

```
#!/usr/bin/env python

"""imgindex.py - Make a PDF index print of an image collection.

Pick all image files inside a directory and place them in a PDF
A4 document, trying to mimick the look of a photo index print.
The layout allows for left, right, top and bottom margins on the
paper (although this is partly still hard-wired in the code).

Slides are rectangular (ideally they should have a square shape)
with arbitrary side length. Inter-slide distance and the slide
margin thickness can be set, too. Row and page wrapping is done
automatically. Simple PDF bookmarks are also used (nested for
pages and rows), containing the image base filename. Labels are
printed on the top (resolution and image type) and the bottom
(filename).

This version does currently not use Platypus (ReportLab's lay-
outing engine). The next version most likely will use it, and
then we can rely on Platypus doing the wrapping for us... as
well as letting us embed a slide collection nicely in some real
surrounding document.

This module depends on the ReportLab toolkit as well as on the
Python Imaging Library (PIL). Note that I am not trying to get
away without PIL, limiting yourself to JPEGs only as you would
be without it.

The command line interface is kept simple on purpose, but might
be enriched later, when 'unwiring' some parameters.

Dinu Gherman, September 2000
"""

__version__ = 0, 4

# Python standard library modules.
import string, glob, os, tempfile, time, getopt, sys

# ReportLab stuff (not that much, actually).
from reportlab.pdfgen import canvas
from reportlab.lib.units import cm
from reportlab.lib.colors import Color

# PIL stuff.
import Image

def printUsage(error=None):
    "Print an info message."

    filename = os.path.basename(sys.argv[0])
    usage = "Usage: %s [options] <inputDir>" % filename
    options = """Options:
    -h      Print this help message.
    -o      Set output file path.
    """

    print string.split(__doc__, '\n')[0]
    print
    if error:
        print 'Error: %s' % error
        print
    print usage
    print options

### Helper functions.
```

```
def makeJPEGThumbnail(imgPath, size, quality=40):
    """Convert an image file to a thumbnail in JPEG format.

    The thumbnail's filename will be derived from the
    original image filename as in the following example:

        image.gif -> image-thumbnail-gif.jpg
    """

    tmpDir = tempfile.gettempdir()
    rest, ext = os.path.splitext(os.path.basename(imgPath))
    outFile = rest + '-thumbnail-' + ext[1:] + '.jpg'
    ourFile = os.path.join(tmpDir, outFile)

    try:
        img = Image.open(imgPath)
        img = img.resize(size)
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img.save(outFile, 'JPEG', quality=quality)
    except IOError:
        print "Cannot convert", imgPath

    return outFile

def identifyImagesInFolder(folder):
    """Which files are images in a given folder?

    Returns a list of filenames, that PIL thinks are
    images. This is independant of file extensions, as
    PIL will sniff into the file headers to find out.
    """

    files = []

    for file in os.listdir(folder):
        try:
            path = os.path.join(folder, file)
            Image.open(path)
            files.append(path)
        except:
            pass

    return files

def getImageFileNames(folder, ext='any'):
    """Find images in a folder, by extension or inspection.

    # Convert ext string into a list.
    ext = string.split(ext, ' ')

    # doubles?, case?

    # If we don't indicate any particular format, PIL
    # will inspect all files and take all the images
    # it finds...
    files = []
    if 'any' in ext:
        files = identifyImagesInFolder(folder)

    # ...else, we pick all files with all extensions
    # as specified in ext, assuming it is a list.
    else:
        for e in ext:
            pat = '*' + e
            pat = os.path.join(folder, pat)
            for f in glob.glob(pat):
```

```
        files.append(f)

    return files

### The real meat.

class Slide:
    """A class representing a single slide.

    Slides are going to be placed on a PDF canvas by some
    external mechanism. An individual slide is composed of
    some frame (with labels) and an image therein. Images
    keep there original aspect ratio if the slide has a
    square size.
    """

    def __init__(self, imgPath, size=None, b=None, c=None):
        """Initialize with some default values if needed."

