**To configure Python SDK for the current project**

* 1. Open the [project settings](https://www.jetbrains.com/pycharm/webhelp/accessing-project-settings.html), and click [Project Interpreter](https://www.jetbrains.com/pycharm/webhelp/project-interpreter.html) page.
  2. In the **Projects pane**, click the desired project.
  3. For the selected project, choose SDK from the list of available Python interpreters and virtual environments.  
     This list includes:
  + Python interpreters, which reside in standard locations.
  + Virtual environments, which reside under the project folder, or under the folder specified as an environment variable WORKON\_HOME.
  + Other Python interpreters, installed locally or remotely.
  + If the desired interpreter is not in the list, click the **Configure interpreters** link, and in the [Python Interpreters](https://www.jetbrains.com/pycharm/webhelp/python-interpreters.html) page configure the desired interpreter as described in the section [Configuring Available Python Interpreters](https://www.jetbrains.com/pycharm/webhelp/configuring-available-python-interpreters.html).
  1. Apply changes.

Pasted from <<https://www.jetbrains.com/pycharm/webhelp/configuring-python-interpreter-for-a-project.html>>

Install OpenCV-Python in Windows

Goals

In this tutorial

* + We will learn to setup OpenCV-Python in your Windows system.

*Below steps are tested in a Windows 7-64 bit machine with Visual Studio 2010 and Visual Studio 2012. The screenshots shows VS2012.*

Installing OpenCV from prebuilt binaries

* + Below Python packages are to be downloaded and installed to their default locations.  
    1.1. [Python-2.7.x](http://python.org/ftp/python/2.7.5/python-2.7.5.msi).  
    1.2. [Numpy](http://sourceforge.net/projects/numpy/files/NumPy/1.7.1/numpy-1.7.1-win32-superpack-python2.7.exe/download).  
    1.3. [Matplotlib](https://downloads.sourceforge.net/project/matplotlib/matplotlib/matplotlib-1.3.0/matplotlib-1.3.0.win32-py2.7.exe) (*Matplotlib is optional, but recommended since we use it a lot in our tutorials*).
  + Install all packages into their default locations. Python will be installed to **C:/Python27/**.
  + After installation, open Python IDLE. Enter import numpy and make sure Numpy is working fine.
  1. Download latest OpenCV release from [sourceforge site](http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.6/OpenCV-2.4.6.0.exe/download) and double-click to extract it.
  2. Goto **opencv/build/python/2.7** folder.
  3. Copy **cv2.pyd** to **C:/Python27/lib/site-packeges**.
  4. Open Python IDLE and type following codes in Python terminal.  
     **>>> import** **cv2**  
     **>>> print** cv2.\_\_version\_\_

If the results are printed out without any errors, congratulations !!! You have installed OpenCV-Python successfully.

Building OpenCV from source

* 1. Download and install Visual Studio and CMake.  
     1.1. [Visual Studio 2012](http://go.microsoft.com/?linkid=9816768)  
     1.2. [CMake](http://www.cmake.org/files/v2.8/cmake-2.8.11.2-win32-x86.exe)
  2. Download and install necessary Python packages to their default locations  
     2.1. [Python 2.7.x](http://python.org/ftp/python/2.7.5/python-2.7.5.msi)  
     2.2. [Numpy](http://sourceforge.net/projects/numpy/files/NumPy/1.7.1/numpy-1.7.1-win32-superpack-python2.7.exe/download)  
     2.3. [Matplotlib](https://downloads.sourceforge.net/project/matplotlib/matplotlib/matplotlib-1.3.0/matplotlib-1.3.0.win32-py2.7.exe) (*Matplotlib is optional, but recommended since we use it a lot in our tutorials.*)

**Note**

In this case, we are using 32-bit binaries of Python packages. But if you want to use OpenCV for x64, 64-bit binaries of Python packages are to be installed. Problem is that, there is no official 64-bit binaries of Numpy. You have to build it on your own. For that, you have to use the same compiler used to build Python. When you start Python IDLE, it shows the compiler details. You can get more [information here](http://stackoverflow.com/q/2676763/1134940). So your system must have the same Visual Studio version and build Numpy from source.

**Note**

Another method to have 64-bit Python packages is to use ready-made Python distributions from third-parties like [Anaconda](http://www.continuum.io/downloads), [Enthought](https://www.enthought.com/downloads/) etc. It will be bigger in size, but will have everything you need. Everything in a single shell. You can also download 32-bit versions also.

* 1. Make sure Python and Numpy are working fine.
  2. Download OpenCV source. It can be from [Sourceforge](http://sourceforge.net/projects/opencvlibrary/) (for official release version) or from [Github](https://github.com/Itseez/opencv) (for latest source).
  3. Extract it to a folder, opencv and create a new folder build in it.
  4. Open CMake-gui (*Start > All Programs > CMake-gui*)
  5. Fill the fields as follows (see the image below):  
     7.1. Click on **Browse Source...** and locate the opencv folder.  
     7.2. Click on **Browse Build...** and locate the build folder we created.  
     7.3. Click on **Configure**.  
     7.4. It will open a new window to select the compiler. Choose appropriate compiler (here, Visual Studio 11) and click **Finish**.  
     7.5. Wait until analysis is finished.
  6. You will see all the fields are marked in red. Click on the **WITH** field to expand it. It decides what extra features you need. So mark appropriate fields. See the below image:
  7. Now click on **BUILD** field to expand it. First few fields configure the build method. See the below image:
  8. Remaining fields specify what modules are to be built. Since GPU modules are not yet supported by OpenCV-Python, you can completely avoid it to save time (But if you work with them, keep it there). See the image below:
  9. Now click on **ENABLE** field to expand it. Make sure **ENABLE\_SOLUTION\_FOLDERS** is unchecked (Solution folders are not supported by Visual Studio Express edition). See the image below:
  10. Also make sure that in the **PYTHON** field, everything is filled. (Ignore PYTHON\_DEBUG\_LIBRARY). See image below:
  11. Finally click the **Generate** button.
  12. Now go to our **opencv/build** folder. There you will find **OpenCV.sln** file. Open it with Visual Studio.
  13. Check build mode as **Release** instead of **Debug**.
  14. In the solution explorer, right-click on the **Solution** (or **ALL\_BUILD**) and build it. It will take some time to finish.
  15. Again, right-click on **INSTALL** and build it. Now OpenCV-Python will be installed.
  16. Open Python IDLE and enter import cv2. If no error, it is installed correctly.

**Note**

We have installed with no other support like TBB, Eigen, Qt, Documentation etc. It would be difficult to explain it here. A more detailed video will be added soon or you can just hack around.

Additional Resources

Exercises

* 1. If you have a windows machine, compile the OpenCV from source. Do all kinds of hacks. If you meet any problem, visit OpenCV forum and explain your problem

Pasted from <<http://docs.opencv.org/trunk/doc/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html>>

Install OpenCV-Python in Windows

Goals

In this tutorial

* 1. We will learn to setup OpenCV-Python in your Windows system.

*Below steps are tested in a Windows 7-64 bit machine with Visual Studio 2010 and Visual Studio 2012. The screenshots shows VS2012.*

Installing OpenCV from prebuilt binaries

* 1. Below Python packages are to be downloaded and installed to their default locations.  
     1.1. [Python-2.7.x](http://python.org/ftp/python/2.7.5/python-2.7.5.msi).  
     1.2. [Numpy](http://sourceforge.net/projects/numpy/files/NumPy/1.7.1/numpy-1.7.1-win32-superpack-python2.7.exe/download).  
     1.3. [Matplotlib](https://downloads.sourceforge.net/project/matplotlib/matplotlib/matplotlib-1.3.0/matplotlib-1.3.0.win32-py2.7.exe) (*Matplotlib is optional, but recommended since we use it a lot in our tutorials*).
  2. Install all packages into their default locations. Python will be installed to **C:/Python27/**.
  3. After installation, open Python IDLE. Enter import numpy and make sure Numpy is working fine.
  4. Download latest OpenCV release from [sourceforge site](http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.4.6/OpenCV-2.4.6.0.exe/download) and double-click to extract it.
  5. Goto **opencv/build/python/2.7** folder.
  6. Copy **cv2.pyd** to **C:/Python27/lib/site-packeges**.
  7. Open Python IDLE and type following codes in Python terminal.  
     **>>> import** **cv2**  
     **>>> print** cv2.\_\_version\_\_

If the results are printed out without any errors, congratulations !!! You have installed OpenCV-Python successfully.

