FPDF Table

Matias Gabriel Martinez Rebori

Jul 24, 2022

CONTENTS:

fpdf-table is a *fast, framework-agnostic library* for generating PDF reports in a similar way as HTML tables are created, everything you draw is inside a table (container with a border), it also allows you to create reports in a more *elegant* and *DRY* way.

CHAPTER

ONE

WHAT IS FPDF-TABLE

fpdf-table is built on top of fpdf2, is somewhat inspired by HTML tables and jspdf-autotable, provides abstraction to manipulate everything in the form of tables, provides unique features and several utilities.

Note: This library does not parse HTML tables to PDF, you can do it through fpdf2 but only in a very limited way.

CHAPTER

TWO

MAIN FEATURES

- Make tables fast
- Make tables with fixed height rows
- Everything that fpdf2 does.

2.1 Usage

2.1.1 Installation

To use fpdf-table, first install it from PyPi using pip:

```
(.venv) $ pip install fpdf-table
```

2.1.2 Minimal Example

Code

```
from fpdf_table import PDFTable


def minimal_example():
    data: list[list[str]] = [
        ['Gerard', 'Martinez', '09/07/1998'],
        ['Amy ', 'Miller', 'July 30, 1969'],
        ['Ferdinand ', 'Varela ', 'November 10, 1988'],
        ['Edén ', 'Mascarenas Benavides', 'May 23, 1990'],
        ['Adrián ', 'Beltrán ', 'December 12, 1977'],
    ]
    # initialize PDFTable, before doing anything, __init__ adds a page, sets font, size␣
↪and colors
    pdf = PDFTable()
    # table header
    pdf.table_header(['First Name', 'Last Name', 'Date of birth'])
    # table rows
    for person in data:
        pdf.table_row(person)
    # file path where to save the pdf
```

(continues on next page)

(continued from previous page)

```
    pdf.output("../pdfs/minimal_example.pdf")


minimal_example()
```

PDF

First Name	Last Name	Date of birth
Gerard	Martinez	09/07/1998
Amy	Miller	July 30, 1969
Ferdinand	Varela	November 10, 1988
Edén	Mascarenas Benavides	May 23, 1990
Adrián	Beltrán	December 12, 1977

minimal_example.pdf

2.1.3 Main Features

Code

```
from fpdf_table import PDFTable, Align


def features_example():
    # initialize PDFTable, before doing anything, __init__ adds a page, sets font, size␣
↪and colors
    pdf = PDFTable()
    """
    table row
    """
    # draw a table header, pass a list with the text, by default width is the same for␣
↪every column
    # and align is to left
    pdf.table_header(['First Name', 'Last Name', 'Date of birth'])
    # draw a table row, by default is only one row with height equal to pdf.default_cell_
↪height
    pdf.table_row(['Gerard', 'Martinez', '09/07/1998'])

    """
    responsive row
    """
    # header with custom width
    pdf.table_header(['Email', 'Address'], [pdf.calculate_width_3(), 2 * pdf.calculate_
↪width_3()])
    # responsive row with custom width
    pdf.table_row(['large_email_example-very_large_email_example-more_large_email_
↪example@example.com',
```

(continues on next page)

