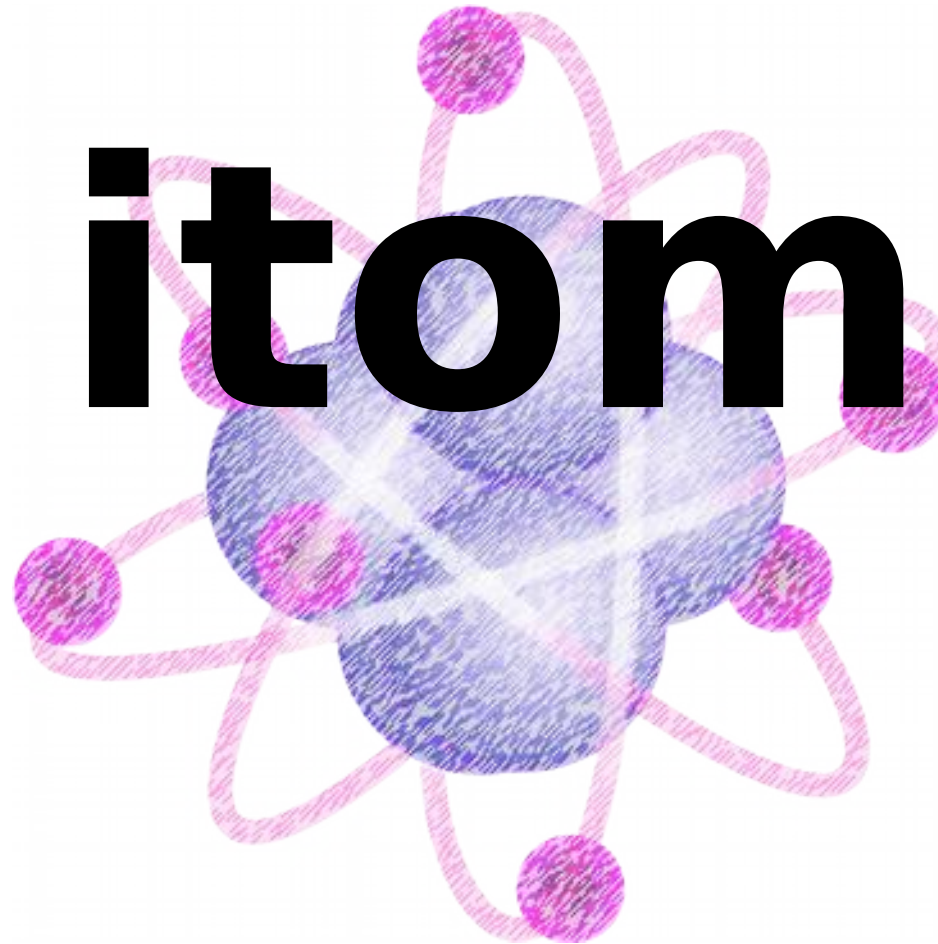




- **Itom** Einführung
 - Warum wurde **itom** entwickelt?
 - **itom** GUI
 - Hauptbestandteile
 - Python Grundlagen und die wichtigsten Module
 - **itom** plugins

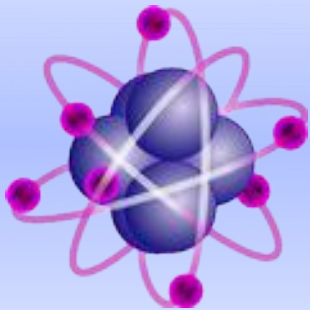
- **Itom** Einführung
 - Warum wurde **itom** entwickelt?
- **itom** GUI
- Hauptbestandteile
- Python Grundlagen und die wichtigsten Module
- **itom** plugins

Matlab

- + Datenverarbeitung
- + Vielfältige mathematische Bibliotheken
- Hardware-Einbindung
- Grafische Benutzeroberflächen

Labview

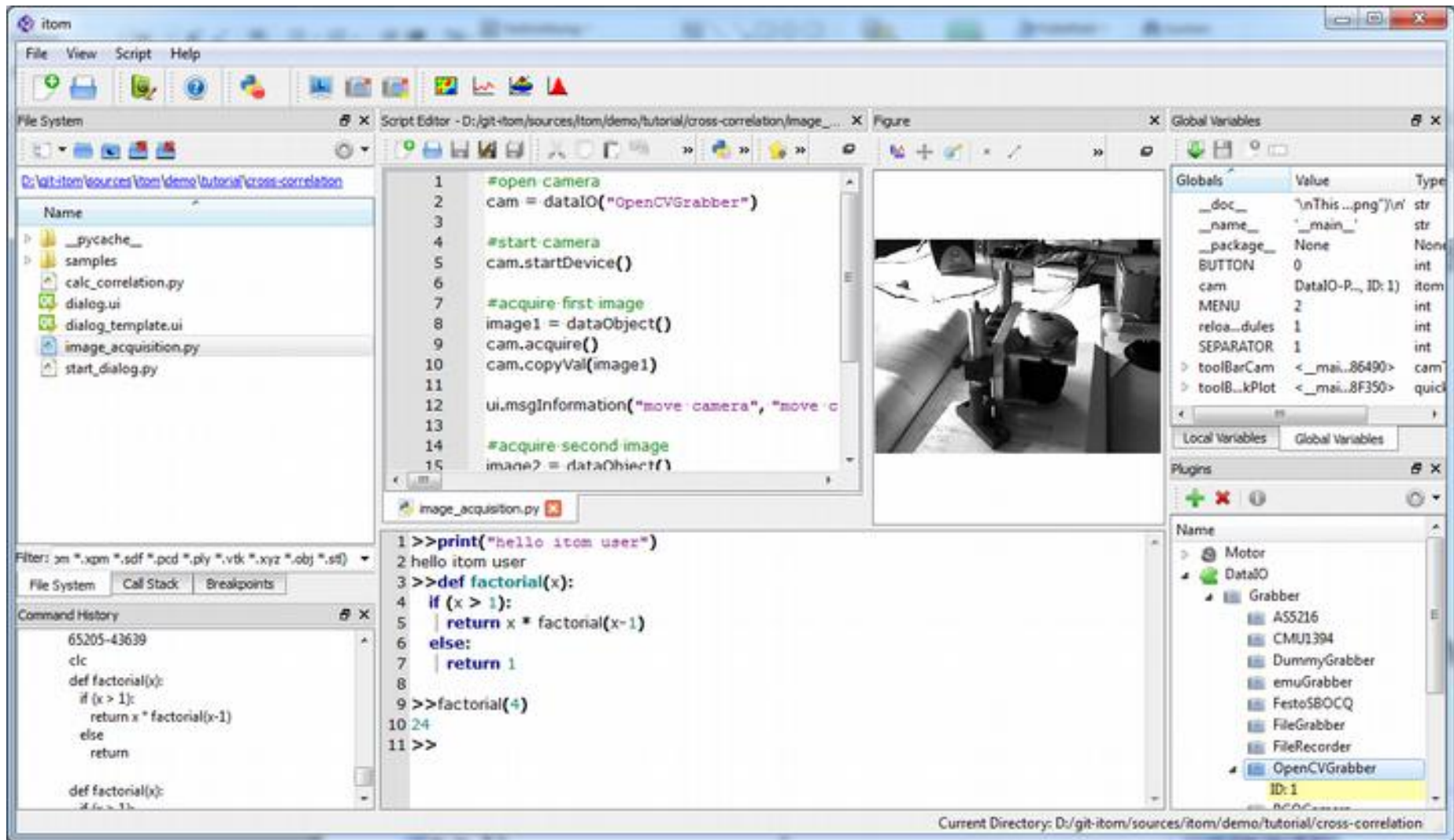
- + Einfache Oberflächen Erstellung
- + Gute Hardware-Unterstützung
- Datenanalyse
- Keine einheitlichen Hardware-Interfaces



