Linsenloses Mikroskop

Benutzerhandbuch

Eric Schmidt

YouTube:

GitHub: https://github.com/Nablaaa/Lensless-Microscope

Bei weiteren Fragen oder Problemen, schauen Sie in die FAQs (Kapitel 10) (frequently asked questions) oder gehen Sie auf meine YouTube oder GitHub Seite. Viel Spass!

Inhaltsverzeichnis

Über dieses Buch:

Dieses Handbuch begleitet Sie Schritt für Schritt auf den Weg zu einem kostengünstigen Mikroskop ohne Linsensystem. Es beinhaltet Informationen zur Hardware (Photodetektor) und zur Software (Python Programme). Es soll Schritt für Schritt durchgearbeitet werden um Probleme zu vermeiden. Falls trotzdem Probleme auftreten sollten, schauen Sie sich bitte die FAQs in Kapitel 10 an oder besuchen Sie die YouTube oder GitHub Seite.
Falls Sie mich bei diesem oder weiteren Projekten unterstützen wollen, geht das auf folgender Seite:

1

SECTION

Kurze Zusammenfassung über das Resultat

Hier kommen Code Stücke und Bilder vom Resultat hin. Außerdem noch eine kurze Zusammenfassung über Hardware und Software. ***Falls ich Text hervorheben will***

1.1 Subsection über Resultat

1.2 Hardware Liste

1.3 Software Liste

1.4 Verweis zu YouTube und GitHub

https://github.com/Nablaaa/Lensless-Microscope

1.5 Verweis zu Förderung

Falls ich einen link angeben will auf den leute spenden abgeben koennen: Farbig hinterlegter Text mit Link[1]

[1] https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes

2

SECTION

Installation und Einrichtung der Programmierumgebung

Wichtig

Dieser Abschnitt ist für diejenigen gedacht, die noch keine Programmierumgebung mit Python auf ihrem System haben. Falls Sie wissen, wie Sie Python Skripte schreiben und ausführen können, dann überspringen Sie diesen Abschnitt.

Um eine Kamera ansteuern zu können, benötigt man eine entsprechende Software. Da es mir darum geht, kostengünstig ein Mikroskop zu bauen, habe ich mich für die OpenSource Programmiersprache Pythonëntschieden. Python ist sehr beliebt unter Wissenschaftlern, da sie relativ einfach zu lernen ist und viele gut dokumentierte "Bibliotheken"hat. Eine Bibliothek im Programmier-Sinn ist ähnlich zu einer Bibliothek im echten Leben. Es ist eine Umgebung, in der viel Wissen gesammelt ist. Manchmal versteht man es, manchmal hat man noch Probleme es zu verstehen, aber immer ist es einem Möglich das Wissen von anderen kompakt nutzen zu können. Außerdem hat Python auch sehr viele Mitglieder, sodass Probleme schnell behoben werden können. Eine sehr empfehlenswerte Community ist Stack Overflow[2].
Um außerdem noch auf andere Programmiersprachen zugreifen zu können, werden wir Anaconda[3] als Entwicklungsumgebung nutzen.

