PdfPug

Release 0.4

Sep 19, 2019

Contents

PdfPug is a tool that makes it **easy** to create **rich beautiful PDFs** from **scratch**. It provides **simple APIs** that allow for quick creation of PDF files without any fuss. Read more about the vision that drives the development efforts of this library *here*.

PdfPug consists of small building blocks like `Table`, `Header`, `OrderedList` etc. and the ability to customise these component to suit different use cases.

Note: The library is still very new and as a result the APIs can be assumed to not be stable. Do not use this library in production. Any bugs found can be reported at the project's Gitlab repo.

Here is a small example to create a basic PDF that contains a header and a paragraph,

```
from pdfpug.modules import Header, Paragraph
from pdfpug import PdfReport

# Create modules
intro_header = Header("Introduction to PdfPug")
para = Paragraph(
    "Lorem Ipsum is <b>simply</b> <u>dummy</u> text of the printing and typesetting "
    "industry. Lorem Ipsum has been the industry's standard dummy text "
    "ever since the 1500s, when an unknown printer took a galley of type"
    " and scrambled it to make a type specimen book. It has survived not "
    "only five centuries, but also the leap into electronic typesetting, "
    "remaining essentially unchanged. It was popularised in the 1960s with "
    "the release of Letraset sheets containing Lorem Ipsum passages, and "
    "more recently with desktop publishing software like Aldus PageMaker "
    "including versions of Lorem Ipsum."
)

# Create a PdfReport object
report = PdfReport()

# Add modules to report
report.add_element(intro_header)
report.add_element(para)

# Generate PDF Report
report.generate_pdf("test-report.pdf")
```

CHAPTER 1

Licensing

PdfPug uses the **MIT** license and as such is open source and welcomes contribution. The license file can be found in the project's Gitlab repo.

1.1 Vision

If you are wondering about the motivation behind creating yet another Python library to create PDF files? The answer is simple. The Python ecosystem does not have a library that is **simple** enough to use while providing the means to create **rich beautiful professional** PDFs. Either they are simple yet lacking certain features or they are fully featured but difficult to use.

That's great. **Is PdfPug the solution to the above problem?** Well, it is **intended** to be. At the moment, it is still in its infancy to match up to that expectation. But that is the vision that drives the development efforts of PdfPug. Also, balancing features with ease of use is a difficult balance to achieve, but something that is worth trying.

How do we measure success? Just about anyone should be able to create professional looking PDFs in a live coding session. It should not be intimidating.

If this is something that excites you, and you are interested in helping make that a reality, then please do check out the *Contributing Guide*.

1.2 Tutorial

This section strives to introduce PdfPug by means of examples. The examples vary in difficult from simple to complex PDF files. By going through these examples, one can get a good idea of the capabilities and ease of use of PdfPug. Each example will provide a detailed explanation of how we arrived at the final output. The links to the full source code and output PDF file will be provided for each example. You are encouraged to download them and explore them. Tweak them. Improve them.

Note: The examples below are also used by the library author to dogfood the library APIs and check if the usage pattern is simple and understandable.

1.2.1 Prerequisites

It is recommended to set up a clean environment before starting the examples. Let's quickly do that before diving into the examples.

1. Create a new folder
2. Inside the newly created folder, set up a Python Virtual Environment
3. Activate Environment
4. Pip install PdfPug

The commands for Linux are provided below for reference,

```
mkdir pdfpug-examples && cd pdfpug-examples
python -m venv venv
source venv/bin/activate
pip install pdfpug
```

Python Wikipedia Article

In this tutorial, we will walk through how to reproduce the Wikipedia article about Python. You will find the online reference here. This article has a good balance of simple elements like *`Paragraph`*, *`Header`* while also housing slightly complex 2 column layouts. This should serve as a good introduction to the PdfPug layout system.

Note: This tutorial focuses on introducing the various PdfPug modules and layouts. As such, the content of the wikipedia article being showcased will be truncated and be a smaller subset of the actual content in the wikipedia article.

The final output would look something like,

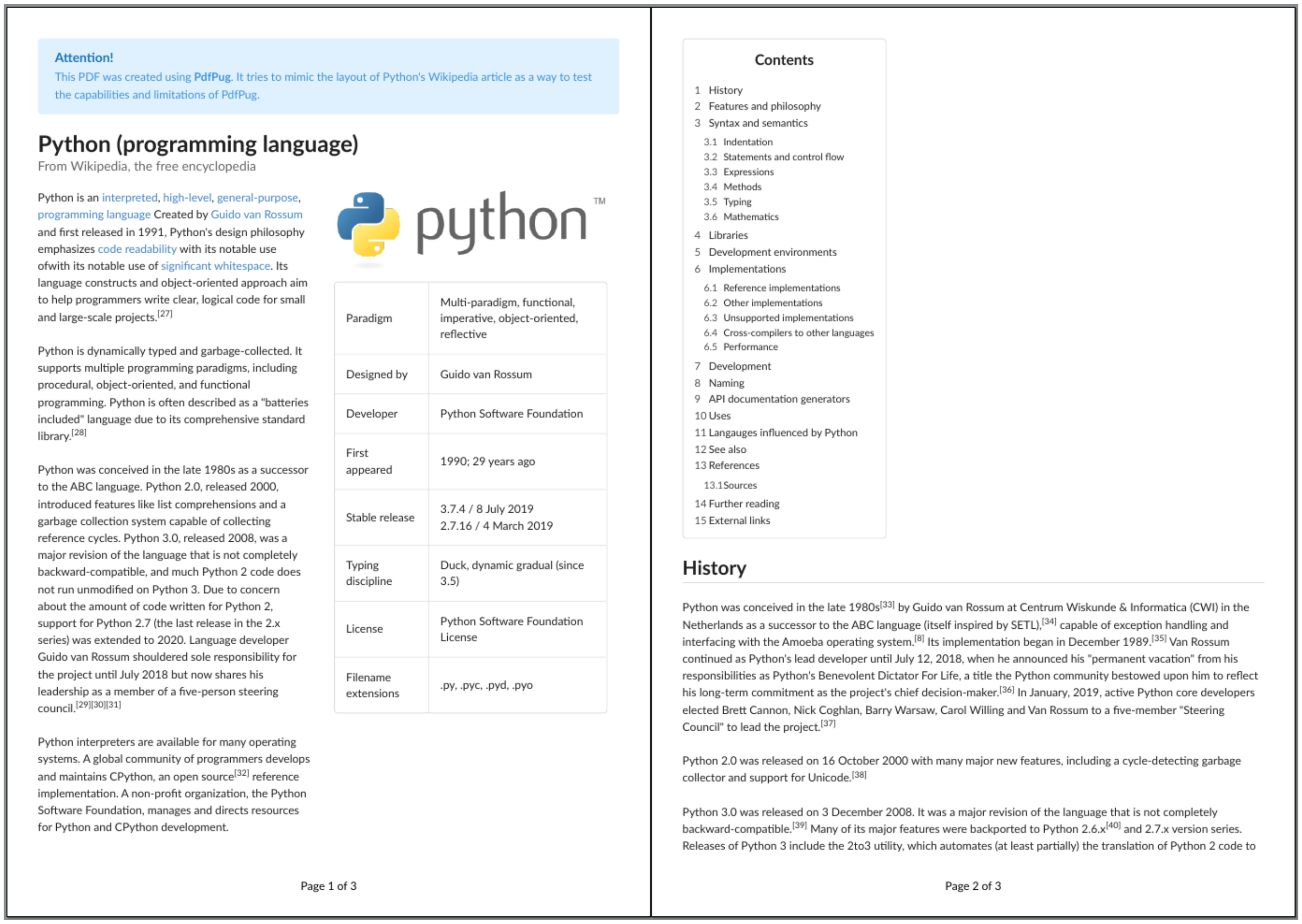
The source code and the output PDF file can be downloaded here. If you notice any discrepancies, do report a bug. `Source Code`, `Output PDF`, `Python Logo Image`

Article Title

The first element to be defined is the article header "Python (programming language)". We can define headers using *`Header`* class. PdfPug header also supports adding a caption (or sub-header) which can be used to add the supporting text "From Wikipedia, the . . . ".

```
from pdfpug.modules import Header
from pdfpug.common import Alignment

main_title = Header(
    "Python (programming language)",
    sub_header="From Wikipedia, the free encyclopedia",
    alignment=Alignment.left,
)
```



Attention!
This PDF was created using **PdfPug**. It tries to mimic the layout of Python's Wikipedia article as a way to test the capabilities and limitations of PdfPug.

Python (programming language)

From Wikipedia, the free encyclopedia

Python is an interpreted, high-level, general-purpose, programming language Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use ofwith its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.[27]

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.[28]

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council.[29][30][31]

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source[32] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

python™

Paradigm	Multi-paradigm, functional, imperative, object-oriented, reflective
Designed by	Guido van Rossum
Developer	Python Software Foundation
First appeared	1990; 29 years ago
Stable release	3.7.4 / 8 July 2019 2.7.16 / 4 March 2019
Typing discipline	Duck, dynamic gradual (since 3.5)
License	Python Software Foundation License
Filename extensions	.py, .pyc, .pyd, .pyo

Page 1 of 3

Contents

History

Python was conceived in the late 1980s[33] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL),[34] capable of exception handling and interfacing with the Amoeba operating system.[8] Its implementation began in December 1989.[35] Van Rossum continued as Python's lead developer until July 12, 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator For Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker.[36] In January, 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.[37]

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.[38]

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible.[39] Many of its major features were backported to Python 2.6.x[40] and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to

Page 2 of 3

Introduction Section

The introduction section is a 2 column layout with an introduction paragraph on the left and a table on the right. There are multiple ways of implementing this layout. We will take the approach of creating the left column first followed by the right column and then add them both to a grid.

Let us start with creating the left column contents which is a `Paragraph` containing URL links and line breaks.