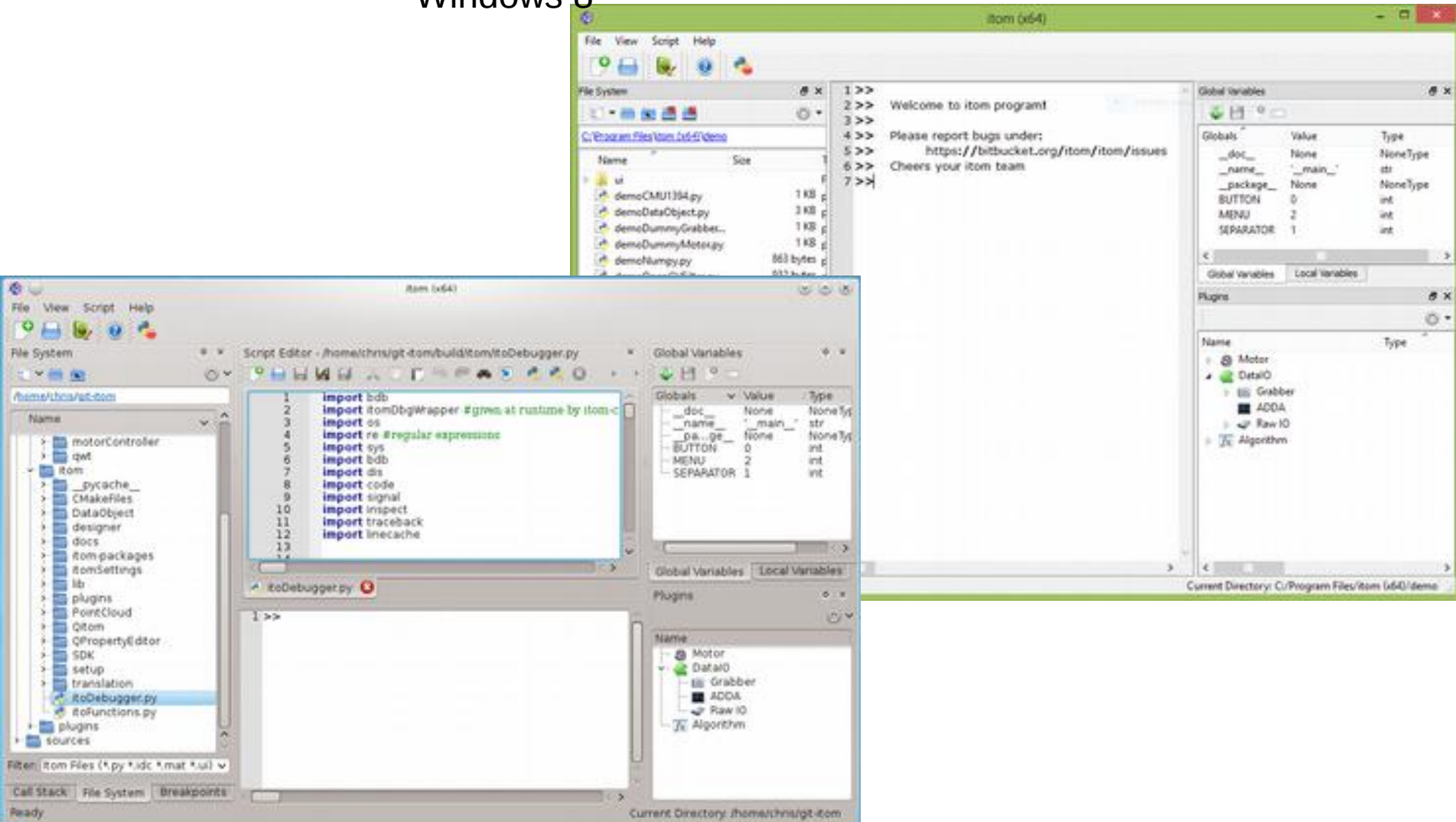
itom

- Schnelle und einfach bedienbare Skriptsprache (Python)
- Einheitliche Hardwareanbindung
- Messablaufsautomatisierung
- Schnelle Datenverarbeitung
- Einfach anpassbar

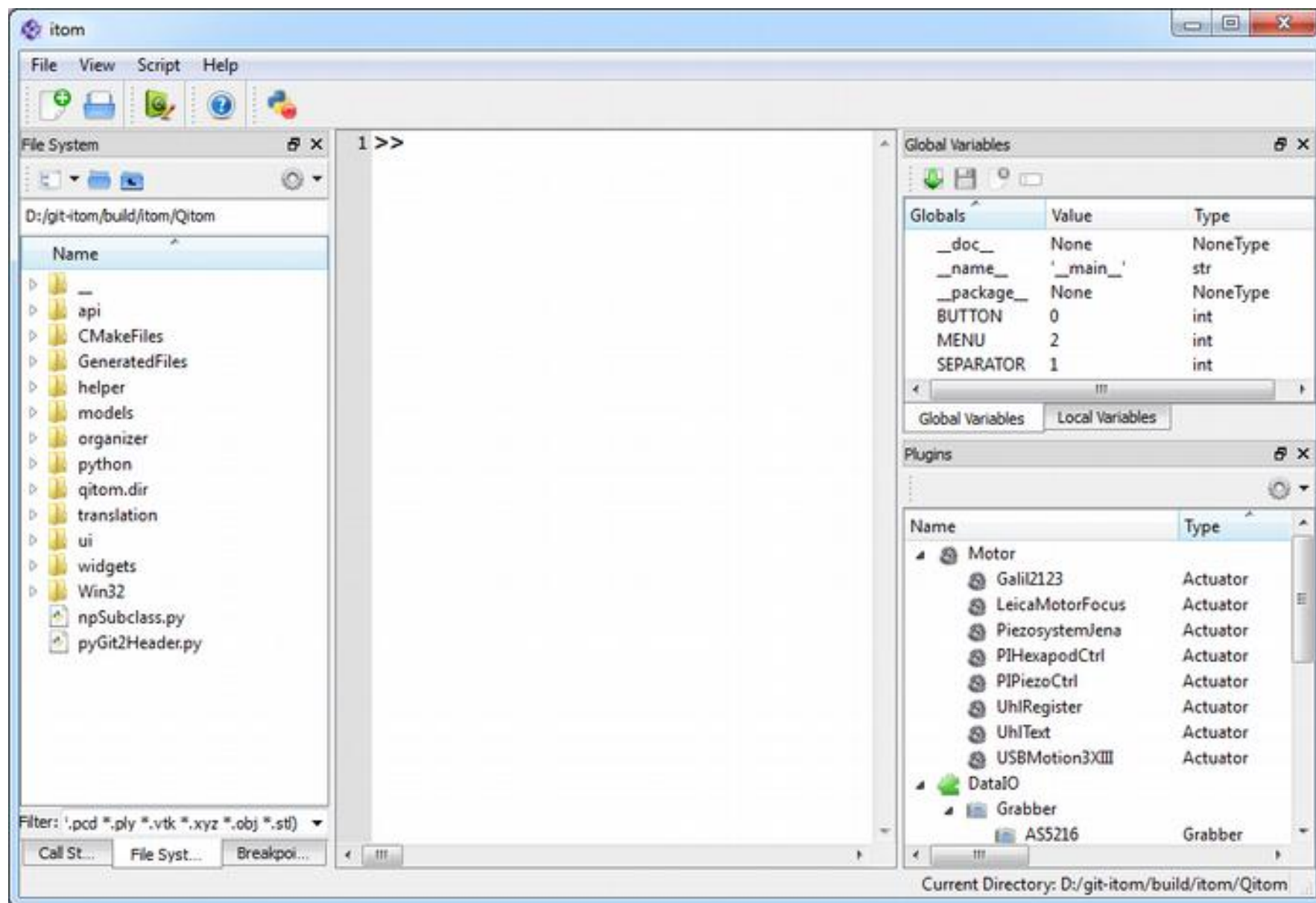
- **Itom** Einführung
 - Warum wurde **itom** entwickelt?
- **itom GUI**
- Hauptbestandteile
- Python Grundlagen und die wichtigsten Module
- **itom** plugins



Windows 8



Debian KDE



GUI – Befehlszeile

The screenshot shows the 'itom' application window. On the left is a 'File System' pane showing a directory tree. The central area is a command prompt where the command 'dataIO(name[, mandparams, optparams]) -> constructor' is entered. On the right, there are two panels: 'Global Variables' and 'Local Variables'. The 'Global Variables' panel contains a table with columns 'Name', 'Value', and 'Type'. The 'Local Variables' panel is currently empty. Below these is a 'Plugins' pane showing a list of available plugins like 'Motor', 'DataIO', and 'Grabber'.

Name	Value	Type
__doc__	None	NoneType
__name__	'_main_'	str
__package__	None	NoneType
BUTTON	0	int
MENU	2	int
SEPARATOR	1	int

Name	Type
Motor	
Gali2123	Actuator
LeicaMotorFocus	Actuator
PIHexapodCtrl	Actuator
PIPiezoCtrl	Actuator
UhlRegister	Actuator
UhlText	Actuator
USBMotion3XIII	Actuator
DataIO	
Grabber	
AS5216	Grabber

- Ein-/Ausgabefenster für Python (Informationen, Warnungen, Fehler)
- Direkte Ausführung von Python-Anweisungen
- Ähnlich Matlab
- Auto-Vervollständigung von Befehlen
- Syntax Hilfe und Highlighting

GUI - Workspace



The screenshot shows the itom GUI workspace. The File System panel on the left displays the directory structure of the project. The Global Variables panel on the right, highlighted with a red circle, shows a table of global variables. The Plugins panel at the bottom right lists various hardware modules.