Pasted from <<http://docs.opencv.org/trunk/doc/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html>>

**Search for Quick start and Hello world to start quickly**

Two versions of unipad => exe one is simpler

Zip one needs execution of the .py code

First set environment variable

System environment variable in path

System properties => folder python add it to envrionment>path

Execution of any code: pyton.exe unipad.py -n

(put in the properties of the file in order to execute it)

**Start simple then go to more complicated ones**

Execut is play button

Py27 folder in c:

Execute and debug

Debug with wxpyton to install it

If pychen is slow => move to unipad

Cvl 2.. Function of package

Pychen > preferenc or setting

Tutorial is available

Command line

Pyton consolue:

Exit() => exit

Identify characters with:

* Tesseact-ocr (location as well) => objects
* OCR: op command line perl
* Python wrapper class to use tesseact-ocr

<http://sourceforge.net/projects/wxpython/files/wxPython/3.0.0.0/wxPython3.0-win64-3.0.0.0-py27.exe/download?use_mirror=superb-dca2>

<https://code.google.com/p/ulipad/>

[**Unofficial Windows Binaries for Python Extension Packages**](http://www.lfd.uci.edu/~gohlke/pythonlibs/)

Pasted from <<http://www.lfd.uci.edu/~gohlke/pythonlibs/>>

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

 have this issue where I try to import cv2 on python and get the following error message.

>>> import cv2  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
ImportError: DLL load failed: %1 is not a valid Win32 application.

I do understand there are many posts about this where it is suggested that the bitness of the package is different from the python package.

However, everything I am running is 64 bits. I am on win7 64 bits, I have the [winpython](http://code.google.com/p/winpython/downloads/detail?name=WinPython-64bit-2.7.3.3.exe) 2.7.3.3, 64 bits distribution, and I compiled opencv in 64 bits with the instruction provided [here](http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html) and placed the cv2.pyd dll in the Lib/site-packages folder of python.

Unfortunately the suggestion of using the 32 bits version of python isn't working for me anymore as I have to handle numpy arrays too large for 32 bits.

Thanks!!!

------ UPDATE

The only thing missing was to add the new opencv binaries path (C:\opencv\build\bin\Release) to the Windows PATH environment variable, restart python.

Everything seems to be working fine now!

Pasted from <<http://stackoverflow.com/questions/14629818/importerror-dll-load-failed-1-is-not-a-valid-win32-application>>

* Tesseract-ocr

C:\Program Files (x86)\Tesseract-OCR

<https://code.google.com/p/tesseract-ocr/>

Work in shell

<http://www.win.tue.nl/~aeb/linux/ocr/tesseract.html>

<http://blog.ayoungprogrammer.com/2012/11/tutorial-installing-tesseract-ocr-30202.html>

<http://misteroleg.wordpress.com/2012/12/19/ocr-using-tesseract-and-imagemagick-as-pre-processing-task/>

**Windows version compiled by VS2008 is available now!**

*remember to*

*1. set PATH: e.g. PATH=%PATH%;C:\PYTHON27*[*Details*](http://pythoncentral.org/how-to-install-python-2-7-on-windows-7-python-is-not-recognized-as-an-internal-or-external-command/)

*2. set c:\python27\python.exe to be compatible to* ***Windows 7*** even though you are using windows 7. Otherwise the program might **crash** during runtime [Details](http://python-tesseract.googlecode.com/files/pythonCompatible.png)

3. *Download and install all of them*

[python-tesseract-win32](https://python-tesseract.googlecode.com/files/python-tesseract_0.8-1.5_i386.deb) [python-opencv](http://www.lfd.uci.edu/~gohlke/pythonlibs/v92kqh5j/opencv-python-2.4.3.win32-py2.7.exe) [numpy](http://www.lfd.uci.edu/~gohlke/pythonlibs/v92kqh5j/numpy-MKL-1.7.0rc1.win32-py2.7.exe)

4. unzip the sample code and keep your fingers crossed [Sample Codes](http://python-tesseract.googlecode.com/files/test-slim.7z)

5. python -u test.py

it is always safer to run python in unbuffered mode especially for windows XP

Pasted from <<https://code.google.com/p/python-tesseract/>>

<http://search.cpan.org/~leocharre/Image-OCR-Tesseract-1.24/lib/Image/OCR/Tesseract.pod>

<http://stackoverflow.com/questions/12278982/get-the-exact-position-of-text-from-image-in-tesseract>

Extracing Frames:

<http://ffmpeg.zeranoe.com/builds/>

**How do I interact with files in python?**

Python comes with libraries that allow your programs to interact with files in your computer. This document covers part of the [os module](http://docs.python.org/2/library/os.html).

**File Systems**

Your computer drive is organized in a hierarchical structure of files and directories.

* **files** -- These contain information. Examples include be csv files, or python files.
* **directories** -- These contain files and directories *inside* of them

Your filesystem starts from a [root directory](http://en.wikipedia.org/wiki/Root_directory), notated by a forward slash / on Unux and by a drive letter C:/ on Windows.

**Absolute and Relative file paths**

**Absolute** file paths are notated by a **leading forward slash or drive label**. For example, /home/example\_user/example\_directory or C:/system32/cmd.exe. An absolute file path describes how to access a given file or directory, starting from the root of the file system. A file path is also called a *pathname*.

**Relative** file paths are notated by a **lack of a leading forward slash**. For example, example\_directory. A relative file path is interpreted from the perspective your current working directory. If you use a relative file path from the wrong directory, then the path will refer to a different file than you intend, or it will refer to no file at all.

In a sense, **whenever you use a relative file path, it is joined with your current directory to create an absolute file path.** That is, if my current working directory is /home/example\_user and I use a relative file path of example\_directory/example\_python\_program, then that is equivalent to using tho absolute file path /home/example\_user/example\_directory/example\_file\_program.

In the following example usage of a Unix command-line shell, the current working directory is initially /home/example\_user/example\_directory. There is a program called example\_python\_program, which prints "this is an example python program". At first, the program can be referenced by the relative file path example\_python\_program. After the directory is changed to /home/example\_user, the relative file path to access the program becomes example\_directory/example\_python\_program. Please note that the $ symbolizes a prompt where the user is allowed to type.

$ pwd  
/home/example\_user/example\_directory  
$ ls  
example\_python\_program.py  
$ python example\_python\_program.py  
this is an example python program  
$ example\_directory cd ..  
$ pwd  
/home/example\_user  
$ python example\_python\_program.py  
python: can't open file 'example\_python\_program.py': [Errno 2] No such file or directory  
$ python example\_directory/example\_python\_program.py  
this is an example python program

**os.getcwd**

When you run a python program, its current working directory is initialized to whatever your current working directory was when you ran the program. It can be retrieved using the function [os.getcwd](http://docs.python.org/2/library/os.html#os.getcwd). Consider the following program, cwd\_printer.py.

import os  
print "The current working directory is", os.getcwd()

The following example usage of the command-line shell illustrates how Python's current working directory is set.

$ pwd  
/home/example\_user/example\_directory  
$ ls  
cwd\_printer.py  
$ python cwd\_printer.py  
The current working directory is /home/example\_user/example\_directory  
$ cd ..  
$ pwd  
/home/example\_user  
$ python example\_directory/cwd\_printer.py  
The current working directory is /home/example\_user

**os.listdir and os.path.join**

The [os.listdir](http://docs.python.org/2/library/os.html#os.listdir) function takes one argument: an absolute or relative pathname, which should refer to a directory. The function returns a list of relative pathnames (strings) of all files/directories inside of the given directory name.

These strings should be used relative to os.listdir's argument. Consider the following situation:

* The current working directory is /home/example\_user.
* There is a directory inside of /home/example\_user called data. That is, this folder's absolute path is /home/example\_user/data, and its relative path is data
* Inside of data is a file called people.txt.

