Methodological guidelines.

The Easy Editor app P. 2

🚀**STORYLINE:**

A representative from the Ministry of Social Development who is preparing a special software package for elderly people asks the ProTeam specialists for help. The package must include simple, yet helpful, apps for users with poor computer skills. One of these apps must be the photo editor called Easy Editor.

To implement the Easy Editor app, developers will program a comprehensive solution using the PyQt5, os, PIL libraries and modules, as well as their own classes.

⚠ **SUMMARY:**

The lesson goal is to program an image from a list of images, to switch between images, and apply a black and white filter.

During the first half of the lesson, the students shall create the ImageProcessor image processing class and implement the display of the selected image. In the second half of the lesson, the students will supplement the ImageProcessor with the first method to process the image and save the result.

Technical note. All work on the Easy Editor is organized in one task in the VS Code.

💾 **LINKS AND ACCESSORIES:**

❏ Presentation,
❏ Lesson tasks: the Easy Editor app (Visual Studio Code).

🎯 LEARNING RESULTS

After the lesson, students will:	*The result is achieved when students:*
• know what a class, a class instance, properties, and a class method are; • know the Image and PIL modules (classes) and methods of working with them; • create their own class with the suggested fields and methods; • know and use the os module commands for uploading files to the program by the path to a file; • program their own class methods (e.g. for selecting image files).	• have participated in the discussions and asked clarifying questions; • used concepts related to the creation of classes; • used the os module commands to refer to the image according to the full path to a file; • have programmed the displaying of image previews; • have created the ImageProcessor class and implemented a black and white filter; • have answered the teacher's questions during the review stage.

RECOMMENDED LESSON STRUCTURE

Time	Stage	Stage aims
5 min	Storyline. Discussion: Project planning	❏ Remind the storyline task: develop an Easy Editor app. ❏ Allocate the tasks of the current working day to the mind map project. ❏ Mark the tasks on the project work checklist.
10 min	Qualification	❏ Organize qualification confirmation for the developers by topic: ❏ "Creating classes"; ❏ "os module and file names"; ❏ "The Image class and its basic methods".
15 min	Brainstorming: "ImageProcessor class"	❏ Arrive to the idea of creating the ImageProcessor class as a means of storing information about the current image and the processor of photos. ❏ Demonstrate the settings of the constructor and methods for uploading and displaying images. ❏ Implement a handler function for displaying a preview of an image when its name is selected in the list widget.
25 min	Platform: "VSC: The Easy Editor app"	❏ Have the students complete task 3 in the exercise "VSC. The Easy Editor app".
5 min	Break	❏ Do a warm-up or change activities.
10 min	Brainstorming: "ImageProcessor class"	❏ Using the method of overlaying the black-and-white filter as an example, discuss processing photos using the ImageProcessor class. ❏ Demonstrate how the methods of image processing and saving the result work. ❏ Implement a handler function that applies a black and white filter to the image.
20 min	Platform: "VSC: The Easy Editor app"	❏ Have the students complete task 4 in the exercise "VSC. The Easy Editor app".
5 min	Wrapping up the lesson. Reflection	❏ Conduct a technical interview based on the brainstorming material. ❏ Announce the content of the next workday.

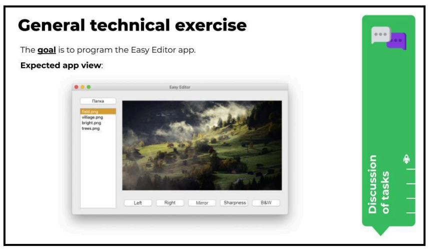
Storyline. Discussion: Project planning

(5 min.)

Open the presentation. The developers do not need their computers yet.

“Hello, colleagues! Today we are continuing the development of the Easy Editor app for photo editing. Last time we planned the project tasks and created a mind map and checklist. Let’s mark the tasks that have been implemented and highlight the tasks for today".