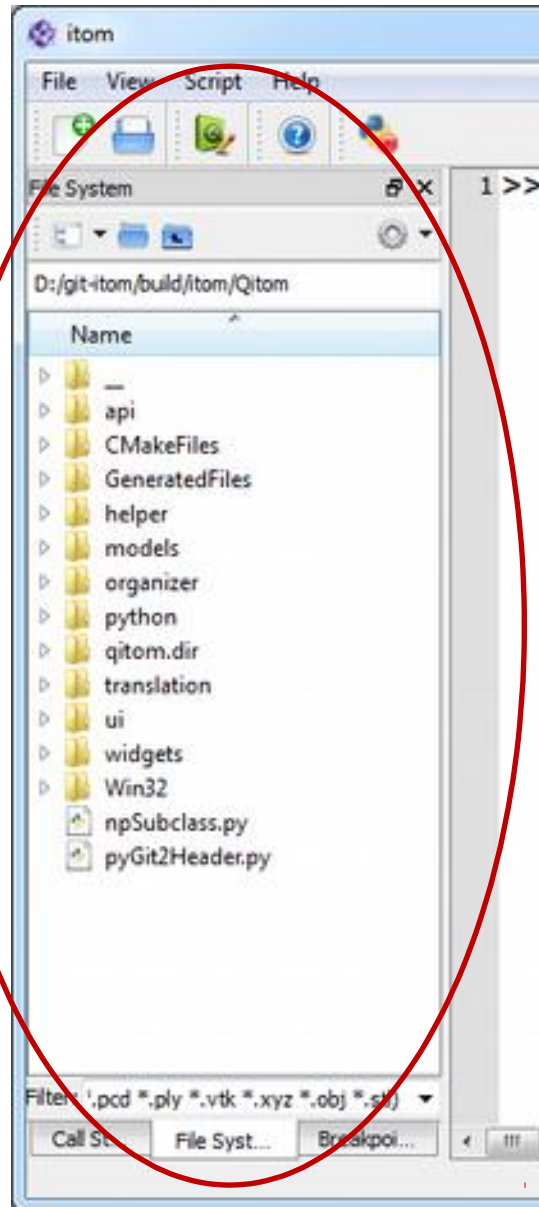
Globals	Value	Type
__doc__	None	NoneType
__name__	'__main__'	str
__package__	None	NoneType
BUTTON	0	int
MENU	2	int
SEPARATOR	1	int

Name	Type
Motor	
Galil2123	Actuator
LeicaMotorFocus	Actuator
PiezosystemJena	Actuator
PIHexapodCtrl	Actuator
PIPiezoCtrl	Actuator
UhlRegister	Actuator
USBMotion3XIII	Actuator
DataIO	
Grabber	
AS5216	Grabber

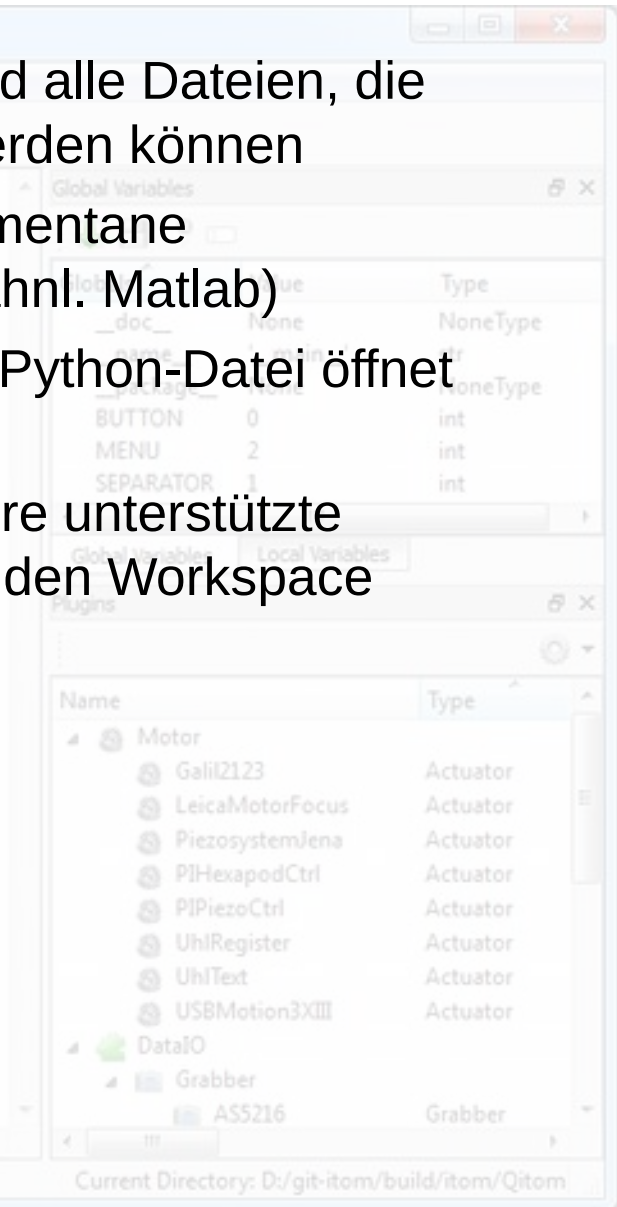
- Global Variables: enthält alle globalen Variablen
- Local Variables: alle lokale Variablen innerhalb einer Funktion (nur im Debug-Modus)
- Direkter Im-/Export von/zum Python Workspace

Current Directory: D:/git-itom/build/itom/Qtom

GUI – Dateisystem

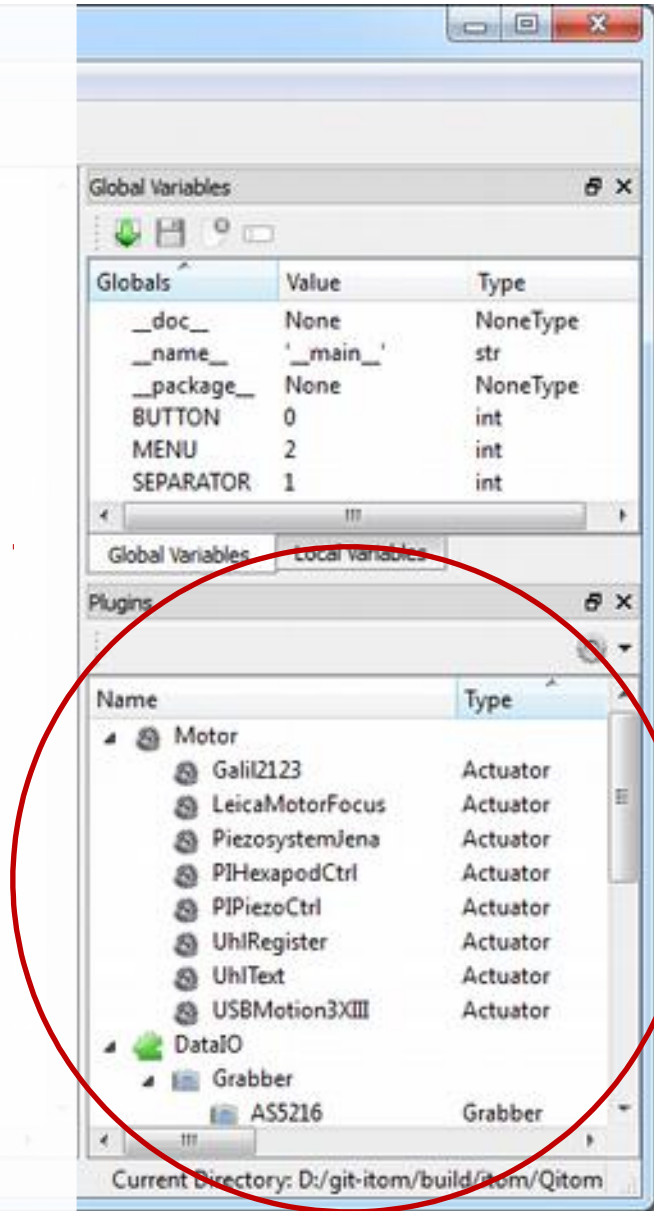
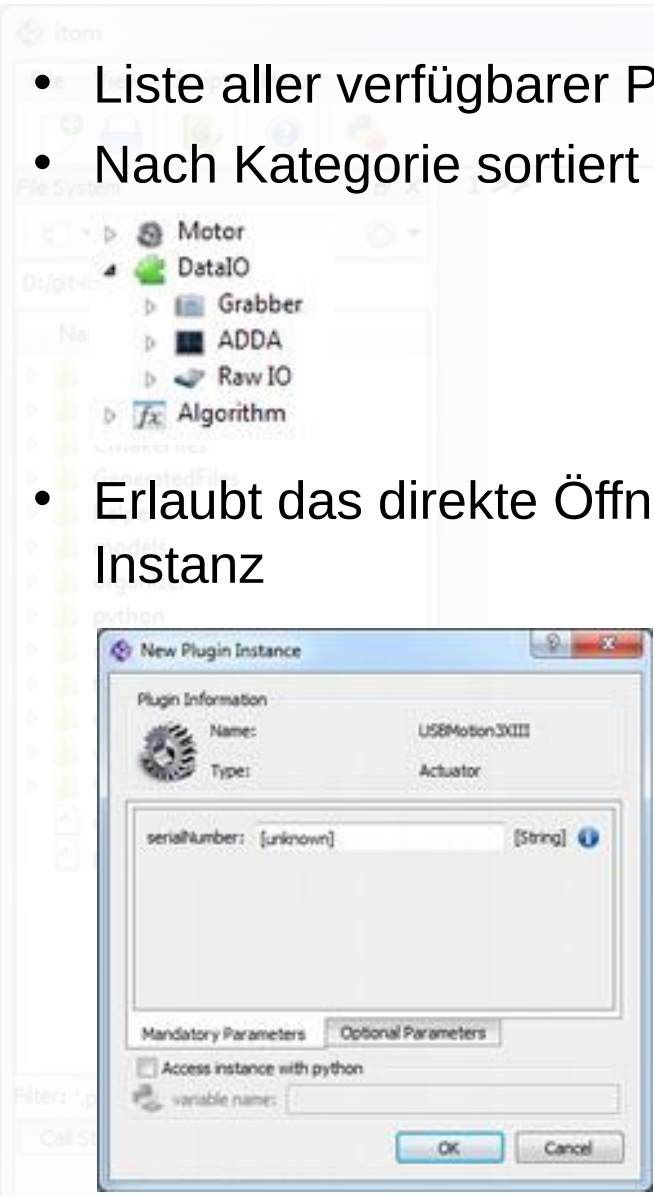