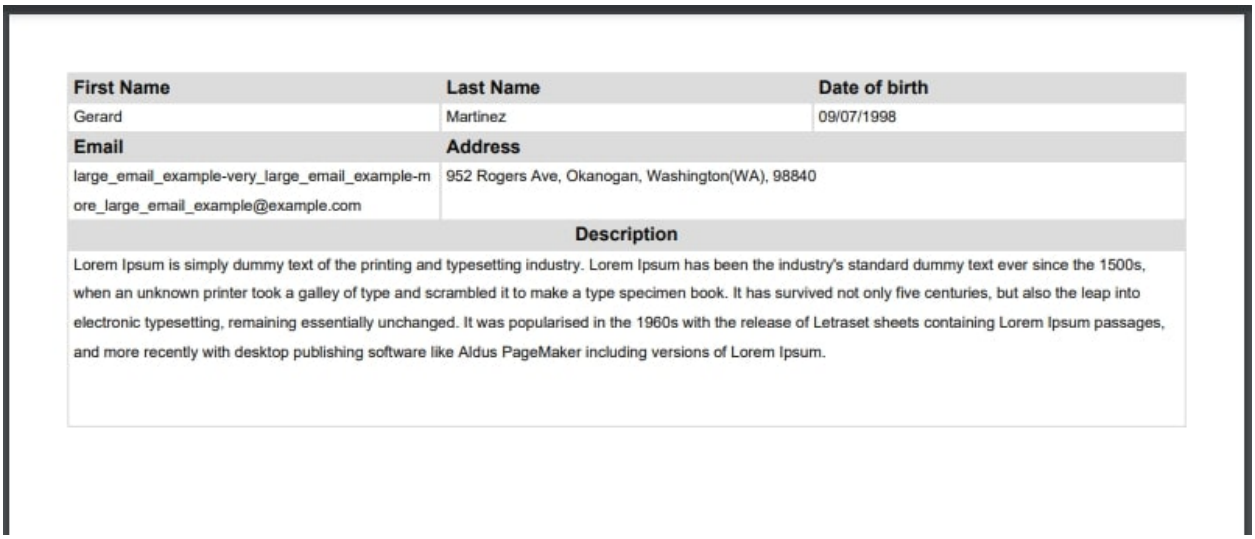
(continued from previous page)

```
                  '952 Rogers Ave, Okanogan, Washington(WA), 98840'],
                 pdf.table_cols(4, 8), option='responsive')

    """
    fixed height row
    """
    # align center, expects a list of alignments but if you pass only one it spreads for␣
↪every column
    pdf.table_header(['Description'], align=Align.C)
    large_text = """Lorem Ipsum is simply dummy text of the printing and typesetting␣
↪industry....."""
    # fixed row needs fixed_height parameter
    pdf.table_row([large_text], option='fixed', fixed_height=6 * pdf.row_height_cell)
    # output
    pdf.output("../pdfs/main_features.pdf")


features_example()
```

PDF



First Name	Last Name	Date of birth
Gerard	Martinez	09/07/1998
Email	**Address**	
large_email_example-very_large_email_example-more_large_email_example@example.com	952 Rogers Ave, Okanogan, Washington(WA), 98840	
Description		
Lorem Ipsum is simply dummy 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.		

main_features.pdf

2.1.4 Image Features

Code

```
from fpdf_table import PDFTable, add_image_local


def image_example():
    # initialize PDFTable, before doing anything, __init__ adds a page, sets font, size␣
↪and colors
```

(continues on next page)