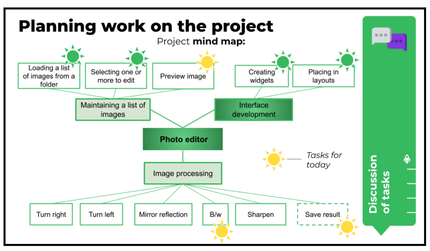
It may seem that today’s workload is much smaller than the set of tasks from the previous day. Note that the task of displaying image previews is more complicated than it seems, as the same set of functions will show both the original images (when switching between file names in the widget) and their processed copies (when pressing the Left, Right, Sharpness and B/W buttons).



General technical exercise

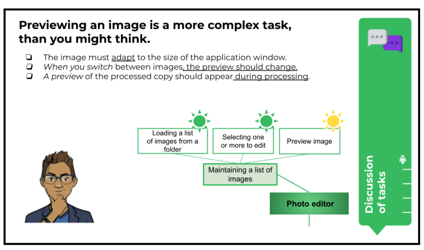
The **goal** is to program the Easy Editor app.

Expected app view:



Planning work on the project

Project **mind map:**



Previewing an image is a more complex task, than you might think.

- The image must adapt to the size of the application window.
- *When you switch* between images, the preview should change.
- *A preview* of the processed copy should appear during processing.

Loading a list of images from a folder
Selecting one or more to edit
Preview image
Maintaining a list of images
Photo editor
Discussion of tasks

Take note that if the developers manage to meet the deadline, their apps will be able to apply a black and white filter to photos as early as today.

Show the tasks to be implemented during the current workday. Then formulate its goal and content.

Qualification

(10 min)

Use the presentation to organize confirmation of the developers’ qualifications before they start working. This time it is on the topics of working with images using PIL and the os module.

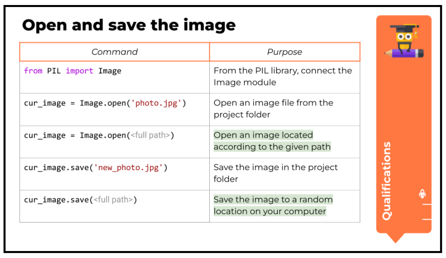
Procedural comment. In the first session of the module, only the images in the project folder were looked at. They did not require knowledge of the full path to the folder or the image itself. As Easy Editor works with a random folder on your computer, images must be accessed by the full path. Therefore, at the practice stage, it can be noted that the open() method works with both the short path to the file (if it locates in the project folder) and the full path (if it is placed in another folder).

There may be students who are not familiar with the computer’s directory hierarchy. In this case, select a folder on your computer and demonstrate the relationship between its location on the system and its full path. Write the full path on the board and refer to this example if you have any future difficulties.

How do we open an image and create an Image type object?

How do we save an image?

Qualifications



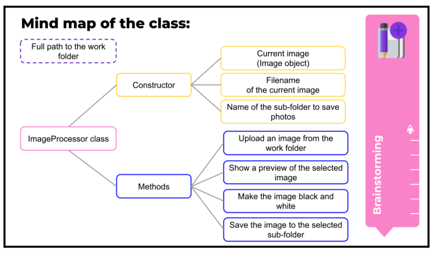
Open and save the image

Command	*Purpose*
`from PIL import Image`	From the PIL library, connect the Image module
`cur_image = Image.open('photo.jpg')`	Open an image file from the project folder
`cur_image = Image.open(<full path>)`	Open an image located according to the given path
`cur_image.save('new_photo.jpg')`	Save the image in the project folder
`cur_image.save(<full path>)`	Save the image to a random location on your computer

Brainstorming: “The ImageProcessor class”

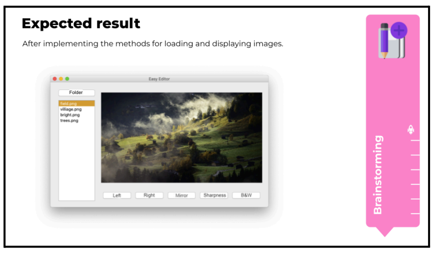
(15 min)

With this stage, we start working on the ImageProcessor class. It will combine both the current image data and methods for loading, displaying, processing, and saving the image. Demonstrate the general layout of the class and move on to a discussion of displaying a preview of the image in the application.