- Zugriff auf Skripte und alle Dateien, die von **itom** geöffnet werden können
- Standard ist das momentane Arbeitsverzeichnis (ähnl. Matlab)
- Doppelklick auf eine Python-Datei öffnet diese im Editor
- Doppelklick auf andere unterstützte Dateien lädt diese in den Workspace



GUI – Plugins

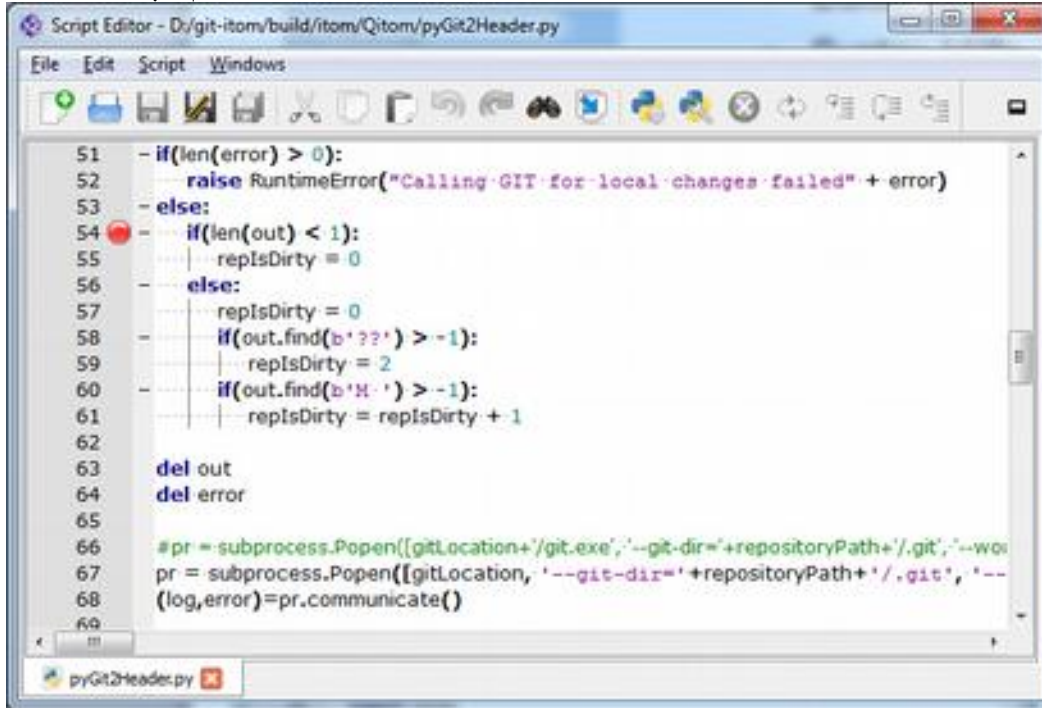
- Liste aller verfügbarer Plugins
- Nach Kategorie sortiert
- Erlaubt das direkte Öffnen einer Plugin-Instanz



Skripteditor



- Editor für Python-Skripte
- Syntax-Hilfe und Highlighting
- Auto-Vervollständigung



- Standard Editor Funktionalität
- Tabben mehrerer Skripte
- In Haupt-Gui dockbar
- Ausführung von Skripten
- Debugger-Funktionalität

Syntax Hilfe und Auto-Vervollständigung

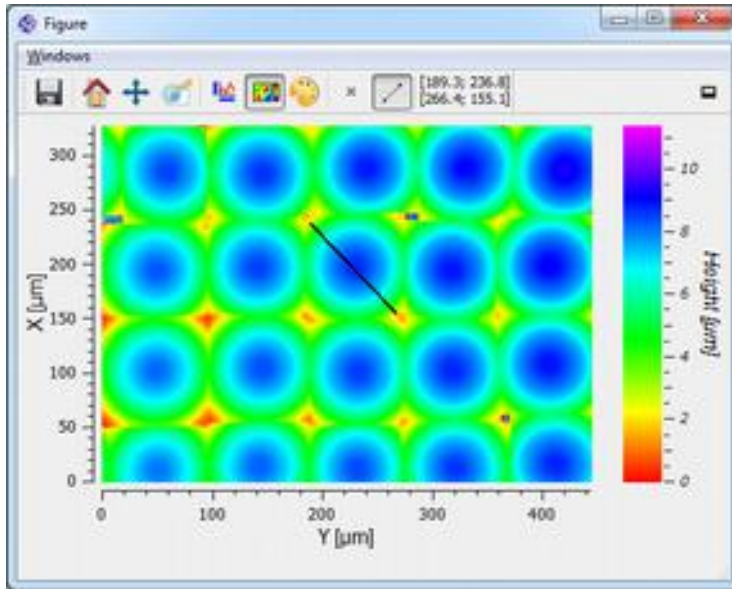
- Auto-Vervollständigung
(Auswahl über tab-Taste)

```
6 dataObj|
7 data (itom.dataObject)
8 data (itom.polygonMesh)
9 dataIO (itom)
10 dataObject (itom)
11 datetime64 (numpy)
12 datetime64 (scipy)
13 datetime_as_string (numpy)
14 datetime_as_string (scipy)
15 datetime_data (numpy)
16 datetime_data (scipy)
```

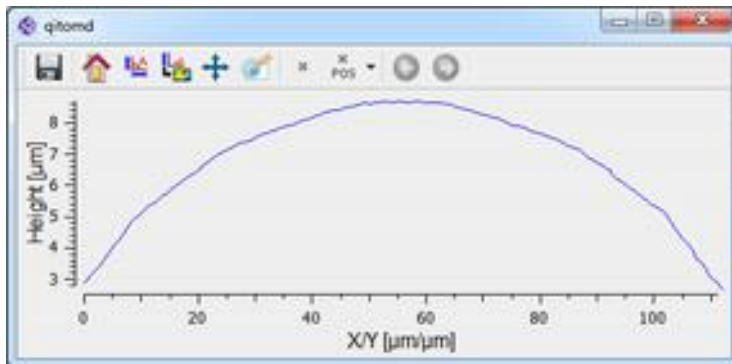
Syntax-Hilfe-Dateien
müssen in den
Einstellungen von itom
hinzugefügt werden, damit
diese Funktionalität
verfügbar ist

