PDFUtils

J.C. Martín

October 1, 2023

Contents

List of Figures

List of Tables

1 Introduction

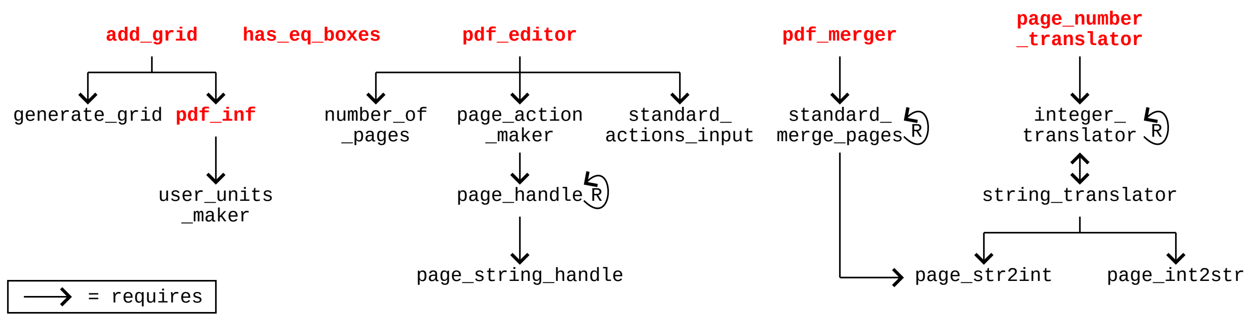
This project was conceived as a set of tools that allow us to create a PDF file from one or more PDF files. It is useful when scanning a long document. In that case, it is possible that the document was scanned in multiple PDF files, with some pages repeated (because you were not sure if those pages were correctly scanned), and also with pages in different orientations. This project has tools to edit PDF files and to obtain a PDF from pages of different PDF files.

This project consists of four main scripts:

- `pdf_editor.py`,
- `pdf_merger.py`,
- `pdf_number_translator.py`, and
- `pdf_inf.py`.

These scripts are interfaces of the functions of the `pdf_transformer.py` module that have the same name, i.e. `pdf_editor`, `pdf_merger`, and so on. Of course, these scripts require the `pdf_transformer.py` module.

Figure 1: Function interactions in `pdf_transformer.py`



2 About the script `pdf_editor.py`

2.1 How it works

To use this script, we have to give values to the following variables:

- `input_filepath`, the path of the input file. Its value must be a string
- `output_filepath`, the path of the output file. Its value must be a string
- `actions`, its value must be a list of tuples and/or lists, this argument defines the edition actions that we will do to each page of `input_filepath` to produce `output_filepath`
- `with_repetitions`, its value must be `True` or `False`, this variable modifies how the script interprets each element in `actions`

This script is intended to produce a PDF file `output_filepath` whose pages are edited versions of the original PDF file `input_filepath`. Each element of `actions` is a tuple or list whose first element determines the pages to be edited, and the other elements define the edition actions to apply to these pages. Let `action` be an element of `actions`. The syntax of `action` is

```
(pages, ed_action_1, ..., ed_action_n)
```

or

```
[pages, ed_action_1, ..., ed_action_n]
```

where $\texttt{n} \geq 1$. For $1 \leq \texttt{i} \leq \texttt{n}$, `ed_action_i` would be `P_D`, or `(k, par_i)`, where `k` is an edition action constant and `par_i` is its parameters. `P_D` is also an edition action constant. It is the only one that does not require parameters. The edition action constants (eac) are given in the following table.

Table 1: Edition action constants. Angular parameters are in degrees counterclockwise. Distance parameters are in Postscript big points (pt). By definition pt = inch/72

eac	**value**	**Description**	**Parameters**
`P_D`	0	Delete page	No parameters
`P_SB`	1	Page scaling by a factor	One parameter: scaling factor (float)
`P_ST`	2	Page scaling to a size	Two parameters: page width and height (floats)
`P_R`	3	Page rotation	One parameter: angle (should be `0`, `90`, `180`, or `270`)
`C_S`	10	Content scaling	Two parameters: scale factors along the x-axis and the y-axis (floats)
`C_T`	11	Content translation	Two parameters: translation along the x-axis and the y-axis (floats)
`C_R`	12	Content rotation	One parameter: angle (float)

The syntax of `ed_action_i` depends on the number of parameters. This syntax is

```
# for 0 parameters
k
# for 1 parameter
(k, parameter)
# for 2 parameters
(k, (parameter_0, parameter_1))
```

where `k` is the eac.