        # Set (default) slide width and height (incl. frame).
        self.aw = size[0] or 3.75*cm
        self.ah = size[1] or 3.75*cm

        # Set (default) inter-slide distance (vert. and horiz.).
        self.b = b or .25*cm

        # Set (default) slide edge thickness.
        self.c = c or 0.25*cm

        # Memo image path.
        self.imgPath = imgPath

    def _getImageSize(self):
        """What is the image size?"

        # Sniff into file, reading image size.
        img = Image.open(self.imgPath)
        iw, ih = map(float, (img.size))

        return iw, ih

    def drawFrame(self, canvas, x, y):
        """Draw a rounded rectangular frame background."

        aw, ah = self.aw, self.ah
        d = self.c
        c = canvas

        gray = 0.75
        c.setFillGray(gray)
        c.setStrokeGray(gray)
        c.setLineWidth(0)
        c.roundRect(x, y, aw, ah, radius=aw/20, stroke=1, fill=1)

        #c.setStrokeGray(0)
        #c.setLineWidth(0)
        #c.rect(x+d, y+d, aw-2*d, ah-2*d)

    def drawImageNameLabel(self, canvas, x, y, imgPath):
        """Draw a label containing the image filename."

        aw, ah = self.aw, self.ah
        th = self.c
        c = canvas

        black = 0
```

```
c.setFillGray(black)
c.setStrokeGray(black)
size = th/cm*16
c.setFont('Helvetica', size)
basename = os.path.basename(imgPath)
c.drawCentredString(x+aw/2, y+size/2, basename)

def drawImageTypeLabel(self, canvas, x, y, imgPath):
    "Draw a label containing the image size and mode."

    aw, ah = self.aw, self.ah
    th = self.c
    c = canvas

    black = 0
    c.setFillGray(black)
    c.setStrokeGray(black)
    h = th/cm*16
    c.setFont('Helvetica', h)

    img = Image.open(imgPath)
    size, mode = img.size, img.mode
    label = '%dx%d %s' % (size[0], size[1], mode)

    c.drawCentredString(x+aw/2, y+ah-h, label)

def drawImage(self, canvas, x, y):
    """Draw a bitmap image on the slide.

    The image will be fit into the surrounding slide frame
    such that a certain distance to the margins and the
    original image aspect ratio will be maintained.
    (The aspect ratio is maintained only if the slide has
    a square shape.)
    """

    aw, ah = self.aw, self.ah
    b, c = self.b, self.c

    # Sniff into file, reading image size.
    iw, ih = self._getImageSize()
    w2h = iw/ih
    h2w = ih/iw

    # Calculate 'real' slide position and width/length for...
    # portrait format...
    if iw <= ih:
        w = (aw-2*c)*w2h
        h = ah-2*c
        x = x+c + (aw-2*c-w)/2
        y = y+c
    # ... or landscape format:
    else:
        w = aw-2*c
        h = (ah-2*c)*h2w
        x = x+c
        y = y+c + (ah-2*c-h)/2

    # Create temp. thumbnail and import it into the PDF canvas.
    r = 3 # increase resolution to allow for some decent zooming.
    g = makeJPEGThumbnail(self.imgPath, size=(r*w, r*h))
    args = (g, x, y, w, h)
    apply(canvas.drawImage, args)
    os.remove(g)

class SlideCollection:
    """A class representing a slide collection.
```

```
"""

def __init__(self, canvas, paperSize, slides=[]):
    "Initialising, adding slides if provided."

    self.canvas = canvas

    self.pageNum = 1
    self.col = 0
    self.row = 0

    # margins: top, bottom, left, right
    self.frame = (.75*cm, .75*cm, 1.25*cm, 1.25*cm)

    # overall page width and height
    self.pageSize = paperSize

    # Add intial slides if provided.
    for slide in slides:
        self.addNewSlide(slide)

def _getNextPos(self, slide):
    "Where should the next image be placed?"