The following example usage shows a wrong and a right way to open people.txt

$ pwd  
/home/example\_user  
$ python  
Python 2.7.3 (default, Sep 26 2012, 21:51:14)   
[GCC 4.7.2] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import os  
>>> filenames = os.listdir("data")  
>>> filenames  
['people.txt']  
>>> file = open(filenames[0])  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
IOError: [Errno 2] No such file or directory: 'people.txt'  
>>> pathname = os.path.join("data", filenames[0])  
>>> pathname  
'data/people.txt'  
>>> open(pathname)  
<open file 'data/people.txt', mode 'r' at 0x7f74ad5d8270>  
>>>

Notice that when the first attempt to open the file was made, python reported that no file exists. Consider that we are trying to open the file with the absolute path of /home/example\_user/data/people.txt. When we try to use the relative path of people.txt (recall that this is interpreted as relative because it does not have a leading /), our computer will combine the relative path with the current working directory, which in this case is/home/example\_user. This generates the incorrect absolute path of /home/example\_user/people.txt

Instead, we can combine data and people.txt using [os.path.join](http://docs.python.org/2/library/os.path.html#os.path.join) to generate the relative path data/people.txt. When we try to use this new relative path, our computer generates the absolute path/home/example\_user/data/people.txt, which is what we want.

**Do not combine paths using string concatenation (+) or anything other than**[**os.path.join**](http://docs.python.org/2/library/os.path.html#os.path.join)**.** Different computers represent paths in different ways. In particular, Windows uses \ as a directory separator in pathnames, while Unix (Mac and Linux) machines use / as a directory separator in pathnames. Your code will not handle pathnames as elegantly and correctly as [os.path.join](http://docs.python.org/2/library/os.path.html#os.path.join) will.

Pasted from <<http://courses.cs.washington.edu/courses/cse140/13wi/file-interaction.html>>

Install different binary files of Python:

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

Install any library that Python gives me error for it

ImShow:

<http://matplotlib.org/1.3.1/users/image_tutorial.html>

Basic Image processing:

<http://pythonvision.org/basic-tutorial>

Sources I used to decompose into frames and save it:

<http://stackoverflow.com/questions/10959141/converting-numpy-array-having-image-data-to-cvmat>

<http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html>

<http://stackoverflow.com/questions/17119544/save-multiple-images-with-opencv-and-python>

<http://www.lfd.uci.edu/~gohlke/pythonlibs/#mahotas>

<http://stackoverflow.com/questions/14452824/saving-image-in-python-pil>

<http://zulko.github.io/blog/2013/09/27/read-and-write-video-frames-in-python-using-ffmpeg/>

\_\_author\_\_ = 'MHE'

FFMPEG\_BIN = "C:/Users/MHE/Desktop/ActiveCourses/Projects/ImageProcessing/ffmpeg-20140320-git-19139d8-win64-shared/ffmpeg-20140320-git-19139d8-win64-shared/bin/ffmpeg.exe"

import subprocess as sp

pipe = sp.Popen([FFMPEG\_BIN,"-i","C:/Users/MHE/Desktop/ActiveCourses/Projects/ImageProcessing/tempad/tv-ads02.mpg",

"-f","image2pipe",

"-pix\_fmt","rgb24",

"-vcodec","rawvideo","-"],

stdin=sp.PIPE,stdout=sp.PIPE)

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import numpy as np

import scipy

import pylab

#import pymorph

import mahotas

from scipy import ndimage

import sys

from PIL import Image

raw\_image = pipe.stdout.read(420\*360\*3) # read 420\*360\*3 bytes (=1 frame)

image=np.fromstring(raw\_image, dtype='uint8').reshape((360,420,3))

#img=mpimg.imread(image)

#imgplot = plt.imshow(image)

dna = mahotas.imread('C:/Users/MHE/Desktop/ActiveCourses/Projects/ImageProcessing/test/firstimage.jpg')

pylab.imshow(dna)

import numpy as np

import cv2

import cv2.cv as cv

import time

cap = cv2.VideoCapture('C:/Users/MHE/Desktop/ActiveCourses/Projects/ImageProcessing/tempad/tv-ads02.mpg')

# Define the codec and create VideoWriter object

#fourcc = cv2.VideoWriter\_fourcc('M','J','P','G')

#out = cv2.VideoWriter('C:/Users/MHE/Desktop/ActiveCourses/Projects/ImageProcessing/tempad/output.avi',fourcc, 20.0, (640,480))

i =0;

while(cap.isOpened()):

ret, frame = cap.read()

if ret==True:

i= i+1

#frame = cv2.flip(frame,0)

cv.SaveImage('C:/Users/MHE/Desktop/ActiveCourses/Projects/ImageProcessing/test/out'+str(i)+'.jpg',cv.fromarray(frame)) ;

cv2.imshow('frame',frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

else:

break

# Release everything if job is finished

cap.release()

#out.release()

cv2.destroyAllWindows()

#imshow( image )

**Getting Started with Videos**

**Goal**

* + Learn to read video, display video and save video.
  + Learn to capture from Camera and display it.
  + You will learn these functions : **cv2.VideoCapture()**, **cv2.VideoWriter()**

**Capture Video from Camera**

Often, we have to capture live stream with camera. OpenCV provides a very simple interface to this. Let’s capture a video from the camera (I am using the in-built webcam of my laptop), convert it into grayscale video and display it. Just a simple task to get started.

To capture a video, you need to create a **VideoCapture** object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Normally one camera will be connected (as in my case). So I simply pass 0 (or -1). You can select the second camera by passing 1 and so on. After that, you can capture frame-by-frame. But at the end, don’t forget to release the capture.

**import** numpy **as** np  
**import** cv2

cap **=** cv2**.**VideoCapture(0)

**while**(True):  
 *# Capture frame-by-frame*  
 ret, frame **=** cap**.**read()

*# Our operations on the frame come here*  
 gray **=** cv2**.**cvtColor(frame, cv2**.**COLOR\_BGR2GRAY)

*# Display the resulting frame*  
 cv2**.**imshow('frame',gray)  
 **if** cv2**.**waitKey(1) **&** 0xFF **==** ord('q'):  
 **break**

*# When everything done, release the capture*  
cap**.**release()  
cv2**.**destroyAllWindows()

cap.read() returns a bool (True/False). If frame is read correctly, it will be True. So you can check end of the video by checking this return value.

Sometimes, cap may not have initialized the capture. In that case, this code shows error. You can check whether it is initialized or not by the method **cap.isOpened()**. If it is True, OK. Otherwise open it using **cap.open()**.

You can also access some of the features of this video using **cap.get(propId)** method where propId is a number from 0 to 18. Each number denotes a property of the video (if it is applicable to that video) and full details can be seen here: [Property Identifier](http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html#videocapture-get). Some of these values can be modified using**cap.set(propId, value)**. Value is the new value you want.

For example, I can check the frame width and height by cap.get(3) and cap.get(4). It gives me 640x480 by default. But I want to modify it to 320x240. Just use ret = cap.set(3,320) andret = cap.set(4,240).

**Note**

If you are getting error, make sure camera is working fine using any other camera application (like Cheese in Linux).

**Playing Video from file**

It is same as capturing from Camera, just change camera index with video file name. Also while displaying the frame, use appropriate time for cv2.waitKey(). If it is too less, video will be very fast and if it is too high, video will be slow (Well, that is how you can display videos in slow motion). 25 milliseconds will be OK in normal cases.

**import** numpy **as** np  
**import** cv2

cap **=** cv2**.**VideoCapture('vtest.avi')

**while**(cap**.**isOpened()):  
 ret, frame **=** cap**.**read()

gray **=** cv2**.**cvtColor(frame, cv2**.**COLOR\_BGR2GRAY)

cv2**.**imshow('frame',gray)  
 **if** cv2**.**waitKey(1) **&** 0xFF **==** ord('q'):  
 **break**

cap**.**release()  
cv2**.**destroyAllWindows()

**Note**

Make sure proper versions of ffmpeg or gstreamer is installed. Sometimes, it is a headache to work with Video Capture mostly due to wrong installation of ffmpeg/gstreamer.

**Saving a Video**

So we capture a video, process it frame-by-frame and we want to save that video. For images, it is very simple, just use cv2.imwrite(). Here a little more work is required.