- Syntax Hilfe

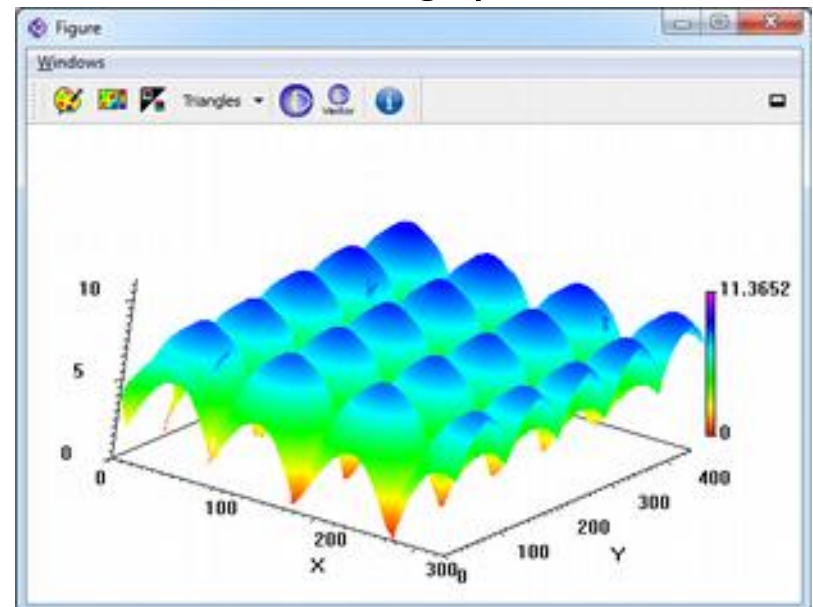
```
45 plot(
46 plot(data, [areaIndex, className]) -> plots a dataObject in the current or given area of this figure
47 plot(data, [className]) -> plots a dataObject in a newly created figure
48 plot(??) [doc: Plot lines and/or markers to the]
```



↓
abhängiger 1D-Linien plot

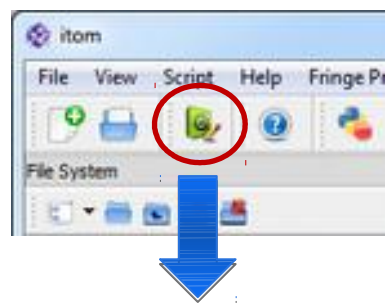


- 1D, 2D, 2.5D plots
- Benutzer angepasste plots können erstellt werden
- Anzeigbar in
 - Einem separaten Fenster
 - In die Haupt-GUI eingedockt
 - In einer benutzerangepassten GUI



Benutzererstellte GUIs (Qt Designer)

- Design eigener GUIs in externem Qt Designer WYSIWYG tool (drag&drop).
- In der GUI erstellte Events (button click) können an Python-Funktionen angebunden werden