Note: The `Paragraph` class supports formatting text (bold, italics, underline, superscript etc.), adding URL and line breaks.

```python
from pdfpug.modules import Paragraph
from pdfpug.common import superscript, url

# Define URLs before to maintain code sanity
interpreted = url("https://en.wikipedia.org/wiki/Interpreted_language", "interpreted")
guido = url("https://en.wikipedia.org/wiki/Guido_van_Rossum", "Guido van Rossum")
readability = url("https://en.wikipedia.org/wiki/Code_readability", "code readability
↪")
high_level = url(
    "https://en.wikipedia.org/wiki/High-level_programming_language", "high-level"
)
general_purpose = url(
    "https://en.wikipedia.org/wiki/General-purpose_programming_language",
    "general-purpose",
)
programming_language = url(
    "https://en.wikipedia.org/wiki/Programming_language", "programming language"
)
whitespace = url(
    "https://en.wikipedia.org/wiki/Off-side_rule", "significant whitespace"
)

intro_para = Paragraph(
    f"Python is an {interpreted}, {high_level}, {general_purpose}, "
    f"{programming_language} Created by {guido} and first released in 1991, "
    f"Python's design philosophy emphasizes {readability} with its notable use of"
    f"with its notable use of {whitespace}. Its language constructs and "
    f"object-oriented approach aim to help programmers write clear, logical code "
    f"for small and large-scale projects.{superscript('[27]')}"
    f"<br><br>Python is dynamically typed and garbage-collected. It supports multiple
↪"
    f"programming paradigms, including procedural, object-oriented, and functional "
    f'programming. Python is often described as a "batteries included" language '
    f"due to its comprehensive standard library.{superscript('[28]')}<br><br>"
)
```

With the content ready, let's add it to a `Column`. Since we need a 2 column layout, the width of both the left and right column need to be specified.

```python
from pdfpug.layouts import Column

intro_para_column = Column(width=7)
intro_para_column.add_element(intro_para)
```

Let's now proceed to build the right column and its contents. As can be seen, the right column consists of an image and a table. PdfPug allows us to add these content types via the `Image` and `Table` class.

```
from pdfpug.modules import Image, Table

# The Image class expects the absolute file path of the image!
python_logo = Image(
    os.path.join(os.path.dirname(os.path.realpath(__file__)), "python-logo.png")
)

intro_table = Table(
    data=[
        [
            "Paradigm",
            "Multi-paradigm, functional, imperative, object-oriented, reflective",
        ],
        ["Designed by", "Guido van Rossum"],
        ["Developer", "Python Software Foundation"],
        ["First appeared", "1990; 29 years ago"],
        ["Stable release", "3.7.4 / 8 July 2019<br>2.7.16 / 4 March 2019"],
        ["Typing discipline", "Duck, dynamic gradual (since 3.5)"],
        ["License", "Python Software Foundation License"],
        ["Filename extensions", ".py, .pyc, .pyd, .pyo"],
    ]
)
```

Let's again build a new column with its contents,

```
intro_table_column = Column(width=7)
intro_table_column.add_element(python_logo)
intro_table_column.add_element(intro_table)
```

With the left and right column created, the final step to creating the 2 column grid is to create a `Grid` and add these columns to it.

```
from pdfpug.layouts import Grid

intro_grid = Grid()
intro_grid.add_layout(intro_para_column)
intro_grid.add_layout(intro_table_column)
```

Table of Contents

One can observe that the table of contents is actually an ordered list. The list is encapsulated within a segment container. Creating this should be fairly simple.

```
from pdfpug.modules import OrderedList, Segment

contents_list = OrderedList(
    [
        "History",
        "Features and philosophy",
        {
            "Syntax and semantics": [
                "Indentation",
                "Statements and control flow",
                "Expressions",
```

(continues on next page)

(continued from previous page)

```
                "Methods",
                "Typing",
                "Mathematics",
            ]
        },
        "Libraries",
        "Development environments",
        {
            "Implementations": [
                "Reference implementations",
                "Other implementations",
                "Unsupported implementations",
                "Cross-compilers to other languages",
                "Performance",
            ]
        },
        "Development",
        "Naming",
        "API documentation generators",
        "Uses",
        "Langauges influenced by Python",
        "See also",
        {"References": ["Sources"]},
        "Further reading",
        "External links",
    ]
)

contents_segment = Segment(
    [Header("Contents", tier=HeaderTier.h3), contents_list],
    spacing=SegmentSpacing.compact,
)
```

Notice that we are setting *SegmentSpacing.compact* as the segment spacing. This ensures that the segment container takes only the required amount of width. Otherwise, it would span the entire page width.

History & Other Sections

```
history_header = Header(
    "History", tier=HeaderTier.h2, style=HeaderStyle.dividing, alignment=Alignment.
↪left
)

history_para = Paragraph(
    f"Python was conceived in the late 1980s{superscript('[33]')} by Guido van Rossum
↪"
    f"at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to␣
↪the "
    f"ABC language (itself inspired by SETL),{superscript('[34]')} capable of "
    f"exception handling and interfacing with the Amoeba operating system."
    f"{superscript('[8]')} Its implementation began in December 1989."
    f"{superscript('[35]')} Van Rossum continued as Python's lead developer until "
    f'July 12, 2018, when he announced his "permanent vacation" from his '
    f"responsibilities as Python's Benevolent Dictator For Life, a title the "
    f"Python community bestowed upon him to reflect his long-term commitment as "
```

(continues on next page)

(continued from previous page)

```
    f"the project's chief decision-maker.{superscript('[36]')} In January, 2019, "
    f"active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw,
↪"
    f'Carol Willing and Van Rossum to a five-member "Steering Council" to lead the '
    f'project.{superscript("[37]")}'
)

library_header = Header(
    "Libraries",
    tier=HeaderTier.h2,
    style=HeaderStyle.dividing,
    alignment=Alignment.left,
)

library_para = Paragraph(
    "Python's large standard library, commonly cited as one of its greatest strengths,
↪"
    "[97] provides tools suited to many tasks. For Internet-facing applications, "
    "many standard formats and protocols such as MIME and HTTP are supported. It "
    "includes modules for creating graphical user interfaces, connecting to␣
↪relational "
    "databases, generating pseudorandom numbers, arithmetic with arbitrary precision "
    "decimals,[98] manipulating regular expressions, and unit testing."
    "<br><br>Some parts of the standard library are covered by specifications "
    "(for example, the Web Server Gateway Interface (WSGI) implementation wsgiref "
    "follows PEP 333[99]), but most modules are not. They are specified by their "
    "code, internal documentation, and test suites (if supplied). However, because "
    "most of the standard library is cross-platform Python code, only a few modules "
    "need altering or rewriting for variant implementations."
    "<br><br>As of March 2018, the Python Package Index (PyPI), the official "
    "repository for third-party Python software, contains over 130,000[100] "
    "packages with a wide range of functionality, including: "
)

library_list = UnorderedList(
    [
        "Graphical user interfaces",
        "Web frameworks",
        "Multimedia",
        "Databases",
        "Networking",
        "Test frameworks",
        "Automation",
        "Web scraping[101]",
        "Documentation",
        "System administration",
        "Scientific computing",
        "Text processing",
        "Image processing",
    ]
)
```

Building the PDF

The final thing involves importing the `PdfReport` class from the PdfPug library and creating an object. This is the main class that will house all the elements we want to add to our PDF file.

```python
from pdfpug import PdfReport

report = PdfReport("PythonWiki.pdf")
report.add_elements(
    [
        main_title,
        intro_grid,
        contents_segments,
        history_header,
        history_para,
        library_header,
        library_para,
        library_list,
    ]
)

report.generate_pdf("python.pdf")
```

Voila! This should generate a PDF file similar to the output shown at the start of this tutorial.

Modern Resume

In this tutorial, we will walk through creating a modern resume. This tutorial is fairly extensive and uses a majority of the PdfPug modules and their properties to achieve the desired look and feel. If you are unfamiliar with some of the basic elements of PdfPug, it is recommended to first go through the *Python Wikipedia Article* tutorial that is easier than and smaller.

The final output would look something like the screenshot below. Doesn't it look great? Let's build that!

Note: The information displayed in the resume may contain factual errors. The point of this tutorial is to explore PdfPug's elements and layouts and showcase its capabilities.

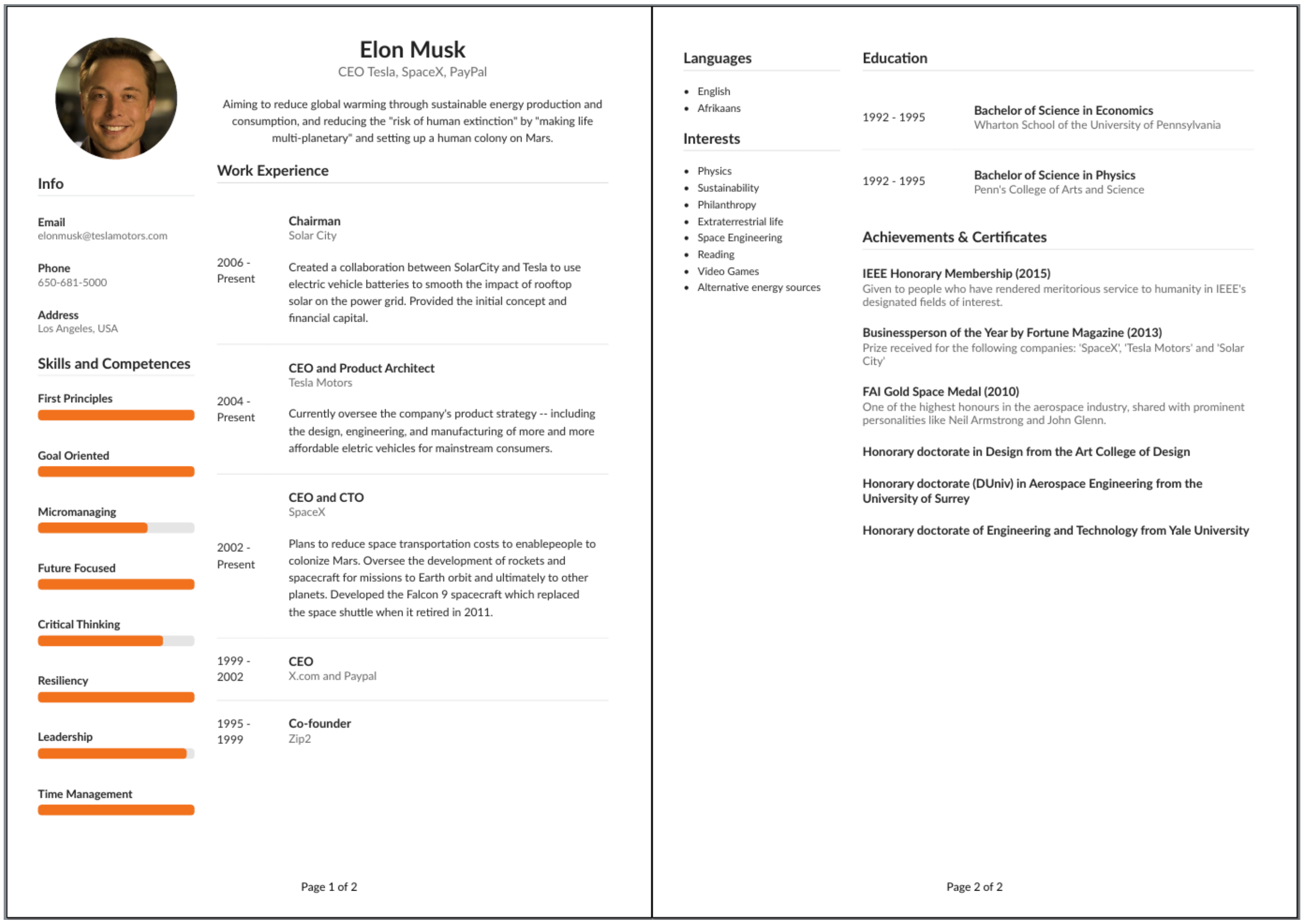
The source code and the output PDF file can be downloaded here. If you notice any discrepancies, do report a bug. `Source Code`, `Output PDF`, `Elon Musk Profile Picture`