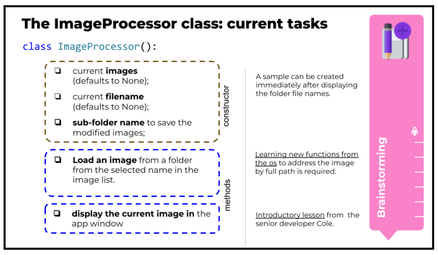
Mind map of the class:

Full path to the work folder
ImageProcessor class
Constructor
Current image (Image object)
Filename of the current image
Name of the sub-folder to save photos
Methods
Upload an image from the work folder
Show a preview of the selected image
Make the image black and white
Save the image to the selected sub-folder
Brainstorming



Expected result

After implementing the methods for loading and displaying images.



The ImageProcessor class: current tasks

```
class ImageProcessor():
```

constructor:
- current **images** (defaults to None);
- current **filename** (defaults to None);
- **sub-folder name** to save the modified images;

methods:
- **Load an image** from a folder from the selected name in the image list.
- **display the current image in** the app window

A sample can be created immediately after displaying the folder file names.

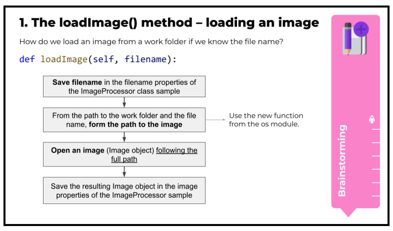
Learning new functions from the os to address the image by full path is required.

Introductory lesson from the senior developer Cole.

Use the diagram to understand the structure of the loadImage() method, which loads images into the program. To load an Image object (and save it in the appropriate field of the ImageProcessor sample) you must generate the full path to the file. This is easily done by knowing the path to the work folder (global variable workdir) and the file name. Retrieving a file name from a list widget is explained below.

Enter the new function of the path section os module — join(). Show how it works in a diagram. Then map the diagrams to the reference code of the loadImage() method.

Technical comment. Optionally, you can make the workdir variable a field of the ImageProcessor class in the next lesson. In the basic version of the program, workdir will remain a global variable.

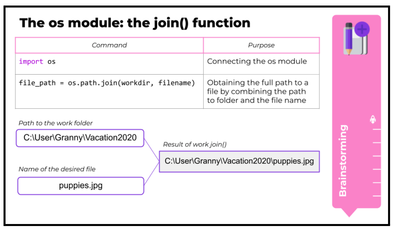


1. The loadImage() method – loading an image

How do we load an image from a work folder if we know the file name?

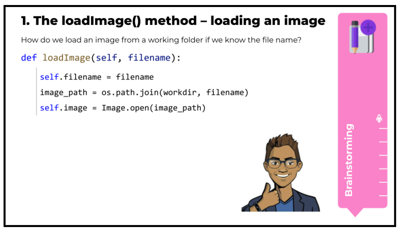
```
def loadImage(self, filename):
```

Save filename in the filename properties of the ImageProcessor class sample
From the path to the work folder and the file name, **form the path to the image**
Use the new function from the os module.
Open an image (Image object) following the full path
Save the resulting Image object in the image properties of the ImageProcessor sample
Brainstorming



The os module: the join() function

Command	*Purpose*
`import os`	Connecting the os module
`file_path = os.path.join(workdir, filename)`	Obtaining the full path to a file by combining the path to folder and the file name



1. The loadImage() method – loading an image

How do we load an image from a working folder if we know the file name?