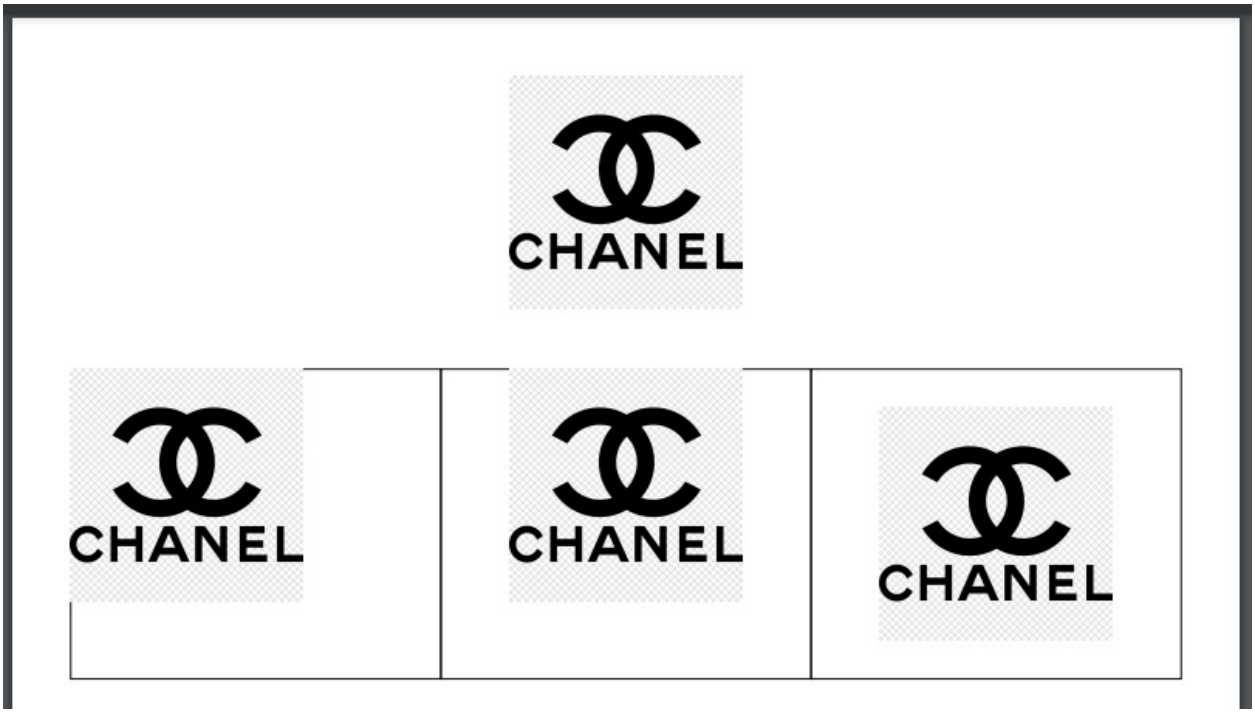
(continued from previous page)

```
    pdf = PDFTable()
    # load image from file
    img, img_width, img_height = add_image_local('../pdfs/logo1.png')
    # set custom width and height
    img_width, img_height = pdf.use_px_to_mm(150), pdf.use_px_to_mm(150)
    # draw image, center on page
    pdf.draw_image_center(img=img, img_width=img_width, img_height=img_height, container_
↪width=pdf.get_width_effective())
    # line breaks
    pdf.ln(img_height)
    pdf.ln(10)
    # get cursor position
    x, y = pdf.get_x(), pdf.get_y()
    # draw a fixed table without content
    table_height = pdf.use_px_to_mm(200)
    # change color of table border
    pdf.set_draw_color(10, 10, 10)
    pdf.table_row(['', '', ''], option='fixed', fixed_height=table_height)
    # draw image no align
    pdf.draw_image_center(img=img, x=x, y=y, img_width=img_width, img_height=img_height)
    # draw image center horizontally
    x = x + pdf.calculate_width_3()
    pdf.draw_image_center(img=img, x=x, y=y, img_width=pdf.use_px_to_mm(150), img_
↪height=pdf.use_px_to_mm(150),
                          container_width=pdf.calculate_width_3())
    # draw image center horizontally and vertically
    x = x + pdf.calculate_width_3()
    pdf.draw_image_center(img=img, x=x, y=y, img_width=pdf.use_px_to_mm(150), img_
↪height=pdf.use_px_to_mm(150),
                          container_width=pdf.calculate_width_3(), container_
↪height=table_height)

    # file path where to save the pdf
    pdf.output("../pdfs/image_example.pdf")


image_example()
```

PDF



image_features.pdf

2.1.5 Usage in web APIs

Please refer to fpdf2 usage in web APIs

2.2 API

class fpdf_table.**PDFTable**

add_fonts_custom(*font_name: str*, *font_extension: str*, *font_dir: str = '/home/docs/checkouts/readthedocs.org/user_builds/fpdf-table/checkouts/latest/docs/fonts'*, *set_default: bool = True*)

add custom fonts, you need the 4 most common styles of the font, and the name needs to be standard.