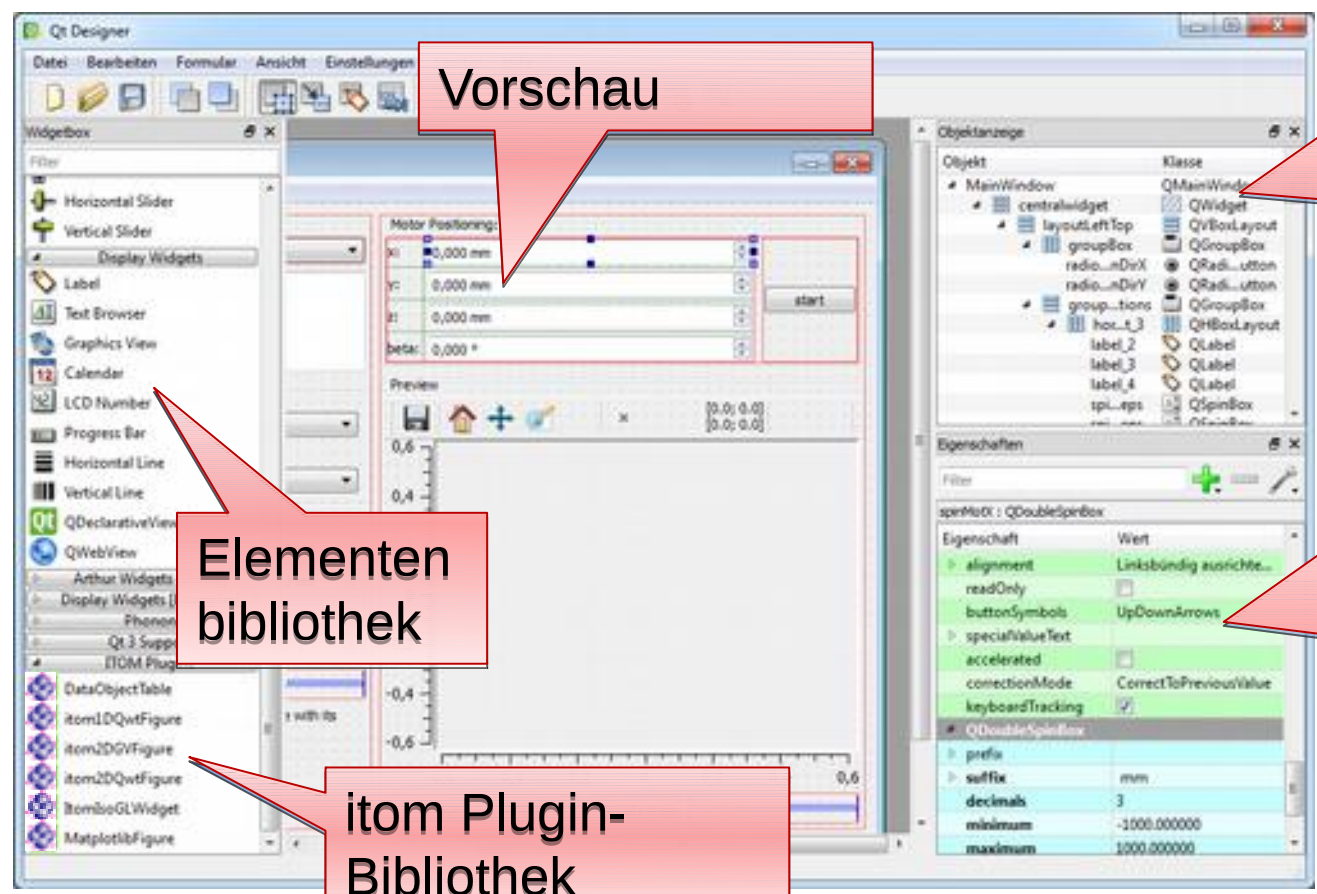
Vorschau

GUI Elemente:
Hierarchisch
Layouts

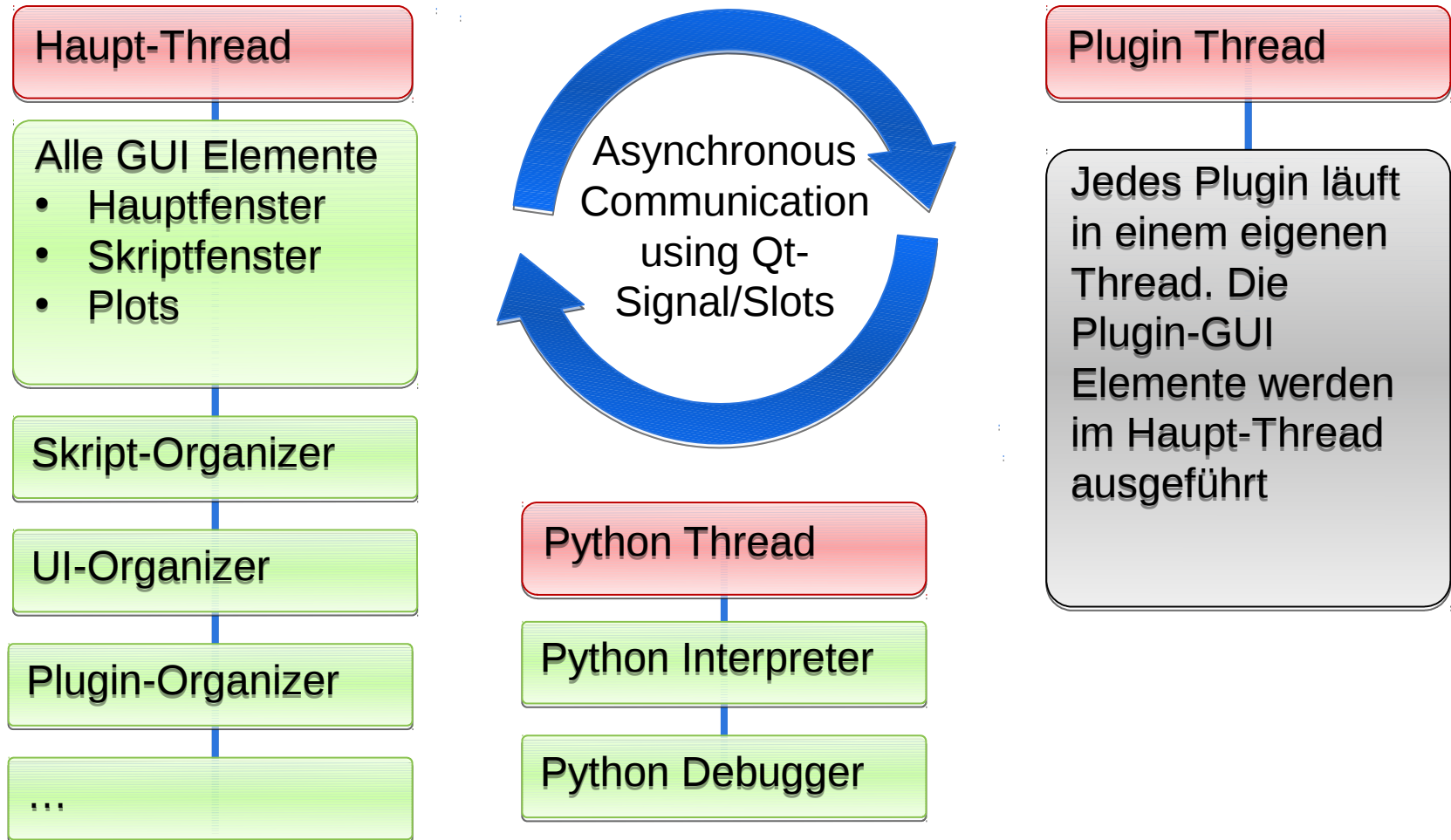
Elementen
bibliothek

Elementeigenschaft
en:
Können durch
Python-Skripte
geändert werden

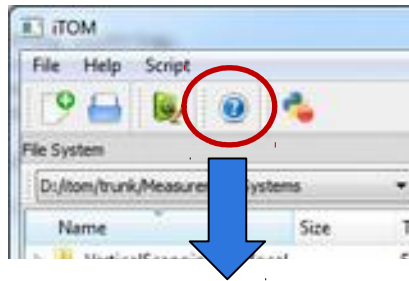
itom Plugin-
Bibliothek



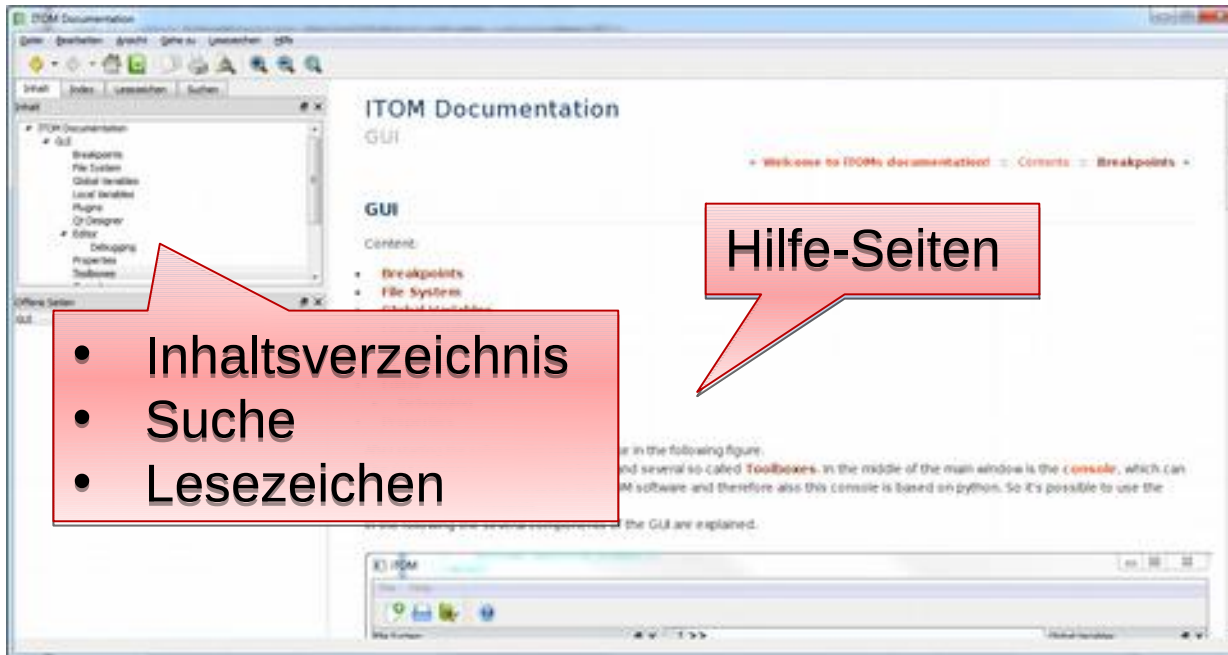
Multithreading



Benutzerdokumentation



- Benutzerdokumentation wird mittels des Qt Assistant angezeigt
- Kann in pdf, html... exportiert werden



itom.bitbucket.org/latest/docs



Zusätzliche Hilfe in Python

1. Syntax-Hilfe und auto-Vervollständigung im Python-Editor
2. Konfigurierbares, inhaltsabhängiges Syntax highlighting
3. Python-internes Hilfesystem über das Kommando *help(...)*
4. Zusätzliche Informationen und Hilfe zu den verfügbaren Plugins über die Kommandos:
pluginHelp(...),
filterHelp(...),
widgetHelp(...)

```
>>liveImage(  
liveImage(dataIO) -> shows camera image in a live window
```

```
#comment  
import sys  
  
def method(arguments):  
    """description of method"""  
    if(2==1):  
        print("crazy")  
    else:  
        print("alright")
```

```
>>help(plot)  
Help on built-in function plot in module itom:  
  
plot(...)  
    plot(dataObject) -> realizes a 2,5D realization in a new figure window.  
    Parameters:  
    - 'dataObject' is the data object whose region of interest should be two-dimensional
```

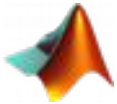
```
>>pluginHelp("PCOPixelFly")  
  
NAME:      PCOPixelFly  
TYPE:      DataIO  
VERSION:   0  
AUTHOR:    ITO  
INFO:      Developed for Windows only. Tested with PixelFlyQE.  
  
DETAILS:  
  
INITIALISATION PARAMETERS:  
    Initialisation function has no mandatory parameters  
  
Optional parameters:  
0   Board Number      int      value: 0   min: 0  
1   restoreLast       int      value: 0   min: 0
```

- **Itom** Einführung
 - Warum wurde **itom** entwickelt?
- **itom** GUI
- Hauptbestandteile
- Python Grundlagen und die wichtigsten Module
- **itom** plugins

- Open-Source Skriptsprache (sehr liberale BSD-Lizenz)
- In C programmiert
- Wird seit 1991 sets weiterentwickelt
- Unterstützt objektorientierte, funktionale und imperative Programmierung
- Version 3.2 und neuer werden in **itom** unterstützt
- Vollständig in itom integriert
- Große Anzahl an Drittanbieter-Modulen frei verfügbar
- Skripte werden vorkompiliert und gecacht für eine optimierte Laufzeit
- Integrierter Python-Debugger

- Variablen haben einen Python internen Typ, hauptsächlich: int, float, complex
- Typ-Casts über die Funktionen *int()*, *float()*...
- Zuweisungen: *a=1 a,b=1,2*
- Vergleichsoperatoren: *==, >, <, <=, >=, !=*
- Bitweise Operatoren: *&, |, ~, ^*
- Einfache Arithmetik: *a = a+1, a += 1, a=a**2*
- Operatoren auch auf viele Konstrukte anwendbar (arrays, Listen, Dictionaries...)