This time we create a **VideoWriter** object. We should specify the output file name (eg: output.avi). Then we should specify the **FourCC** code (details in next paragraph). Then number of frames per second (fps) and frame size should be passed. And last one is **isColor** flag. If it is True, encoder expect color frame, otherwise it works with grayscale frame.

[FourCC](http://en.wikipedia.org/wiki/FourCC) is a 4-byte code used to specify the video codec. The list of available codes can be found in[fourcc.org](http://www.fourcc.org/codecs.php). It is platform dependent. Following codecs works fine for me.

* + In Fedora: DIVX, XVID, MJPG, X264, WMV1, WMV2. (XVID is more preferable. MJPG results in high size video. X264 gives very small size video)
  + In Windows: DIVX (More to be tested and added)
  + In OSX : *(I don’t have access to OSX. Can some one fill this?)*

FourCC code is passed as cv2.VideoWriter\_fourcc('M','J','P','G') orcv2.VideoWriter\_fourcc(\*'MJPG) for MJPG.

Below code capture from a Camera, flip every frame in vertical direction and saves it.

**import** numpy **as** np  
**import** cv2

cap **=** cv2**.**VideoCapture(0)

*# Define the codec and create VideoWriter object*  
fourcc **=** cv2**.**VideoWriter\_fourcc(**\***'XVID')  
out **=** cv2**.**VideoWriter('output.avi',fourcc, 20.0, (640,480))

**while**(cap**.**isOpened()):  
 ret, frame **=** cap**.**read()  
 **if** ret**==**True:  
 frame **=** cv2**.**flip(frame,0)

*# write the flipped frame*  
 out**.**write(frame)

cv2**.**imshow('frame',frame)  
 **if** cv2**.**waitKey(1) **&** 0xFF **==** ord('q'):  
 **break**  
 **else**:  
 **break**

*# Release everything if job is finished*  
cap**.**release()  
out**.**release()  
cv2**.**destroyAllWindows()

Pasted from <<http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html>>

**The Best Python IDEs You Can Use for Development**

Published: Monday 30th January 2012

Last Updated: Tuesday 17th September 2013

[Share on facebook](http://www.pythoncentral.io/the-best-python-ides-you-can-use-for-development/)[Share on google\_plusone\_share](http://www.addthis.com/bookmark.php?v=300&winname=addthis&pub=ra-4ddccf8c3564f5c8&source=tbx32-300&lng=en-US&s=google_plusone_share&url=http%3A%2F%2Fwww.pythoncentral.io%2Fthe-best-python-ides-you-can-use-for-development%2F&title=The%20Best%20Python%20IDEs%20You%20Can%20Use%20for%20Development%20%7C%20Python%20Central&ate=AT-ra-4ddccf8c3564f5c8/-/-/53397c3c1b53534b/2&frommenu=1&uid=53397c3c5b80911d&ufbl=1&ct=1&pre=https%3A%2F%2Fwww.google.com%2F&tt=0&captcha_provider=nucaptcha)[Share on twitter](http://www.pythoncentral.io/the-best-python-ides-you-can-use-for-development/)[Share on stumbleupon](http://www.addthis.com/bookmark.php?v=300&winname=addthis&pub=ra-4ddccf8c3564f5c8&source=tbx32-300&lng=en-US&s=stumbleupon&url=http%3A%2F%2Fwww.pythoncentral.io%2Fthe-best-python-ides-you-can-use-for-development%2F&title=The%20Best%20Python%20IDEs%20You%20Can%20Use%20for%20Development%20%7C%20Python%20Central&ate=AT-ra-4ddccf8c3564f5c8/-/-/53397c3c1b53534b/3&frommenu=1&uid=53397c3c54fd5208&ufbl=1&ct=1&pre=https%3A%2F%2Fwww.google.com%2F&tt=0&captcha_provider=nucaptcha)[Share on printShare on email](http://www.pythoncentral.io/the-best-python-ides-you-can-use-for-development/)

This article was written by Sergio Tapia Gutierrez. If you're after another opinion by Jason Fruit, checkout our other IDE article [Comparison of Python IDEs for Development](http://www.pythoncentral.io/comparison-of-python-ides-development/). We also have a [Review of Python's Best Text Editors](http://www.pythoncentral.io/best-text-editors-for-python-development/).

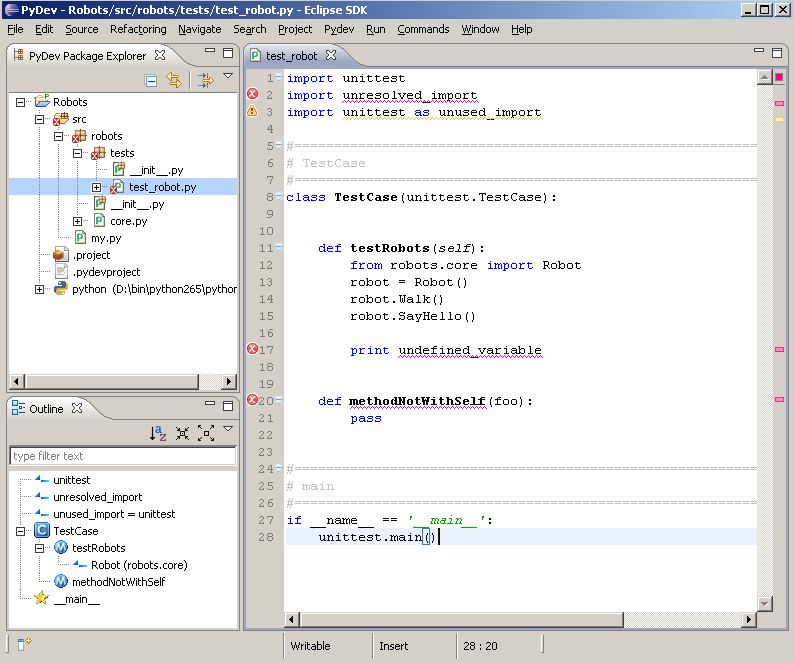
An IDE (*Integrated Development Environment*) is one of the best tools a programmer can wield. It allows developers to work efficiently and forget about the boilerplate.

While some programmers scoff at the idea of using anything more than a text editor, when you are working on a very big project consisting of many files, an IDE will have features that will make your life as a developer much easier.

Some features you can expect a good IDE to have are:

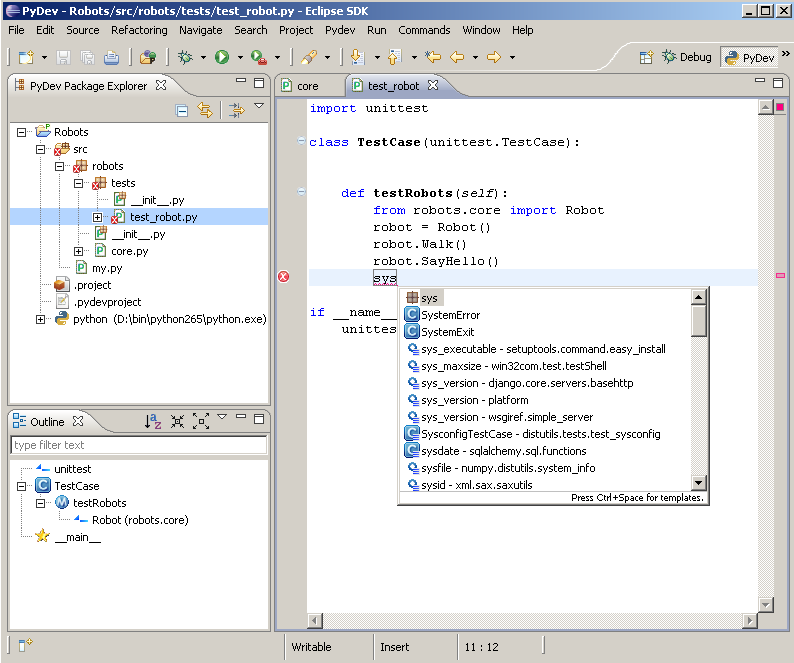
* Code completion
* Syntax highlighting
* Templates for common code
* Source Control support (eg. *Subversion, Mercurial or Git*)

Let's see which Python IDE's are the best and how they stack up against each other.



