

Computergrafik 2: Übung 1

Eclipse, PyDev, NumPy, Matplotlib

Überblick

1. Einrichten der Entwicklungsumgebung
2. Python-Techniken
3. Bildverarbeitung Numpy und Matplotlib

Python, Numpy, Matplotlib installieren

Windows

- **Python** (Skriptsprache)
 - Version 2.7.3 (**32bit**) herunterladen:
<http://www.python.org/ftp/python/2.7.3/python-2.7.3.msi>
- **Numpy** (Matrix-Erweiterung + Mathematische Library)
 - <http://sourceforge.net/projects/numpy/files/NumPy/1.6.1/numpy-1.6.1-win32-superpack-python2.7.exe/download>
- **Matplotlib** (Erzeugen von Visualisierungen)
 - <http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.1.0/matplotlib-1.1.0.win32-py2.7.exe/download>

Python, Numpy, Matplotlib installieren

Mac OS X

- **Python** (Skriptsprache)
 - <http://python.org/ftp/python/2.7.3/python-2.7.3-macosx10.6.dmg>
- **Numpy** (Matrix-Erweiterung + Mathematische Library)
 - <http://sourceforge.net/projects/numpy/files/NumPy/1.6.1/numpy-1.6.1-py2.7-python.org-macosx10.6.dmg/download>
- **Matplotlib** (Erzeugen von Visualisierungen)
 - <http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.1.0/matplotlib-1.1.0-py2.7-python.org-macosx10.6.dmg/download>
- **Linux:** Über Paketmanager o.g. Pakete installieren

Tutorials

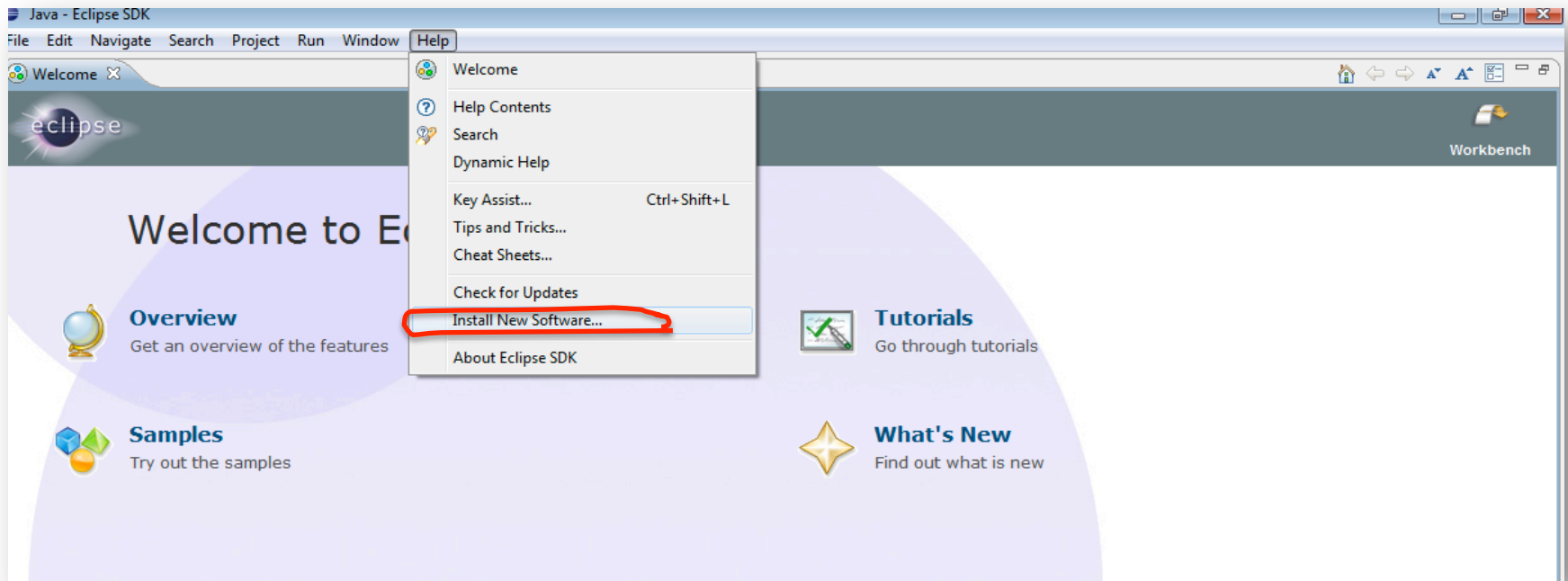
- Wir empfehlen, die folgenden Tutorials durchzuarbeiten
- **Python** (Skriptsprache)
 - <http://docs.python.org/tutorial/>
- **Numpy** (Matrix-Erweiterung + Mathematische Library)
 - http://www.scipy.org/Tentative_NumPy_Tutorial
- **Matplotlib** (Erzeugen von Visualisierungen)
 - http://matplotlib.sourceforge.net/users/pyplot_tutorial.html

Eclipse installieren

- JDK oder JRE installieren, falls nicht vorhanden
 - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
 - Ggf. den Pfad zum Java /bin Verzeichnis dem Windows-Path hinzufügen
- Eclipse
 -  **Eclipse Classic 3.7.2**
 - <http://www.eclipse.org/downloads/packages/eclipse-classic-372/indigosr2>
 - In ein Verzeichnis entpacken
 - z.B. Windows: c:\eclipse