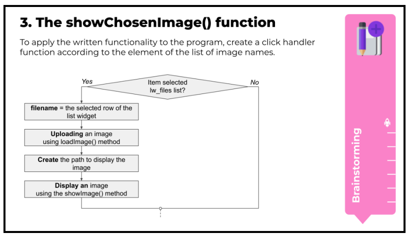
```
def loadImage(self, filename):
    self.filename = filename
    image_path = os.path.join(workdir, filename)
    self.image = Image.open(image_path)
```

Brainstorming

Present the work of the following showImage() method in the form of a masterclass from the developer Cole. To display an image properly in the application interface, you need to introduce a QPixmap object. The problems regarding the QPixmap class are outside the scope of this lesson. However, it can be said that with QPixmap it is possible to “insert” a random sized image into the application window.

To apply the written functionality to the program, create a click handler function according to the element of the list of image names showChosenImage(). The methods for working with the list widget are already familiar to developers from the Smart Notes project.

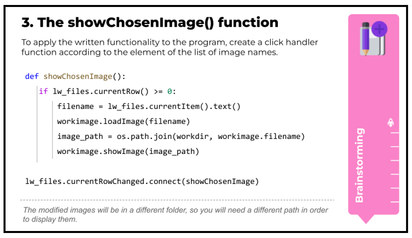
Describe the operation of the function using a flowchart and show the reference code. Then demonstrate how the described functionality can be implemented in the project.



3. The showChosenImage() function

To apply the written functionality to the program, create a click handler function according to the element of the list of image names.

Item selected lw_files list?
Yes
No
filename = the selected row of the list widget
Uploading an image using loadImage() method
Create the path to display the image
Display an image using the showImage() method
Brainstorming



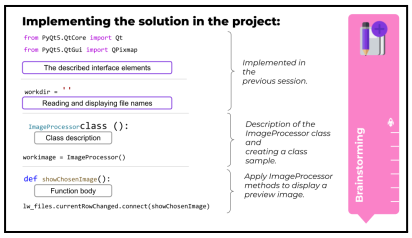
3. The showChosenImage() function

To apply the written functionality to the program, create a click handler function according to the element of the list of image names.

```
def showChosenImage():
    if lw_files.currentRow() >= 0:
        filename = lw_files.currentItem().text()
        workimage.loadImage(filename)
        image_path = os.path.join(workdir, workimage.filename)
        workimage.showImage(image_path)

lw_files.currentRowChanged.connect(showChosenImage)
```

The modified images will be in a different folder, so you will need a different path in order to display them.

Brainstorming



Implementing the solution in the project:

```
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QPixmap
[The described interface elements]

workdir = ''
[Reading and displaying file names]

ImageProcessorclass ():
    [Class description]

workimage = ImageProcessor()

def showChosenImage():
    [Function body]
lw_files.currentRowChanged.connect(showChosenImage)
```

Implemented in the previous session.

Description of the ImageProcessor class and creating a class sample.

Apply ImageProcessor methods to display a preview image.

Wrap up the discussion, formulate the task of the next stage, and start working on the VS Code.

Platform: “VSC. The Easy Editor app”

(20 min)

Arrange the work in the VS Code for the third task of the Easy Editor project. Remind the developers that all the technical documentation and tips on the interface development are available on the platform, and they can view any forgotten stuff there.

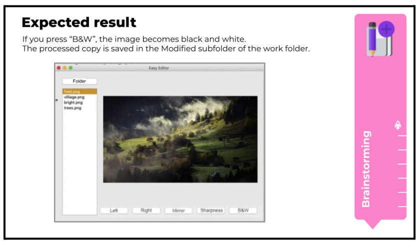
A link to the archive with the solution to task 3 is available at the end of the methodological guidelines.

Brainstorming: “The ImageProcessor class”

(10 min)

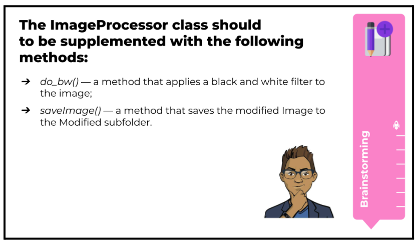
Move on to programming the process of image processing. Remind the developers that they programmed the ImageEditor class when studying image processing with PIL. Processing in ImageProcessor will be handled in the same way, but with the addition of a method that saves the edited copy to the Modified subfolder of the work folder.