Eclipse with PyDev

Eclipse with PyDev



Eclipse with PyDev

**PyDev's Official Site:**[**http://pydev.org/**](http://pydev.org/)

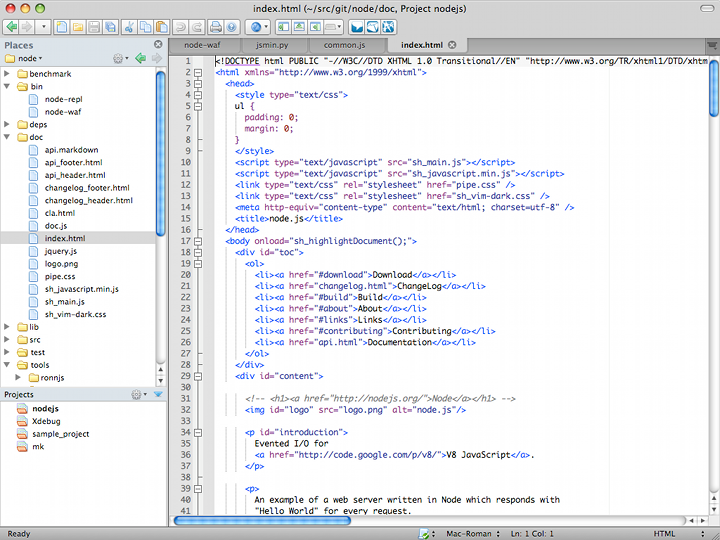
Eclipse is a very versatile IDE that's been around for a very long time. It's a time tested offering and is very solid all around. Eclipse is like a sandbox IDE; it can support any language as long as somebody has baked in the support through a package. Such is the case with PyDev, a package that allows you to turn Eclipse into a very useful Python IDE.

It's a completely free IDE that offer a wide array of features such as:

* Django integration
* Code completion
* Code completion with auto import
* Syntax highlighting
* Code analysis
* Go to definition
* Refactoring
* Mark occurrences
* Debugger
* Remote Debugger
* Tokens browser
* Interactive browser
* Unit test integration
* Code coverage
* and many many more

I use this when coding in Python on my Windows machine, it just works with minimal configuration.

Komodo Edit



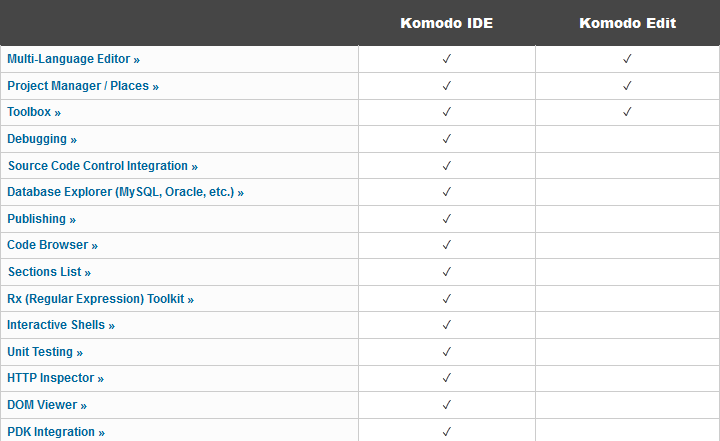
Komodo Edit - Available on Windows, Linux and Mac.

**Komodo Edit's Official site:**[**http://www.activestate.com/komodo-edit**](http://www.activestate.com/komodo-edit)

Komodo Edit is a very clean, professional Python IDE. It doesn't have fluff and instead focuses on putting the thing you need right in front of you. No digging through random submenus looking for an option. It's code completion is very good and fast; it pops up as you type with minimal loading time.

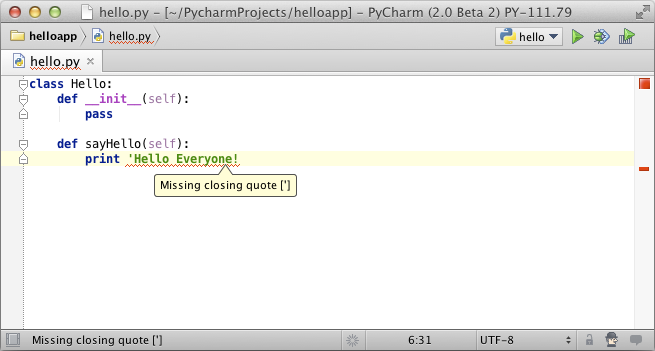
ActiveState offers a commercial version of their IDE, called [Komodo IDE](http://www.activestate.com/komodo-ide).

The differences between the two version are as follows:



Komodo Edit vs. Komodo IDE

PyCharm



PyCharm by JetBrains

**PyCharm's Official Site:**[**http://www.jetbrains.com/pycharm/**](http://www.jetbrains.com/pycharm/)

PyCharm is an IDE created by [JetBrains](http://www.jetbrains.com/). You might remember these guys as the authors of [ReSharper](http://www.jetbrains.com/resharper/), one of the best investments a .NET developer can make. Well PyCharm is no exception, and continuing with their outstanding pedigree, JetBrains has released another excellent tool to the developer ecosystem.

Said to have the absolute best code completion technology, this one is worth at least a trial on your part.

* Cross Platfom
* Commercial
* Automatic Code-completion
* Integrated Python Debugging
* Error Markup
* Source Control integration
* Smart Indent
* Bracket Matching
* Line Numbering
* Code Folding
* Unit Testing

Pasted from <<http://www.pythoncentral.io/the-best-python-ides-you-can-use-for-development/>>

Converting Image to Grayscale with Python + OpenCV

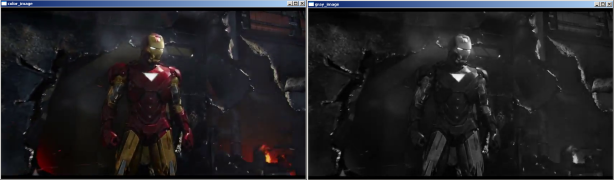
Posted: September 23, 2012 in [**Python**](http://extr3metech.wordpress.com/category/programming/python/)

Tags: [**convert image to grayscale**](http://extr3metech.wordpress.com/tag/convert-image-to-grayscale/), [**opencv**](http://extr3metech.wordpress.com/tag/opencv/), [**python**](http://extr3metech.wordpress.com/tag/python-2/), [**tutorial**](http://extr3metech.wordpress.com/tag/tutorial/)

[**3**](http://extr3metech.wordpress.com/2012/09/23/convert-photo-to-grayscale-with-python-opencv/#comments)

Hey guys, been reading OpenCV for python and thought of posting a tutorial on Programming a Grayscale Image Convertor. I encourage you to google them , there are lots and lots of examples and code snippets. This is on how to a convert any image to gray scale using Python and OpenCV.

A sample input image and output image are shown below (**YEah, I am big Iron Man Fan! :B**). You can click on image to enlarge:

[](http://extr3metech.files.wordpress.com/2012/09/pic.png)

I have placed an image named “ironman.png” in the current working directory (i.e. I have the original image in the same directory as the  place where I have saved my Python code).

First we **import**the **cv2**module:

|  |  |
| --- | --- |
| 1 | **import** cv2 |

Then ,**read** the image **to a variable** named “**image**” :

|  |  |
| --- | --- |
| 1 | image **=** cv2.imread('ironman.png') |

Now, to convert to gray-scale image and store it to another variable named “**gray\_image**” use the function**cv2.cvtColor()** with parameters as  the “**image**” variable and  “**cv2.COLOR\_BGR2GRAY**” :

|  |  |
| --- | --- |
| 1 | gray\_image **=** cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) |

So, now the variable “gray\_image” will hold the gray-scale version of the input image.