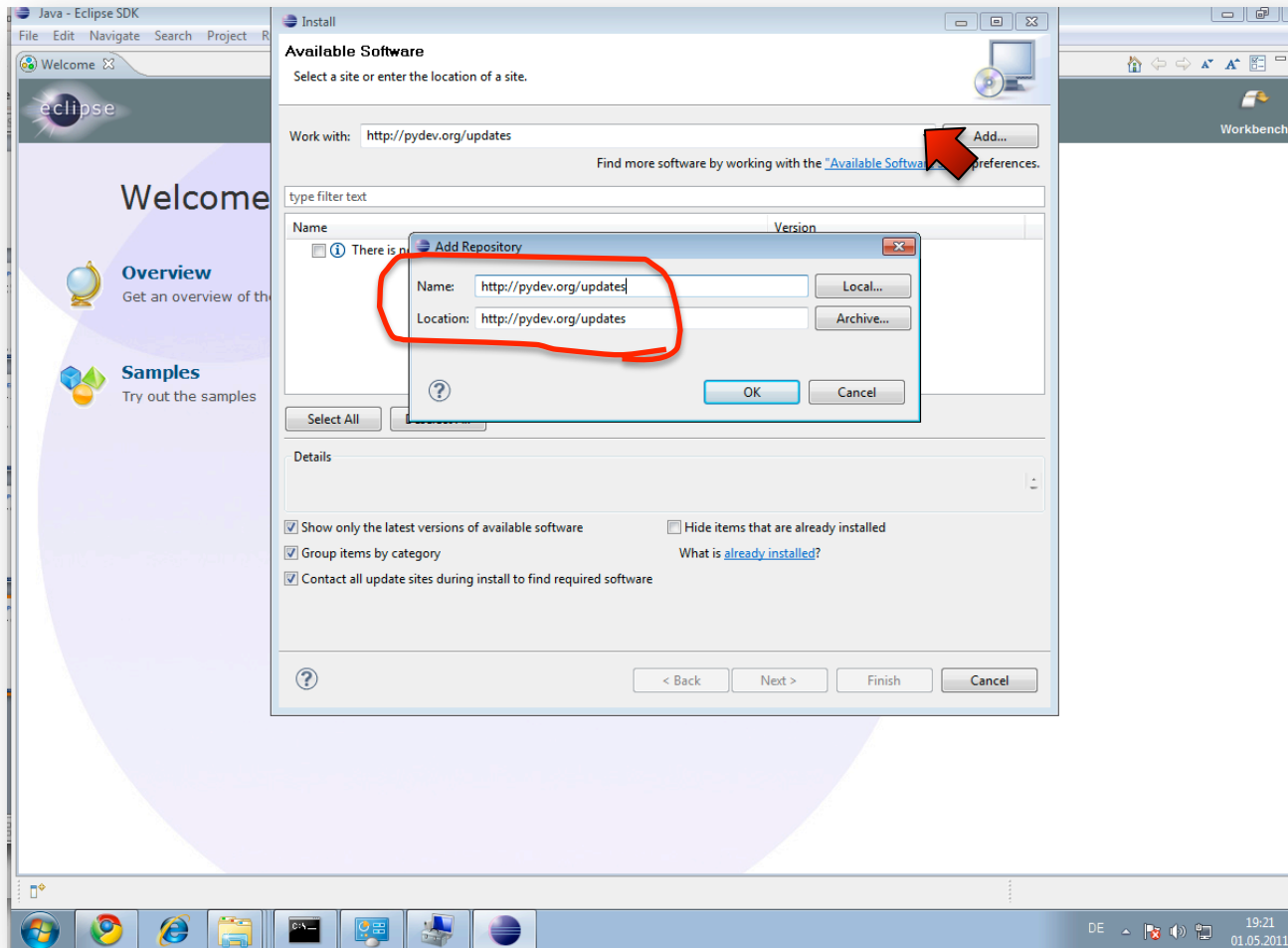
PyDev installieren

- **PyDev** (Eclipse-Plugin für Python)
 - Syntax-Highlighting, Code-Completion, Debugger
- **Achtung:** zuerst Python, NumPy, Matplotlib installieren!