The normal font hast to be only the name, for the bold append: -Bold, for italic: -Oblique, bolditalic: -BoldOblique. i.e. Arial.ttf,Arial-Bold.ttf,Arial-Oblique.ttf, Arial-BoldOblique.ttf

Parameters

- **font_name** – name of the font without extension
- **font_extension** – extension of the font, ttf ot otf
- **font_dir** – directory to find the font, defaults to current_working_directory/fonts, find the cwd with os.getcwd().
- **set_default** – set custom font as default

Returns

`calculate_align_list`(*align: fpdf.enums.Align | list[fpdf.enums.Align]*, *columns_count: int*, *default_value: Align = Align.J*) → list[fpdf.enums.Align]

make list of alignments.

Parameters

- **align** – list of alignment or one alignment value
- **columns_count** – columns count
- **default_value** – if is an empty list use align passed here

Returns

`calculate_center_code39_x`(*text: str*) → float

calcula la posicion donde se debe dibujar el codigo de barras para que este centrado.

Parameters
text – texto del codigo de barras.

Returns
posicion de x

`static calculate_center_generic`(*start: float*, *container_length: float*, *element_length: float*) → float

calcular posicion para centrar un objeto.

Parameters

- **start** – posicion de inicio inicial.
- **container_length** – longitud del container del objeto.
- **element_length** – longitud del elemento.

Returns
posicion de inicio para que el elemento quede centrado

`calculate_center_x`(*start: Optional[float] = None*, *container_length: Optional[float] = None*, *element_length: Optional[float] = None*) → float

calcular posicion para centrar un objeto en horizontal.

Parameters

- **start** – posicion de inicio inicial.
- **container_length** – longitud del container del objeto.
- **element_length** – longitud del elemento.

Returns
posicion de inicio para que el elemento quede centrado

`calculate_text_fragments`(*w=0*, *txt=''*, *row_quantity=1*, *justify=True*, *markdown=False*) → tuple[list[fpdf.line_break.TextLine], bool]

dado un texto y su longitud, dividir el texto en arrays cada que debe haber un salto de linea. devuelve también si el texto se dividió

Parameters

- **w** – longitud del container.
- **txt** – texto.
- **row_quantity** – cantidad de filas.

- **justify** – justificar texto.
- **markdown** –

Returns

calculate_text_rows(*w: float = 0*, *txt=''*, *justify=True*, *markdown=False*)

calculate how many rows will take the given text in the given width.

Parameters

- **w** – longitud del container.
- **txt** – texto
- **justify** – justify
- **markdown** – markdown

Returns

calculate_width_2() → float

get width for 2 columns of same width.

Returns
width of one column

calculate_width_3() → float

get width for 2 columns of same width.

Returns
width of one column

static calculate_width_code39(*quantity: int*) → float

calcula la longitud del codigo de barras en mm.

Parameters
quantity – cantidad de caracteres.

Returns
mm

calculate_width_list(*width_list: list[float]*, *columns_count: int*) → list[float]

if width_list is not empty check the total width ,if width_list is empty make list of equals width´s.

Parameters

- **width_list** – list of width for every column
- **columns_count** – columns count

Returns

Raises
NumberColumnsTextDoesntMatchError – columns count doesn't match text count

calculate_width_n(*n: int*) → float

get width for n columns of same width.

Returns
width of one column

`cell`(*w=0*, *h: Optional[float] = None*, *txt=''*, *border=1*, *ln='DEPRECATED'*, *align=Align.L*, *fill=False*, *link=''*, *center='DEPRECATED'*, *markdown=False*, *new_x=XPos.RIGHT*, *new_y=YPos.TOP*, *line_break=False*)

Prints a cell (rectangular area) with optional borders, background color and character string. The upper-left corner of the cell corresponds to the current position. The text can be aligned or centered. After the call, the current position moves to the selected *new_x*/*new_y* position. It is possible to put a link on the text.