Approach

Looking at the output, at a high level, this is a 2 column grid that contains a mixture of elements like headers, list, paragraphs, tables and even progress bars to indicate skill level. A layout like this should be implemented one at a time to take an organised approach.

A possible starting point could be the left column that is fairly simple and then moving on to the right column that is slightly more complex due to the table that contains other elements i.e header inside a cell inside a table. Inception!

Warning: There is a known bug where a `Grid` that bleeds to the next page causes the layout to go haywire. Due to this limitation, in this tutorial 2 grid were used. One for the first page and the other for the second page.



Elon Musk

CEO Tesla, SpaceX, PayPal

Aiming to reduce global warming through sustainable energy production and consumption, and reducing the "risk of human extinction" by "making life multi-planetary" and setting up a human colony on Mars.

Info

Email
elonmusk@teslamotors.com

Phone
650-681-5000

Address
Los Angeles, USA

Skills and Competences

First Principles

Goal Oriented

Micromanaging

Future Focused

Critical Thinking

Resiliency

Leadership

Time Management

Work Experience

2006 - Present	**Chairman** Solar City Created a collaboration between SolarCity and Tesla to use electric vehicle batteries to smooth the impact of rooftop solar on the power grid. Provided the initial concept and financial capital.
2004 - Present	**CEO and Product Architect** Tesla Motors Currently oversee the company's product strategy -- including the design, engineering, and manufacturing of more and more affordable eletric vehicles for mainstream consumers.
2002 - Present	**CEO and CTO** SpaceX Plans to reduce space transportation costs to enablepeople to colonize Mars. Oversee the development of rockets and spacecraft for missions to Earth orbit and ultimately to other planets. Developed the Falcon 9 spacecraft which replaced the space shuttle when it retired in 2011.
1999 - 2002	**CEO** X.com and Paypal
1995 - 1999	**Co-founder** Zip2

Page 1 of 2

Languages

- English
- Afrikaans

Interests

- Physics
- Sustainability
- Philanthropy
- Extraterrestrial life
- Space Engineering
- Reading
- Video Games
- Alternative energy sources

Education

1992 - 1995	**Bachelor of Science in Economics** Wharton School of the University of Pennsylvania
1992 - 1995	**Bachelor of Science in Physics** Penn's College of Arts and Science

Achievements & Certificates

IEEE Honorary Membership (2015)
Given to people who have rendered meritorious service to humanity in IEEE's designated fields of interest.

Businessperson of the Year by Fortune Magazine (2013)
Prize received for the following companies: 'SpaceX', 'Tesla Motors' and 'Solar City'

FAI Gold Space Medal (2010)
One of the highest honours in the aerospace industry, shared with prominent personalities like Neil Armstrong and John Glenn.

Honorary doctorate in Design from the Art College of Design

Honorary doctorate (DUniv) in Aerospace Engineering from the University of Surrey

Honorary doctorate of Engineering and Technology from Yale University

Page 2 of 2

First Page - Left Column

The first element we need to build is an image that should be circular and centered to the left column layout.

```python
from pdfpug.modules import Image
from pdfpug.common import ImageLayout, ImageStyle, Size

profile_pic = Image(
    os.path.join(os.path.dirname(os.path.realpath(__file__)), "musk.jpeg"),
    style=ImageStyle.circular,
    size=Size.small,
    layout=ImageLayout.centered,
)
```

This is followed by the the info section which comprises of just headers. In the code block below, playing with the *HeaderTier*, *Alignment* and adding a sub-header helped achieved the style. In order to have a dividing horizontal line be drawn after the info header, a dividing *HeaderStyle* is used.

```python
from pdfpug.modules import Header
from pdfpug.common import HeaderTier, HeaderStyle, Alignment

info_header = Header(
    "Info", tier=HeaderTier.h3, style=HeaderStyle.dividing, alignment=Alignment.left
)

email = Header(
    "Email",
    sub_header="elonmusk@teslamotors.com",
    alignment=Alignment.left,
    tier=HeaderTier.h5,
)
```

Next up is the skills and competences section. Although this requires an unconventional element, it appears to be the best fit for the use case. The *ProgressBar* element supports various modifications to its default style like *Size*, *Color*, title etc.

```python
from pdfpug.modules import ProgressBar
from pdfpug.common import Color

skills_header = Header(
    "Skills and Competences",
    tier=HeaderTier.h3,
    style=HeaderStyle.dividing,
    alignment=Alignment.left,
)

resiliency = ProgressBar(100, title="Resiliency", size=Size.small, color=Color.orange)
```

With the content created, we can add them all to a column.

```python
from pdfpug.layouts import Column

first_page_left_column = Column(width=4)
first_page_left_column.add_element(profile_pic)
first_page_left_column.add_element(info_header)
first_page_left_column.add_element(email)
first_page_left_column.add_element(skills_header)
```

(continues on next page)

(continued from previous page)

```
first_page_left_column.add_element(resiliency)
```

First Page - Right Column

In the right column, there is the resume title that displays the name and the current designation. There is a subtle difference in this header size. It is bigger than a *h1* tier header. How do one achieve that? Using the *size* attribute that takes in `Size` enum.

Warning: It is important to note that the header size can be defined either using the *tier* or *size* attribute but not both!

```
name_header = Header(
    "Elon Musk", sub_header="CEO Tesla, SpaceX, PayPal", size=Size.huge, tier=None
)
```

This is followed by a brief abstract that can be easily implemented using the `Paragraph` element with one minor adjustment to the *alignment* attribute to ensure that the content is centered.

```
summary = Paragraph(
    "Aiming to reduce global warming through sustainable energy production and "
    'consumption, and reducing the "risk of human extinction" by '
    '"making life multi-planetary" and setting up a human colony on Mars.',
    alignment=ParagraphAlignment.center,
)
```

Now comes the tricky work experience section. At a quick glance, it is fairly obvious that this is a `Table`. However, looking closer, there are cells that would need to house other PdfPug elements like header, paragraph to achieve the desired appearance. This requires us to use the `Cell` element to implement that inception of elements.

Going by the bottom top approach, the contents can be created using a header and a paragraph. This hybrid content need to displayed in a vertical layout which can be achieved using a `Segment` element designed to group content together. However, the style should be set to *SegmentType.basic* to ensure that it does not draw any borders. Finally, this element should added to a `Cell` which in turn is the basic building block of a `Table`.

```
work_header = Header(
    "Work Experience",
    tier=HeaderTier.h3,
    style=HeaderStyle.dividing,
    alignment=Alignment.left,
)

work_exp = Table(
    data=[
        [
            "2006 - Present",
            Cell(
                Segment(
                    [
                        Header(
                            "Chairman",
                            sub_header="Solar City",
                            alignment=Alignment.left,
```

(continues on next page)

(continued from previous page)

```
                                tier=HeaderTier.h4,
                            ),
                            Paragraph(
                                "Created a collaboration between SolarCity and Tesla "
                                "to use electric vehicle batteries to smooth the "
                                "impact of rooftop solar on the power grid. Provided "
                                "the initial concept and financial capital."
                            ),
                        ],
                        segment_type=SegmentType.basic,
                        spacing=SegmentSpacing.compact,
                    )
                ),
            ]
    ],
    spacing=TableSpacing.compact,
    table_type=TableType.bare,
)
```

Oh, another minor detail to notice is that the table style is set to *TableType.bare* to ensure no boundaries are drawn. Take a look at `TableType` for other table styles.

Finally,

```
report = PdfReport()
report.add_element(first_page_grid)
report.generate_pdf("modern_resume_tutorial.pdf")
```

This is where the tutorial can be wrapped up. The contents and layout of page 2 are fairly simple to implement yourself. Give it a try. If you are stuck, you can always refer to the source code linked at the start of this tutorial.

1.3 API Documentation

Modules are the building blocks of a PDF report. PdfPug provides several modules like `Header`, `OrderedList` that can be used to put together a PDF report. Modules can also take other modules as inputs.

1.3.1 PdfReport

class pdfpug.**PdfReport**(***kwargs*)

This is the main class that assembles the elements together to create the final PDF output. All the PdfPug elements defined in the *API Documentation* section need to be added to this class.

Parameters **theme** (*Optional[Theme]*) – PDF file theme

```
>>> from pdfpug.modules import Header
>>> from pdfpug import PdfReport
>>> header = Header('PdfPug Header')
>>> report = PdfReport()
>>> report.add_element(header)
>>> report.generate_pdf('pug-report.pdf')
```

PdfPug ships with a predefined themes that can be used to further style and modernise the output pdf file.

```
>>> from pdfpug.common import Theme
>>> report = PdfReport(theme=Theme.mood_swing)
>>> report.generate_pdf('pug-report.pdf')
```

add_elements (*element*)
Add an element to the PDF file

Parameters element (BasePugElement) – Object instance of the different modules supported by PdfPug

Raises TypeError – If object instance is not a PdfPug element

Return type None

add_elements (*elements*)
Add multiple elements in one call to the PDF file

Parameters elements (List[BasePugElement]) – Each element must be an object instance supported by PdfPug

Raises TypeError – If object instance is not a PdfPug element

Return type None

set_meta_information (*title=None, description=None, authors=None, keywords=None*)
Set the document's meta information such as title, description, author etc.