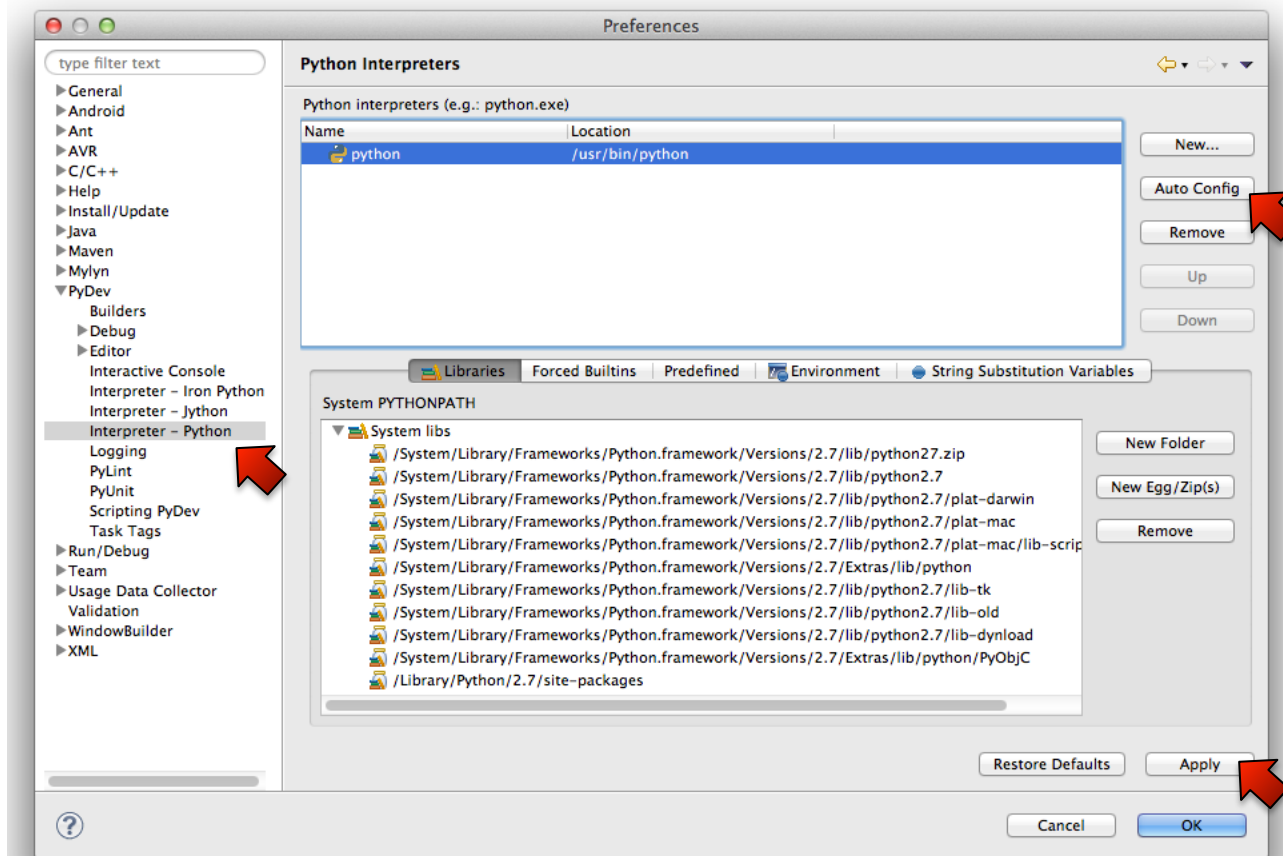
PyDev installieren

- <http://pydev.org/updates>



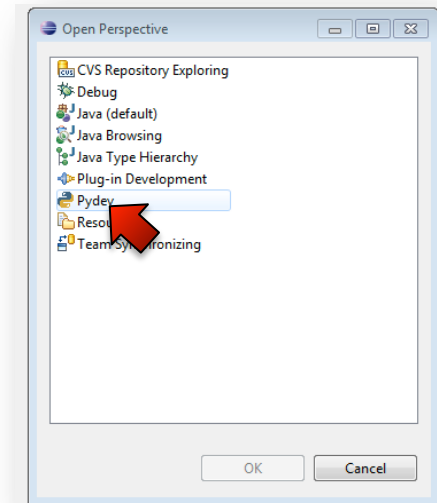
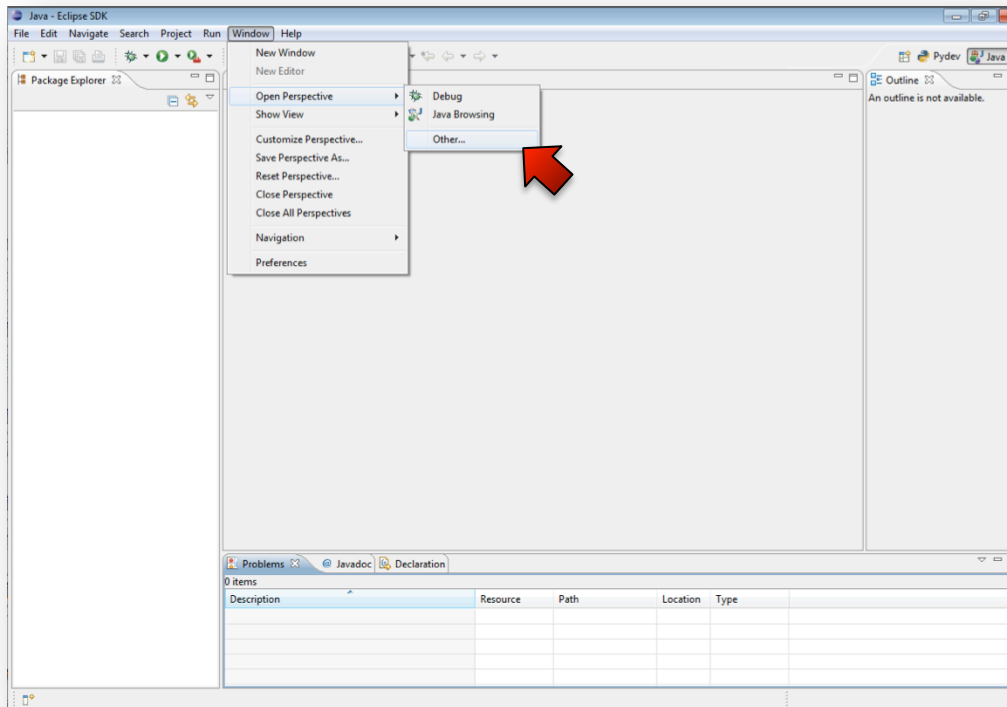
Python Interpreter für PyDev auswählen

- Eclipse-Preferences → PyDev
- „Auto Config“ klappt, falls Python im Path gefunden wird

