```python
"""
Representation and utils for ranges of PDF file pages.

Copyright (c) 2014, Steve Witham <switham_github@mac-guyver.com>.
All rights reserved. This software is available under a BSD license;
see https://github.com/py-pdf/PyPDF2/blob/main/LICENSE
"""

import re
from typing import Any, List, Tuple, Union

from .errors import ParseError

_INT_RE = r"(0|-?[1-9]\d*)"  # A decimal int, don't allow "-0".
PAGE_RANGE_RE = "^({int}|({int}?(:{int}?(:{int}?)?)))$".format(int=_INT_RE)
# groups:         12     34     5 6     7 8


class PageRange:
    """
    A slice-like representation of a range of page indices.

    For example, page numbers, only starting at zero.

    The syntax is like what you would put between brackets [ ].
    The slice is one of the few Python types that can't be subclassed,
    but this class converts to and from slices, and allows similar use.

      -  PageRange(str) parses a string representing a page range.
      -  PageRange(slice) directly "imports" a slice.
      -  to_slice() gives the equivalent slice.
      -  str() and repr() allow printing.
      -  indices(n) is like slice.indices(n).

    """

    def __init__(self, arg: Union[slice, "PageRange", str]) -> None:
        """
        Initialize with either a slice -- giving the equivalent page range,
        or a PageRange object -- making a copy,
        or a string like
            "int", "[int]:[int]" or "[int]:[int]:[int]",
            where the brackets indicate optional ints.
        Remember, page indices start with zero.
        Page range expression examples:
            :     all pages.                   -1    last page.
            22    just the 23rd page.          :-1   all but the last page.
            0:3   the first three pages.       -2    second-to-last page.
            :3    the first three pages.       -2:   last two pages.
            5:    from the sixth page onward.  -3:-1 third & second to last.
        The third, "stride" or "step" number is also recognized.
            ::2       0 2 4 ... to the end.    3:0:-1    3 2 1 but not 0.
            1:10:2    1 3 5 7 9                2::-1     2 1 0.
            ::-1      all pages in reverse order.
        Note the difference between this notation and arguments to slice():
            slice(3) means the first three pages;
            PageRange("3") means the range of only the fourth page.
            However PageRange(slice(3)) means the first three pages.
        """
        if isinstance(arg, slice):
            self._slice = arg
            return

        if isinstance(arg, PageRange):
            self._slice = arg.to_slice()
            return

        m = isinstance(arg, str) and re.match(PAGE_RANGE_RE, arg)
        if not m:
            raise ParseError(arg)
        elif m.group(2):
            # Special case: just an int means a range of one page.
            start = int(m.group(2))
            stop = start + 1 if start != -1 else None
            self._slice = slice(start, stop)
        else:
            self._slice = slice(*[int(g) if g else None for g in m.group(4, 6, 8)])

    @staticmethod
    def valid(input: Any) -> bool:
        """True if input is a valid initializer for a PageRange."""
        return isinstance(input, (slice, PageRange)) or (
            isinstance(input, str) and bool(re.match(PAGE_RANGE_RE, input))
        )

    def to_slice(self) -> slice:
        """Return the slice equivalent of this page range."""
        return self._slice

    def __str__(self) -> str:
        """A string like "1:2:3"."""
        s = self._slice
        indices: Union[Tuple[int, int], Tuple[int, int, int]]
        if s.step is None:
            if s.start is not None and s.stop == s.start + 1:
                return str(s.start)

            indices = s.start, s.stop
        else:
            indices = s.start, s.stop, s.step
        return ":".join("" if i is None else str(i) for i in indices)

    def __repr__(self) -> str:
        """A string like "PageRange('1:2:3')"."""
        return "PageRange(" + repr(str(self)) + ")"

    def indices(self, n: int) -> Tuple[int, int, int]:
        """
        n is the length of the list of pages to choose from.

        Returns arguments for range().  See help(slice.indices).
        """
        return self._slice.indices(n)

    def __eq__(self, other: Any) -> bool:
        if not isinstance(other, PageRange):
            return False
        return self._slice == other._slice

    def __add__(self, other: "PageRange") -> "PageRange":
        if not isinstance(other, PageRange):
            raise TypeError(f"Can't add PageRange and {type(other)}")
        if self._slice.step is not None or other._slice.step is not None:
            raise ValueError("Can't add PageRange with stride")
        a = self._slice.start, self._slice.stop
        b = other._slice.start, other._slice.stop

        if a[0] > b[0]:
            a, b = b, a

        # Now a[0] is the smallest
        if b[0] > a[1]:
            # There is a gap between a and b.
            raise ValueError("Can't add PageRanges with gap")
        return PageRange(slice(a[0], max(a[1], b[1])))


PAGE_RANGE_ALL = PageRange(":")  # The range of all pages.


def parse_filename_page_ranges(
    args: List[Union[str, PageRange, None]]
) -> List[Tuple[str, PageRange]]:
    """
    Given a list of filenames and page ranges, return a list of (filename,
page_range) pairs.

    First arg must be a filename; other ags are filenames, page-range
    expressions, slice objects, or PageRange objects.
    A filename not followed by a page range indicates all pages of the file.
    """
    pairs: List[Tuple[str, PageRange]] = []
    pdf_filename = None
    did_page_range = False
    for arg in args + [None]:
        if PageRange.valid(arg):
            if not pdf_filename:
                raise ValueError(
                    "The first argument must be a filename, not a page range."
                )

            pairs.append((pdf_filename, PageRange(arg)))
            did_page_range = True
        else:
            # New filename or end of list--do all of the previous file?
            if pdf_filename and not did_page_range:
                pairs.append((pdf_filename, PAGE_RANGE_ALL))

            pdf_filename = arg
            did_page_range = False
    return pairs


PageRangeSpec = Union[str, PageRange, Tuple[int, int], Tuple[int, int, int],
List[int]]
```