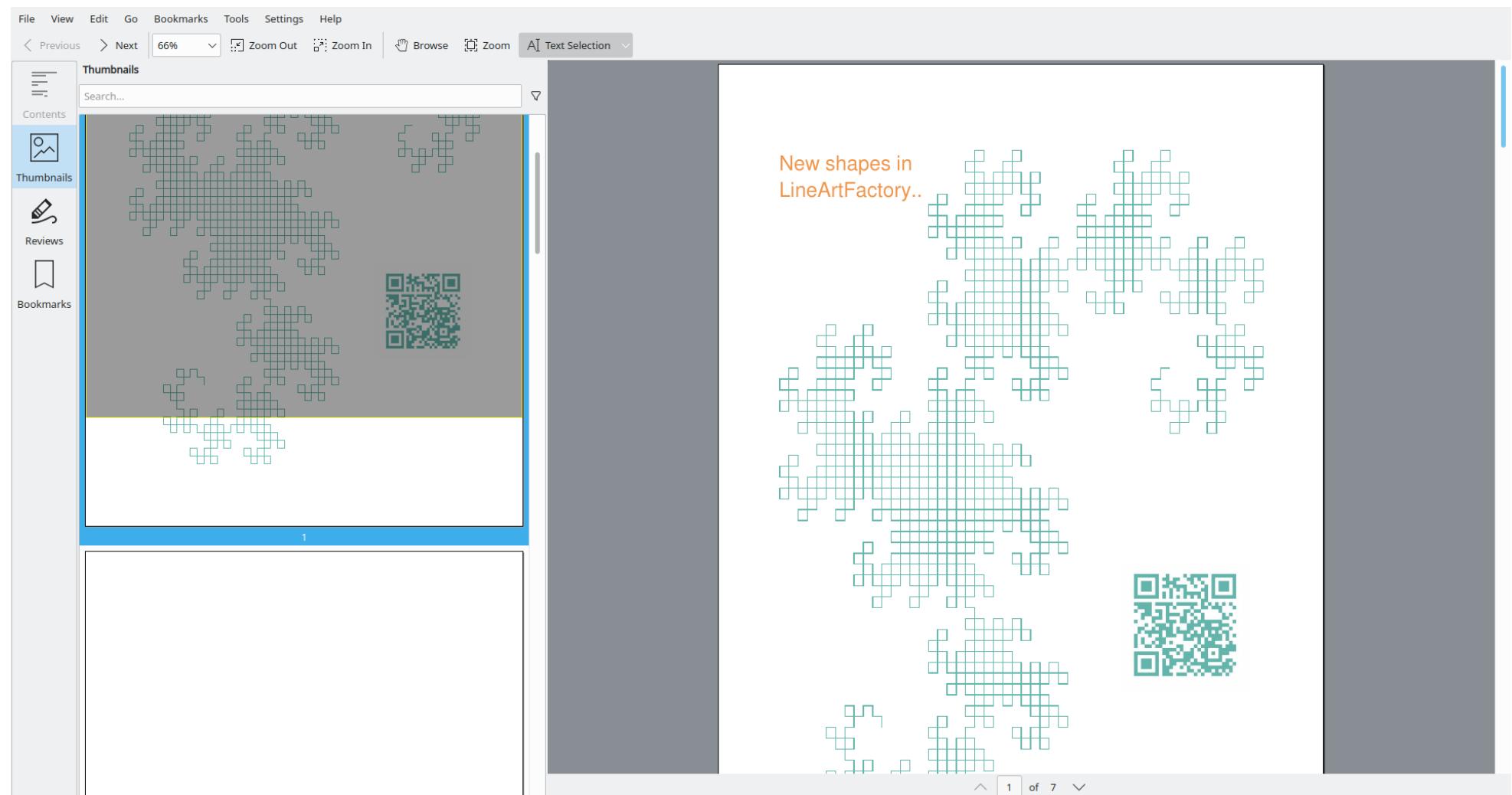
So why do we take the long route? Why not directly paint on the pdf `Canvas`? `Shape` plays nicely with our layout algorithms. If the content needs to be resized or translated during the layout process, `Shape will do so.