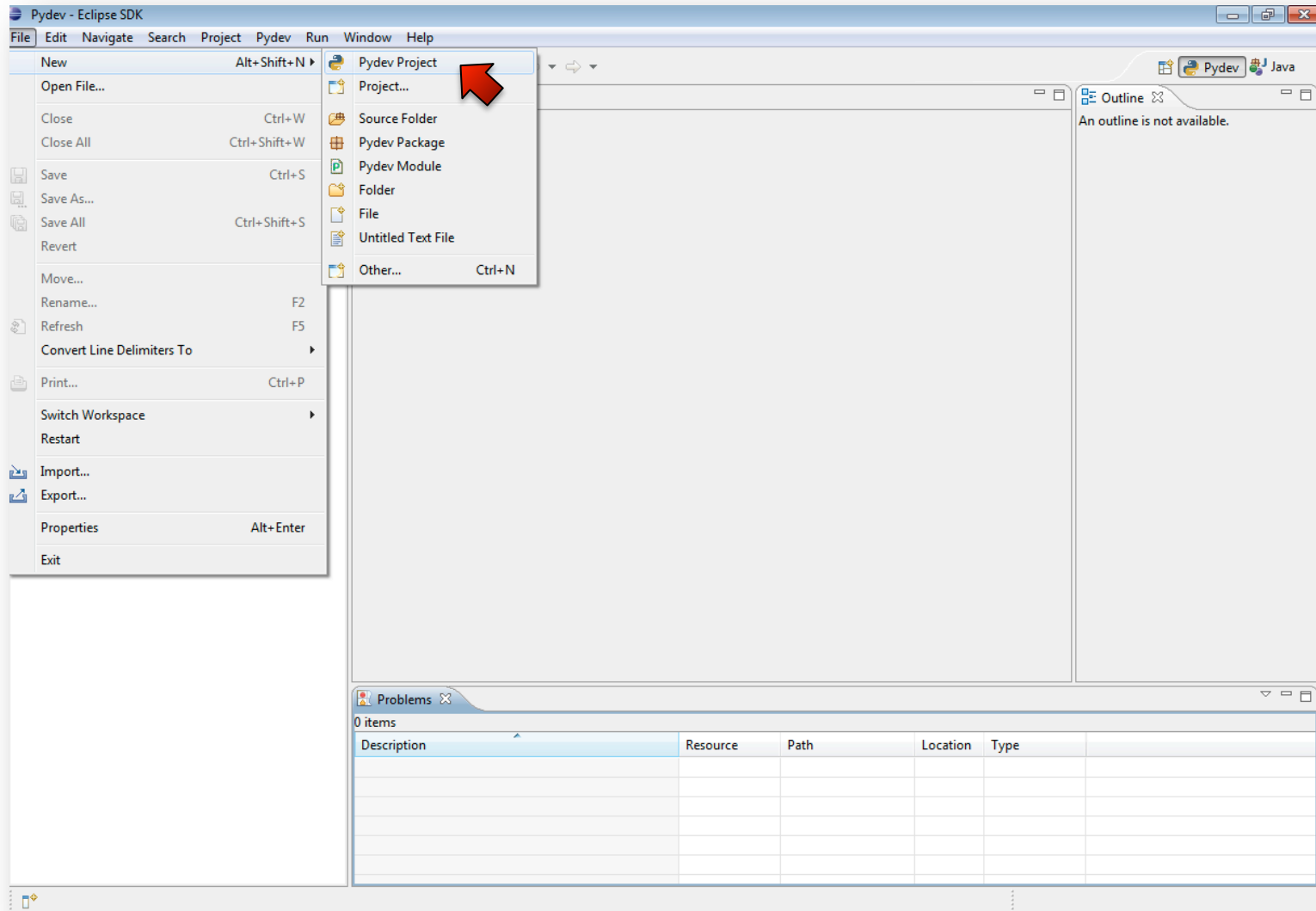


Arbeiten mit Pydev

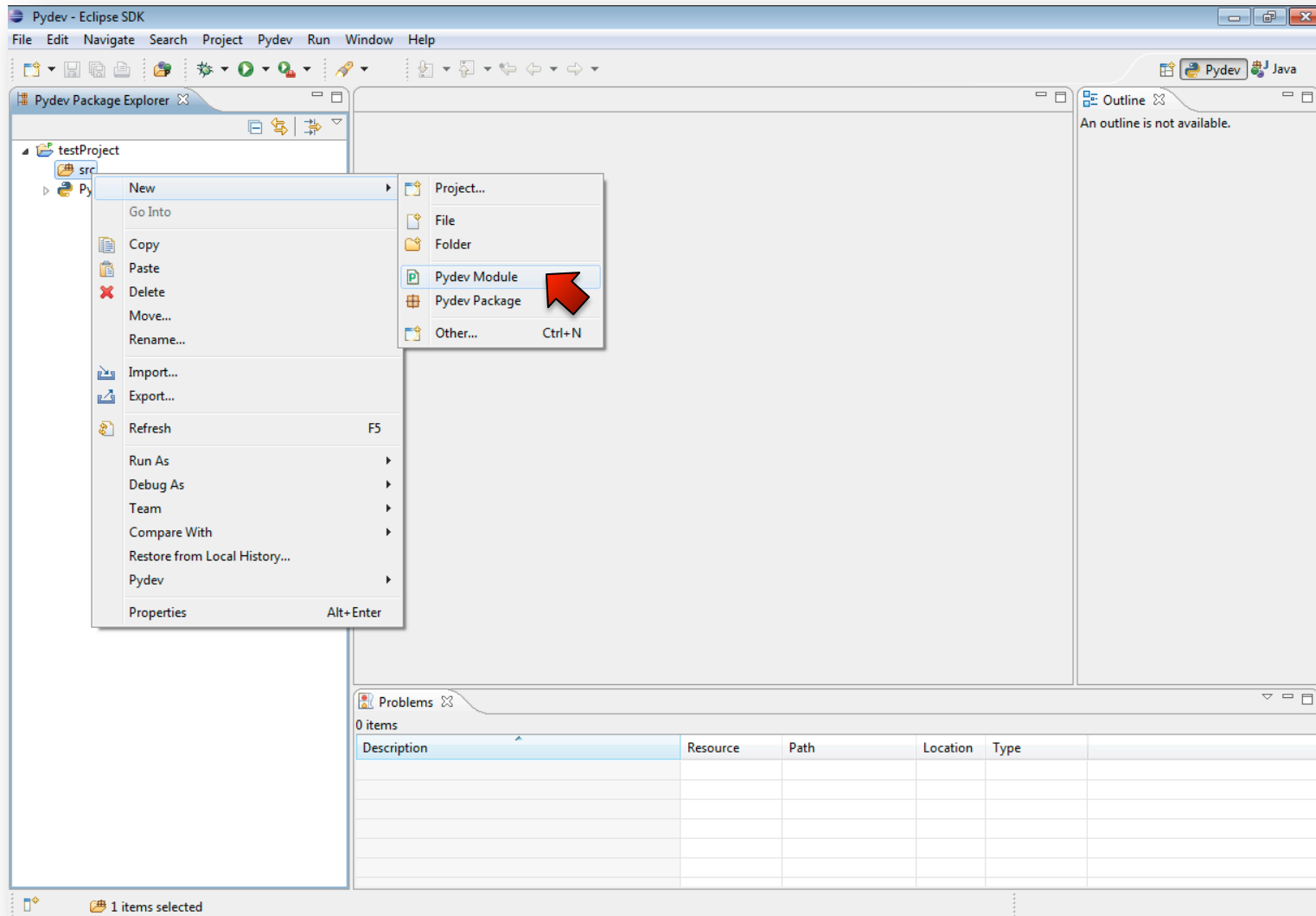
- PyDev-Perspective öffnen



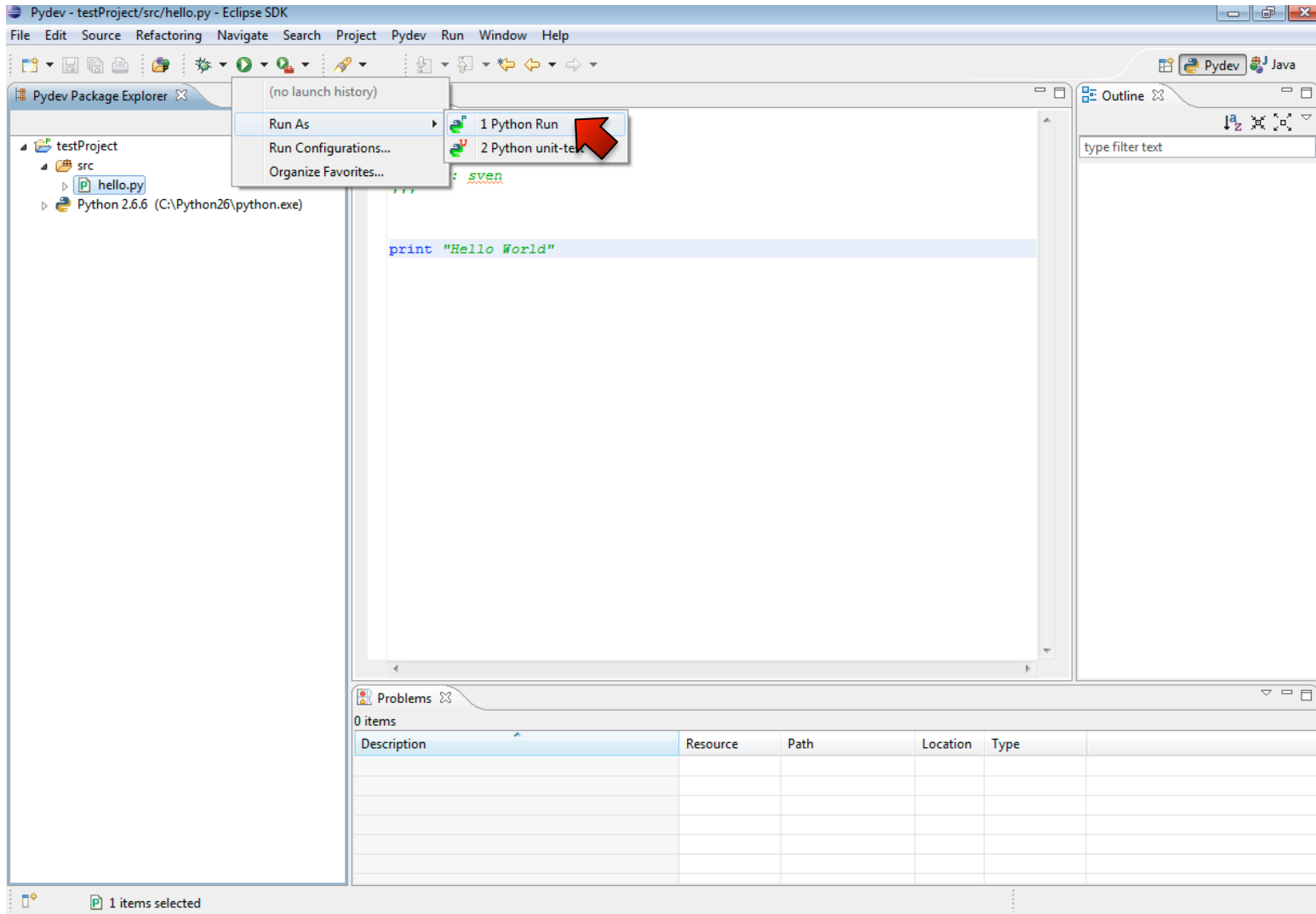
Neues PyDev-Projekt anlegen



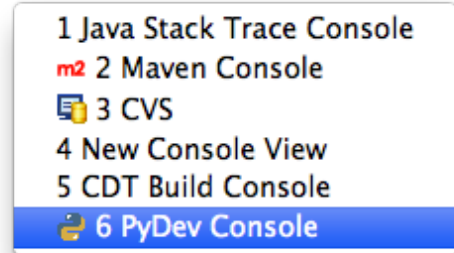
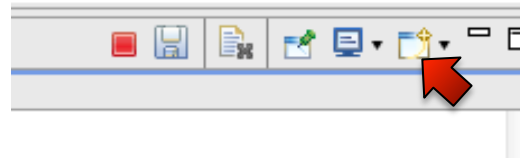
Neues Python-Modul anlegen



