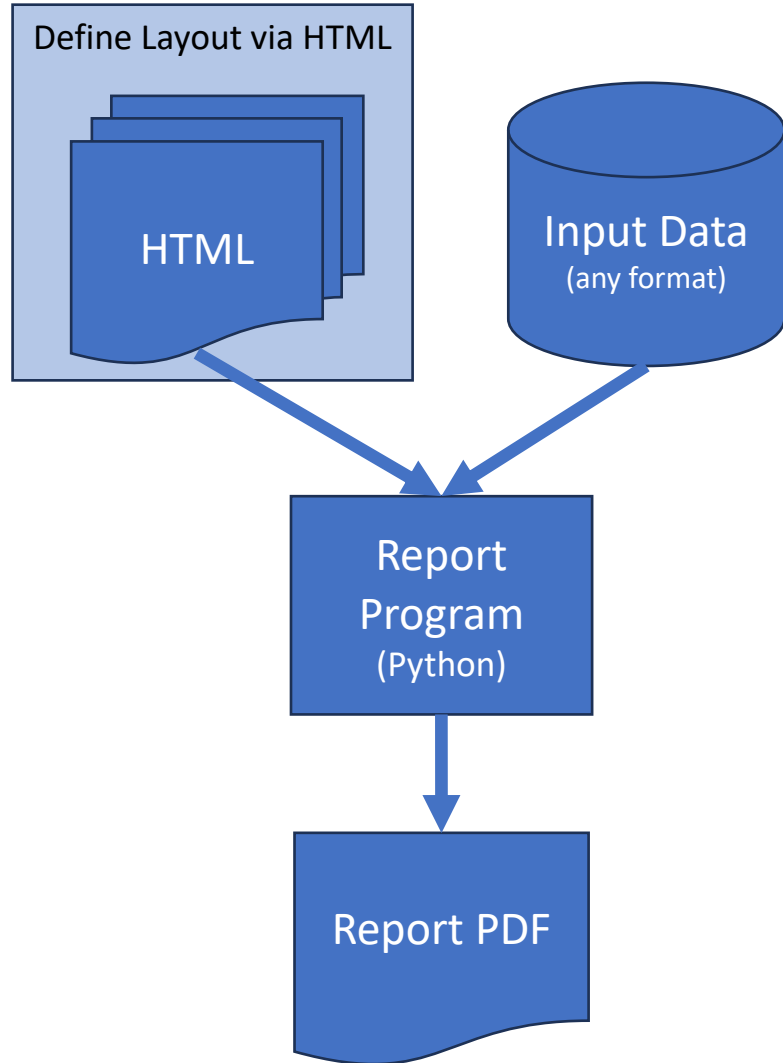


Overview: Effective Report Generation Using PyMuPDF



Simple, intuitive approach:

- Define the layout via HTML and CSS sources
- Develop the report program in simple steps:
 - Implement access to databases
 - Convert HTML to ***building blocks*** (logo, header, text, table, footer, etc.)
 - Compose the report by simply naming the ***building blocks***
- Execute PDF generation

PyMuPDF Reporting: Features

PyMuPDF Reporting supports multiple advanced features

- The report layout is defined using the powerful HTML and CSS languages.
- Multiple HTML sources can be used to deal with report sections separately like header, footer, images or tables.
- Based on each piece of text, automatic selection of the right font from Google's NOTO fonts (includes Latin, CJK, Hindi, Tamil, and more).
- User fonts can be included via appropriate CSS definitions. Elegant support for pymupdf-fonts.
- Support for multi-column pages and multiple page formats in the same report.
- Report data may reside on any Python-supported storage like databases or JSON- and CSV-files, or containers like dictionaries, lists, pandas DataFrames and more.
- Easy variable substitution using the Story interface.
- Table building blocks support optional top-row repetition, ***alternating row*** and ***final row*** background colors and images in table cells.
- Changing page size, page orientation, columns per page, etc. **require no coding** effort: any layout adjustments are automatically carried out by the underlying Story.

PyMuPDF Report Creation Overview

Reports are composed from **building blocks**

- Identify layout segments
 - Header, footer, logo
 - Text (prolog, intermediate, trailer, etc.)
 - Tabular data
- Identify input data sources (SQL, CSV, JSON, DataFrames (pandas), text, etc.)
- Code
 - HTML / CSS sources for layout segments
 - Accessing external data
- Define building blocks
 - Header, footer, text: **Block**
 - Tabular data: **Table**
- Compose the report object
 - Assign **report.header**, **report.footer** to their building blocks
 - Assign **report.sections** to the list of building blocks
- Generate the report
report.run(filename)

PyMuPDF Reports Have a Common Structure

```
# import required objects
from fitz.reports import Report, Table, Block
# define the report object
report = Report(mediabox) # choose report page size
```

```
header = Block(html=header_html, report=report)
footer = Block(html=footer_html, report=report)
prolog = Block(html=prolog_html, report=report)
epilog = Block(html=epilog_html, report=report)
```

```
def fetch_rows():
    """Access databases and return row items."""
    ...
    return rows
```

```
items = Table(
    report=report, # point to owning report
    html=items_html, # HTML definition of table items
    top_row="toprow", # name of top row in table
    fetch_rows=fetch_rows, # call this to get item data
    # alternating background colors of rows
    alternating_bg=("#ccc", "#aaa", "#fff"),
)
```

```
report.header = header
report.footer = footer
report.sections = [prolog, [items, 2], epilog]
report.run("output.pdf")
```

To construct a **simple** block, the HTML source is sufficient

Access any data format supported by Python to return a list of item rows

A table supports several parameters

This report will output the prolog, then the items table (two columns per page), and finally the epilog.