If automatic page breaking is enabled and the cell goes beyond the limit, a page break is performed before outputting.

Args:

w (float): Cell width. Default value: None, meaning to fit text width.
If 0, the cell extends up to the right margin.

h (float): Cell height. Default value: None, meaning an height equal
to the current font size.

txt (str): String to print. Default value: empty string. border: Indicates if borders must be drawn around the cell.

The value can be either a number (*0*: no border ; *1*: frame) or a string containing some or all of the following characters (in any order): *L*: left ; *T*: top ; *R*: right ; *B*: bottom. Default value: 0.

new_x (fpdf.enums.XPos, str): New current position in x after the call. Default: RIGHT new_y (fpdf.enums.YPos, str): New current position in y after the call. Default: TOP ln (int): **DEPRECATED since 2.5.1**: Use *new_x* and *new_y* instead. align (fpdf.enums.Align, str): Allows to center or align the text inside the cell.

Possible values are: *L* or empty string: left align (default value) ; *C*: center; *X*: center around current x; *R*: right align

fill (bool): Indicates if the cell background must be painted (*True*)
or transparent (*False*). Default value: False.

link (str): optional link to add on the cell, internal
(identifier returned by *add_link*) or external URL.

center (bool): DEPRECATED since 2.5.1:
Use align="C" or align="X" instead.

markdown (bool): enable minimal markdown-like markup to render part
of text as bold / italics / underlined. Default to False.

Returns: a boolean indicating if page break was triggered

`cell_fixed`(*container_width: float*, *container_height: float*, *txt: str = ''*, *align=Align.L*, *line_break: bool = False*, *inline: bool = False*)

draw a fixed size table border.

Parameters

- **`container_width`** – container_width
- **`container_height`** – container_height
- **`txt`** – text
- **`align`** – text align
- **`line_break`** – perform a new line

- **inline** – next Y with be in the same line

Returns

check_width_available(*total_width: float*)

check if some calculated width fits in the available page width.

Parameters
total_width – calculated width.

Returns

Raises
WidthOverflowError – calculated width doesn't fit in available page space

draw_image_center(*img: any*, *x: Optional[float] = None*, *y: Optional[float] = None*, *img_width: float = 0*, *img_height: float = 0*, *container_width: Optional[float] = None*, *container_height: Optional[float] = None*)

draw an image and center its position in a given container.

Parameters

- **img** – either a string representing a file path to an image, a URL to an image, an io.BytesIO, or an instance of *PIL.Image.Image*.
- **x** – optional horizontal position where to put the image on the page. If not specified or equal to None, the current abscissa is used.
- **y** – optional vertical position where to put the image on the page. If not specified or equal to None, the current ordinate is used. After the call, the current ordinate is moved to the bottom of the image
- **img_width** – optional width of the image. If not specified or equal to zero, it is automatically calculated from the image size. Pass *pdf.epw* to scale horizontally to the full page width.
- **img_height** – optional height of the image. If not specified or equal to zero, it is automatically calculated from the image size. Pass *pdf.eph* to scale horizontally to the full page height.
- **container_width** – with of the rectangle that contains the image. If not specified or equal to None, the current image width is used.
- **container_height** – height of the rectangle that contains the image If not specified or equal to None, the current image height is used.

Returns

draw_row_fixed(*text_list: list[str]*, *width_list: list[float]*, *align: fpdf.enums.Align | list[fpdf.enums.Align]*, *fixed_height: Optional[float] = None*, *line_break: bool = False*)

draw n columns in the same row, columns height is fixed.