Modul ausführen



Python Tipps



- Interaktiver Modus und Modulbefragung:

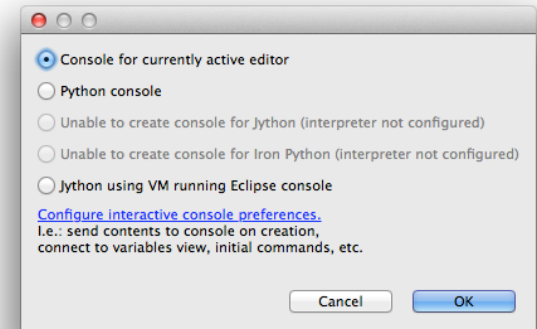
```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh',
'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'hypot', 'isinf', 'isnan',
'ldexp', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt',
'tan', 'tanh', 'trunc']
>>> help(math.atan2)
```

Help on built-in function atan2 in module math:

```
atan2(...)
    atan2(y, x)
```

Return the arc tangent (measured in radians) of y/x .
Unlike $\text{atan}(y/x)$, the signs of both x and y are considered.

```
>>>
```



Python Tipps

- `print` ist Dein Freund

```
>>> a,b,c = (1.0, "foo", ["bar"])
>>> print a,b,c      # Komma hinter der letzten Variable unterdrückt das Leerzeichen
1.0 foo ['bar']
>>> print "x %d y %d z %d" % (1,4,9)
x 1 y 4 z 9
>>>
```

- Modul als Hauptprogramm ausführen
(nützlich um tests zu schreiben)

```
if __name__=="__main__":
    # code, der nur ausgeführt wird wenn das Modul als hauptmodul startet
```

Python Tipps

- Listen

```
>>> liste1 = [1,2,3,4,5]
>>> liste2 = [6,7,8,9]
# '+' konkateniert zwei listen. Vorsicht: erzeugt keine neuen elemente
# weitere Operatoren und eingebaute Listenmethoden in der Doku zu finden
>>> liste1 + liste2
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- “Unpacking” von Unterelementen

```
>>> koordinaten = [[1,0],[0,1],[2,3]]
>>> for x,y in koordinaten:
...     print 'x',x,'y',y
x 1 y 0
x 0 y 1
x 2 y 3
```

- List-Comprehensions

- Erzeugen eine neue Liste aus einer Eingabeliste
- Ersparen das Implementieren vieler for-Schleifen

```
>>> [2**x for x in range(10)]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```


Python Tipps

- List-Slicing: Einfache Bildung von Unterlisten

```
>>> liste = [1,2,3,4,5,6,7,8]
```

```
# vom Listenkopf abschneiden
```

```
>>> liste[3:]  
[4, 5, 6, 7, 8]
```

```
# vom Listenende abschneiden
```

```
>>> liste[:3]  
[1, 2, 3]
```

```
# segment der Liste auswählen
```

```
>>> liste[2:5]  
[3, 4, 5]
```

```
# Listenadressierung für Fortgeschrittene ;)
```

```
>>> liste[-4:]  
[5, 6, 7, 8]  
>>> liste[:-4]  
[1, 2, 3, 4]
```

Python Tipps

- Funktionen

```
def zone_f2(coords, center, k=0.001):  
    deltaX = center[0] - coords[0]  
    deltaY = center[1] - coords[1]  
    return np.cos(k*(deltaX*deltaX + deltaY*deltaY))
```

Python Tipps (optional)

- map / filter und anonyme (Lambda-) funktionen
 - map bildet listenelemente auf eine Funktion ab und erstellt dabei eine neue liste
 - filter kann über ein bool'sches Entscheidungskriterium Elemente aus eine Liste herausfiltern. Funktioniert jedoch auch mit list comprehensions.
 - Anstelle einer Lambda-Funktion kann man in den unteren Beispielen auch den namen einer bestehenden Funktion übergeben. Aber: mit Lambda lassen sich zur Laufzeit Funktionen dynamisch generieren

```
>>> koordinaten = [[1,0],[0,1],[2,3]]
>>> map (lambda k : k[0] + k[1], koordinaten)
[1, 1, 5]
>>> zahlen = range(10)
>>> zahlen
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> filter (lambda x: x < 5, zahlen)
[0, 1, 2, 3, 4]
```

NumPy Arrays

- importieren: `import numpy as np`

- NumPy Arrays

```
>>> import numpy as np
>>> a = np.ones( (4, 3) )
>>> a.dtype
dtype('float64')
>>> a.shape
(4, 3)
```

```
>>> b = a.transpose()
>>> b.shape
(3, 4)
>>> b.size
12
```

```
>>> a[:,2]
```

```
>>> a[0,3:5]
array([3,4])
```

Adressierung Elemente in Array:
Zeile, Spalte
Indexierung beginnt bei 0

- Array Slicing

```
>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

```
>>> a[4:,4:]
array([[44, 45],
       [54, 55]])
```

NumPy Vektoroperationen

- `>>> a = np.ones((4,4))`