    # Set some variables to ease typing.
    tm, bm, lm, rm = self.frame
    pw, ph = self.pageSize
    s = slide
    aw, ah = s.aw, s.ah
    b, c = s.b, s.c
    col, row = self.col, self.row

    # Wrap around right frame margin, if needed.
    if lm + (col + 1)*(s.aw + s.b) - s.b > pw - rm:
        col = 0
        row = row + 1

    # Wrap around bottom frame margin, if needed.
    if bm + (row + 1)*(s.ah + s.b) - s.b > ph - bm:
        row = 0
        self.canvas.showPage()
        self.pageNum = self.pageNum + 1

    # If the slide larger than paper size, raise exception.
    if s.ah > pw or s.aw > ph:
        details = 'Slide too big to fit on paper.'
        raise ValueError, details

    # If the slide does still not fit, raise exception.
    if s.ah > pw-rm-lm or s.aw > ph-bm-tm:
        details = 'Slide too big to fit within page borders.'
        raise ValueError, details

    # Place the slide on the canvas.
    x = lm + col*(aw + b)
    y = ph - tm - row*(ah + b) - ah

    # Increase 'slide counter'.
    self.col, self.row = col + 1, row

    return x, y

def _bookmarkSlide(self, slide, x, y):
    """Bookmark a slide.

    Add a bookmark to the individual slide, as well as
    to each new row of images and each new page.
    """
```

```
cv = self.canvas
s = slide

# Add page bookmark.
if self.col == 1 and self.row == 0:
    nameTag = 'Page %d' % self.pageNum
    cv.bookmarkPage(nameTag)
    cv.addOutlineEntry(nameTag, nameTag, 0, closed=1)

# Add row bookmark.
if self.col == 1:
    label = 'Row %d' % (self.row + 1)
    tag = 'p:%s row:%s' % (self.pageNum, self.row + 1)
    cv.bookmarkHorizontalAbsolute(tag, y + s.ah)
    cv.addOutlineEntry(label, tag, 1, closed=1)

# Add slide bookmark.
label = os.path.basename(s.imgPath)
tag = "p:%s x:%s y:%s" % (self.pageNum, x, y)
cv.bookmarkHorizontalAbsolute(tag, y + s.ah)
cv.addOutlineEntry(label, tag, 2)

def addNewSlide(self, slide, size=None):
    "Add a new slide to the collection."

    # Add only slides specified by a path or real slides.
    if type(slide) == type(''):
        imgPath = slide
        s = Slide(imgPath, size)
    elif isinstance(slide) and slide.__class__ == Slide:
        s = slide
    else:
        return

    x, y = self._getNextPos(s)
    c = self.canvas
    s.drawFrame(c, x, y)
    s.drawImage(c, x, y)
    s.drawImageNameLabel(c, x, y, imgPath)
    s.drawImageTypeLabel(c, x, y, imgPath)

    self._bookmarkSlide(s, x, y)

def makeIndex(imgFolder, pdfPath=None):
    """Convert image files into one PDF index file.

    If no pdfPath is given a file index.pdf is created
    in the input folder.
    """

    # Create PDF file.
    if not pdfPath:
        pdfPath = os.path.join(imgFolder, 'index.pdf')

    paperSize = (21*cm, 29.5*cm)
    canv = canvas.Canvas(pdfPath, paperSize)
    canv.setPageCompression(1)

    # Create slide collection and fill it with images
    # as found in a specified folder.
    coll = SlideCollection(canv, paperSize)
    files = getImageFileNames(imgFolder)
    files.sort()

    for f in files:
        coll.addNewSlide(f, size=(3.5*cm, 3.5*cm))
```

```
    canv.showPage()
    canv.save()

### Main.

def main():
    "Run with values provided on the command line."

    opts, args = getopt.getopt(sys.argv[1:], 'ho:')
    outFile = None

    for o, a in opts:
        if o == '-h':
            printUsage()
            sys.exit(0)
        elif o == '-o':
            outFile = a

    if len(args) != 1:
        printUsage('Exactly one input directory allowed!')
        sys.exit(0)

    imgFolder = args[0]

    if not os.path.isdir(imgFolder):
        printUsage('Not a valid directory: %s' % imgFolder)
        sys.exit(0)
    else:
        makeIndex(imgFolder, outFile)

if __name__ == '__main__':
    main()
```