Parameters

- **text_list** – list of the texts to write
- **width_list** – list of width for every column
- **fixed_height** – height of every column
- **line_break** – perform a line break
- **align** – alignment

Returns

draw_row_line(*text_list: list[str]*, *width_list: list[float]*, *align: fpdf.enums.Align | list[fpdf.enums.Align]*, *line_break: bool = False*)

draw n columns in the same row, columns height are 1 column.

Parameters

- **text_list** – list of the texts to write
- **width_list** – list of width for every column
- **line_break** – perform a line break
- **align** – alignment

Returns

draw_row_responsive(*text_list: list[str]*, *width_list: list[float]*, *align: fpdf.enums.Align | list[fpdf.enums.Align]*, *line_break: bool = False*)

draw n columns in the same row, every column has height equals to the column with maximum height.

Parameters

- **text_list** – list of the texts to write
- **width_list** – list of width for every column
- **line_break** – perform a line break
- **align** – alignment

Returns

fit_text_fixed_height(*txt: str*, *row_height: float*, *container_width: float*, *container_height: float*, *linesep: str = '\n'*, *ellipsis: bool = False*) → tuple[str, str]

divide the text in two string, the first string contains the piece of text that fits in the container, the second string contains the remaining text that doesn't it.

Parameters

- **container_width** – width of the container
- **txt** – text
- **row_height** – height of every row
- **container_height** – total height of the container
- **linesep** – os new line representation
- **ellipsis** – truncate text and add ellipsis

Returns
list with two strings

get_width_effective()

effective page width: the page width minus its horizontal margins. :return:

multi_cell(*w=0*, *h: Optional[float] = None*, *txt=''*, *border=1*, *align=Align.J*, *fill=False*, *split_only=False*, *link=''*, *ln='DEPRECATED'*, *max_line_height=None*, *markdown=False*, *print_sh=False*, *new_x=XPos.RIGHT*, *new_y=YPos.TOP*, *line_break=False*)

This method allows printing text with line breaks. They can be automatic (breaking at the most recent space or soft-hyphen character) as soon as the text reaches the right border of the cell, or explicit (via the *n* character). As many cells as necessary are stacked, one below the other. Text can be aligned, centered or justified. The cell block can be framed and the background painted.

Args:

w (float): cell width. If 0, they extend up to the right margin of the page. h (float): cell height. Default value: None, meaning to use the current font size. txt (str): string to print. border: Indicates if borders must be drawn around the cell.

The value can be either a number (*0*: no border ; *1*: frame) or a string containing some or all of the following characters (in any order): *L*: left ; *T*: top ; *R*: right ; *B*: bottom. Default value: 0.

align (fpdf.enums.Align, str): Allows to center or align the text.
Possible values are: *J*: justify (default value); *L* or empty string: left align; *C*: center; *X*: center around current x; *R*: right align

fill (bool): Indicates if the cell background must be painted (*True*)
or transparent (*False*). Default value: False.

split_only (bool): if *True*, does not output anything, only perform
word-wrapping and return the resulting multi-lines array of strings.

link (str): optional link to add on the cell, internal
(identifier returned by *add_link*) or external URL.

new_x (fpdf.enums.XPos, str): New current position in x after the call. Default: RIGHT new_y (fpdf.enums.XPos, str): New current position in y after the call. Default: NEXT ln (int): **DEPRECATED since 2.5.1**: Use *new_x* and *new_y* instead. max_line_height (float): optional maximum height of each sub-cell generated markdown (bool): enable minimal markdown-like markup to render part

of text as bold / italics / underlined. Default to False.

print_sh (bool): Treat a soft-hyphen (u00ad) as a normal printable
character, instead of a line breaking opportunity. Default value: False

Using *new_x=XPos.RIGHT, new_y=XPos.TOP, maximum height=pdf.font_size* is useful to build tables with multiline text in cells.