For example, `P_D` means delete the pages in `pages`, `(P_R, 90)` means rotate 90° counterclockwise the pages in `pages`, and `(C_S, (300, 200))` means scaling the content of the pages in `pages` to a width of 300pt and a height of 200pt.

Now we will talk about the syntax of `pages`. If a PDF file has n pages, its pages are numbered from 0 to $n-1$. `pages` does reference to the pages of the PDF file `input_filepath`. `pages` could be one of the following objects

- An non negative integer. It represents that page.
- A two-tuple of non negative integers `(a, b)`. This two-tuple represents the pages in the list `list(range(a, b))`.
- A three-tuple of integers `(a, b, c)`, with `a` ≥ 0 and `b` ≥ 0. It represents the pages in the list `list(range(a, b, c))`.
- A string `'a'`, with `a` a non negative integer. It represents that page.
- A string `'a:b'`, with `a` and `b` non negative integers. It represents the pages in the list `list(range(a, b + 1))`.
- A string of comma-separated elements in the above strings. It represents the union of the pages in the comma-separated elements, e.g. `'0:4, 12, 17:20'` represents the pages list `[0, 1, 2, 3, 4, 12, 17, 18, 19, 20]`, and `'3:6, 8, 4'` represents the pages list `[3, 4, 4, 5, 6, 8]`.
- The list `'all'`. It represents all the pages.
- A list with the above-mentioned elements. It represents the union of the pages in its elements. In this list also can be a list of the above-mentioned elements, and so on.

These unions accept the repetition of elements, but the order does not matter.

Note. We can use `['all', 8]` but not `'all, 8'`.

The pages in the PDF file produced by this script, `output_filepath`, will be in the same order as the pages in `input_filepath`. The deleted pages will not be in the output file. Each non-deleted page in the input file will be only one time in the output file. If `pages` has integers that do not correspond with a page of the input file, they will be ignored.

If `pages` has repeated page numbers then the behavior of `action` depends on the value of `with_repetitions`. If it is `True` the corresponding actions `ed_action_1`, ..., `ed_actions_n` will be done to each page in `pages` as many times as this page number is in `pages`. But if it is `False`, these repetitions in `pages` are ignored. For example, if

```
actions = [([(0, 4), 2], (P_R, 90), (P_SB, 0.5))]
with_repetitions = True
```

the third page will be rotated by 180° and escaled by 0.25, because the integer `2` is two times in `[(0, 4), 2]`. If `with_repetitions = False`, then the first four pages are rotated by 90° and scaled by 0.5.

If a page number is in the first element of two different elements of `actions`, then the corresponding actions of these two elements of `actions` will be done to this page in the order that they are no matter the value of `with_repetitions`. For example, if

```
actions = [[(3, 13), (C_S, (0.9, 0.75))], [[7, 15], (P_R, 30),
           (C_S, (0.95, 0.8))]]
with_repetitions = False
```

page number 7 has the following sequence of edition actions `(C_S, (0.9, 0.75))`, `(P_R, 30)`, and `(C_S, (0.95, 0.8))`.