```
>>> a + 4
```

```
array([[ 5.,  5.,  5.,  5.],  
       [ 5.,  5.,  5.,  5.],  
       [ 5.,  5.,  5.,  5.],  
       [ 5.,  5.,  5.,  5.]])
```

```
>>> a *2
```

```
array([[ 2.,  2.,  2.,  2.],  
       [ 2.,  2.,  2.,  2.],  
       [ 2.,  2.,  2.,  2.],  
       [ 2.,  2.,  2.,  2.]])
```

- Obige Operationen **elementweise**.
Matrixmultiplikation mit `numpy.dot(A,B)`

```
>>> b = np.arange(16).reshape((4,4))
```

```
>>> b
```

```
array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15]])
```

```
>>> np.dot(a,b)
```

```
array([[ 24.,  28.,  32.,  36.],  
       [ 24.,  28.,  32.,  36.],  
       [ 24.,  28.,  32.,  36.],  
       [ 24.,  28.,  32.,  36.]])
```

Bilder laden und plotten mit Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
```

```
a = plt.imread('./Kitten.png')
plt.imshow(a)
plt.show()
```

```
print a.shape      → RGB-Bild: (300, 400, 3)
                   → B&W-Bild: (300, 400)
print a.dtype      → float32
```



- `a` ist ein Numpy-Vektor (300x400x3), der entsprechend modifiziert werden kann:

```
a = plt.imread('./Kitten.png')
a = np.mean(a,2) # axis 2 ist die RGB-Achse
plt.imshow(a)
plt.show()
```



Mehrere “Subfigures”

```
import numpy as np
import matplotlib.pyplot as plt
```

```
a = plt.imread('./CG2/src/hand2.png')
print a.shape
```

```
plt.subplot(121)
plt.imshow(a)
```

```
plt.subplot(122)
b = np.mean(a,2)
plt.gray()
plt.imshow(b)
```