It also means we don't have to worry about the precise coordinates when we're building our `Shape`. We can just pretend our figure starts at `(0, 0)` and have `Shape` do the heavy lifting.

Back to our tutorial. We need to store the PDF:

```
# attempt to store PDF
with open("output.pdf", "wb") as in_file_handle:
    PDF.dumps(in_file_handle, pdf)
```

The result should be something like this:



Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.

2.8.2 Using `DisjointShape` to display a maze

Let's put `DisjointShape` to the test by generating a maze and adding it to a PDF. First we'll need some code to generate a `Maze`:

```
class Maze:

    def __init__(self, width: int, height: int):
        assert width > 0
        assert height > 0
        self.width: int = width
        self.height: int = height
        self.cells = [
            [210 for _ in range(0, self.height)] for _ in range(0, self.width)
        ]
        # build the maze
        self._build_maze()
        # pick a start
        self._make_gap()
        # pick an exit
        self._make_gap(reverse_scan_order=True)
```

Each cell of the `Maze` has 4 walls. Each wall is represented by a prime (north=2, east=3, south=5, west=7). Each cell can therefore be represented using 1 number, which is 210 initially.

This algorithm is based on https://en.wikipedia.org/wiki/Maze_generation_algorithm

```
def _unvisited_neighbours(self, x: int, y: int):
    nbs: typing.List[typing.Tuple[int, int]] = []
    for i in range(-1, 2):
        for j in range(-1, 2):
            # self is not a valid neighbour
```

```

        if i == 0 and j == 0:
            continue
        if abs(i) == abs(j) == 1:
            continue
        # check out-of-bounds
        if x + i >= self.width or x + i < 0:
            continue
        if y + j >= self.height or y + j < 0:
            continue
        if self.cells[x + i][y + j] == 210:
            nbs.append((x + i, y + j))
    return nbs

```

This function returns the walls representing the Maze

```

def get_walls(self, cell_size: int) -> typing.List[typing.Tuple[typing.Tuple[Decimal, Decimal],
typing.Tuple[Decimal, Decimal]]]:
    walls: typing.List[typing.Tuple[typing.Tuple[Decimal, Decimal], typing.Tuple[Decimal, Decimal]]] = []
    for i in range(0, self.width):
        for j in range(0, self.height):
            c = self.cells[i][j]
            if c % 2 == 0:
                walls.append([(i * cell_size, j * cell_size), [(i + 1) * cell_size, j * cell_size]])
            if c % 3 == 0:
                walls.append([(i + 1) * cell_size, j * cell_size], [(i + 1) * cell_size, (j + 1) * cell_size]])
            if c % 5 == 0:
                walls.append([(i * cell_size, (j + 1) * cell_size), [(i + 1) * cell_size, (j + 1) * cell_size]])
            if c % 7 == 0:
                walls.append([(i * cell_size, j * cell_size), [i * cell_size, (j + 1) * cell_size]])
    return walls

```

This function actually generates the Maze :

```
def _build_maze(self) -> None:

    # find first cell
    stk: typing.List[typing.Tuple[int, int]] = []
    for i in range(0, self.width):
        for j in range(0, self.height):
            if self.cells[i][j] == 210:
                stk.append((i, j))
                break
        if len(stk) > 0:
            break

    while len(stk) > 0:
        # pop a cell from the stack and make it the current cell
        current_cell: typing.Tuple[int, int] = stk[-1]
        stk.pop(-1)
        # If the current cell has any neighbours which have not been visited
        nbs = self._unvisited_neighbours(current_cell[0], current_cell[1])
        if len(nbs) > 0:
            # Push the current cell to the stack
            stk.append(current_cell)
            # Choose one of the unvisited neighbours
            nb = random.choice(nbs)
            # Remove the wall between the current cell and the chosen cell
            if current_cell[0] == nb[0]:
                if current_cell[1] > nb[1]:
                    self.cells[current_cell[0]][current_cell[1]] /= 2
                    self.cells[nb[0]][nb[1]] /= 5
                elif nb[1] > current_cell[1]:
                    self.cells[current_cell[0]][current_cell[1]] /= 5
                    self.cells[nb[0]][nb[1]] /= 2
```

```

        elif current_cell[1] == nb[1]:
            if current_cell[0] > nb[0]:
                self.cells[current_cell[0]][current_cell[1]] /= 7
                self.cells[nb[0]][nb[1]] /= 3
            elif nb[0] > current_cell[0]:
                self.cells[current_cell[0]][current_cell[1]] /= 3
                self.cells[nb[0]][nb[1]] /= 7
        # Mark the chosen cell as visited and push it to the stack
        stk.append((nb[0], nb[1]))
    
```