2.2 The script

The content of the script is the following

```
import pdf_transformer as pt

"""
This script is an interface for the function 'pdf_editor' defined in
pdf_transformer.py.

DOCSTRING OF PDF_EDITOR FUNCTION:

This function takes each page of 'i_filename', applies to it the
actions in 'actions' for that page, and adds all those pages to the
'o_filename' file.

The argument 'actions' is a list of tuples (or lists). Those tuples
or lists have at least two elements. The first element of those
tuples is an integer, list, tuple, or string that represents a page
number or set of page numbers, see below. The page numbers start with
0. The others element of these tuples (or lists) are the constant P_D
(page delete) or two-tuples (or two-lists) that represents an action
with its parameters, in this two-tuples the first element is one of
the following constants: P_SB (page scale by), P_ST (page scale to),
P_R (page rotation), C_S (contents scale), C_T (contents
translation), and C_R (contents rotation). All the rotations are
counter-clockwise.

The number of parameters by action is:
P_D: no parameters;
P_SB: the scaling factor, float (one parameter);
P_ST: width and height, floats (two parameters);
P_R: angle (in degrees, multiples of 90), integers (one parameter);
C_S: the scale factors along the x-axis and y-axis, floats
     (two parameters);
C_T: the translations along the x-axis and y-axis, floats (two
     parameters);
C_R: angle (in degrees), float (one parameter), e.g. (C_R, 30)

Instead of page numbers, we can use:
-> a list of integers that represents a set of page numbers, e.g.
   [2, 5, 8] for pages 2, 5, and 8
-> a two-tuple of integers that represents a range of numbers, e.g.
   (2, 5) is equal to [2, 3, 4], i.e. list(range(2, 5))
-> a three-tuple of integers that represents a range of numbers, e.g.
   (1, 7, 2) is equal to [1, 3, 5], i.e. list(range(1, 7, 2))
```

```
-> a string with page numbers comma-separated, e.g. '2, 4, 7, 5',
   ranges, e.g. '5:12', or 'all' which means all the pages. We can
   use strings with comma-separated numbers and ranges, e.g. '0, 4:7,
   9:11' gives pages 0, 4, 5, 6, 7, 9, 10, and 11
We can use lists that contain all those elements. For example the
list [2, 5, (8, 11), '5, 12:16, 24, 22', [28, (31, 35)]] is equivalent
to [2, 5, 5, 8, 9, 10, 12, 13, 14, 15, 16, 22, 24, 28, 31, 32, 33, 34].
Note that we can use ['all', 8] but not 'all, 8'.

If there are actions for page numbers that 'i_filename' does not have,
these page numbers are ignored. For example, if 'i_filename' has forty
pages the action ((-3, 5), P_D) deletes the first five pages (pages 0
to 4).

If the argument 'with_repetitions' is True and a page number is
repeated in the first element of a tuple or list that belongs to
'actions' then the actions on that tuple or list are also repeated on
that page. If one page number is in two elements of 'actions', both
actions will be executed in the order given by 'actions', regardless
of the value of 'with_repetitions'.

Example of actions list:
[([2, 8, (10, 13)], (P_R, -90), (C_T, (12.4, -21.8))),
([(14, 17), '1:5, 25', 27], (P_SB, 0.97), (C_R, 10)),
(8, (P_ST, (590, 840))),
([0, 35], P_D)].
Note that there are four actions ('P_R', 'C_T', 'P_SB', and 'C_R') on
page 2. There are three actions on page 8.

Requirements: PyPDF2.PdfReader, PyPDF2.PdfWriter,
PyPDF2.Transformation, number_of_pages, page_action_maker,
standard_actions_input.

Required by: no one

:param i_filename: A string, the input file name
:param o_filename: A string, the output file name
:param actions: A list of tuples or lists, see above
:param with_repetitions: Boolean, see above, by default True
:return: None
"""

# CONSTANTS
# For transformation:
P_D = 0  # Page delete
```

```
P_SB = 1  # Page scale by
P_ST = 2  # Page scale to
P_R = 3  # Page rotation
C_S = 10  # Contents scale
C_R = 11  # Contents rotation
C_T = 12  # Contents translation

# ARGUMENTS:
input_filepath = "input\\filename.pdf"
output_filepath = "output\\filename-ed.pdf"
actions = [(0, (P_SB, 1))]
with_repetitions = True

if __name__ == '__main__':
    # Verify constants
    const_pt = [pt.P_D, pt.P_SB, pt.P_ST, pt.P_R, pt.C_S, pt.C_R,
                pt.C_T]
    const_here = [P_D, P_SB, P_ST, P_R, C_S, C_R, C_T]
    if const_here == const_pt:
        pt.pdf_editor(input_filepath, output_filepath, actions,
                      with_repetitions)
    else:
        print("The constants in pt and here do not coincide!")
```

2.3 Examples and notes

Some examples follow.