Returns: a boolean indicating if page break was triggered,
or if *split_only* == *True*: *txt* splitted into lines in an array

`multi_cell_fixed`(*w: float*, *txt: str*, *row_height: float*, *container_height: float*, *align: str | fpdf.enums.Align = Align.J*, *line_break: bool = False*, *ellipsis: bool = False*, *inline: bool = False*)

draw a fixed size cell, if the text is larger than the cell (container), it will draw the text until it fits and will return the text that doesn't fit for later use.

Parameters

- `w` – container width
- `txt` – text
- `row_height` – height of every row
- `container_height` – total height of the container
- `align` – alignment
- `line_break` – add a trailing new line
- `ellipsis` – truncate text and add ellipsis
- `inline` – next Y with be in the same line

Returns

`set_defaults()`

return to default values.

Returns

`table_cols`(**args: float*) → list[float]

calculate widths like bootstrap grid system :param args: bootstrap column widths :return: list of calculated bootstrap columns widths

`table_header`(*text_list: list[str]*, *width_list: list[float] = []*, *align: list[fpdf.enums.Align] | fpdf.enums.Align = Align.L*, *fill: bool = True*, *border: int = 1*)

draw a table header for a table.

Parameters

- **text_list** – list of the texts to write
- **width_list** – list of width´s for every column
- **align** – alignment

Returns

`table_row`(*text_list: list[str]*, *width_list: list[float] = []*, *align: list[fpdf.enums.Align] | fpdf.enums.Align = Align.L*, *option: str = 'line'*, *fixed_height: Optional[float] = None*)

draw a row for a table.

Parameters

- **text_list** – list of the texts to write
- **width_list** – list of width´s for every column
- **option** – define what type of row to draw
- **fixed_height** – height if option is fixed
- **align** – alignment

Returns

Raises

- **MissingValueError** – a value was expected and wasn't found
- **HeightError** – height cannot be smaller than default cell height
- **MismatchValueError** – undefined option

`static use_mm_to_px`(*mm: float*) → int

convertir mm to px.

Parameters
mm – unidad en milimetros.

Returns
unidad en pixeles.

`static use_object_or_dash`(*obj*)

si un objeto no tiene contenido devuelve un dash -.

Parameters
obj – objeto

Returns
objeto o string

`static use_object_or_empty(obj)`

si un objeto no tiene contenido devuelve un string vacio.

Parameters
obj – objeto

Returns
objeto o string

`static use_object_or_text(obj, text: str)`

si un objeto no tiene contenido devuelve un texto.

Parameters

- **obj** – objeto
- **text** – texto

Returns
objeto o string

`static use_px_to_mm(px: int)` → float

convertir px to mm.

Parameters
px – unidad en pixeles.

Returns
unidad en milimetros.

`static use_px_to_pt(px: int)` → float

convertir px to pt.

Parameters
px – unidad en pixeles.

Returns
unidad en points.

2.3 Creating recipes

To retrieve a list of random ingredients, you can use the `lumache.get_random_ingredients()` function:

`lumache.get_random_ingredients(kind=None)`

Return a list of random ingredients as strings.

Parameters
kind (*list[str] or None*) – Optional "kind" of ingredients.

Raises
lumache.InvalidKindError – If the kind is invalid.

Returns
The ingredients list.

Return type
list[str]

The `kind` parameter should be either `"meat"`, `"fish"`, or `"veggies"`. Otherwise, *lumache.get_random_ingredients()*

#:py:func:*fpdf_table.main.PDFTable.barcode* will raise an exception. # .. py:exception:: lumache.InvalidKindError

Raised if the kind is invalid.

2.3.1 Welcome to Lumache's documentation!

Lumache (/lu'make/) is a Python library for cooks and food lovers that creates recipes mixing random ingredients. It pulls data from the Open Food Facts database and offers a *simple* and *intuitive* API.

Check out the *Usage* section for further information, including how to *install* the project.

Hint: This is a note admonition. This is the second line of the first paragraph.

Important: This is a note admonition. This is the second line of the first paragraph.

Tip: This is a note admonition. This is the second line of the first paragraph.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX