```
#---------------------------------------------------------------------------
#
#   PyGUI - TextEditor Printing - Gtk
#
#---------------------------------------------------------------------------

import re
from Printing import Printable

class TextEditorPrintView(Printable):

    def __init__(self, base_view, page_setup):
        self.base_view = base_view; self.width = page_setup.page_width; self.page_height =
page_setup.page_height; self.lay_out_text(); lines_per_page = int(page_setup.page_height /
base_view.font.line_height); self.lines_per_page = lines_per_page; num_lines = len(self.lines); self.num_pages
= (num_lines + lines_per_page - 1) // lines_per_page
        self.height = self.num_pages * self.page_height
        print "TextEditorPrintView: lines_per_page =", self.lines_per_page ###
        print "...num_lines =", num_lines ###
        print "...height =", self.height ###
        print "...page_height =", self.page_height ###

    def lay_out_text(self):
        base_view = self.base_view
        font = base_view.font
        space_width = font.width(" ")
        tab_spacing = base_view.tab_spacing
        page_width = self.width
        wrap = not base_view.get_hscrolling()
        if wrap:
            pat = re.compile(r"[ \t]|[^ \t]+")
        else:
            pat = re.compile(r"\t|[^ \t]+")
        lines = []
        line = []
        x = 0
        for text_line in base_view.text.splitlines():
            for match in pat.finditer(text_line):
                item = match.group()
                if item == " ":
                    item_width = space_width
                    item = ""
                elif item == "\t":
                    item_width = tab_spacing - x % tab_spacing
                    item = ""
                else:
                    item_width = font.width(item)
                if wrap and x + item_width > page_width and x > 0:
                    lines.append(line); line = []; x = 0
                line.append((x, item))
                x += item_width
            lines.append(line); line = []; x = 0
        self.lines = lines

    def get_print_extent(self):
        return self.width, self.height

    def draw(self, canvas, page_rect):
        l, t, r, b = page_rect
        page_no = int(t / self.page_height)
        print "TextEditorPrintView.draw: page_no =", page_no ###
        n = self.lines_per_page
        i = page_no * n
        font = self.base_view.font
        y = t + font.ascent
        dy = font.line_height
        for line in self.lines[i : i + n]:
            for x, item in line:
                canvas.moveto(x, y)
                canvas.show_text(item)
            y += dy

#---------------------------------------------------------------------------
#
#   PyGUI - Printing - Generic
#
```

```
#---------------------------------------------------------------------------

from __future__ import division
from math import ceil
import cPickle as pickle
from Properties import overridable_property
from Globals import application

class PageSetup(object):
	"""Holder of information specified by the "Page Setup" command."""

	paper_name = overridable_property('paper_name')
	paper_width = overridable_property('paper_width')
	paper_height = overridable_property('paper_height')
	left_margin = overridable_property('left_margin')
	top_margin = overridable_property('top_margin')
	right_margin = overridable_property('right_margin')
	bottom_margin = overridable_property('bottom_margin')
	orientation  = overridable_property('orientation')

	paper_size = overridable_property('paper_size')
	margins = overridable_property('margins')
	page_width = overridable_property('page_width')
	page_height = overridable_property('page_height')
	page_size = overridable_property('page_size')
	page_rect = overridable_property('page_rect')
	printable_rect  = overridable_property('printable_rect') # May not work
	printer_name = overridable_property('printer_name')

	_pickle_attributes = ['paper_name', 'paper_size', 'margins',
		'printer_name', 'orientation']

	def __getstate__(self):
		state = {}
		for name in self._pickle_attributes:
			state[name] = getattr(self, name)
		return state

	def __setstate__(self, state):
		for name, value in state.iteritems():
			setattr(self, name, value)

	def from_string(s):
		"""Restore a pickled PageSetup object from a string."""
		return pickle.loads(s)

	from_string = staticmethod(from_string)

	def to_string(self):
		"""Pickle the PageSetup object and return it as a string."""
		return pickle.dumps(self, 2)

	def get_paper_size(self):
		return self.paper_width, self.paper_height

	def set_paper_size(self, x):
		self.paper_width, self.paper_height = x

	def get_margins(self):
		return self.left_margin, self.top_margin, self.right_margin, self.bottom_margin

	def set_margins(self, x):
		self.left_margin, self.top_margin, self.right_margin, self.bottom_margin = x

	def get_page_width(self):
		return self.paper_width - self.left_margin - self.right_margin

	def get_page_height(self):
		return self.paper_height - self.top_margin - self.bottom_margin

	def get_page_size(self):
		return (self.page_width, self.page_height)

	def get_page_rect(self):
		lm, tm, rm, bm = self.margins
		pw, ph = self.paper_size
```

```
        return (lm, tm, pw - rm, ph - bm)

#------------------------------------------------------------------------------

class Printable(object):
    """Mixin class for components implementing the "Print" command."""

    printable = overridable_property('printable', "Whether this component should handle the 'Print' command.")
    page_setup = overridable_property('page_setup', "The PageSetup object to use for printing.")
    print_title = overridable_property('print_title', "Title for print job.")

    _printable = True

    def get_printable(self):
        return self._printable

    def set_printable(self, x):
        self._printable = x

    def get_print_title(self):
        window = self.window
        if window:
            return window.title
        else:
            return ""

    def get_page_setup(self):
        result = None
        model = getattr(self, 'model', None)
        if model:
            result = getattr(model, 'page_setup', None)
        if not result:
            result = application().page_setup
        return result

    def setup_menus(self, m):
        if self.printable:
            m.print_cmd.enabled = True

    def print_cmd(self):
        if self.printable:
            page_setup = self.page_setup
            if page_setup:
                self.print_view(page_setup)
        else:
            self.pass_to_next_handler('print_cmd')

#------------------------------------------------------------------------------

class Paginator(object):
    """Used internally. A generic pagination algorithm for printing."""

    def __init__(self, view, page_setup):
        self.view = view
        extent_width, extent_height = view.get_print_extent()
        #paper_width, paper_height = page_setup.paper_size
        #lm, tm, rm, bm = page_setup.margins
        #page_width = int(paper_width - lm - rm)
        #page_height = int(paper_height - tm - bm)
        page_width, page_height = page_setup.page_size
        if page_width <= 0 or page_height <= 0:
            from AlertFunctions import stop_alert
            stop_alert("Margins are too large for the page size.")
            return
        self.extent_rect = (0, 0, extent_width, extent_height)
        self.page_width = page_width
        self.page_height = page_height
        self.left_margin = page_setup.left_margin
        self.top_margin = page_setup.top_margin
        self.pages_wide = int(ceil(extent_width / page_width))
        self.pages_high = int(ceil(extent_height / page_height))
        self.num_pages = self.pages_wide * self.pages_high

    def draw_page(self, canvas, page_num):
        row, col = divmod(page_num, self.pages_wide)
        view_left = col * self.page_width
```

```
view_top = row * self.page_height
view_right = view_left + self.page_width
view_bottom = view_top + self.page_height
view_rect = (view_left, view_top, view_right, view_bottom)
dx = self.left_margin - view_left
dy = self.top_margin - view_top
canvas.translate(dx, dy)
canvas.rectclip(self.extent_rect)
canvas.rectclip(view_rect)
canvas._printing = True
self.view.draw(canvas, view_rect)
```