Example 1. Suppose that `my_file.pdf` has 25 pages. If we use the script `pdf_editor.py` with the following arguments

```
input_filepath = "input\\my_file.pdf"
output_filepath = "output\\my_file-ed.pdf"
actions = [([5, 6, 12, 17, 19, 24], P_D),
           ("1:4, 7", (P_SB, 0.7), (P_R, 90)),
           ([0, (8, 12), 10, (13, 17), 18], (P_R, 90))
          ]
with_repetitions = True
```

Then pages 5, 6, 12, 17, 19, and 24 (the last one) of the file will not be in the output file. Pages 1, 2, 3, 4, and 7 will be scaled by 0.7 and rotated 90° counter-clockwise. Pages 0, 8, 9, 11, 13, 14, 15, 16, and 18 will be rotated 90°. Page 10 will be rotated 180°. The remaining pages 20, 21, 22, and 23 will be in the output file with no change. So, `my_file-ed.pdf` will have 19 pages in the same order as in `my_file.pdf`, but with some changes in 15 pages.

Example 2. Suppose that `my_file.pdf` has 15 pages. If we use the script with the following arguments

```
input_filepath = "input\\my_file.pdf"
output_filepath = "output\\my_file-ed.pdf"
actions = [("1:4, 7", (P_R, 90)),
           ([0, 4, (8, 12), 10, (13, 17), 18], (P_R, 90))
          ]
with_repetitions = False
```

Then pages 0, 1, 2, 3, 7, 8, 9, 10, 11, 13, and 14 will be rotated 90°, page 4 will be rotated 180°, and pages 5, 6, and 12 will be in the output file without changes. Note that pages 15, 16, and 18 do not exist in the input file.

3 About the script `pdf_merger.py`

3.1 How it works

To use this script, we have to give values to the following variables:

- `input_pdfs`, the PDF file paths of the input files. Its value must be a list of strings
- `output_pdf`, the PDF file path of the output file. Its value must be a string
- `pages`, gives the information on what pages of what files will be in the output PDF file, `output_pdf`. Its value must be `None` or a list

The purpose of this script is to produce the output PDF file with the pages of the input PDF files that the value of `pages` determines. If

```
input_pdfs = [pdf_1, ..., pdf_k]
output_pdf = pdf_0
pages = None
```

then `pdf_0` has first all the pages of `pdf_1`, then all the pages of `pdf_2`,and so on. The behavior with `pages = []` is the same as with `pages = None`.

If the value of `pages` is a nonempty list, then its elements must be nonnegative integers, tuples, and/or lists. If an element of `pages` is the integer `i` then it represents all the pages of the PDF file `input_pdfs[i]`, and so $0 \leq$ `i` $<$ `k`. If an element of `pages` is a tuple or list, say `x`, then the first element of `x`, `x[0]`, must be an integer `i`, with $0 \leq$ `i` $<$ `k`, that represents the index of and PDF file in `input_pdfs`, and the other elements of `x` give pages of this PDF file. In each PDF file, the pages are counted from 0 on.

The elements of `x[1:]` must be nonnegative integers, two-tuples of nonnegative integers, or three-tuples of integers whose two first elements are nonnegative. An integer `j` represents that page number of the PDF file `input_pdfs[i]`, a two-tuple `(a, b)` represents the page numbers in `list(range(a, b))` of the same PDF file, and a three-tuple `(a, b, c)` represents the page numbers in `list(range(a, b, c))` of the same PDF file.

If `x` is a list with only one element that element must be an integer `i`, with $0 \leq$ `i` $<$ `k`, and it is equivalent to `x[0]`, i.e. if `x` is `[i]` or `i` give the same. So it represents all the pages of `input_pdfs[i]`.

3.2 The script

The content of the script is the following

```
import pdf_transformer as pt

"""
This script is an interface for the function 'pdf_merger' defined in
pdf_transformer.py.

DOCSTRING OF PDF_MERGER FUNCTION:

This function creates a PDF file named 'o_filename' with the content
of the files whose file names are in the list 'i_filenames'.

The 'pages' argument determines the content of 'o_filename'. If
'pages' is None or an empty list, then  'o_filename' contains all the
pages of the files whose names are in 'i_filenames' in increasing
order. First the pages of the first file, then of the second one, and
so on.

If 'pages' argument is not None nor an empty list, it should be a list
whose elements are integers, lists, or tuples. If an element of 'pages'
is an integer 'i' it represents all the pages of 'i_filenames[i]'. If
an element 'x' of 'pages' is a list or tuple, the first element of 'x'
represents the index of the source PDF file in 'i_filenames', and the
other elements of 'x' represents the pages of this PDF file. In each
PDF file, the pages are counted from 0 on. These other elements of 'x'
are integers, two-tuples, or three-tuples. The integers represent the
corresponding page, the two-tuple (a, b) represents the pages in
range(a, b), and the three-tuple (a, b, c) represents the pages in
range(a, b, c). If 'x' is a list with only one integer, it is equivalent
to this integer, i.e. 'x[0]', so it represents all the pages of
'i_filenames[x[0]]'.

For example, if 'pages' is [(1, 0, 1, (6, 10), 0), (0, (2, 10, 2)), 2]
it means that the output PDF file contains the pages 0, 1, 6 to 9, and
0 of the second PDF in 'i_filenames' ('i_filenames[1]'), then the pages
2, 4, 6, and 8 of the first PDF in 'i_filenames' ('i_filenames[0]), and
then all the pages of the third PDF file in 'i_filename'
('i_filenames[2]) in this order. Note that the first page of the second
PDF is two times in the output PDF file.

Requirements: PyPDF2.PdfWriter, and standard_merge_pages.

Required by: no one
```

```
:param i_filenames: A list of input PDF filenames
:param o_filename: A string, the filename of the output PDF file
:param pages: A list or None, by default None, see above
:return: None
"""

input_pdfs = ['input\\file1.pdf', 'input\\file2.pdf', 'input\\file3.pdf']
output_pdf = 'output\\merged.pdf'
pages = []

if __name__ == '__main__':
    pt.pdf_merger(input_pdfs, output_pdf, pages)
```

3.3 Examples and notes

Some examples follow.

Example 3. If the values of the variables are

```
input_pdfs = ['input\\file1.pdf', 'input\\file2.pdf', 'input\\file3.pdf']
output_pdf = 'output\\merged.pdf'
pages = [0, (1, (0, 10), (14, 20, 2)), (2, 0)]
```

then the script produces a PDF file `merged.pdf` that first contains all the pages of `file1.pdf`, then follows the pages 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 14, 16, and 18 of `file2.pdf`, and finally, the first page of `file3.pdf`.

Example 4. If the values of the variables are

```
input_pdfs = ['input\\file1.pdf', 'input\\file2.pdf', 'input\\file3.pdf']
output_pdf = 'output\\merged.pdf'
pages = [[1, 0, 1, (6, 10), 0], [0, (2, 10, 2)], 2]
```

then `merged.pdf` contains first the pages 0, 1, 6 to 9, and 0 of `file2.pdf`, then the pages 2, 4, 6, and 8 of `file1.pdf`, and then all the pages of `file3.pdf`.

Example 5. If we scan the course notes which have 32 pages in a file `file1.pdf`. We realize that we missed scanning page 12 and page 23[1] was wrong scanned. Then we scan those two pages in a file `file2.pdf`. With the following values for the variables, this script will produce a file with all the pages in the correct order.

```
input_pdfs = ['input\\file1.pdf', 'input\\file2.pdf']
output_pdf = 'output\\CourseNotes.pdf'
pages = [(0, (0, 12)), (1, 0), (0, (12, 22)), (1, 1), (0, (23, 31))]
```

[1] These two numbers correspond to the pages from the course notes numbered from 0 to 31.

4 About the script page_number_translator.py

4.1 How it works

In the functions `pdf_editor` and `pdf_merger` of the module `pdf_transformer.py` the pages of the PDF files are numbered from 0 on. But when working with some PDF files, it could be more natural, that the page numbers are the same as the impressed on its pages or simply start with 1 instead of 0. The function `page_number_translator` of `pdf_transformer.py` and the script `page_number_translator.py` helps with this.

As with the previous scripts, the script `page_number_translator.py` is an interface for the function `page_number_translator` of the module `pdf_transformer.py`. In this case, it is more useful the function than the script. In fact, we can use this function in the scripts `pdf_editor.py` and `pdf_merger.py`.

Note that the page information in the function or script `pdf_editor` is in the parameter, or variable, `actions`. Specifically in the first elements of its elements. For the function and script `pdf_merger`, the page information is in the parameter, or variable, `pages`. Specifically in all the elements but the first of its elements. That is, the page information is in `actions[i][0]` (resp. `pages[i][1:]`), for $0 \leq$ `i` $<$ `n`, where `n` is `len(actions)` (resp. `len(pages)`). Note also that `actions` and `pages` are lists.

We say that a list is *actions-like* if the only difference between it and a proper argument to the `actions` parameter of the function `pdf_editor` is that the page numbers start with an integer that can be different from 0.

We say that a list is *pages-like* if the only difference between it and a proper argument to the `pages` parameter of the function `pdf_merger` is that the page numbers start with an integer that can be different from 0 and the file index reference start with 1 instead of 0. But the start page number for every file in the `input_pdfs` value is the same.

The function `page_number_translator` has the following parameters

- `i_list`, the input list. It must be as an actions-like list or a pages-like list
- `start`, the starting page number, by default `1`
- `kind`, it determines the kind of list that `i_list` is, the possible values are `'actions'` and `'pages'`, the default value is `'actions'`
- `inverse`, it must be a boolean, by default it is `False`

This function returns a list of the same kind as `i_list`. If `i_list` is an actions-like (resp. pages-like) list for `pdf_editor` (resp. `pdf_merger`), then the `kind` argument must be `'actions'` (resp. `'pages'`).

For each page number reference `i` in `i_list` it must be a corresponding page number reference `j` in `o_list`, where

```
o_list = page_number_translator(i_list, m, kind)
```

or

```
o_list = page_number_translator(i_list, m, kind, inverse=False)
```

The relation between `i` and `j` is given by `j` = `i` − `m`.

So if `i_list` is an actions-like (resp. pages-like) argument of the function `pdf_editor` (resp. `pdf_merger`) but whose starting page number is `m`, instead of `0`, then `o_list` is an appropriate actions (resp. pages) argument for the `pdf_editor` (resp. `pdf_merger`) function.

If

$$F : \texttt{x} \mapsto \texttt{page\_number\_translator}(\texttt{x}, \texttt{m}, \texttt{kind}),$$
$$G : \texttt{x} \mapsto \texttt{page\_number\_translator}(\texttt{x}, \texttt{m}, \texttt{kind}, \texttt{inverse} = \texttt{True}),$$

Then $G = F^{-1}$, i.e. G is the inverse of F.

When `kind='pages'`, the function not only changes the page number references, it also changes the `input_pdfs` index references as follows. If

```
o_list = page_number_translator(i_list, m, 'pages')
```

then `o_list[i][1]` = `i_list[i][1]` − 1, for 0 ≤ `i` < `len(i_list)`.

That means that if we want to use `start` as the first-page number in our PDF files when using `pdf_merger`, then we have to use as the number references for the `input_pdfs` the numbers from 1 on, instead of from 0 on.

If we want to start the page number references of an actions-like list with `m` (≠ `0`), then we can edit the script `pdf_editor.py` as follows

```
actions_like = [(m, (P_SB, 1))]
actions = pt.page_number_translator(actions_like, m)
```

If we want to start the page number references of a pages-like list with `m`, then we can edit the script `pdf_merger.py` as follows

```
pages_like = [(1, m), (2, m)]
pages = pt.page_number_translator(pages_like, m, 'pages')
```

4.2 The script

The content of the script is the following

```
import pdf_transformer as pt

"""
This script is an interface for the function 'page_number_translator'
defined in pdf_transformer.py. This function could be used in the
scripts pdf_editor.py and pdf_merger.py, for example:
input_filepath = "input\\filename.pdf"
output_filepath = "output\\filename-ed.pdf"
# First page is 1
actions = [('1:5, 8', (P_SB, 0.5)), (6, P_D)]
# First page is 0
actions = pt.page_number_translator(actions)
```

pdf_editor(input_pdf, output_pdf, actions)

Note: For use the constants pt.P_D, pt.P_SB, etc. in this script, you need 'pt.'.

This script output to the console the returned value of the function.

DOCSTRING OF PAGE_NUMBER_TRANSLATOR FUNCTION:

The first argument (i_list) should be a list whose elements are lists or tuples. These elements should have at least two elements. The second argument (start) should be an integer, its default value is 1. The third argument (kind) should be one of the following strings: 'actions' or 'pages', its default value is 'actions'. Finally, the last argument (inverse) should be a boolean, its default value is False.

Assume that the argument 'inverse' is False. The argument 'actions' of the 'pdf_editor' function and the argument 'pages' of the 'pdf_merger' function should be lists that reference the PDF files' page numbers. For these functions, 0 is the first page of a PDF file. If we want to start with another page number, say -10, we can use this function, 'page_number_translator', to obtain the argument for those functions.

The argument 'pages' of 'pdf_merger' also has references to the different PDF files in the argument 'i_filenames'. These references are by the index in the list 'i_filenames', so the first PDF file is referenced by 0, and so on. This function starts the PDF files reference by 1 instead of 0.

The third argument, 'kind', says if the first argument 'i_list' is like the 'actions' argument of 'pdf_editor' (kind='actions') or the 'pages' argument of 'pdf_merger' (kind='pages'). The default value for this parameter is 'actions'. See the docstrings of the functions 'pdf_editor' and 'pdf_merger' to know how should be 'i_list' elements.

For example, if 'i_list' is [('1, -3, -7:-2, all', (1,2))], 'start' is -12, and 'kind' is 'actions', then the function returns [['13, 9, 5:10, all', (1, 2)]].

If the fourth parameter, 'inverse', is True, this function is the inverse of the above-explained function. For example, if 'start' is -12, 'kind' is actions, 'i_list' is [['13, 9, 5:10, all', (1, 2)]] and 'inverse' is True, then it returns [('1, -3, -7:-2, all', (1,2))].

If 'inverse' is False and 'kind' is 'actions', this function subtracts

```
from every integer in the first element of each element of 'i_list' the
value of 'start', with the sole exception of the third element of
three-tuples, which remain the same, because they represent steps, not
pages. For example, if 'start' is -10 and 'i_list' is
[('4:7, 2', 1, 2), (-2, 3), [(0, 10, 4), 7]], the function returns
[['14:17, 12', 1, 2], [8, 3], [(10, 20, 4), 7]].

If 'inverse' is False and 'kind' is 'pages', the function subtracts 1
from the first element of every element in 'i_list', and subtracts
'start' from every integer in the other elements of each element in
'i_list', with the sole exception of the third element of three-tuples.
For example, if 'i_list' is [(1, 1, 2), (3, 3), [2, (0, 10, 4), 7]],
'start' is -10, and 'kind' is 'pages', then the function returns
[[0, 11, 12], [2, 13], [1, (10, 20, 4), 17]].

Requirements: integer_translator

Required by: no one

:param i_list: A list, see above
:param start: An integer, by default 1
:param kind: A string, 'actions' or 'pages', by default the first
:param inverse: A boolean, by default False
:return: A list
"""

actions = [(1, (pt.P_SB, 1))]
pages = [(1, "all"), (2, "all")]
start = 1  # Default value 1
kind = 'actions'  # Options: 'actions' (default) and 'pages'
inverse = False  # Default False

if __name__ == '__main__':
    if kind == 'actions':
        print(pt.page_number_translator(actions, start, kind='actions',
                                        inverse=inverse))
    elif kind == 'pages':
        print(pt.page_number_translator(pages, start, kind='pages',
                                        inverse=inverse))
```

4.3 Examples and notes

Some examples follow.

Example 6. Suppose that we have a PDF file whose pages 25, 31, 47, and 79 are tables rotated

90° counterclockwise. But these page numbers are the ones that we can read on the page, and page number 1 is the twelfth page of the PDF. If we start to number the pages of the PDF file with -10 then the twelfth page of the PDF has page number one. If we want to rotate -90° these four pages then we can use the following variables in the `pdf_editor.py` script.

```
actions_like = [([25, 31, 47, 79], (P_R, -90))]
actions = pt.page_number_translator(actions_like, -10)
```

In this case, `actions` is `[([35, 41, 57, 89], (3, -90))]`, remember that `P_R` is `3`, see Table 1.

Example 7. If

```
pages_like = [(1, 1, (3, 7), (9, 20, 3)), (2, (1, 5), 7, 12)]
pages = pt.page_number_translator(pages_like, 1, kind='pages')
```

then `pages` is equal to `[[0, 0, (2, 6), (8, 19, 3)], [1, (0, 4), 6, 11]]`.

5 About the script `pdf_inf.py`

5.1 How it works

This script is an interface for the `pdf_inf` function of the `pdf_transformer.py` module. This script only requires giving value to the variable `input_filepath` which has to be a valid path to a PDF file. The script prints to the terminal information of this PDF file.

The output information is given in the following table. A PS Big point is equal to $\frac{1}{72}$`inch`.

Table 2: The information given by the script `pdf_inf.py`

Label	**Description**
`num_pages`	Number of pages of the PDF file, an integer
`user_units`	User units in PS Big points, a positive integer or a dictionary, see `user_units_maker`
`min_width`	Minimum page width, a positive number
`max_width`	Maximum page width, a positive number
`average_width`	Average width, a positive number
`i_average_width`	Round of average width, a positive number
`min_height`	Minimum page height, a positive number
`max_height`	Maximum page height, a positive number
`average_height`	Average height, a positive number
`i_average_height`	Round of average height, a positive number
`pages_size`	A list of page sizes, list of two-tuples

5.2 The script

The content of the script is the following

```
from PyPDF2 import PdfReader
import pdf_transformer as pt

"""
This script is an interface for the function 'pdf_inf' defined in
pdf_transformer.py.

DOCSTRING OF PDF_INF FUNCTION:

This function returns a dictionary with information about the pages of
the argument, a PdfReader object of PyPDF2. The information is under
the following keys:
'num_pages' (number of pages, integer)
'user_units' (user units in PS Big Points, positive integer or
              dictionary, see user_units_maker)
'min_width' (minimum page width, positive number)
'max_width' (maximum page width, positive number)
'average_width' (average width, positive number)
'i_average_width' (round of average width, positive number)
'min_height' (minimum page height, positive number)
'max_height' (maximum page height, positive number)
'average_height' (average height, positive number)
'i_average_height' (round of average height, positive number)
'pages_size' (the list of page sizes, list of four-tuples)

Note: A PS Big Point is equal to 1/72 inch

Requirements: PyPDF2.PdfReader and user_units_maker.

Required by: add_grid

:param pdf_reader: A PdfReader Object
:return: A dictionary
"""

# ARGUMENTS:
input_filepath = "input\\filename.pdf"

if __name__ == '__main__':
    reader = PdfReader(input_filepath)
    pdf_inf = pt.pdf_inf(reader)
    for key in pdf_inf.keys():
        print(f"{key}: {pdf_inf[key]}")
```

5.3 Examples and notes

Some examples follow.

Example 8. The terminal output for a PDF file whose pages have different dimensions

```
num_pages: 192
user_units: 1
min_width: 421
max_width: 478
average_width: 449.3020833333333
i_average_width: 449
min_height: 605
max_height: 669
average_height: 638.9895833333334
i_average_height: 639
pages_size: [(478, 650), (450, 651), (443, 649), (429, 647), (436, 645),
... (31 lines)
(476, 651)]
```

Example 9. The terminal output for a PDF file whose pages have two different dimensions

```
num_pages: 229
user_units: 1
min_width: 476.22
max_width: 490.394
average_width: 476.2818951965065502183406114
i_average_width: 476
min_height: 680.315
max_height: 697.323
average_height: 680.3892707423580786026200873
i_average_height: 680
pages_size: [(Decimal('490.394'), Decimal('697.323')),
(Decimal('476.22'), Decimal('680.315')),
... (226 lines)
(Decimal('476.22'), Decimal('680.315'))]
```

6 Installation

You need Python 3.9. Download the project folder `PDFUtils`, create a virtual environment with `pip` and `requirements.txt`, and run with this virtual environment the script that you want. To uninstall it, simply delete the folders of the project and the virtual environment.