Parameters

- **title** (Optional[str]) – Document title
- **description** (Optional[str]) – Document description
- **authors** (Optional[List[~T]]) – Document authors
- **keywords** (Optional[List[~T]]) – Document keywords

Return type None

generate_pdf (*pdf_file_path*)
Generate PDF file

Parameters pdf_file_path (str) – Absolute path of the PDF file to be created

Return type None

1.3.2 Header

class pdfpug.modules.**Header** (*text, **kwargs*)
A *header* element provides a short summary of the body text

It is separated from the body element and has a strong distinct style to stand above all other elements. Headers give a sense of orientation to the reader.

This class supports a wide variety of customisation that can be applied to a header from changing the header weight to the horizontal placement, color or style.

Instantiating a header is as simple as the following,

```
>>> from pdfpug.modules import Header
>>> intro_header = Header('Introduction')
```

Want to customise the header weight and color?

```
>>> from pdfpug.common import HeaderTier, Color
>>> intro_header.tier = HeaderTier.h2
>>> intro_header.color = Color.red
```

Note: The header size can be set using either the `tier` or `size` parameter. **Do not set both!** Doing so will result in a `ValueError` being raised! By default, header tier is set to `HeaderTier.h1`. When setting `size`, be sure to set `tier` to None.

Parameters

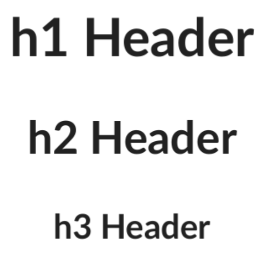
- **text** (`str`) – Header text
- **sub_header** (*Optional[str]*) – Caption (sub header) below the header
- **tier** (`HeaderTier`) – Header weight (defaults to `HeaderTier.h1`)
- **alignment** (`Alignment`) – Horizontal placement (defaults to `Alignment.center`)
- **size** (*Optional[Size]*) – Size of header
- **color** (*Optional[Color]*) – Color of the header text
- **style** (*Optional[HeaderStyle]*) – Visual style of header

class pdfpug.common.**HeaderTier**

Enum Weights to set the hierarchy of a header

The weights are compatible with Markdown levels such as h1, h2, h3 etc.

```
>>> from pdfpug.modules import Header
>>> from pdfpug.common import HeaderTier
>>> h1_header = Header('h1 Header', tier=HeaderTier.h1)
>>> h2_header = Header('h2 Header', tier=HeaderTier.h2)
>>> h3_header = Header('h3 Header', tier=HeaderTier.h3)
```


h1 Header

h2 Header

h3 Header

h1 = 'h1'

Page level header. Equivalent to a markdown h1 header.

h2 = 'h2'

Section level header. Equivalent to a markdown h2 header.

h3 = 'h3'

Paragraph level header. Equivalent to a markdown h3 header.

h4 = 'h4'

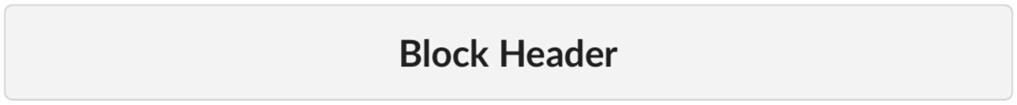
Paragraph level header. Equivalent to a markdown h4 header.

h5 = 'h5'
Paragraph level header. Equivalent to a markdown h5 header.

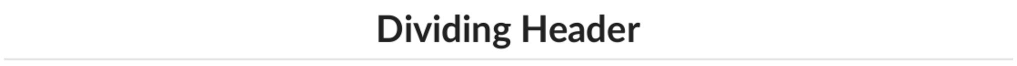
class pdfpug.common.**HeaderStyle**
Enum header styles

```
>>> from pdfpug.modules import Header
>>> from pdfpug.common import HeaderStyle
>>> block_header = Header('Block Header', style=HeaderStyle.block)
```

block = 'block'
The header is formatted to appear inside a content block



dividing = 'dividing'
The header is formatted to divide itself from the content below it using a horizontal line



1.3.3 Paragraph

class pdfpug.modules.**Paragraph**(*text*, ***kwargs*)
Paragraphs are considered as one of the core elements of any report with each paragraph being a self-contained unit around a central idea.

Parameters

- **text** (str) – Paragraph text
- **alignment** (ParagraphAlignment) – Horizontal paragraph alignment (defaults to ParagraphAlignment.left)

Instantiating a paragraph is as simple as the following,

```
>>> from pdfpug.modules import Paragraph
>>> para = Paragraph("Lorem Ipsum is simply dummy text of the printing industry")
```

This component supports rich HTML formatting options like <b>, <i>, <u> tags.

```
>>> para = Paragraph("Lorem Ipsum is <b>simply</b> <u>dummy</u> text!")
```

class pdfpug.common.**ParagraphAlignment**
Enum Alignment options

center = 'center'
Center align content

left = 'left'
Left align content

right = 'right'
Right align content

1.3.4 List

class pdfpug.modules.**OrderedList**(*items*, ***kwargs*)

OrderedList is used to represent an *ordered* list of items in numerical order format.

Parameters

- **items** (*List*) – List items
- **orientation** (Orientation) – Orientation of list (defaults to `Orientation.vertical`)
- **size** (Size) – Size (defaults to `Size.small`)

Instantiating a list is as simple as the following,

```
>>> from pdfpug.modules import OrderedList
>>> chapters = OrderedList(['Introduction', {'Chapter 1': ['Header 1', 'Header 2
↪']}])
```

class pdfpug.modules.**UnorderedList**(*items*, ***kwargs*)

UnorderedList is used to represent an *unordered* list of items in bullet format.

Parameters

- **items** (*List*) – List items
- **orientation** (Orientation) – Orientation of list (defaults to `Orientation.vertical`)
- **size** (Size) – Size (defaults to `Size.small`)

Instantiating a list is as simple as the following,

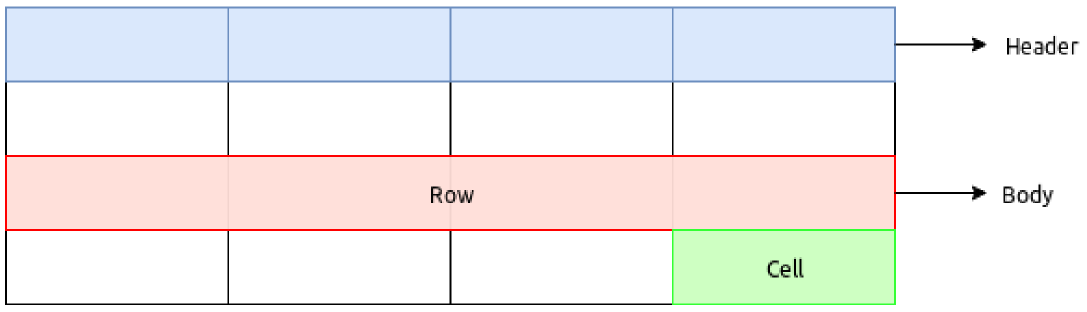
```
>>> from pdfpug.modules import UnorderedList
>>> fruits = UnorderedList(['Apples', 'Oranges', 'Grapes'])
```

1.3.5 Table

class pdfpug.modules.**Table**(*data*, ***kwargs*)

A Table lists data in organised manner making it easier to digest large amounts of data. It is made up of *Row* and *Cell* as shown in the screenshot.

It is also worth noting that the header and body of a table are also comprised of the same. The header and body attributes exist primarily for style changes. Header contents have a stronger style by being in bold and allow the reader to be informed of what the categories of data are. The body counterpart places more emphasis on placing the content in an organised manner so to speak.

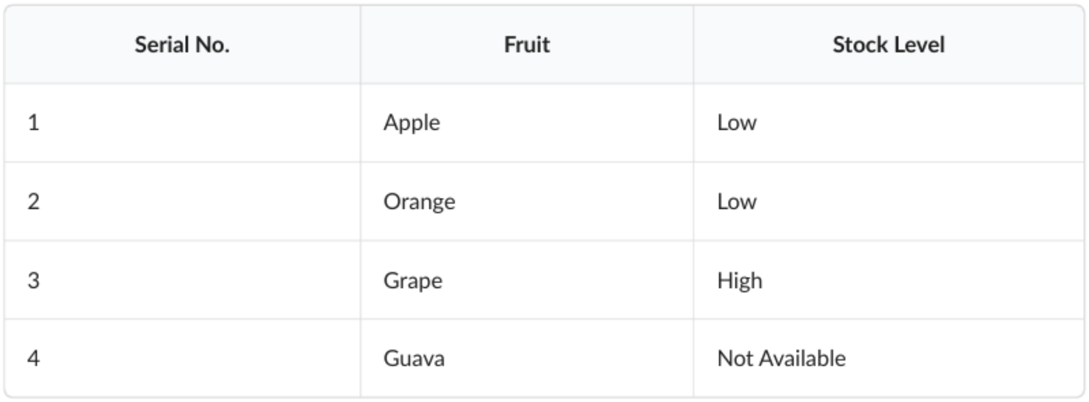


Parameters

- **header** (*Optional[List]*) – Header row
- **data** (*List[List]*) – Body rows
- **spacing** (`TableSpacing`) – Table spacing (defaults to `TableSpacing.comfortable`)
- **striped** (*Optional[*`TableRowStyle`*]*) – Table row style
- **table_type** (`TableType`) – Table type (defaults to `TableType.celled`)
- **color** (*Optional[*`Color`*]*) – Table color
- **column_width_rule** (*Optional[*`TableColumnWidth`*]*) – Table column width

A simple table consisting of just strings and numbers can be created as shown below.

```
>>> from pdfpug.modules import Table
>>> basic_table = Table(
...     header=['Serial No.', 'Fruit', 'Stock Level'],
...     data=
...      [
...          [1, 'Apple', 'Low'],
...          [2, 'Orange', 'Low'],
...          [3, 'Grape', 'High'],
...          [4, 'Guava', 'Not Available']
...      ],
... )
```



Serial No.	Fruit	Stock Level
1	Apple	Low
2	Orange	Low
3	Grape	High
4	Guava	Not Available

More formatting options are unlocked if the *Row* and *Cell* are used. A Cell allows for embedding of other elements like *Header* etc thereby providing more control of the content layouts and style.

A more advanced table would looks something like the following where the cell content alignment is modified. Also, the table has alternate row colored different and uses a compact style.

```
>>> from pdfpug.modules import Cell, Row
>>> from pdfpug.common import TableSpacing, TableRowStyle, State, Alignment
>>> advanced_table = Table(
>>>     header=['Player', 'Hero', 'Role', 'K/D/A'],
>>>     data=
...     [
...         Row(
...             ['Kuro', 'Lion', Cell('Support', row_span=2), '2/10/15'],
```

(continues on next page)

(continued from previous page)

```
...                 alignment=Alignment.center,
...                 state=State.negative
...             ),
...             Row(['Gh', 'Oracle', '3/7/6'], alignment=Alignment.center),
...             Row(['Miracle', 'Void', 'Carry', '9/2/4'], alignment=Alignment.
↪center),
...             Row(['W33', 'Timber', 'Midlaner', '5/8/2'], alignment=Alignment.
↪center)
...         ],
...     spacing=TableSpacing.compact,
...     striped=TableRowStyle.striped,
... )
```

Player	Hero	Role	K/D/A
Kuro	Lion	Support	2/10/15
Gh	Oracle		3/7/6
Miracle	Void	Carry	9/2/4
Mind Control	Timbersaw	Offlaner	5/8/2

class `pdfpug.modules.`**`Row`**(*data*, ***kwargs*)

A Row is the next higher order element above `Cell`. Multiple Rows make up a `Table` similar to how multiple `Cell` make a Row.

Parameters

- **`data`** (*`List`*) – Row contents
- **`row_type`** (`TableRowType`) – Row type (defaults to `TableRowType.body`)
- **`state`** (*`Optional[State]`*) – Row state
- **`alignment`** (*`Optional[Alignment]`*) – Horizontal alignment of row contents

```
>>> from pdfpug.modules import Row, Cell, Header
>>> row = Row(
...     ['Cell 1', 'Cell 2', Cell(Header('Inception'))], alignment=Alignment.left
... )
```

class `pdfpug.modules.`**`Cell`**(*data*, ***kwargs*)

A Cell is the most basic unit (lowest denominator) of a `Table`. A group of cells together form a `Row`.

Parameters

- **`BasePugElement] data`** (*`Union[str,`*) – Cell content
- **`cell_type`** (`TableRowType`) – Cell type (defaults to `TableRowType.body`)
- **`width`** (*`Optional[int]`*) – Cell width (should be in the range of 1-16 & only set for `TableRowType.header` cell type)
- **`row_span`** (*`Optional[int]`*) – Cell span across rows
- **`column_span`** (*`Optional[int]`*) – Cell span across columns

- **`state`** (*`Optional[State]`*) – Cell content state
- **`alignment`** (*`Optional[Alignment]`*) – Cell content horizontal alignment

It can contain a simple string to complex elements like *`Header`*, *`OrderedList`* etc. This allows for embedding all kinds of data in a Cell.

```
>>> from pdfpug.modules import Cell, Header
>>> header_cell = Cell(Header('Header Inside Cell'))
```

A Cell has various customisation attributes that enable data to be represented accurately. For instance, if certain content need to be represented positively, one can do the following,

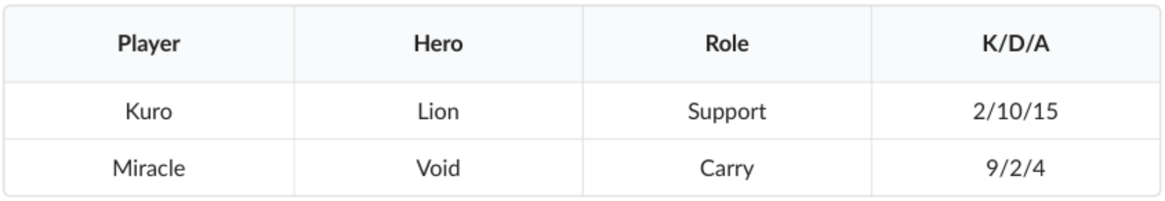
```
>>> from pdfpug.common import State
>>> pos_cell = Cell('Available', state=State.positive)
```

class `pdfpug.common.`**`TableType`**

Enum Table types

`celled = 'celled'`

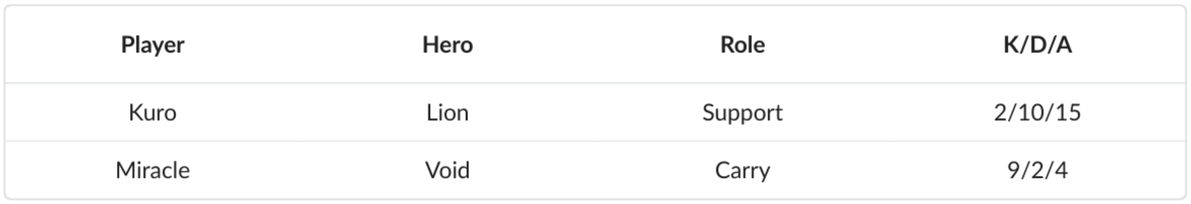
Default table style with each cell clearly visible due to separators



Player	Hero	Role	K/D/A
Kuro	Lion	Support	2/10/15
Miracle	Void	Carry	9/2/4

`simple = 'basic'`

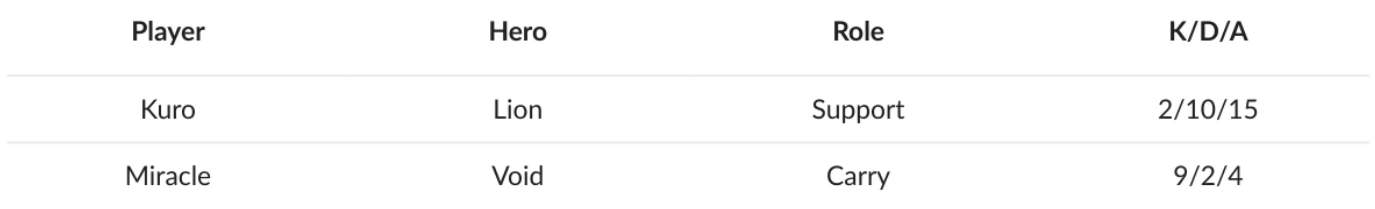
Bare minimum row separating lines with table border



Player	Hero	Role	K/D/A
Kuro	Lion	Support	2/10/15
Miracle	Void	Carry	9/2/4

`bare = 'very basic'`

Bare minimum row separating lines and no table border



Player	Hero	Role	K/D/A
Kuro	Lion	Support	2/10/15
Miracle	Void	Carry	9/2/4

class `pdfpug.common.`**`TableColumnWidth`**

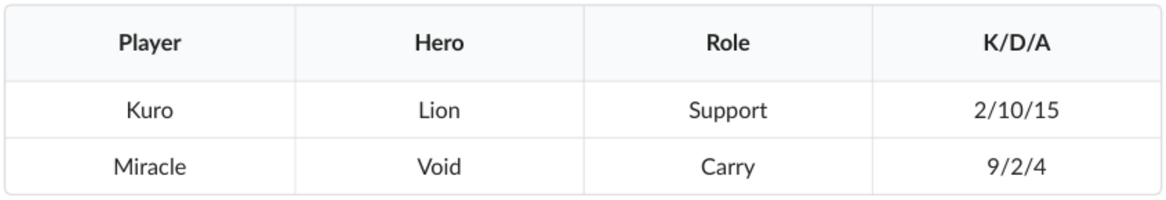
Enum Table column width rules

`fixed = 'fixed'`

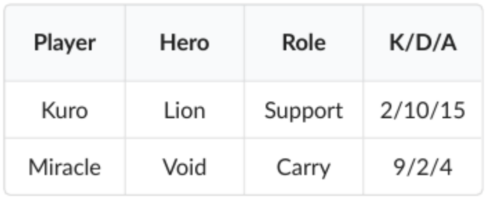
Equal widths for all columns

`minimum = 'collapsing'`

Minimum width for each column based on their content



Player	Hero	Role	K/D/A
Kuro	Lion	Support	2/10/15
Miracle	Void	Carry	9/2/4



Player	Hero	Role	K/D/A
Kuro	Lion	Support	2/10/15
Miracle	Void	Carry	9/2/4

class pdfpug.common.**TableSpacing**
Enum Table row spacing

tight = 'very compact'
Tight spacing of row content

compact = 'compact'
Compact spacing of row content

comfortable = 'padded'
Good spacing of row content

spacious = 'very padded'
Spacious padding of row content

class pdfpug.common.**TableRowStyle**
Table row style

striped = 'striped'
Set if alternate rows should be colored differently

class pdfpug.common.**TableRowType**
Table row type

header = 'th'
Header row

body = 'td'
Body row

1.3.6 Image

class pdfpug.modules.**Image**(*path*, ***kwargs*)
Embed picturesque visuals using the `Image` class with different styles

Parameters

- **path** (`str`) – Absolute path of image
- **size** (*Optional[Size]*) – Size of image
- **style** (*Optional[ImageStyle]*) – Render style
- **layout** (*Optional[ImageLayout]*) – Layout options

Instantiating an image is as simple as the following,

```
>>> from pdfpug.modules import Image
>>> img = Image('/home/johndoe/image.png', size=Size.small, style=ImageStyle.
↪rounded)
```

class pdfpug.modules.**Images**(*images*, ***kwargs*)

Embed a row of images together using the `Images` class.

Parameters

- **images** (List[Image]) – Group of images
- **size** (*Optional*[*Size*]) – Common size of group images

```
>>> from pdfpug.modules import Image, Images
>>> images = Images(
...     [
...         Image('/home/johndoe/image1.png'),
...         Image('/home/johndoe/image2.png'),
...         Image('/home/johndoe/image3.png')
...     ],
...     size=Size.small
... )
```

class pdfpug.common.**ImageStyle**

Enum Image Style

avatar = 'avatar'

Image which appears inline as an avatar (circular image)

rounded = 'rounded'

Image with rounded edges

circular = 'circular'

Crop image into a circular shape. The input image should have the same width and height for this style to work.

class pdfpug.common.**ImageLayout**

Enum Image Layouts

left_float = 'left float'

Float to the left of neighbouring content

right_float = 'right float'

Float to the right of neighbouring content

centered = 'centered'

Horizontally center the image

1.3.7 Segment

class pdfpug.modules.**Segment**(*data*, ***kwargs*)

A segment is used to create a grouping of related content.