Now, **to** write/**save** the **converted gray-scale image to the hard disk**, we **use** the function “**cv2.imwrite()**” with parameters as “the name of converted image” and the variable “gray\_image” to which the converted image was stored:

|  |  |
| --- | --- |
| 1 | cv2.imwrite('gray\_image.png',gray\_image) |

So, now if you open the directory where you saved your python code, you can see a new image there : gray\_image.png! Wow! :D

Now, **to display** the original and the gray-scale ,we use function “**cv2.imshow()**” with parameters as the “window title” and the “image variable” :

|  |  |
| --- | --- |
| 1  2 | cv2.imshow('color\_image',image)  cv2.imshow('gray\_image',gray\_image) |

**Complete Code for GrayScale Image convertor**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # GrayScale Image Convertor  # [http://extr3metech.wordpress.com](http://extr3metech.wordpress.com/)    **import** cv2  image **=** cv2.imread('ironman.png')  gray\_image **=** cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  cv2.imwrite('gray\_image.png',gray\_image)  cv2.imshow('color\_image',image)  cv2.imshow('gray\_image',gray\_image)  cv2.waitKey(0)                 # Waits forever for user to press any key  cv2.destroyAllWindows()        # Closes displayed windows    #End of Code |

Hope you enjoyed this post, feel free to comment and don’t forget to follow my blog! :D

**ΞXΤЯ3МΞ**

Pasted from <<http://extr3metech.wordpress.com/2012/09/23/convert-photo-to-grayscale-with-python-opencv/>>

Python useful Scripts:

you can also do this:

section = "C\_type"  
new\_section = "Sec\_%s" % section

This allows you not only append, but also insert wherever in the string:

section = "C\_type"  
new\_section = "Sec\_%s\_blah" % section

Pasted from <<http://stackoverflow.com/questions/2711579/concatenate-strings-in-python-2-4>>

|  |  |
| --- | --- |
| up vote10down voteaccepted | Obviously the file is too large to be read into memory all at once.  Why not just use:  with open("data.txt") as myfile:  for line in myfile:  do\_something(line.rstrip("\n"))  or, if you're not on Python 2.6 and higher:  myfile = open("data.txt") for line in myfile:  do\_something(line.rstrip("\n"))  In both cases, you'll get an iterator that can be treated much like a list of strings.  EDIT: Since your way of reading the entire file into one large string and then splitting it on newlines will remove the newlines in the process, I have added a .rstrip("\n") to my examples in order to better simulate the result. |

Pasted from <<http://stackoverflow.com/questions/2396238/memory-error-due-to-the-huge-input-file-size>>

Try the [rstrip](http://docs.python.org/2/library/stdtypes.html#str.rstrip) method.

>>> 'test string\n'.rstrip()  
'test string'

Note that Python's rstrip method strips *all* kinds of whitespace by default, not just newlines as Perl does with chomp. To strip only newlines:

>>> 'test string \n'.rstrip('\n')  
'test string '

There is also the lstrip and strip methods.

>>> s = " \n abc def "  
>>> s.strip()  
'abc def'  
>>> s.rstrip()  
' \n abc def'  
>>> s.lstrip()  
'abc def '  
>>>

Pasted from <<http://stackoverflow.com/questions/275018/how-can-i-remove-chomp-a-newline-in-python>>

Reading and Writing Files in Python

Overview

In Python, you don't need to import any library to read and write files.

The first step is to get a file object.

The way to do this is to use the open function.

File Types

A file is usually categorized as either text or binary.

A text file is often structured as a sequence of lines and a line is a sequence  
of characters.

The line is terminated by a EOL (End Of Line) character.

The most common line terminator is the \n , or the newline character.

The backslash character indicates that the next character will be treated as a  
newline.

A binary file is basically any file that is not a text file. Binary files can  
only be processed by application that know about the file's structure.

Open ( )

To open a file for writing use the built-i open() function. open() returns a  
file object, and is most commonly used with two arguments.

The syntax is:  
file\_object = open(filename, mode) where file\_object is the variable to put the  
file object.

The second argument describes the way in which the file will be used.

Mode

The mode argument is optional; 'r' will be assumed if it’s omitted.

**The modes can be:**

'r' when the file will only be read

'w' for only writing (an existing file with the same name will be erased)

'a' opens the file for appending; any data written to the file is automatically  
added to the end.

'r+' opens the file for both reading and writing.

>>> f = open('workfile', 'w')  
>>> print f

Next the file objects functions can be called. The two most common functions are  
read and write.

Create a text file

Let's first create a new text file. You can name it anything you like,  
in this example we will name it "newfile.txt".

file = open("newfile.txt", "w")

file.write("hello world in the new file\n")

file.write("and another line\n")

file.close()

If we now look in the newfile.txt, we can see the text that we wrote:

$ cat newfile.txt   
hello world in the new file  
and another line

How to read a text file

To read a file, we can use different methods.

file.read( )

If you want to return a string containing all characters in the file, you can  
use file.read().

file = open('newfile.txt', 'r')

print file.read()

**Output:**

hello world in the new file  
and another line

We can also specify how many characters the string should return, by using  
file.read(n), where "n" determines number of characters.

This reads the first 5 characters of data and returns it as a string.

file = open('newfile.txt', 'r')  
print file.read(5)

**Output:**

hello

file.readline( )

The readline() function will read from a file line by line (rather than pulling  
the entire file in at once).

Use readline() when you want to get the first line of the file, subsequent calls  
to readline() will return successive lines.

Basically, it will read a single line from the file and return a string  
containing characters up to \n.

file = open('newfile.txt', 'r')

print file.readline():

**Output:**

hello world in the new file

file.readlines( )

readlines() returns the complete ?le as a list of strings each separated by \n

file = open('newfile.txt', 'r')

print file.readlines()

**Output:**

['hello world in the new file\n', 'and another line\n']

Looping over a file object

For reading lines from a file, you can loop over the file object.

This is memory efficient, fast, and leads to simple code.

file = open('newfile.txt', 'r')

for line in file:  
 print line,

Output:

hello world in the new file  
and another line

file.write( )

The write method takes one parameter, which is the string to be written.

To start a new line after writing the data, add a \n character to the end.

file = open("newfile.txt", "w")

file.write("This is a test\n")

file.write("And here is another line\n")

file.close()

Close ( )

When you’re done with a file, call f.close() to close it and free up any system  
resources taken up by the open file.

After calling f.close(), attempts to use the file object will automatically fail.

File Handling Usages

Let's show some example on how to use the different file methods

To open a text file, use:

**fh = open("hello.txt", "r")**

To read a text file, use:

**fh = open("hello.txt","r")  
print fh.read()**

To read one line at a time, use:

**fh = open("hello".txt", "r")  
print fh.readline()**

To read a list of lines use:

**fh = open("hello.txt.", "r")  
print fh.readlines()**

To write to a file, use:

**fh = open("hello.txt","w")  
fh.write("Hello World")  
fh.close()**

To write to a file, use:

**fh = open("hello.txt", "w")  
lines\_of\_text = ["a line of text", "another line of text", "a third line"]  
fh.writelines(lines\_of\_text)  
fh.close()**

To append to file, use:

**fh = open("Hello.txt", "a")  
fh.write("Hello World again")  
fh.close**

To close a file, use

**fh = open("hello.txt", "r")  
print fh.read()  
fh.close()**

With Statement

Another way of working with file objects is the With statement.

It is good practice to use this statement.

With the "With" statement, you get better syntax and exceptions handling.

In addition, it will automatically close the file. The with statement provides a  
way for ensuring that a clean-up is always used.

Opening a file using with is as simple as:

**with open(filename) as file:**

Let's take a look at some examples

**with open("newtext.txt") as file:        *# Use file to refer to the file object*  
 data = file.read()  
 do something with data**

You can of course also loop over the file object:

with open("newfile.txt") as f:  
 for line in f:  
 print line,

Notice, that we didn't have to write "file.close()". That will automatically be  
called.

With Examples

Let's show some examples on how we can use this in our every day programming.

Write to a file using With

Write to a file using the With statement

with open("hello.txt", "w") as f:  
        f.write("Hello World")

Read a file line by line into an list

This will read the file hello.txt and save the content into "data".

with open(hello.txt) as f:  
 data = f.readlines()

Splitting Lines

As a last example, we will show how to split lines from a text file.

The split function in our example, splits the string contained in the variable  
data, whenever it sees a space character.