Therefore, the saveImage() method, the do_bw() method (overlaying a black and white filter) must be programmed. The clicking on the "B/W" button will also be handled by the do_bw() method.



Expected result

If you press “B&W”, the image becomes black and white.
The processed copy is saved in the Modified subfolder of the work folder.



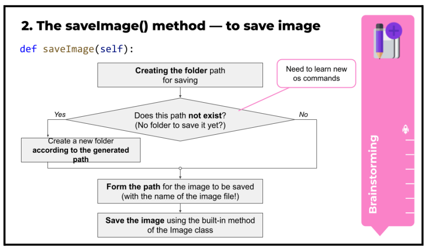
The ImageProcessor class should to be supplemented with the following methods:

- *do_bw()* — a method that applies a black and white filter to the image;
- *saveImage()* — a method that saves the modified Image to the Modified subfolder.

Brainstorming

Take turns sorting out the design of the new functions and methods.

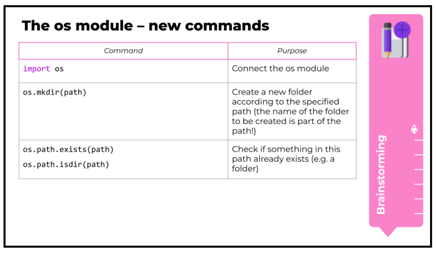
Pay particular attention to the saveImage() method and the new os module commands. Remind the students that according to the terms of reference, the images must be saved to the Modified subfolder of the selected working directory. If this folder does not exist, it must be created. The names of the files to be saved must be the same as the original.



2. The saveImage() method — to save image

```
def saveImage(self):
```

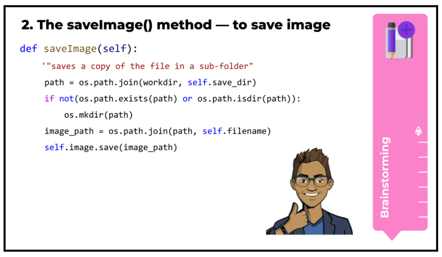
Creating the folder path for saving
Need to learn new os commands
Does this path not exist? (No folder to save it yet?)
Yes
No
Create a new folder according to the generated path
Form the path for the image to be saved (with the name of the image file!)
Save the image using the built-in method of the Image class
Brainstorming



The os module – new commands

Command	Purpose
`import os`	Connect the os module
`os.mkdir(path)`	Create a new folder according to the specified path (the name of the folder to be created is part of the path!)
`os.path.exists(path)` `os.path.isdir(path)`	Check if something in this path already exists (e.g. a folder)

Brainstorming

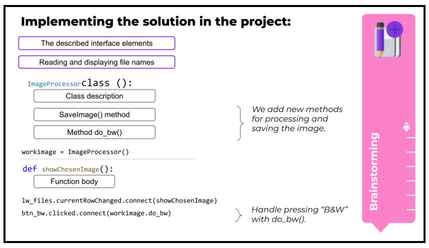


2. The saveImage() method — to save image

```
def saveImage(self):
    '"saves a copy of the file in a sub-folder"
    path = os.path.join(workdir, self.save_dir)
    if not(os.path.exists(path) or os.path.isdir(path)):
        os.mkdir(path)
    image_path = os.path.join(path, self.filename)
    self.image.save(image_path)
```

Brainstorming

Demonstrate how to complete the project code with the written methods, formulate the objectives of the next stage, and move on to programming.



Implementing the solution in the project:

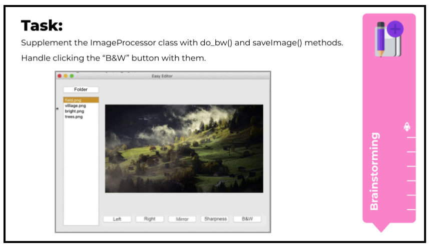
The described interface elements
Reading and displaying file names

```
ImageProcessorclass ():
    Class description
    SaveImage() method
    Method do_bw()

workimage = ImageProcessor()

def showChosenImage():
    Function body

lw_files.currentRowChanged.connect(showChosenImage)
btn_bw.clicked.connect(workimage.do_bw)
```

We add new methods for processing and saving the image.