Parameters

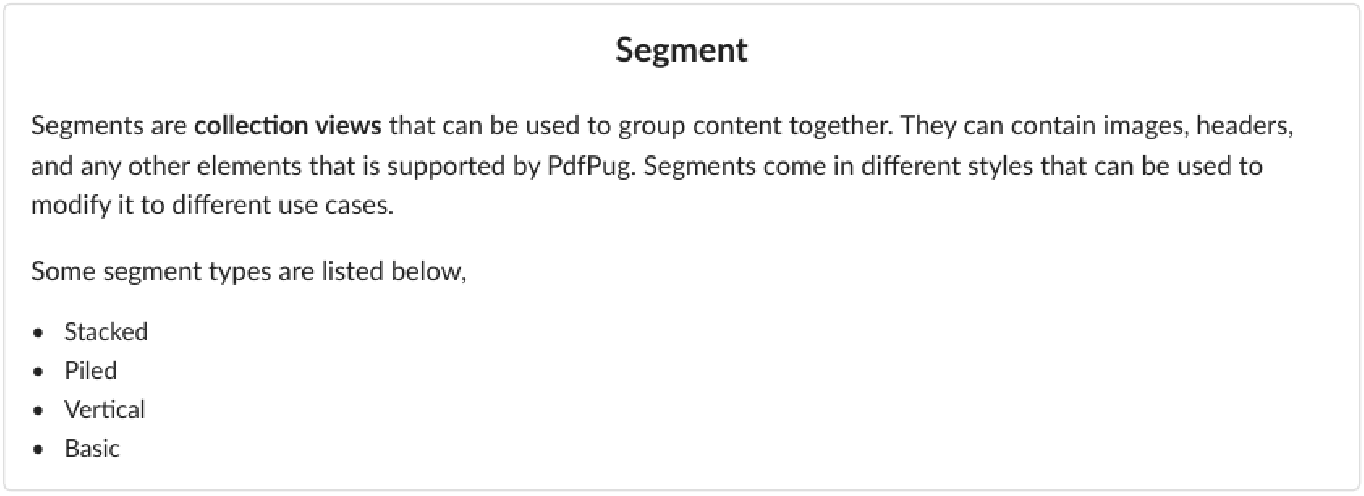
- **data** – Content to be grouped
- **segment_type** (*Optional[SegmentType]*) – Visual style
- **aligment** (*Optional[Alignment]*) – Horizontal alignment of all content
- **spacing** (*Optional[SegmentSpacing]*) – Padding around the content
- **emphasis** (*Optional[SegmentEmphasis]*) – Emphasis strength of segment

```
>>> from pdfpug.modules import Segment, Header, Paragraph, UnorderedList
>>> from pdfpug.common import HeaderTier
>>> segment = Segment(
```

(continues on next page)

(continued from previous page)

```
...     [
...         Header('Segment', tier=HeaderTier.h3),
...         Paragraph(
...             'Segments are <b>collection views</b> that can be used to group '
...             'content together. They can contain images, headers, and any '
...             'other elements that is supported by PdfPug. Segments come in '
...             'different styles that can be used to modify it to different use '
...             'cases.'
...         ),
...         Paragraph('Some segment types are listed below,'),
...         UnorderedList(['Stacked', 'Piled', 'Vertical', 'Basic'])
...     ],
... )
```



Segment

Segments are **collection views** that can be used to group content together. They can contain images, headers, and any other elements that is supported by PdfPug. Segments come in different styles that can be used to modify it to different use cases.

Some segment types are listed below,

- Stacked
- Piled
- Vertical
- Basic

The appearance of segments can be styled for different use cases and preferences,

```
>>> from pdfpug.common import SegmentType
>>> segment.segment_type = SegmentType.stacked
```

class pdfpug.modules.**Segments**(*segments*, ***kwargs*)

A group of *Segment* can be formatted to appear together using *Segments*.

Parameters

- **segments** (List[Union[Segment, Segments]]) – Group of segments
- **segments_type** (*Optional*[SegmentType]) – Visual style
- **orientation** (*Optional*[Orientation]) – Orientation of elements

class pdfpug.common.**SegmentType**

Enum Segment Type

basic = 'basic'

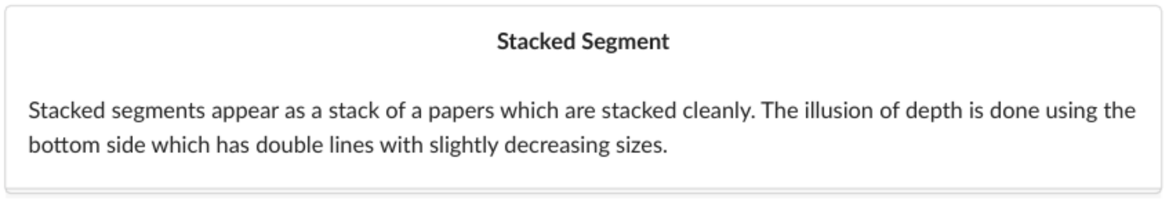
Basic segment type with no special formatting

stacked = 'stacked'

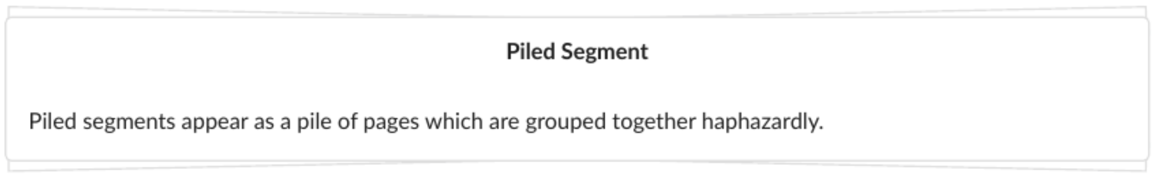
Segment that appears to contain multiple pages which are stacked cleanly

piled = 'piled'

Segment that appears to look like a pile of papers

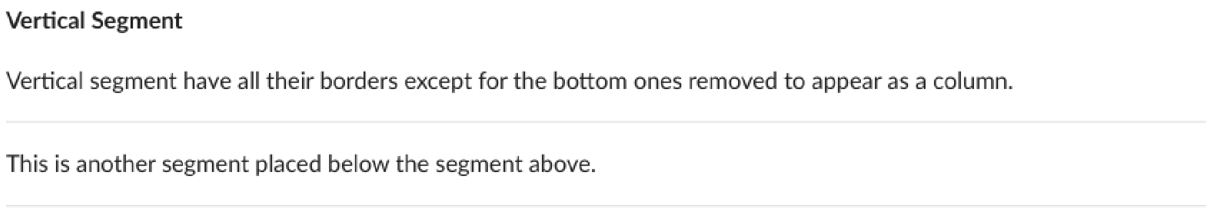

Stacked Segment

Stacked segments appear as a stack of a papers which are stacked cleanly. The illusion of depth is done using the bottom side which has double lines with slightly decreasing sizes.


Piled Segment

Piled segments appear as a pile of pages which are grouped together haphazardly.

vertical = 'vertical'
Segment type that formats the content to be aligned as part of a vertical group


Vertical Segment

Vertical segment have all their borders except for the bottom ones removed to appear as a column.

This is another segment placed below the segment above.

circular = 'circular'
Circular segment type. For a circle, ensure content has equal width and height

class pdfpug.common.**SegmentSpacing**
Enum Segment Spacing

compact = 'compact'
Segment will take up only as much space as is necessary

padded = 'padded'
Segment will add good amount of padding on all sides making it look more spacious

class pdfpug.common.**SegmentEmphasis**
Enum Segment Emphasis

secondary = 'secondary'
Lesser emphasis than the normal standard

tertiary = 'tertiary'
Lesser emphasis than secondary elements

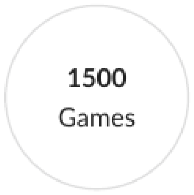
1.3.8 LineBreak

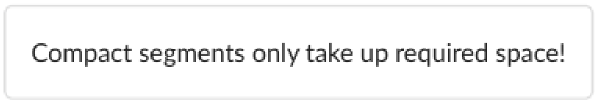
class pdfpug.modules.**LineBreak**(*lines_count=1*)
Add a line break (blank line)

Parameters **lines_count** (int) – No of blank lines to add

```
>>> from pdfpug.modules import LineBreak
>>> from pdfpug import PdfReport
```

(continues on next page)




Compact segments only take up required space!

(continued from previous page)

```
>>> report = PdfReport()
>>> report.add_element(LineBreak())
```

1.3.9 Message Box

class pdfpug.modules.**MessageBox**(*body*, *header=None*, ***kwargs*)

A MessageBox can be used to display information in a distinct style that captures the attention of the reader.

Parameters

- **header** (*Optional[str]*) – title
- **List] body** (*Union[str,*) – message
- **color** (*Optional[Color]*) – color
- **size** (*Optional[Size]*) – size
- **state** (*Optional[MessageState]*) – state

```
>>> from pdfpug.modules import MessageBox
>>> from pdfpug.common import MessageState
>>> message = MessageBox(
...     header="Important Announcement",
...     body="MessageBox is really good at capturing the attention of the reader!
↪",
...     state=MessageState.info
... )
```

class pdfpug.common.**MessageState**

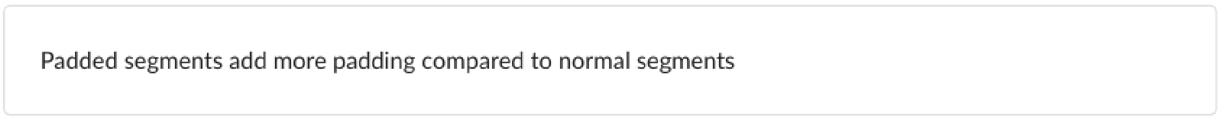
Enum Message box style options

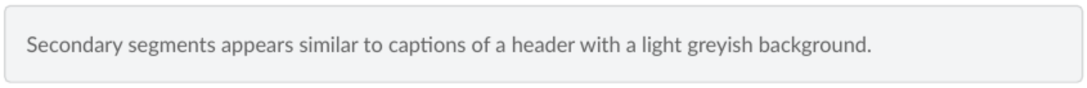
positive = 'positive'

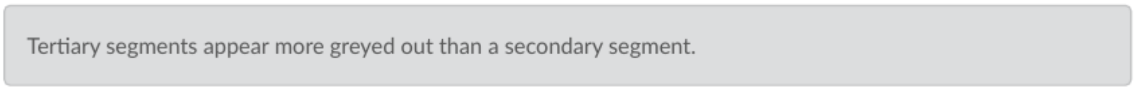
Positive message

negative = 'negative'

Negative Message


Padded segments add more padding compared to normal segments


Secondary segments appears similar to captions of a header with a light greyish background.