Example: Factorial



```
function ret = factorial(x)
  if(x > 1)
    ret = x * factorial(x-1);
  else
    ret = 1;
  end
end
```



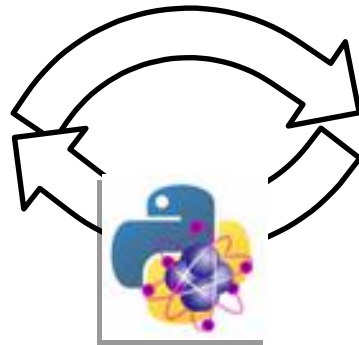
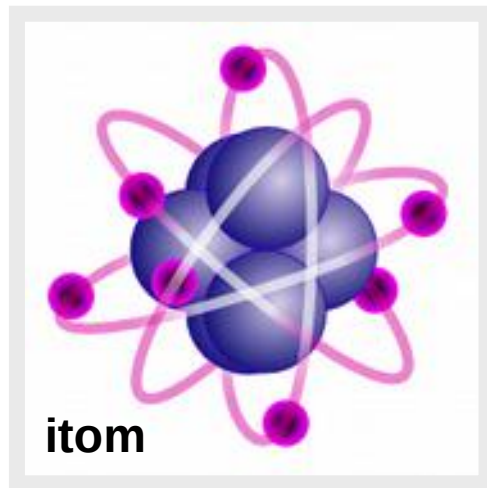
```
int factorial(int x)
{
  if (x > 1) {
    return x * factorial(x-1);
  } else {
    return 1;
  }
}
```



```
def factorial(x):
  if (x > 1):
    return x * factorial(x-1)
  else:
    return 1
```

Python - Pakete

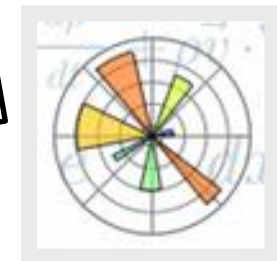
- Python ist in **itom** integriert
- **itom** can über ein Modul von Python gesteuert werden
- Python ist durch Pakete erweiterbar



Numpy
numerische Bibliothek



Scipy
wissenschaftl. Bibliothek



Matplotlib
plots und Grafiken



scikit-image
Bildverarbeitung

Python-Modul *itom*



„Die Brücke zwischen **itom** und Python“

```
>>> from itom import * <<<
```



- Menüs und Toolbars können zur **itom** Gui hinzugefügt werden und mit Python-Funktionen verknüpft werden



- Plugin Ansteuerung (*dataIO*, *actuator*)
- Algorithmen-Ausführung
- Online Hilfe für Plugins



- GUIs Erstellung zur Laufzeit mittels eines WYSIWYG editors
- Widget-Signale können mit Python-Funktionen verknüpft werden
- Widget-Eigenschaften ändern





Numerik Paket

- Unterstützt große mehrdimensionale Arrays
- Große Bibliothek von mathematischen Funktionen und Operatoren
- **Itom** DatenObjekt ist mit Numpy Arrays kompatibel

Beispiel: Löse $Ax=b$

```
from numpy import *
from numpy.linalg import solve

# The system of equations we want to solve for
# (x0,x1,x2):
# 3 * x0 + 1 * x1 + 5 * x2 = 6
# 1 * x0 + 8 * x2 = 7
# 2 * x0 + 1 * x1 + 4 * x2 = 8

a = array([[3,1,5],[1,0,8],[2,1,4]])
b = array([6,7,8])
x = solve(a,b)
print(x) # This is our solution
[-3.28571429  9.42857143  1.28571429]
```

- Array Erzeugung und Manipulation
- Binäre Operationen
- Lineare Algebra
- Masken
- Polynome
- Sortieren, Suchen, Zählen
- Fourier-Transformation
- ...

Wissenschaftl. Algorithmen

- Erweiterung von *numpy*
- Weitere Funktionen aus den Bereichen Numerik, Statistik und Optimierung
- Erweiterbar durch *scikits*

- Optimierung
- Lineare Algebra
- Integration
- Interpolation
- FFT
- Signal Verarbeitung
- Grundlegende Bildverarbeitung
- Sparse Matrizen

Example:

Find root of $x + 2\cos(x) = 0$ around $x = 0.3$

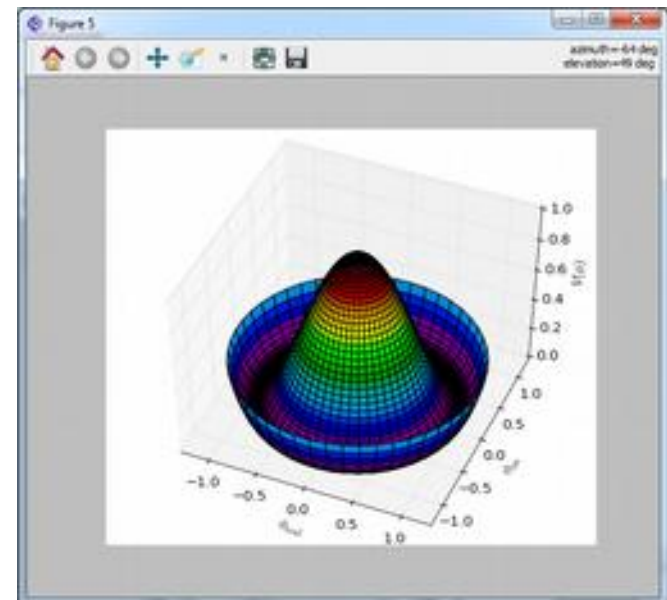
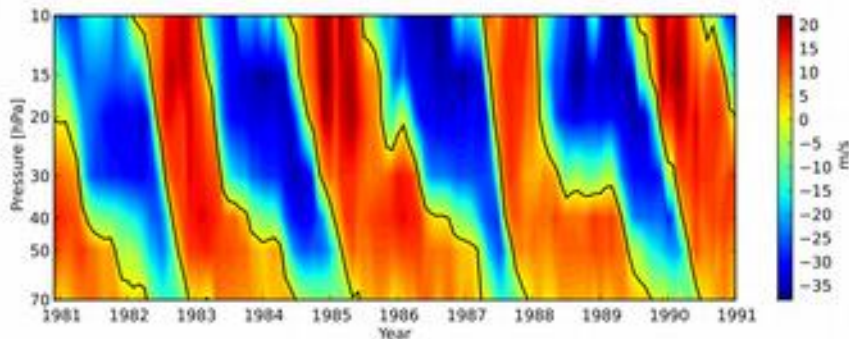
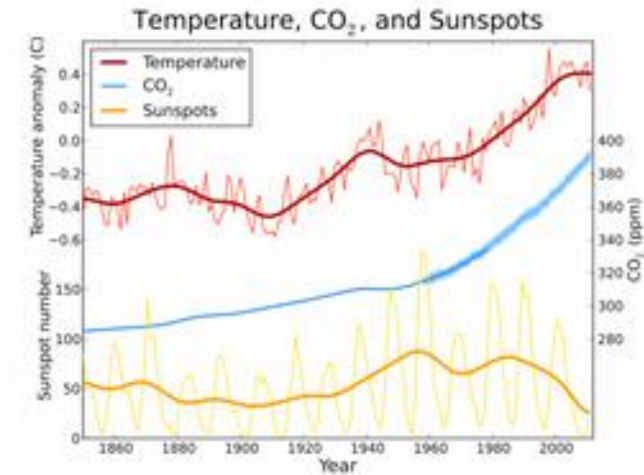
```
import numpy as np
from scipy.optimize import root

def func(x):
    return x + 2 * np.cos(x)

sol = np.root(func, 0.3)
sol.x
>>> array([-1.02986653])
sol.fun
>>> array([-6.66133815e-16])
```

Grafikpaket

- Python Paket für mathematische Plots
- Basiert auf *numpy*
- Syntax ähnlich Matlab
- Export in verschiedenen Bildformate: png, pdf, eps...
- Voll in **itom** integriert
- Kann in benutzererstellte GUIs integriert werden



Bildverarbeitungspaket

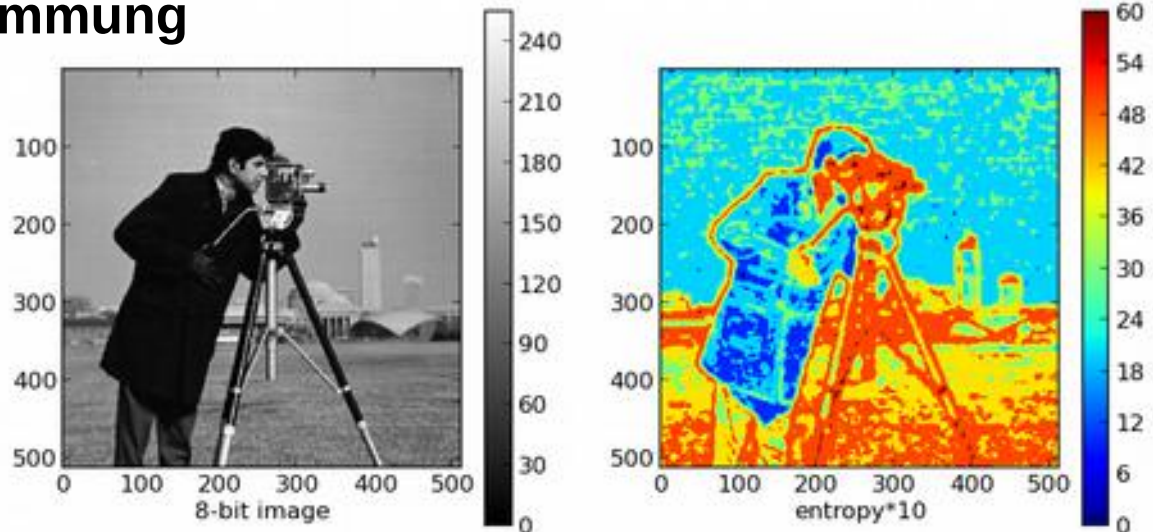
- Basiert auf Numpy arrays
- Algorithmen in Python und C geschrieben
- Benutzt Matplotlib zur grafischen Ausgabe

- Segmentierung
- Transformationen
- Morphologie
- Messen
- IO
- Filter
- Rank-Filter
- Feature Erkennung

Beispiel: Entropy Bestimmung

```
from skimage import data
from skimage.filter.rank import entropy
from skimage.morphology import disk
from skimage.util import img_as_ubyte

# defining a 8- and a 16-bit test images
a8 = img_as_ubyte(data.camera())
a16 = a8.astype(np.uint16) * 4
```



- **Itom** Einführung
 - Warum wurde **itom** entwickelt?
- **itom** GUI
- Hauptbestandteile
- Python Grundlagen und die wichtigsten Module
- **itom plugins**

- 🧩 Plugins erweitern die **itom** Funktionalität. Jedes plugin ist eine C++ Bibliothek (.dll, .so)
- 🧩 Jedes Plugin leitet sich von einer der drei Basisklassen ab (*DataIO*, *Actuator*, *Algorithm*)
- 🧩 Plugins (z.B. Kameras, Antriebe ...) können von Python aus geöffnet werden oder direkt über die **itom**-GUI

DataIO

- Kameras
- A/D-Wandler
- Serielle Schnittstellen

Actuator

- Motoren
- Mehrachs-Maschinen

Algorithm

- Algorithmen
- Filter
- Komplizierte GUIs

Hauptfunktionalität

- `getParam(..)` □ Parameter lesen
- `setParam(..)` □ Parameter setzen
- `startDevice()` □ Kamera starten
- `stopDevice()` □ Kamera stoppen
- `acquire()` □ Bild aufnehmen
- `getVal(..) / copyVal(..)` □ Bild an Python übertragen
- ...

Implementierungen

- Standard-USB Kameras
- CMU1394
- PCO Pixelfly
- PCO Kamera Interface
- Vistek GigE
- Ximea (USB3)
- PMD Kamera (Lynkeus)
- Allied Vision (Firewire)
- Dummy-Kamera

Live-Bilder können in einem separaten Fenster angezeigt oder in eigene GUIs integriert werden

Interface “actuator”

Hauptfunktionalität

- `getParam(..)` □ Parameter lesen
- `setParam(..)` □ Parameter setzen
- `getStatus(..)` □ Achsen Status lesen
- `getPos(..)` □ Position lesen
- `setPosAbs/Rel()` □ verfahren
- ...

Implementierungen

- Leica MZ12xx Actuator
- USB Motion 3XIII
- Uhl-Actuator (x,y,z)
- Galil DMC2123
- PI Piezo Controller
- PI-Hexapod
- Dummy-Motor
- Piezosysteme Jena Actuator
- CF30 Piezo Controller

Aktuatoren Positions- und Status-Signale können in der Haupt-GUI angezeigt werden

Interface “algo”

„Algo“ plugins beinhalten

- Numerische algorithmen
- GUI Elemente

Aufruf:

- Durch ein Python-Skript
- Durch andere Plugins

Jede Methode hat:

- Pflichtparameter (Typ, Beschreibung...)
- Optionale Parameter
- Rückgabewerte

Algorithmen

- Streifenprojektion
- Rauheitsparameter
- Numerische Filter (fft...)
- Fitting
- IO-Methoden
- ...

GUIs

- Visualisierung von 3D-Punktwolken
- ...

- **Itom** Einführung
 - Warum wurde **itom** entwickelt?
- **itom** GUI
- Hauptbestandteile
- Python Grundlagen und die wichtigsten Module
- **itom** plugins
- **Lizenz**

- **itom** (Hauptprogramm) ist “Open Source” (**LGPL**)
- **itom-SDK** (Standard Module der Hauptanwendung und der Plugins) werden unter der **LGPL-Lizenz** + **itom-Ausnahme**. Die itom Ausnahme erlaubt das Einbinden und Linken der SDK Komponenten in andere Komponenten unabhängig derer Lizenz.
- **Plugins** können beliebige Lizenzen haben (auch closed source). Das ITO stellt verschiedene Plugins unter der **LGPL** zur Verfügung.
- **Designer-Plugins** (plots...), ähnlich den anderen Plugins, können unter beliebiger Lizenz veröffentlicht werden.



- **Itom** Einführung
 - Warum wurde **itom** entwickelt?
- **itom** GUI
- Hauptbestandteile
- Python Grundlagen und die wichtigsten Module
- **itom** plugins
- Lizenz
- **Datenobjekt**

Ziel:

- Diverse Grundlegende Datentypen (auch komplexwertige)
- Verarbeitung großer, mehrdimensionaler Datensätze (z.B. Bilderstapel)
- Kompatibel mit Matlab, Numpy, OpenCV

Implementierung:

- *DataObject* ist sehr ähnlich wie OpenCV Datenstrukturen
- Unterstützte Datentypen: *int8*, *uint8*, *int16*, *uint16*, *int32*, *uint32*, *float*, *double*, *complex(float)*, *complex(double)*
- *DataObject* kann tags enthalten (Achseneinheiten, Beschreibungen, Titel...)

Bilderstapel (2D-Bilder)



3D Datenstapel

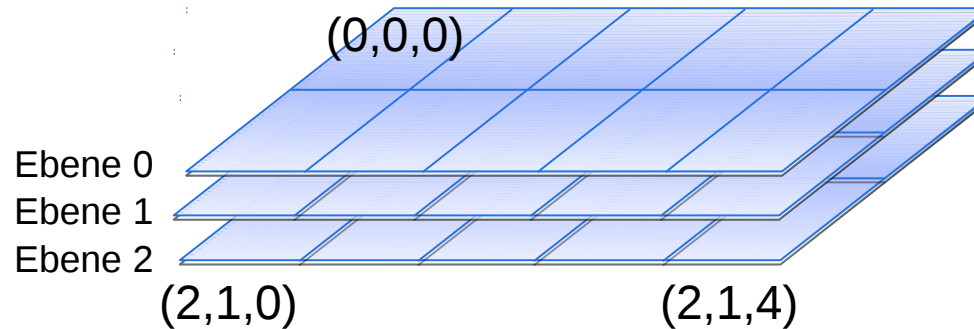
Ebene 0

Ebene 1

Ebene 2



Annahme: Sequenz von 2D-Bildern ($3 \times 2 \times 5$)



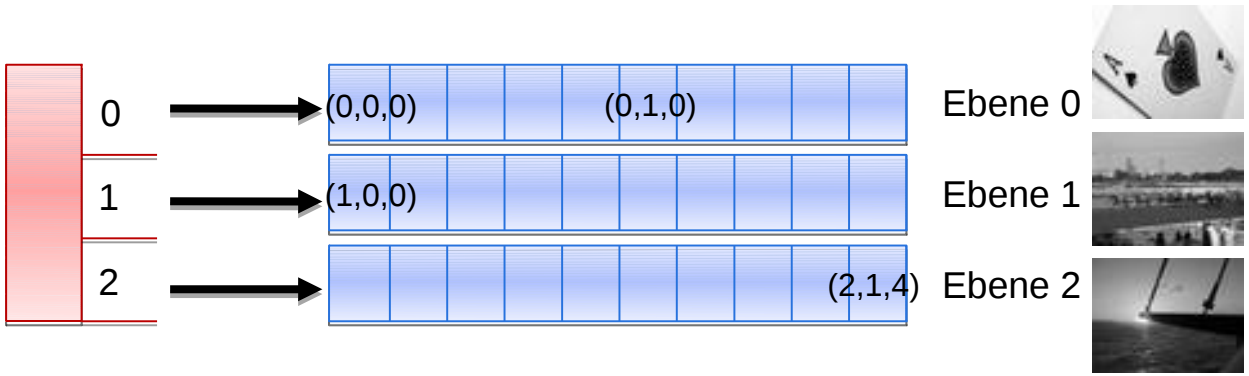
C / Matlab: kontinuierlicher Speicherbereich



- + Einheitlicher, schneller und einfacher Zugriff auf mehrdimensionale arrays
- Problem bei der Erzeugung „großer“ arrays

Datenobjekt

DataObject:



- + weniger Probleme bei der Erzeugung durch verteilte Speicherbereiche
- Etwas aufwändigerer Speicherzugriff

DataObject (kontinuierlich): Kompatibel mit C-arrays