You can split by whatever you wish, line.split(":") would split the line using  
colons.

with open('data.txt', 'r') as f:  
 data = f.readlines()

for line in data:  
 words = line.split()  
 print words

**Output:**

Because multiple values are returned by split, they are returned as an array.  
['hello', 'world,', 'how', 'are', 'you', 'today?']  
['today', 'is', 'saturday']

More Reading

<http://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files>  
<http://www.pythonforbeginners.com/cheatsheet/python-file-handling/>  
<http://en.wikibooks.org/wiki/Non-Programmer's_Tutorial_for_Python_3/>  
<http://chryswoods.com/beginning_python/>

Pasted from <<http://www.pythonforbeginners.com/files/reading-and-writing-files-in-python>>

7. Input and Output

There are several ways to present the output of a program; data can be printed in a human-readable form, or written to a file for future use. This chapter will discuss some of the possibilities.

7.1. Fancier Output Formatting

So far we’ve encountered two ways of writing values: *expression statements* and the [**print**](https://docs.python.org/2/reference/simple_stmts.html#print) statement. (A third way is using the **write()** method of file objects; the standard output file can be referenced as sys.stdout. See the Library Reference for more information on this.)

Often you’ll want more control over the formatting of your output than simply printing space-separated values. There are two ways to format your output; the first way is to do all the string handling yourself; using string slicing and concatenation operations you can create any layout you can imagine. The string types have some methods that perform useful operations for padding strings to a given column width; these will be discussed shortly. The second way is to use the[**str.format()**](https://docs.python.org/2/library/stdtypes.html#str.format) method.

The [**string**](https://docs.python.org/2/library/string.html#module-string) module contains a [**Template**](https://docs.python.org/2/library/string.html#string.Template) class which offers yet another way to substitute values into strings.

One question remains, of course: how do you convert values to strings? Luckily, Python has ways to convert any value to a string: pass it to the [**repr()**](https://docs.python.org/2/library/repr.html#module-repr) or [**str()**](https://docs.python.org/2/library/functions.html#str)functions.

The [**str()**](https://docs.python.org/2/library/functions.html#str) function is meant to return representations of values which are fairly human-readable, while [**repr()**](https://docs.python.org/2/library/repr.html#module-repr) is meant to generate representations which can be read by the interpreter (or will force a [**SyntaxError**](https://docs.python.org/2/library/exceptions.html#exceptions.SyntaxError) if there is no equivalent syntax). For objects which don’t have a particular representation for human consumption, [**str()**](https://docs.python.org/2/library/functions.html#str) will return the same value as [**repr()**](https://docs.python.org/2/library/repr.html#module-repr). Many values, such as numbers or structures like lists and dictionaries, have the same representation using either function. Strings and floating point numbers, in particular, have two distinct representations.

Some examples:

>>>

**>>>** s = 'Hello, world.'  
**>>>** str(s)  
'Hello, world.'  
**>>>** repr(s)  
"'Hello, world.'"  
**>>>** str(1.0/7.0)  
'0.142857142857'  
**>>>** repr(1.0/7.0)  
'0.14285714285714285'  
**>>>** x = 10 \* 3.25  
**>>>** y = 200 \* 200  
**>>>** s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'  
**>>> print** s  
The value of x is 32.5, and y is 40000...  
**>>>** *# The repr() of a string adds string quotes and backslashes:*  
**...** hello = 'hello, world**\n**'  
**>>>** hellos = repr(hello)  
**>>> print** hellos  
'hello, world\n'  
**>>>** *# The argument to repr() may be any Python object:*  
**...** repr((x, y, ('spam', 'eggs')))  
"(32.5, 40000, ('spam', 'eggs'))"

Here are two ways to write a table of squares and cubes:

>>>

**>>> for** x **in** range(1, 11):  
**...**  **print** repr(x).rjust(2), repr(x\*x).rjust(3),  
**...**  *# Note trailing comma on previous line*  
**...**  **print** repr(x\*x\*x).rjust(4)  
**...**  
 1 1 1  
 2 4 8  
 3 9 27  
 4 16 64  
 5 25 125  
 6 36 216  
 7 49 343  
 8 64 512  
 9 81 729  
10 100 1000

**>>> for** x **in** range(1,11):  
**...**  **print** '{0:2d} {1:3d} {2:4d}'.format(x, x\*x, x\*x\*x)  
**...**  
 1 1 1  
 2 4 8  
 3 9 27  
 4 16 64  
 5 25 125  
 6 36 216  
 7 49 343  
 8 64 512  
 9 81 729  
10 100 1000

(Note that in the first example, one space between each column was added by the way [**print**](https://docs.python.org/2/reference/simple_stmts.html#print) works: it always adds spaces between its arguments.)

This example demonstrates the [**str.rjust()**](https://docs.python.org/2/library/stdtypes.html#str.rjust) method of string objects, which right-justifies a string in a field of a given width by padding it with spaces on the left. There are similar methods [**str.ljust()**](https://docs.python.org/2/library/stdtypes.html#str.ljust) and [**str.center()**](https://docs.python.org/2/library/stdtypes.html#str.center). These methods do not write anything, they just return a new string. If the input string is too long, they don’t truncate it, but return it unchanged; this will mess up your column lay-out but that’s usually better than the alternative, which would be lying about a value. (If you really want truncation you can always add a slice operation, as in x.ljust(n)[:n].)

There is another method, [**str.zfill()**](https://docs.python.org/2/library/stdtypes.html#str.zfill), which pads a numeric string on the left with zeros. It understands about plus and minus signs:

>>>

**>>>** '12'.zfill(5)  
'00012'  
**>>>** '-3.14'.zfill(7)  
'-003.14'  
**>>>** '3.14159265359'.zfill(5)  
'3.14159265359'

Basic usage of the [**str.format()**](https://docs.python.org/2/library/stdtypes.html#str.format) method looks like this:

>>>

**>>> print** 'We are the {} who say "{}!"'.format('knights', 'Ni')  
We are the knights who say "Ni!"

The brackets and characters within them (called format fields) are replaced with the objects passed into the [**str.format()**](https://docs.python.org/2/library/stdtypes.html#str.format) method. A number in the brackets refers to the position of the object passed into the [**str.format()**](https://docs.python.org/2/library/stdtypes.html#str.format) method.

>>>

**>>> print** '{0} and {1}'.format('spam', 'eggs')  
spam and eggs  
**>>> print** '{1} and {0}'.format('spam', 'eggs')  
eggs and spam

If keyword arguments are used in the [**str.format()**](https://docs.python.org/2/library/stdtypes.html#str.format) method, their values are referred to by using the name of the argument.

>>>

**>>> print** 'This {food} is {adjective}.'.format(  
**...**  food='spam', adjective='absolutely horrible')  
This spam is absolutely horrible.

Positional and keyword arguments can be arbitrarily combined:

>>>

**>>> print** 'The story of {0}, {1}, and {other}.'.format('Bill', 'Manfred',  
**...**  other='Georg')  
The story of Bill, Manfred, and Georg.

'!s' (apply [**str()**](https://docs.python.org/2/library/functions.html#str)) and '!r' (apply [**repr()**](https://docs.python.org/2/library/repr.html#module-repr)) can be used to convert the value before it is formatted.

>>>

**>>> import** **math**  
**>>> print** 'The value of PI is approximately {}.'.format(math.pi)  
The value of PI is approximately 3.14159265359.  
**>>> print** 'The value of PI is approximately {!r}.'.format(math.pi)  
The value of PI is approximately 3.141592653589793.

An optional ':' and format specifier can follow the field name. This allows greater control over how the value is formatted. The following example rounds Pi to three places after the decimal.

>>>

**>>> import** **math**  
**>>> print** 'The value of PI is approximately {0:.3f}.'.format(math.pi)  
The value of PI is approximately 3.142.

Passing an integer after the ':' will cause that field to be a minimum number of characters wide. This is useful for making tables pretty.

>>>

**>>>** table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}  
**>>> for** name, phone **in** table.items():  
**...**  **print** '{0:10} ==> {1:10d}'.format(name, phone)  
**...**  
Jack ==> 4098  
Dcab ==> 7678  
Sjoerd ==> 4127

If you have a really long format string that you don’t want to split up, it would be nice if you could reference the variables to be formatted by name instead of by position. This can be done by simply passing the dict and using square brackets '[]' to access the keys

>>>

**>>>** table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}  
**>>> print** ('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '  
**...**  'Dcab: {0[Dcab]:d}'.format(table))  
Jack: 4098; Sjoerd: 4127; Dcab: 8637678

This could also be done by passing the table as keyword arguments with the ‘\*\*’ notation.