Tertiary segments appear more greyed out than a secondary segment.

error = 'error'
Error message

success = 'success'
Success message

warning = 'warning'
Warning message

info = 'info'
Info message

1.3.10 PageBreak

class pdfpug.modules.**PageBreak**
Add a page break

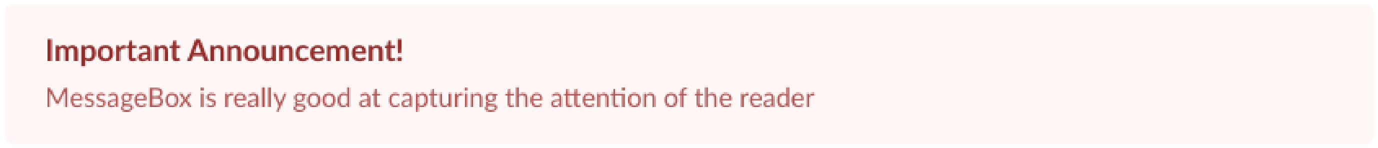
```
>>> from pdfpug.modules import PageBreak
>>> from pdfpug import PdfReport
>>> report = PdfReport()
>>> report.add_element(PageBreak())
```

1.3.11 Progress Bar

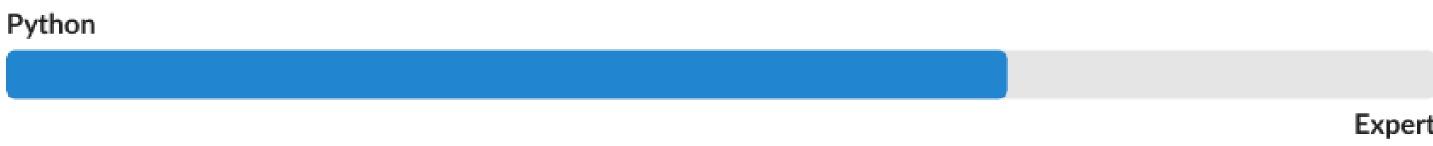
class pdfpug.modules.**ProgressBar**(*percent*, ***kwargs*)
Progress bar is a slightly unconventional element, but is surprisingly useful in some scenarios. For instance, consider a resume where one would like showcase the amount of experience in a language or technology. This can be expressed visually using a progress bar as seen nowadays in many modern resume styles.

Parameters

- **percent** (Union[int, float]) – Amount of progress in percentage
- **title** (*Optional[str]*) – Describes the progress bar
- **subtitle** (*Optional[str]*) – Describes the maximum range value
- **color** (*Optional[Color]*) – Color of progress bar
- **size** (*Optional[Size]*) – Size of progress bar


Important Announcement!
MessageBox is really good at capturing the attention of the reader

```
>>> from pdfpug.modules import ProgressBar
>>> from pdfpug.common import Color
>>> python_skill = ProgressBar(
...     75, title="Python", subtitle="Expert", color=Color.blue
... )
```



1.3.12 Table of Contents

class pdfpug.modules.**TableOfContents**(***kwargs*)

The TableOfContents element automatically searches the entire document for tier *h1* and *h2* headers and compiles the overall structure of the document. One needs to only add the table of contents element to the `PdfReport` class using the `add_element()` function.

```
>>> from pdfpug.modules import TableOfContents, Header
>>> from pdfpug import PdfReport
>>> toc = TableOfContents()
>>> report = PdfReport()
>>> report.add_element(toc)
>>> report.add_element(Header('PdfPug'))
>>> report.generate_pdf('pdfpug.pdf')
```

1.3.13 Enums

class pdfpug.common.**Alignment**

Enum Alignment options

right = 'right aligned'
Right align content

left = 'left aligned'
Left align content

justified = 'justified'
Justify content across the line

center = 'center aligned'
Center align content

class pdfpug.common.**Color**

Enum Colors

red = 'red'
Red

orange = 'orange'
Orange

yellow = 'yellow'
Yellow

olive = 'olive'
Olive

green = 'green'
Green

teal = 'teal'
Teal

blue = 'blue'
Blue

purple = 'purple'
Purple

violet = 'violet'
Violet

pink = 'pink'
Pink

brown = 'brown'
Brown

grey = 'grey'
Grey

class `pdfpug.common.`**Size**
Enum Size options

mini = 'mini'
Mini

tiny = 'tiny'
Tiny

small = 'small'
Small

medium = 'medium'
Medium

large = 'large'
Large

big = 'big'
Big

huge = 'huge'
Huge

massive = 'massive'
Massive

class `pdfpug.common.`**Orientation**
Enum Orientation options

horizontal = 'horizontal'
Layout elements horizontally

vertical = 'vertical'
Layout elements vertically

class pdfpug.common.**State**

Enum content state options

positive = 'positive'

Positive content

negative = 'negative'

Negative content

error = 'error'

Error content

warning = 'warning'

Warning content

active = 'active'

Active content

disabled = 'disabled'

Disabled content

1.3.14 Text Formatters

The text formatter functions help set various styles to text like making it bold, underline, strikethrough, superscript etc. They can be used within any of PdfPug's modules to further customize them to suit different use cases.

For example,

```
from pdfpug.modules import Paragraph, Header

# Italicized header
header = Header(italic("PdfPug"))

# Paragraph with a URL
para = Paragraph(f"{url('https://pypi.org/project/pdfpug/', 'PdfPug PyPi Page')}")
```

pdfpug.common.**url**(*hyperlink*, *text=None*)

Create a hyperlink

Note: If *text* is not provided, the hyperlink will be used as the visible text

Parameters

- **hyperlink** (str) – URL to resource
- **text** (Optional[str]) – Text displayed instead of hyperlink

Return type str

pdfpug.common.**bold**(*text*)

Formats the text to appear bold

Parameters **text** (str) – Text to be bold formatted

Return type str

pdfpug.common.**italic**(*text*)

Formats the text to appear italicized

Parameters **`text`** (`str`) – Text to be italic formatted

Return type `str`

`pdfpug.common.`**`underline`**(*text*)

Formats the text to appear underlined

Parameters **`text`** (`str`) – Text to be underline formatted

Return type `str`

`pdfpug.common.`**`strike`**(*text*)

Formats the text to appear striked through

Parameters **`text`** (`str`) – Text to be strike through formatted

Return type `str`

`pdfpug.common.`**`superscript`**(*text*)

Formats the text to appear as a superscript

Parameters **`text`** (`str`) – Text to be superscript formatted

Return type `str`

`pdfpug.common.`**`subscript`**(*text*)

Formats the text to appear as a subscript

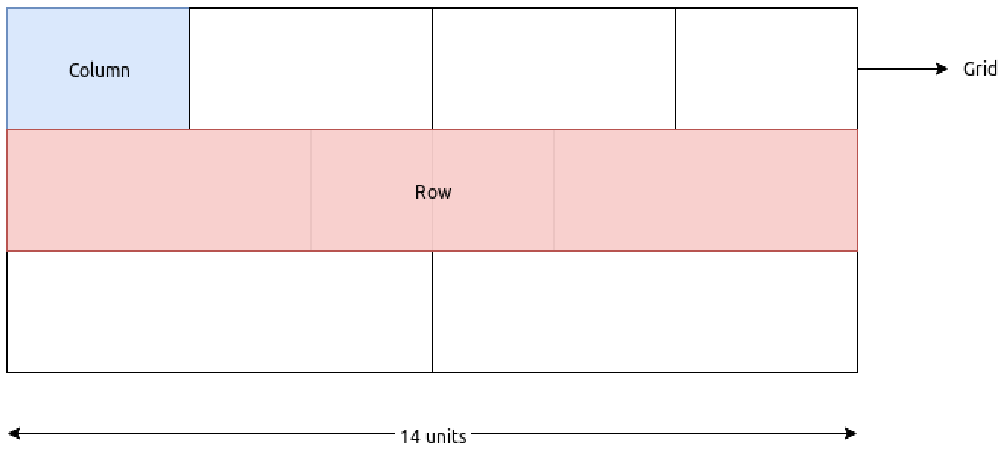
Parameters **`text`** (`str`) – Text to be subscript formatted

Return type `str`

1.3.15 Layouts

`class` `pdfpug.layouts.`**`Grid`**

A grid is a tabular structure that is divided vertically into `Row` and horizontally into `Column`. This allows for creating complex layouts that would otherwise not be possible. The grid system is illustrated below for more clarity.



The grid system supports a maximum horizontal size of **14** units. For instance, 2 columns of width 7 units can be placed in a single row. Or a single column of width 14 units. If the width of the columns in a row exceed 14 units, the extra columns will automatically flow to the next row.

Note: Only layouts like `Row` or `Column` can be added to the grid layout.

```
>>> from pdfpug.layouts import Grid, Column
>>> from pdfpug.modules import Paragraph, OrderedList
>>> # Create left column and its contents
>>> para = Paragraph('Python 3.x has several releases as listed,')
>>> left_column = Column(width=5)
>>> left_column.add_element(para)
>>> # Create right column and its contents
>>> releases = OrderedList(['3.0', '3.1', '3.2', '3.3', '3.4', '3.5', '3.6', '3.7
↪'])
>>> right_column = Column(width=5)
>>> right_column.add_element(releases)
>>> # Construct grid
>>> grid = Grid()
>>> grid.add_layout(left_column)
>>> grid.add_layout(right_column)
```

add_layout (*layout*)
Add a Row/Column to the grid

Parameters **Row] layout** (*Union[Column,*) – layout to be added to the grid

Return type None

class pdfpug.layouts.**Column** (***kwargs*)
The grid system divides horizontal space into indivisible units called Columns. The *Column* layout is the one that contain the actual content like *Paragraph* etc. Think of it as a container that holds content in a vertical layout.