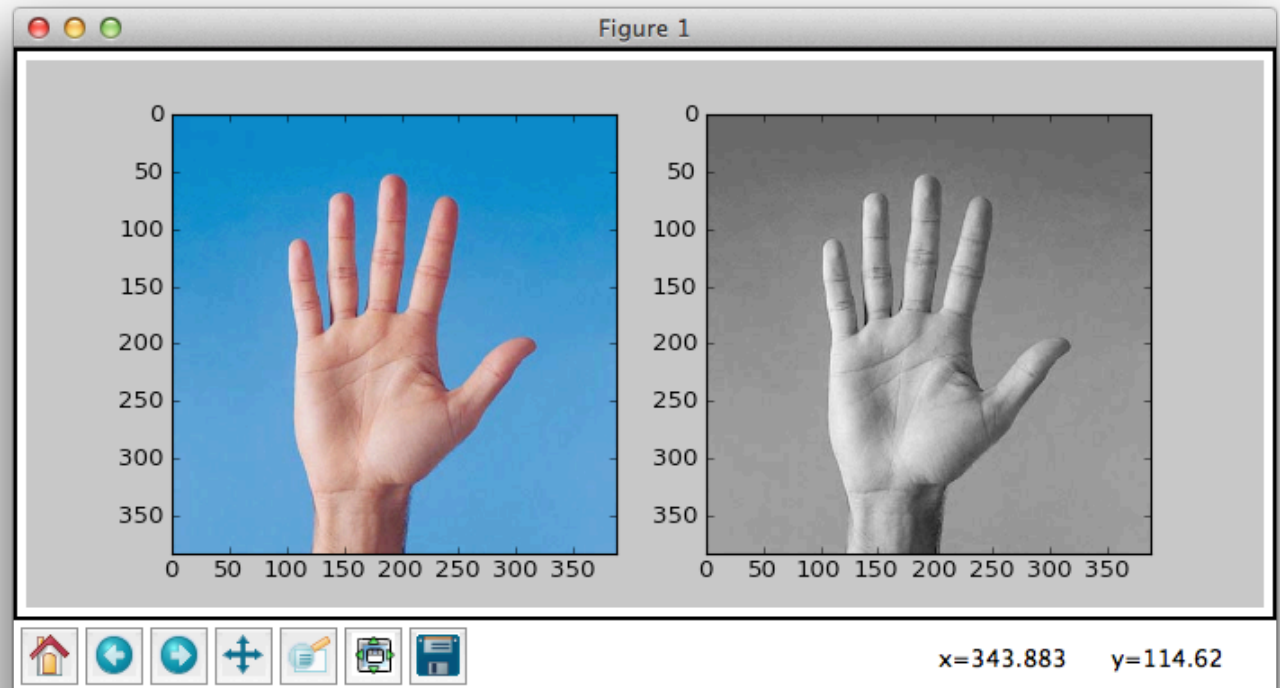
```
plt.show()
```

`plt.subplot(121)`

1 Zeile

2 Spalten

Subfigure 1



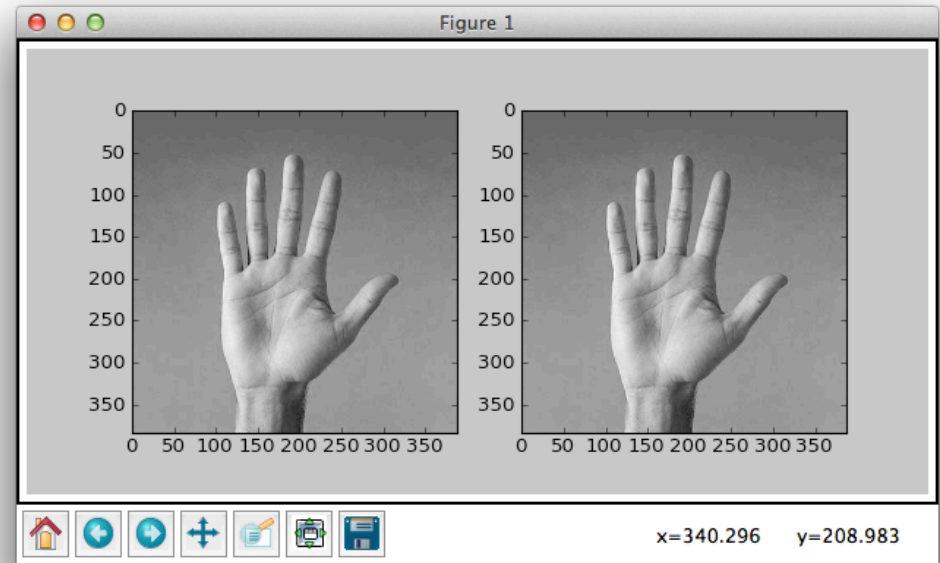
Skalierung Helligkeit

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
plt.gray()
a = plt.imread('./CG2/src/hand2.png')
b = np.mean(a,2)
```

```
plt.subplot(121)
plt.imshow(b)
```

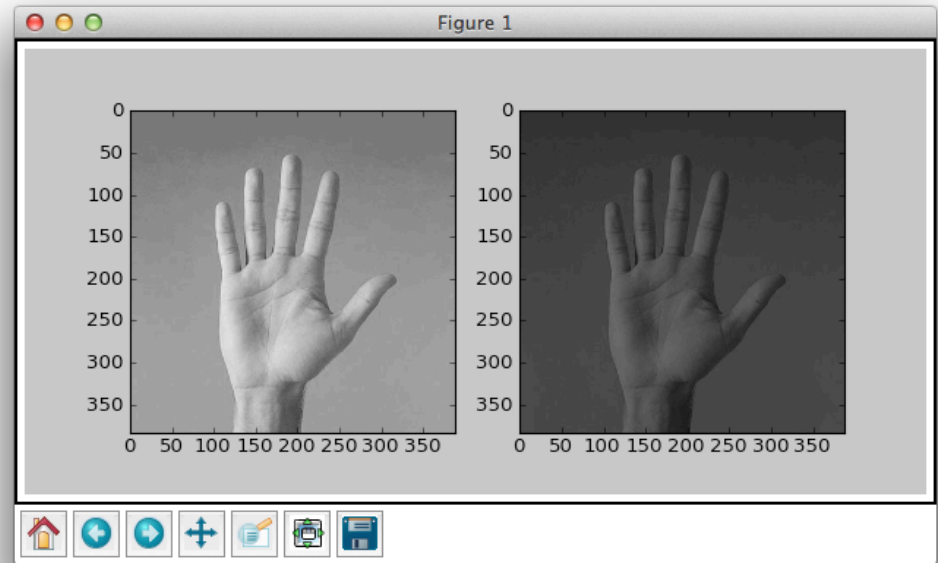
```
plt.subplot(122)
c = 0.4 * b
plt.imshow(c)
```

```
plt.show()
```



Skalierung Helligkeit

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
plt.gray()
a = plt.imread('./CG2/src/hand2.png')
b = np.mean(a,2)
```



```
plt.subplot(121)
plt.imshow(b, norm = matplotlib.colors.NoNorm())
```

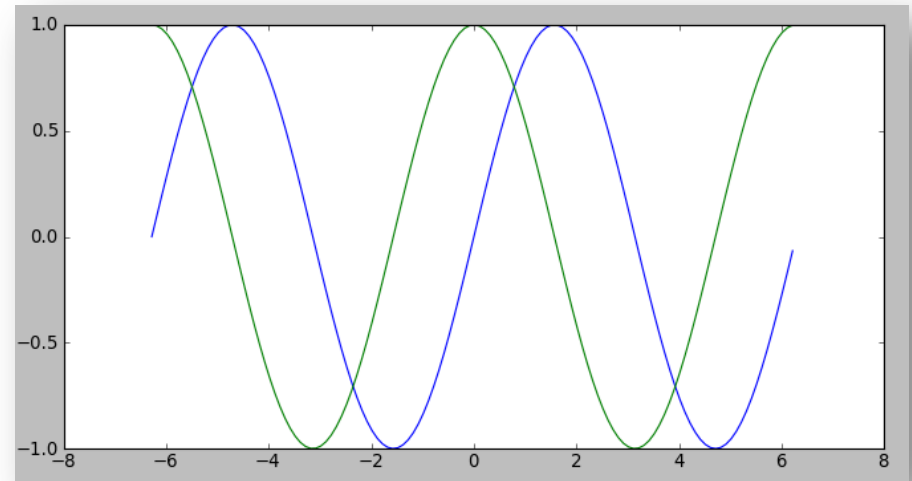
```
plt.subplot(122)
c = 0.4 * b
plt.imshow(c, norm = matplotlib.colors.NoNorm())
```

```
plt.show()
```

Plotten von Daten

- Funktionen

```
rng = np.arange(-2*np.pi, 2*np.pi, 0.1)
sin = np.array([np.sin(x) for x in rng])
cos = np.array([np.cos(x) for x in rng])
plt.plot(rng, sin)
plt.plot(rng, cos)
plt.show()
```



- Histogramme

```
a = plt.imread([...])
red = a[:, :, 0]
green = a[:, :, 1]
blue = a[:, :, 2]
r = red.reshape(red.size, 1)
g = green.reshape(green.size, 1)
b = blue.reshape(blue.size, 1)
plt.hist(r, bins= 255, color='red')
plt.hist(g, bins= 255, color='green')
plt.hist(b, bins= 255, color='blue')
plt.show()
```