Handle pressing "B&W" with do_bw().

Brainstorming



Task:

Supplement the ImageProcessor class with do_bw() and saveImage() methods.

Handle clicking the "B&W" button with them.

Platform: "VSC. The Easy Editor app"

(20 min)

Arrange the work in the VS Code for the fourth task of the Easy Editor project.

A link to the archive with the solution to task 4 is available at the end of the methodological guidelines.

Wrapping up the lesson

(5 min.)

Have the developers turn away from their computers, then organize a technical interview about the brainstorming material. Propose theoretical documentation to the developers.

Answers to exercises

Exercise “VSC. The Easy Editor app”

Link to the archive with task 3.

```
import os
from PyQt5.QtWidgets import (
   QApplication, QWidget,
   QFileDialog, # Dialogue for opening files (and folders)
   QLabel, QPushButton, QListWidget,
   QHBoxLayout, QVBoxLayout
)

from PyQt5.QtCore import Qt # needs a Qt.KeepAspectRatio constant to resize while maintaining proportions
from PyQt5.QtGui import QPixmap # screen-optimised

from PIL import Image

app = QApplication([])
win = QWidget()
win.resize(700, 500)
win.setWindowTitle('Easy Editor')
lb_image = QLabel("Image")
btn_dir = QPushButton("Folder")
lw_files = QListWidget()

btn_left = QPushButton("Left")
btn_right = QPushButton("Right")
btn_flip = QPushButton("Mirror")
btn_sharp = QPushButton("Sharpness")
btn_bw = QPushButton("B/W")

row = QHBoxLayout()          # Main line
col1 = QVBoxLayout()         # divided into two columns
col2 = QVBoxLayout()
col1.addWidget(btn_dir)      # in the first - directory selection button
col1.addWidget(lw_files)     # and file list
col2.addWidget(lb_image, 95) # in the second - image
row_tools = QHBoxLayout()    # and button bar
row_tools.addWidget(btn_left)
row_tools.addWidget(btn_right)
row_tools.addWidget(btn_flip)
row_tools.addWidget(btn_sharp)
row_tools.addWidget(btn_bw)
col2.addLayout(row_tools)
```

```
row.addLayout(col1, 20)
row.addLayout(col2, 80)
win.setLayout(row)

win.show()

workdir = ''

def filter(files, extensions):
   result = []
   for filename in files:
       for ext in extensions:
           if filename.endswith(ext):
               result.append(filename)
   return result

def chooseWorkdir():
   global workdir
   workdir = QFileDialog.getExistingDirectory()

def showFilenamesList():
   extensions = ['.jpg','.jpeg', '.png', '.gif', '.bmp']
   chooseWorkdir()
   filenames = filter(os.listdir(workdir), extensions)

   lw_files.clear()
   for filename in filenames:
       lw_files.addItem(filename)

btn_dir.clicked.connect(showFilenamesList)

class ImageProcessor():
   def __init__(self):
       self.image = None
       self.dir = None
       self.filename = None
       self.save_dir = "Modified/"

   def loadImage(self, dir, filename):
       '''When loading, remember the path and file name'''
       self.dir = dir
       self.filename = filename
       image_path = os.path.join(dir, filename)
       self.image = Image.open(image_path)

   def showImage(self, path):
       lb_image.hide()
```

```
        pixmapimage = QPixmap(path)
        w, h = lb_image.width(), lb_image.height()
        pixmapimage = pixmapimage.scaled(w, h, Qt.KeepAspectRatio)
        lb_image.setPixmap(pixmapimage)
        lb_image.show()

workimage = ImageProcessor()

def showChosenImage():
   if lw_files.currentRow() >= 0:
       filename = lw_files.currentItem().text()
       workimage.loadImage(workdir, filename)
       image_path = os.path.join(workimage.dir, workimage.filename)
       workimage.showImage(image_path)

lw_files.currentRowChanged.connect(showChosenImage)

app.exec()
```