This function creates a gap in the `Maze` wall on an edge (representing the start or exit):

```

def _make_gap(self, reverse_scan_order: bool = False):
    xs = ([x for x in reversed(range(0, self.width))] if reverse_scan_order else [x for x in range(0, self.width)])
    ys = ([x for x in reversed(range(0, self.height))] if reverse_scan_order else [x for x in range(0, self.height)])
    for i in xs:
        for j in ys:
            if i == 0 or i == self.width - 1 or j == 0 or j == self.height - 1:
                if self.cells[i][j] != -1:
                    # mark as start
                    if i == 0:
                        self.cells[i][j] /= 7
                        return
                    if i == self.width - 1:
                        self.cells[i][j] /= 3
                        return
                    if j == 0:
                        self.cells[i][j] /= 2
                        return
                    if j == self.height - 1:
                        self.cells[i][j] /= 5
    
```

```

        return
    elif self.cells[i][j] != -1 and (
        self.cells[i - 1][j] == -1
        or self.cells[i + 1][j] == -1
        or self.cells[i][j - 1] == -1
        or self.cells[i][j + 1] == -1
    ):
        if self.cells[i - 1][j] == -1:
            self.cells[i][j] /= 7
            return
        if self.cells[i + 1][j] == -1:
            self.cells[i][j] /= 3
            return
        if self.cells[i][j - 1] == -1:
            self.cells[i][j] /= 2
            return
        if self.cells[i][j + 1] == -1:
            self.cells[i][j] /= 5
            return

```

Finally, with all of that out of the way, let's get to PDF:

```

# create document
pdf = Document()

# add page
page = Page()
pdf.append_page(page)

# set layout
layout = SingleColumnLayout(page)

```

```
# add title
layout.add(
    Paragraph(
        "AMAZING MAZES",
        font="TimesRoman",
        font_size=Decimal(20),
        font_color=HexColor("274029"),
    )
)

# add subtitle
layout.add(
    Paragraph(
        """
        Can you solve this maze?
        Try going from (lower) left to (upper) right.
        Good luck
        """,
        respect_newlines_in_text=True,
    )
)

# generate maze
m = Maze(20, 20)

# add maze
layout.add(
    DisjointShape(
        m.get_walls(Decimal(10)),
        stroke_color=HexColor("315C2B"),
        line_width=Decimal(1),
    )
)
```

The end result should be something like this (keeping in mind the maze is generated randomly, so it might be different on your machine):

A screenshot of a web browser window displaying a maze puzzle. The browser's interface includes a top navigation bar with File, View, Edit, Go, Bookmarks, Tools, Settings, and Help menus, along with Fit Width, Zoom Out, Zoom In, Browse, and Zoom buttons. On the left side, there is a sidebar with icons for Contents, Thumbnails, Reviews, and Bookmarks. The main content area features a large title "AMAZING MAZES" at the top. Below it, there is a text block with instructions: "Can you solve this maze? Try going from (lower) left to (upper) right. Good luck". A large, complex black-and-white maze is centered in the page. The bottom of the browser window shows a page number "1" and a navigation footer with arrows and the text "1 of 1".

Check out the `tests` directory to find more tests like this one, and discover what you can do with `pText`.