>>>

**>>>** table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}  
**>>> print** 'Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab: {Dcab:d}'.format(\*\*table)  
Jack: 4098; Sjoerd: 4127; Dcab: 8637678

This is particularly useful in combination with the built-in function [**vars()**](https://docs.python.org/2/library/functions.html#vars), which returns a dictionary containing all local variables.

For a complete overview of string formatting with [**str.format()**](https://docs.python.org/2/library/stdtypes.html#str.format), see [*Format String Syntax*](https://docs.python.org/2/library/string.html#formatstrings).

7.1.1. Old string formatting

The % operator can also be used for string formatting. It interprets the left argument much like a **sprintf()**-style format string to be applied to the right argument, and returns the string resulting from this formatting operation. For example:

>>>

**>>> import** **math**  
**>>> print** 'The value of PI is approximately *%5.3f*.' % math.pi  
The value of PI is approximately 3.142.

More information can be found in the [*String Formatting Operations*](https://docs.python.org/2/library/stdtypes.html#string-formatting) section.

7.2. Reading and Writing Files

[**open()**](https://docs.python.org/2/library/functions.html#open) returns a file object, and is most commonly used with two arguments: open(filename, mode).

>>>

**>>>** f = open('workfile', 'w')  
**>>> print** f  
<open file 'workfile', mode 'w' at 80a0960>

The first argument is a string containing the filename. The second argument is another string containing a few characters describing the way in which the file will be used. *mode* can be 'r' when the file will only be read, 'w' for only writing (an existing file with the same name will be erased), and 'a' opens the file for appending; any data written to the file is automatically added to the end. 'r+' opens the file for both reading and writing. The *mode* argument is optional; 'r'will be assumed if it’s omitted.

On Windows, 'b' appended to the mode opens the file in binary mode, so there are also modes like 'rb', 'wb', and 'r+b'. Python on Windows makes a distinction between text and binary files; the end-of-line characters in text files are automatically altered slightly when data is read or written. This behind-the-scenes modification to file data is fine for ASCII text files, but it’ll corrupt binary data like that in JPEG or EXE files. Be very careful to use binary mode when reading and writing such files. On Unix, it doesn’t hurt to append a 'b' to the mode, so you can use it platform-independently for all binary files.

7.2.1. Methods of File Objects

The rest of the examples in this section will assume that a file object called f has already been created.

To read a file’s contents, call f.read(size), which reads some quantity of data and returns it as a string. *size* is an optional numeric argument. When *size* is omitted or negative, the entire contents of the file will be read and returned; it’s your problem if the file is twice as large as your machine’s memory. Otherwise, at most *size* bytes are read and returned. If the end of the file has been reached, f.read() will return an empty string ("").

>>>

**>>>** f.read()  
'This is the entire file.\n'  
**>>>** f.read()  
''

f.readline() reads a single line from the file; a newline character (\n) is left at the end of the string, and is only omitted on the last line of the file if the file doesn’t end in a newline. This makes the return value unambiguous; if f.readline() returns an empty string, the end of the file has been reached, while a blank line is represented by '\n', a string containing only a single newline.

>>>

**>>>** f.readline()  
'This is the first line of the file.\n'  
**>>>** f.readline()  
'Second line of the file\n'  
**>>>** f.readline()  
''

For reading lines from a file, you can loop over the file object. This is memory efficient, fast, and leads to simple code:

>>>

**>>> for** line **in** f:  
 print line,

This is the first line of the file.  
Second line of the file

If you want to read all the lines of a file in a list you can also use list(f) or f.readlines().

f.write(string) writes the contents of *string* to the file, returning **None**.

>>>

**>>>** f.write('This is a test**\n**')

To write something other than a string, it needs to be converted to a string first:

>>>

**>>>** value = ('the answer', 42)  
**>>>** s = str(value)  
**>>>** f.write(s)

f.tell() returns an integer giving the file object’s current position in the file, measured in bytes from the beginning of the file. To change the file object’s position, use f.seek(offset, from\_what). The position is computed from adding *offset* to a reference point; the reference point is selected by the *from\_what*argument. A *from\_what* value of 0 measures from the beginning of the file, 1 uses the current file position, and 2 uses the end of the file as the reference point.*from\_what* can be omitted and defaults to 0, using the beginning of the file as the reference point.

>>>

**>>>** f = open('workfile', 'r+')  
**>>>** f.write('0123456789abcdef')  
**>>>** f.seek(5) *# Go to the 6th byte in the file*  
**>>>** f.read(1)  
'5'  
**>>>** f.seek(-3, 2) *# Go to the 3rd byte before the end*  
**>>>** f.read(1)  
'd'

When you’re done with a file, call f.close() to close it and free up any system resources taken up by the open file. After calling f.close(), attempts to use the file object will automatically fail.

>>>

**>>>** f.close()  
**>>>** f.read()  
Traceback (most recent call last):  
 File "<stdin>", line 1, in ?  
**ValueError**: I/O operation on closed file

It is good practice to use the [**with**](https://docs.python.org/2/reference/compound_stmts.html#with) keyword when dealing with file objects. This has the advantage that the file is properly closed after its suite finishes, even if an exception is raised on the way. It is also much shorter than writing equivalent [**try**](https://docs.python.org/2/reference/compound_stmts.html#try)-[**finally**](https://docs.python.org/2/reference/compound_stmts.html#finally) blocks:

>>>

**>>> with** open('workfile', 'r') **as** f:  
**...**  read\_data = f.read()  
**>>>** f.closed  
True

File objects have some additional methods, such as [**isatty()**](https://docs.python.org/2/library/stdtypes.html#file.isatty) and [**truncate()**](https://docs.python.org/2/library/stdtypes.html#file.truncate) which are less frequently used; consult the Library Reference for a complete guide to file objects.

7.2.2. Saving structured data with [**json**](https://docs.python.org/2/library/json.html#module-json)

Strings can easily be written to and read from a file. Numbers take a bit more effort, since the **read()** method only returns strings, which will have to be passed to a function like [**int()**](https://docs.python.org/2/library/functions.html#int), which takes a string like '123' and returns its numeric value 123. When you want to save more complex data types like nested lists and dictionaries, parsing and serializing by hand becomes complicated.

Rather than having users constantly writing and debugging code to save complicated data types to files, Python allows you to use the popular data interchange format called [JSON (JavaScript Object Notation)](http://json.org/). The standard module called [**json**](https://docs.python.org/2/library/json.html#module-json) can take Python data hierarchies, and convert them to string representations; this process is called *serializing*. Reconstructing the data from the string representation is called *deserializing*. Between serializing and deserializing, the string representing the object may have been stored in a file or data, or sent over a network connection to some distant machine.

**Note**

The JSON format is commonly used by modern applications to allow for data exchange. Many programmers are already familiar with it, which makes it a good choice for interoperability.

If you have an object x, you can view its JSON string representation with a simple line of code:

>>>

**>>>** json.dumps([1, 'simple', 'list'])  
'[1, "simple", "list"]'

Another variant of the [**dumps()**](https://docs.python.org/2/library/json.html#json.dumps) function, called [**dump()**](https://docs.python.org/2/library/json.html#json.dump), simply serializes the object to a file. So if f is a [*file object*](https://docs.python.org/2/glossary.html#term-file-object) opened for writing, we can do this:

json.dump(x, f)

To decode the object again, if f is a [*file object*](https://docs.python.org/2/glossary.html#term-file-object) which has been opened for reading:

x = json.load(f)

This simple serialization technique can handle lists and dictionaries, but serializing arbitrary class instances in JSON requires a bit of extra effort. The reference for the [**json**](https://docs.python.org/2/library/json.html#module-json) module contains an explanation of this.

**See also**

[**pickle**](https://docs.python.org/2/library/pickle.html#module-pickle) - the pickle module

Contrary to [*JSON*](https://docs.python.org/2/tutorial/inputoutput.html#tut-json), *pickle* is a protocol which allows the serialization of arbitrarily complex Python objects. As such, it is specific to Python and cannot be used to communicate with applications written in other languages. It is also insecure by default: deserializing pickle data coming from an untrusted source can execute arbitrary code, if the data was crafted by a skilled attacker.

Pasted from <<https://docs.python.org/2/tutorial/inputoutput.html>>