Parameters **width** (*int*) – Width of the column (should be in the range of 1-14)

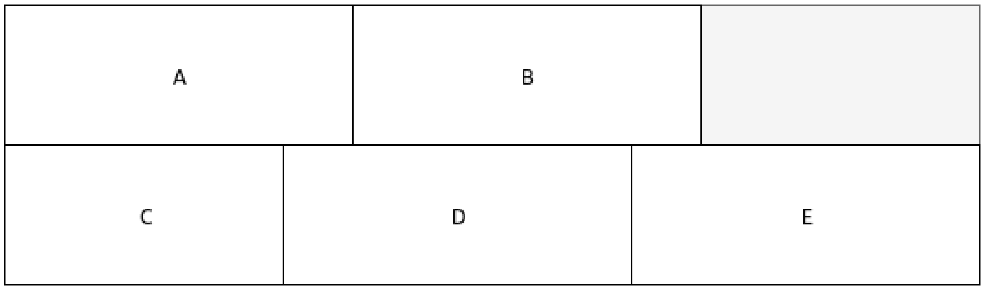
add_element (*element*)
Add element to the column

Parameters **element** (*BasePugElement*) – Element to be added to the column

Return type None

class pdfpug.layouts.**Row** (***kwargs*)
Rows are groups of columns which are aligned horizontally. When a group of columns exceed the grid width (14 units), the content automatically flows to the next row which is to say that rows are created automatically as required.

However, if explicit control is required for achieving a particular layout it can be declared with columns added to it. For instance, in the illustration below, the first row has 2 columns A, B which occupy a total of 10 units. If the row was not explicitly declared, then column C would be placed in the first row due to available space.



add_column (*column*)
Add column to the row

Parameters **column** (Column) – Column to be added to the row

Return type `None`

1.3.16 Theme

PdfPug comes with a set of curated themes that work beautiful with all PdfPug elements like `Header` and others defined in the *API Documentation* section.

Using these themes is as simple as passing it as an argument to the `PdfReport` class as shown below,

```
>>> from pdfpug import PdfReport
>>> from pdfpug.common import Theme
>>> report = PdfReport(theme=Theme.mood_swing)
>>> report.generate_pdf("pdfpug.pdf")
```

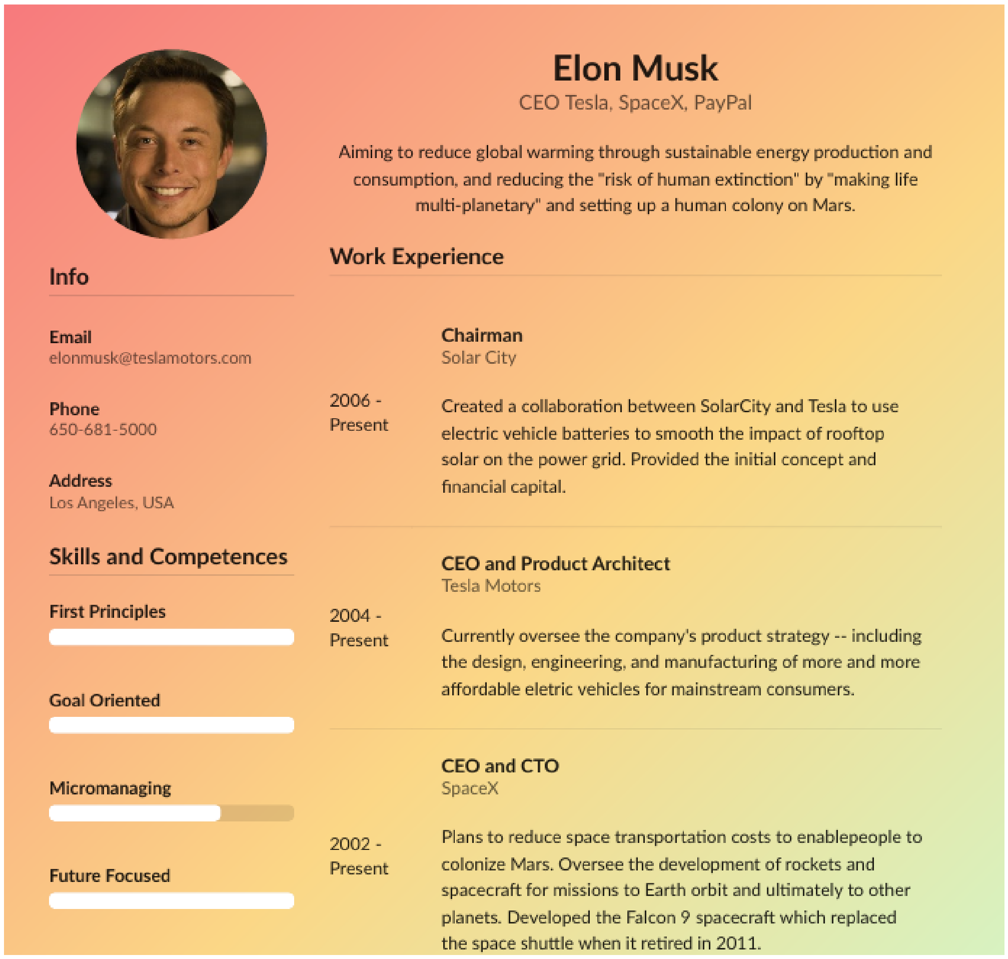
Warning: Themes are an **experimental** feature that is in a state of flux. Expect frequent API breakage!

class pdfpug.common.**Theme**

Predefined theme collection

mood_swing = 'moodswing'

Mood Swing Theme



Elon Musk

CEO Tesla, SpaceX, PayPal

Aiming to reduce global warming through sustainable energy production and consumption, and reducing the "risk of human extinction" by "making life multi-planetary" and setting up a human colony on Mars.

Info

Email
elonmusk@teslamotors.com

Phone
650-681-5000

Address
Los Angeles, USA

Skills and Competences

First Principles

Goal Oriented

Micromanaging

Future Focused

Work Experience

2006 - Present

Chairman
Solar City

Created a collaboration between SolarCity and Tesla to use electric vehicle batteries to smooth the impact of rooftop solar on the power grid. Provided the initial concept and financial capital.

2004 - Present

CEO and Product Architect
Tesla Motors

Currently oversee the company's product strategy -- including the design, engineering, and manufacturing of more and more affordable eletric vehicles for mainstream consumers.

2002 - Present

CEO and CTO
SpaceX

Plans to reduce space transportation costs to enablepeople to colonize Mars. Oversee the development of rockets and spacecraft for missions to Earth orbit and ultimately to other planets. Developed the Falcon 9 spacecraft which replaced the space shuttle when it retired in 2011.

1.4 Changelog

1.4.1 0.4

- Support setting meta information of document like title, description, authors and keywords
- Support customizing column widths of `Table`
- Support headless `Table`
- Add `ProgressBar` element
- Add `TableOfContents` element
- Add text formatter helpers for `bold()`, `italic()`. `underline()`, `strike()`, `superscript()`, `subscript()` and `url()`
- Add support for customizing the appearance of reports using predefined themes. *Mood Swing* is the first theme to be included in `Theme`
- Documented `TableType` with images for better clarity
- Add tutorials to demonstrate usage of APIs and showcase sample PDF files created using PdfPug

1.4.2 0.3 (03-08-2019)

- Add basic layout system `Grid`, `Row` and `Column`
- Add support for `Header` captions (subheader)
- Fixed `Table` not being inserted correctly into other elements like `Segment`
- Improved documentation of `Table`
- Moved all elements to *pdfpug.modules* namespace to improve clarity and avoid naming conflicts
- Style and position enums can now only be accessed through *pdfpug.common* to improve clarity

1.4.3 0.2 (26-07-2019)

- Add new elements `Image`, `PageBreak`, `Segment` and `LineBreak`
- Add `h4` and `h5` header tiers
- Setup pytest, Gitlab CI/CD and pre-commit
- Removed unnecessary dependencies like pandas, markdown etc

1.4.4 0.1 (20-07-2019)

- Add new elements `Header`, `Table`, `OrderedList`, `UnorderedList`, and `Paragraph` elements
- Setup documentation system using Sphinx
- Project init

1.5 Contributing Guide

Looking to contribute? Thank you very much! **All contributions are welcome** even if they are as small as fixing a typo in the documentation. Just the fact that you are reading this document to learn how to contribute is a big deal to the library author!

A bit of of a confession. This document is a stub with limited exposure to the outside world. If you hit a road block anywhere or find the instructions to be outdated or incorrect, do report it as a bug here or even better create a pull request with the fix.

Let's get started.

The first step towards contributing either code or documentation changes involves grabbing the library's source code. You can do that by forking the Gitlab repo.

Note: When contributing new features, it is best to discuss the solution before diving into the implementation. The simplicity and ease of use of the public APIs are important goals in achieving the *vision* of PdfPug. This is something that can be verified only by discussing it with others.

1.5.1 Code Contributions

PdfPug at its core functions by providing an easy to use API for users to create .pug files which is parsed by PyPugJS and then converted into a PDF using WeasyPrint. All the modules provided by PdfPug essentially create mini pug snippets that are then compiled together to create the final PDF.

1. Start by installing the development dependencies using *pip install -r requirements.txt*.
2. Run the tests to confirm if they all pass on your system.
3. Write tests that demonstrate your bug or feature. Ensure that they fail (Test Driven Development - TDD).
4. Make your change.
5. Run the entire test suite again, confirming that all tests pass.
6. Send a pull request to the master branch of the Gitlab repo.

1.5.2 Documentation Contributions

The documentation lives in the *docs* directory. They are written in reStructuredText, and use Sphinx to generate the full suite of documentation.

1. Start by installing the development dependencies using *pip install -r requirements.txt*.
2. Navigate to the *docs* directory
3. Make changes to the documentation.
4. Generate the documentation by running *make html*
5. Open the *index.html* that is generated in the *docs/build* folder. This should show you the updated documentation with your changes.

Index