[Link to the archive with the solution to task 4.]

```
import os
from PyQt5.QtWidgets import (
   QApplication, QWidget,
   QFileDialog, # Dialogue for opening files (and folders)
   QLabel, QPushButton, QListWidget,
   QHBoxLayout, QVBoxLayout
)
from PyQt5.QtCore import Qt # needs a Qt.KeepAspectRatio constant to resize while maintaining proportions
from PyQt5.QtGui import QPixmap # screen-optimised

from PIL import Image

app = QApplication([])
win = QWidget()
win.resize(700, 500)
win.setWindowTitle('Easy Editor')
lb_image = QLabel("Image")
btn_dir = QPushButton("Folder")
lw_files = QListWidget()

btn_left = QPushButton("Left")
btn_right = QPushButton("Right")
btn_flip = QPushButton("Mirror")
btn_sharp = QPushButton("Sharpness")
```

```
btn_bw = QPushButton("B/W")

row = QHBoxLayout()          # Main line
col1 = QVBoxLayout()         # divided into two columns
col2 = QVBoxLayout()
col1.addWidget(btn_dir)      # in the first - directory selection button
col1.addWidget(lw_files)     # and file list
col2.addWidget(lb_image, 95) # in the second - image
row_tools = QHBoxLayout()    # and button bar
row_tools.addWidget(btn_left)
row_tools.addWidget(btn_right)
row_tools.addWidget(btn_flip)
row_tools.addWidget(btn_sharp)
row_tools.addWidget(btn_bw)
col2.addLayout(row_tools)

row.addLayout(col1, 20)
row.addLayout(col2, 80)
win.setLayout(row)

win.show()

workdir = ''

def filter(files, extensions):
   result = []
   for filename in files:
       for ext in extensions:
           if filename.endswith(ext):
               result.append(filename)
   return result

def chooseWorkdir():
   global workdir
   workdir = QFileDialog.getExistingDirectory()

def showFilenamesList():
   extensions = ['.jpg','.jpeg', '.png', '.gif', '.bmp']
   chooseWorkdir()
   filenames = filter(os.listdir(workdir), extensions)

   lw_files.clear()
   for filename in filenames:
       lw_files.addItem(filename)

btn_dir.clicked.connect(showFilenamesList)
```

```
class ImageProcessor():
   def __init__(self):
       self.image = None
       self.dir = None
       self.filename = None
       self.save_dir = "Modified/"

   def loadImage(self, dir, filename):
       '''When loading, remember the path and file name'''
       self.dir = dir
       self.filename = filename
       image_path = os.path.join(dir, filename)
       self.image = Image.open(image_path)

   def do_bw(self):
       self.image = self.image.convert("L")
       self.saveImage()
       image_path = os.path.join(self.dir, self.save_dir, self.filename)
       self.showImage(image_path)

   def saveImage(self):
       '"saves a copy of the file in a sub-folder"
       path = os.path.join(self.dir, self.save_dir)
       if not(os.path.exists(path) or os.path.isdir(path)):
           os.mkdir(path)
       image_path = os.path.join(path, self.filename)
       self.image.save(image_path)

   def showImage(self, path):
       lb_image.hide()
       pixmapimage = QPixmap(path)
       w, h = lb_image.width(), lb_image.height()
       pixmapimage = pixmapimage.scaled(w, h, Qt.KeepAspectRatio)
       lb_image.setPixmap(pixmapimage)
       lb_image.show()

def showChosenImage():
   if lw_files.currentRow() >= 0:
       filename = lw_files.currentItem().text()
       workimage.loadImage(workdir, filename)
       image_path = os.path.join(workimage.dir, workimage.filename)
       workimage.showImage(image_path)

workimage = ImageProcessor() #current workimage for work
lw_files.currentRowChanged.connect(showChosenImage)

btn_bw.clicked.connect(workimage.do_bw)
```

```
app.exec()
```