[2] https://stackoverflow.com/
[3] https://www.anaconda.com/products/individual

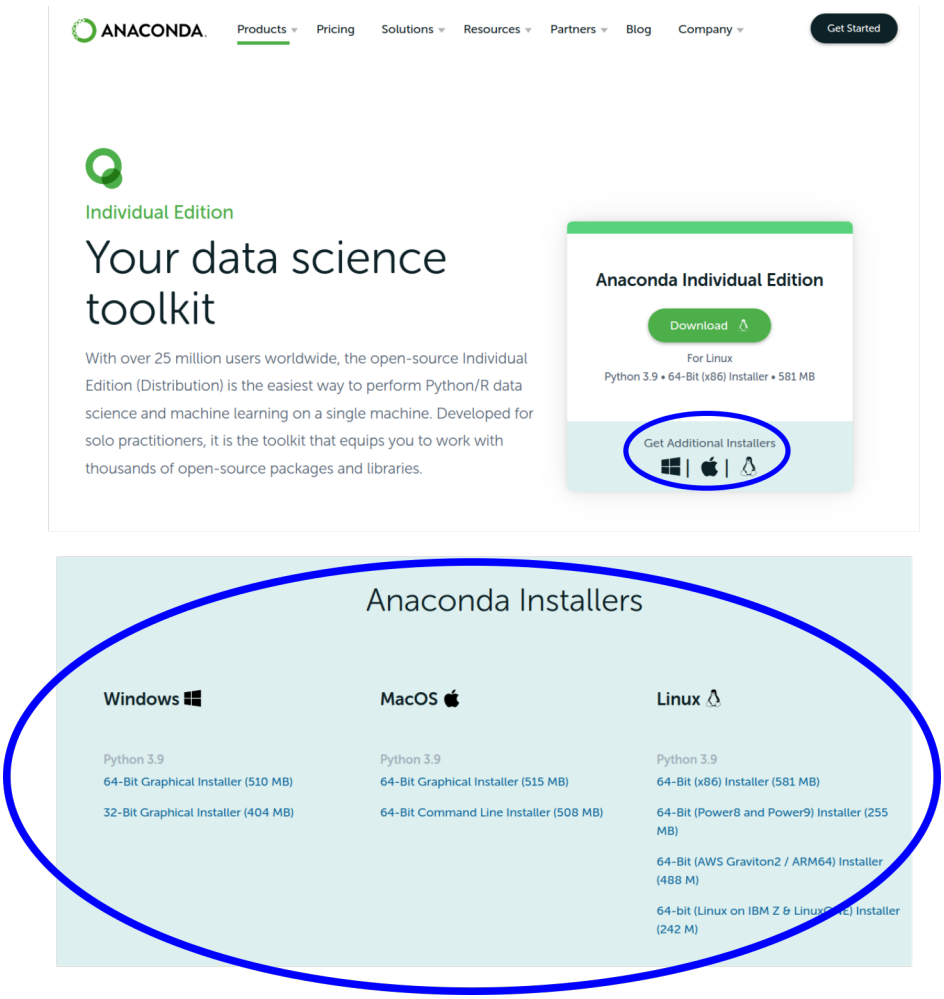


Abbildung 1: Installationsmöglichkeiten von Anaconda auf ihrer Webpage: `https://www.anaconda.com/products/individual` . Wählen Sie bitte ihr entsprechendes Gerät und den dazugehörigen Installierer aus.

2.1 Installiere Anaconda

Um Anaconda zu installieren, geben wir auf die Seite `https://www.anaconda.com/products/individual` und klicken entweder direkt auf Download, wenn die Voreinstellungen passen (in meinem Fall steht: For Linux, Python 3.9, 64-Bit"was genau zu meinem Computer passt) oder klicken Sie auf eines der Symbole der zusätzlichen Installierer (Windows, Mac, Linux). Falls das auch nicht klappt, dann scrollen Sie tiefer auf der Seite bis zu Änaconda Installers". Das ganze sieht dann in etwa so aus wie in Abbildung 1.
Ich vermute, dass der Großteil der Leser dieses Buches einen Windows Installierer für die 64-Bit Version benötigen. Falls weitere Probleme auftreten, schauen Sie bitte in die FAQs in Kapitel 10.1

2.2 Erstelle einen Arbeitsplatz im Desktop

Wichtig

Ich werde in diesem Abschnitt Kommandozeilen schwarz hinterlegen, z.B. `Beispiel` *. Diese Zeilen geben Sie bitte ein und bestätigen sie danach mit der ENTER Taste.*

Nun da Anaconda heruntergeladen ist, haben Sie Zugriff auf richtig viel Wissen. Sie wissen es nur noch nicht. Wir werden nun etwas machen, was sie vielleicht nicht auf Anhieb voll umfänglich verstehen werden, aber ich gebe mir Mühe, es einfach zu halten.
Als erstes beginnen wir mit etwas einfachem. Erstellen Sie einen Ordner auf Ihrem Startbildschirm (Desktop). Geben Sie dem Ordner einen Namen, z.B. 'Mikroskop'. In diesem Ordner werden wir alles machen, was mit diesem Projekt hier zu tun hat.
Nun beginnt der Teil, der für fast jeden Windows Benutzer seltsam ist, aber zugleich auch viel Potential hat. Es lohnt sich also sich damit zu befassen... Egal welches Betriebssystem Sie haben, öffnen Sie nun die 'Konsole', bzw. ist es bei Linux das 'Terminal'. Eine ausführlichere Beschreibung finden Sie in den FAQs in Kaptiel 10.2. Navigieren Sie nun mit den Befehlen (change directory):

Windows: `cd %HOMEPATH%`
Linux: `cd ~` (also mit einer Tilde hinter cd)
MacOS: angeblich hilft die Tastenkombination Fn+Shift+Links Pfeil
in Ihr 'Home' Verzeichnis und fahren Sie dann fort mit:
`cd Desktop/`
`cd Mikroskop/`

zu dem gewünschten Ordner. Falls eine Fehlermeldung kommen sollte dann lesen Sie sich diese durch und versuchen das Problem selbst zu verstehen. Beispielsweise würde die Fehlermeldung 'No such file or directory' kommen, wenn der Ordner 'Mikroskop' noch nicht erstellt ist oder ein Tippfehler im Namen aufgetreten ist.

2.3 Richte eine Programmierumgebung ein

Nun können Sie Anaconda mit dem Befehl: `conda activate` aktivieren (wieder im Terminal/in der Konsole) und eine Programmierumgebung erstellen. Diese Umgebung mit dem Namen 'Mikroskop_Umgebung' erstellen Sie mit dem Befehl: `conda create --name Mikroskop_Umgebung`
Nun ist die Umgebung erstellt, aber wir sind noch nicht drin. Um rein zu kommen tippen wir `conda activate Mikroskop_Umgebung` ein und geben dann als erstes: `conda install python` ein, um Python herunterzuladen (Sie werden die Frage erhalten, ob Sie es wirklich installieren wollen. Klicken Sie den Buchstaben 'y' auf der Tastatur und bestätigen mit ENTER.). Außerdem geben wir noch `conda install jupyter` ein, da wir es später benötigen (Sie werden wieder gefragt, ob Sie es wirklich installieren wollen. Auch hier klicken Sie 'y' und ENTER). Alles weitere wird in den entsprechenden Kapiteln erklärt.

Jetzt sind wir in der komplett frischen Programmierumgebung. Diese ist noch fast leer und somit frei von Komplikationen. Das ist perfekt um einfach drauf los zu programmieren!
Probieren Sie es aus. Sie sind in Ihrer Konsole/Ihrem Terminal und Ihre Umgebung ist aktiviert. Geben Sie ein:
`python` und dann als erstes
`print('Hello World')`
Da Sie ja immer hinter schwarz hinterlegten Kommandozeilen schön ENTER drücken, wird Ihnen nun in der nächsten Zeile 'Hello World' angezeigt. Ein anderer Test ist der Befehl:
`print(7*3 + 5)`
Das Ergebnis sollten Sie nun in der nächsten Zeile sehen. Ich hoffe es stimmt mit dem Überein, was Sie erwartet haben.
Um nun wieder aus python raus zu kommen, tippen Sie ein:
`exit()` und bestätigen Sie mit ENTER.
Falls etwas nicht so richtig funktioniert, schauen Sie bitte noch einmal auf meinem YouTube Kanal[4] nach.

2.4 Texteditor zum Programmieren aussuchen und ersten Code schreiben

Um nun tatsächlich zu programmieren benötigen wir etwas übersichtlicheres als die Kommandozeile. Persönlich favorisiere ich es, einen normalen vorinstallierten Texteditor auf Linux zum Programmieren zu wählen. Auf Windows kann ich Geany[5] empfehlen. Geany ist auf Linux bereits vorinstalliert. Bei Installationsproblemen schauen Sie bitte in die FAQs in Kapitel 10.3

[4] Platzhalter:LinkzuInstallationsvideos
[5] https://www.geany.org/download/releases/

3

SECTION

Kamera Testprogramm

Wichtig

Falls Sie einen Prozess in der Konsole unterbrechen wollen (z.B. falls sie die Kamera schließen wollen) dann können Sie jederzeit die Tastenkombination `Strg + C` *in der Konsole drücken. Damit unterbricht man den Prozess. Später lernen Sie noch eine andere Möglichkeit.*

Wir kommen nun zur ersten Action. Packen Sie das Kameramodul aus Ihrer Box aus und verbinden Sie die Kamera mit dem USB Eingang Ihres Gerätes. Unter der Annahme, dass die Kamera richtig verbunden ist, können wir nun das Texteditorprogramm unserer Wahl öffnen (Falls Sie das nicht können, schauen Sie in die Installationsanweisungen in Kapitel 2.4).
Starten Sie ein neues Dokument und nennen Sie es zum Beispiel: '01_Videos_mit_Kamera.py' wobei die Endung '.py' dem Texteditor sagt, dass es sich um die Programmiersprache Python handelt. Speichern Sie das Dokument mit diesem Namen. Nun kann das Programmieren beginnen.

3.1 Lade Bibliotheken herunter

Wir wollen in diesem Handbuch das Kameramodul mit der Bibliothek cv2[6] von opencv bedienen. Außerdem wollen wir noch Zeitmessungen durchführen. Dafür verwenden wir die Bibliothek time[7]. Im Normalfall ist *time* schon vorinstalliert, aber opencv muss erst noch heruntergeladen werden.
Nun kommt die Magie von Anaconda Umgebungen und Python. Um eine Bibliothek herunterladen zu können, müssen wir lediglich die Umgebung mit unserem Terminal/der Konsole öffnen (Der Befehl ist bereits bekannt: `conda activate Mikroskop_Umgebung`) und in der aktivierten Umgebung tippen wir einfach

```
pip install opencv-python
```

ein. Falls das unter Windows nicht klappt, probieren Sie es mit

```
pip3 install opencv-python
```

3.2 Importiere Bibliotheken

Als aller erstes macht es beim Programmieren Sinn, die Bibliotheken (oder 'Module') zu installieren, die man permanent braucht. Die ersten Zeilen Code sind daher: für den Zugriff auf die

```
import cv2
import time
```

Kamera und für Zeitmessungen. Wenn wir später unser erstes Modul selber schreiben werden, werden Sie besser verstehen, was genau passiert. Für jetzt lassen wir es einfach unkommentiert.

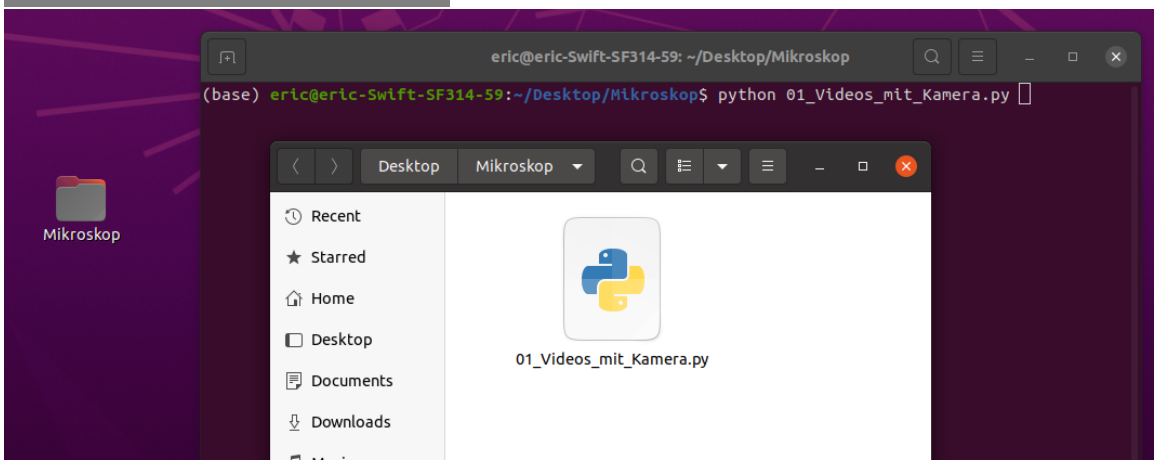
3.3 Schreibe das erste Programm und führe es aus

Greifen wir also nun auf cv2 zu, um unsere Kamera zu initialisieren. Dafür müssen wir dem Programm sagen, welche Kamera wir nutzen wollen. Ich arbeite an einem Laptop. Dort hat die eingebaute Kamera die Nummer 'Null'. Falls Sie am Laptop arbeiten, probieren Sie daher kameraNummer = 1 oder 2 oder 3 oder ... um das USB Kameramodul anzuschließen. Sie werden es herausfinden!

```
kameraNummer = 0
Kamera = cv2.VideoCapture(kameraNummer) # Initialisierung

while True:
        success, img = Kamera.read() # Foto machen
        cv2.imshow("Image",img) # Foto anzeigen
        cv2.waitKey(1) # 1 ms Bild einblenden, bevor es weiter geht
```

[6] https://pypi.org/project/opencv-python/
[7] https://docs.python.org/3/library/time.html

Beachten Sie den Hinweis am Anfang des Kapitels!
Schreiben/kopieren Sie diese Zeilen und speichern Sie die Datei ab. Je nachdem, wie Sie Ihren Texteditor gewählt und eingerichtet haben müssen Sie jetzt entweder (bei Geany) nur auf den grünen dreieckigen Startbutton klicken oder falls Sie das Programm über eine Konsole laufen lassen wollen, dann müssen Sie nun zu dem Ordner navigieren (z.B. mit `cd Desktop/Mikroskop/`). Im Ordner angelangt müssen Sie nun das Programm mit Python zum Laufen bringen. Das geht mit
`python 01_Videos_mit_Kamera.py` (Linux, Mac) oder
`python3 01_Videos_mit_Kamera.py` (Windows)



Nun startet das Programm und dann sollten Sie auf Ihrem Computer das sehen, was das Kameramodul sieht. Falls Sie die Fehlermeldung 'can't open camera by index' erhalten, dann müssen Sie die 'kameraNummer' Variable verändern. Probieren Sie einfach die natürlichen Zahlen durch.

3.4 Füge dem Video eine Bildrate hinzu

Herzlichen Glückwunsch! Nun haben Sie Zugriff auf die Kamera. Jetzt können wir noch ein paar Modifikationen machen, um ein Gefühl für die Aufnahmezeit zu bekommen. Eine typische Größe dafür sind "Bilder pro Sekunde"(frames per second, fps). Ein Fernseher oder Handy spielt Bilder mit circa 60 fps ab, sodass das menschliche Auge nicht mehr realisieren kann, dass es einzelne Bilder sind. Wir sprechen dann von einem Video. Um die Zeit pro Bild zu bestimmen berechnen wir einfach die Differenz der Zeit vor dem Bild und nach dem Bild. Die Bildrate in Bildern pro Sekunde ist dann das multiplikative Inverse von diesem Wert. Man braucht für die Bildrate also immer eine vorherige Zeit und eine aktuelle Zeit direkt nach der Aufnahme oder wahlweise auch genau bevor das nächste Foto gemacht wird (praktisch liegt darin kaum ein Unterschied bei der Kürze des Programmes). Die aktuelle Zeit nach dem Bild wird gleichzeitig die vorherige Zeit für das darauf folgende Bild. Das ganze sieht dann in etwa so aus:

```
kameraNummer = 0
Kamera = cv2.VideoCapture(kameraNummer) # Initialisierung
vorherige_Zeit = 0

while True:
        success, img = Kamera.read() # Foto machen
        aktuelle_Zeit = time.time()
        zeit_differenz = aktuelle_Zeit - vorherige_Zeit
        fps = 1/zeit_differenz

        vorherige_Zeit = aktuelle_Zeit # aktuelle Zeit von Bild i wird zur
                                       # vorherigen Zeit von Bild i + 1
        print(fps) # Gebe Bildrate aus

        cv2.imshow("Image",img) # Foto anzeigen
        cv2.waitKey(1) # 1 ms Bild einblenden, bevor es weiter geht
```

Hier wird die Bildrate berechnet und in der Konsole/in dem Texteditor mit dem print Befehl angezeigt. Eine elegantere Version ist es allerdings, die Bildrate direkt im Bild anzuzeigen, wenn man in der Testphase ist.
Man kann das mit dem Befehl erreichen. Ersetzen Sie einfach den print Befehl im Programm.

```
cv2.putText(img, str(fps), (10, 70), cv2.FONT_HERSHEY_PLAIN,
                           3, (255, 100, 0), 3)
```

Die *cv2.putText* Funktion schreibt den Text (in unserem Fall ist es die Variable fps als string=text Format) in das Bild (hier: img) in der Position (10,70), mit der Schriftart Hershey plain, mit einer Größe von 3, einer Farbe von (BGR) = (blau grün rot) = (255, 100, 0) und einer Strichdicke von 3. Probieren Sie es einfach aus und verändern die Werte ein bisschen.

3.5 Baue einen Timer für das Video

Manchmal ist es sinnvoll, ein Video nur für ein paar Sekunden zu zeigen und es dann wieder zu schließen. Das kann man mit einem Timer realisieren. Eine einfache Möglichkeit ist es, die while=True Schleife zu modifizieren. Z.B. kann man folgende Bedingung einbauen:

```
# stelle ein, wie lange die Kamera laufen soll
startzeit = time.time()
laufzeit = 5 # s
schleife = True

while schleife == True:
    # wenn die laufzeit ueberschritten wird, wird die schleife auf
    # False gesetzt und endet somit
    if time.time() - startzeit >= laufzeit:
        schleife = False
```

Man definiert sich also erste eine Laufzeit und setzt die Variable SSchleifeäuf wahr und befindet sich dann so lange in der while Schleife, bis die Laufzeit überschritten wird. Dann wird die schleife auf False gesetzt und somit die while Bedingung beendet.
Der komplette Code sieht dann so aus:

```
# importiere "Bibliotheken" die einen Grossteil der Arbeit machen
import cv2 # Zugriff auf Kamera
import time # Zeitmessungen

# suche eine Kamera aus
# (0 is die eingebaute kamera im laptop, 1,2,3,... sind weitere Anschluesse)
Kamera = cv2.VideoCapture(2) # der name Kamera ist nun bereit und kann
                             # verandert werden. Zum Beispiel kann die
                             # Aufloesung eingestellt werden

# stelle die Aufloesung in pixel ein
wCam, hCam = 640, 480 # weite und hoehe des Bildes
Kamera.set(3, wCam)
Kamera.set(4, hCam)

# optional wollen wir hier die Anzahl der Bilder pro Sekunde (fps)
# berechnen. Dazu muss die Zeit gemessen werden, die fuer ein Bild
# gebraucht werden.
#Hier wird die Zeit initialisiert.
vorherige_Zeit = 0
```

```
# stelle ein, wie lange die Kamera laufen soll
startzeit = time.time()
laufzeit = 5 # s
schleife = True

# Fuehre den folgenden Abschnitt in einer Schleife aus, bis der Wert False
# uebergeben wird
while schleife:
        # Mache ein Foto und speichere es als "img"
        success, img = Kamera.read()

        # Messe die Zeit, die pro bild gebraucht wird.
        aktuelle_Zeit = time.time()
        zeit_differenz = aktuelle_Zeit - vorherige_Zeit

        # Reziproker Wert ist die Anzahl der Bilder pro zeit
        fps = 1/zeit_differenz  # (frames per second)

        vorherige_Zeit = aktuelle_Zeit

        # schreibe die Zeit in das Display
        text = str(int(fps)) # string = textformat
                             # int = integer (Ganze Zahl... ohne Komma)

        # Befehl, um Text in das Bild (img) einzubauen
        # Position, Schriftart, Schriftgroesse, (BGR) Farbcode, Strichdicke
        cv2.putText(img, text, (10, 70), cv2.FONT_HERSHEY_PLAIN,
                            3, (255, 100, 0), 3)


        # Zeige das Foto an
        cv2.imshow("Image",img)
        cv2.waitKey(1)

        # wenn die laufzeit ueberschritten wird, wird die
        #  schleife auf False gesetzt und endet somit
        if time.time() - startzeit >= laufzeit:
                schleife = False
```

4

SECTION

Videos speichern

In diesem Abschnitt wollen wir nun die Videoaufnahmen auch speichern. Starten Sie dazu ein neues Dokument und nennen Sie es zum Beispiel: '02_Speichere_Videos.py' wobei auch hier wieder die Endung '.py' dem Texteditor sagt, dass es sich um die Programmiersprache Python handelt. Speichern Sie das Dokument mit diesem Namen. Nun kann das Programmieren beginnen.

4.1 Lade Bibliotheken herunter

Aus Teil 1 kennen wir bereits cv2 und time als Bibliotheken, jetzt benutzen wir noch os[8] und imageio[9] für das erstellen von einem Speicherordner und für das erstellen von Videodateien.
Diese können wie immer mit Hilfe von

```
pip install os
pip install imageio
```

direkt in der Konsole heruntergeladen werden (siehe Kamera Testprogramm).
Falls das unter Windows nicht klappt, probieren Sie es wieder mit pip3 an Stelle von pip.

[8] https://www.geeksforgeeks.org/os-module-python-examples/
[9] https://imageio.readthedocs.io/en/stable/

4.2 Importiere Bibliotheken

Beginnen wir nun das Programmieren wieder mit dem Installieren von den Modulen:

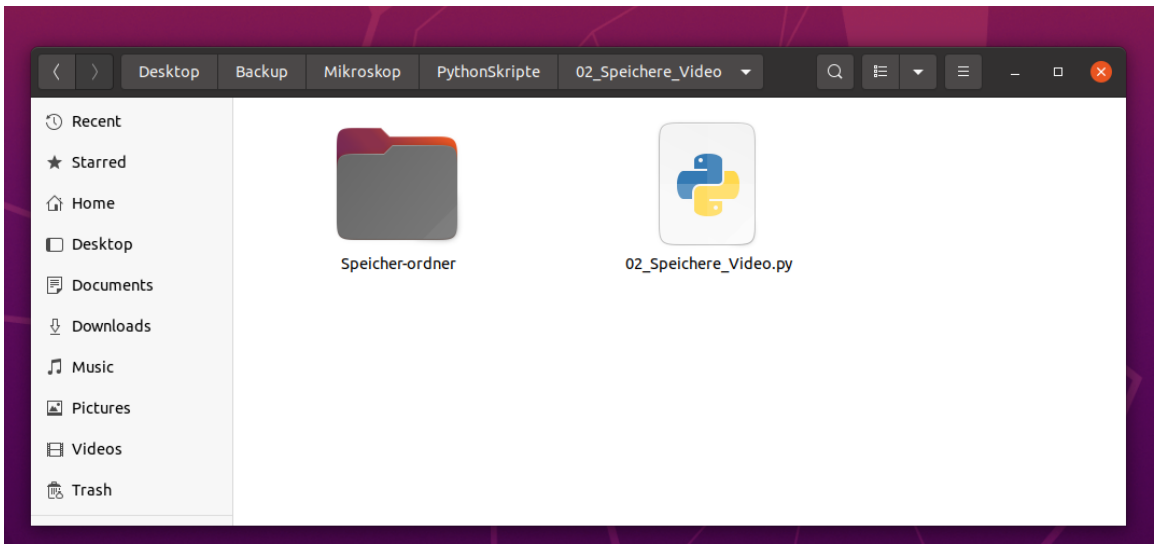
```
import cv2
import time
import os
import imageio
```

4.3 Erstelle einen Speicherordner

Als erstes erstellen wir einen Speicherordner für unsere Dateien. Das machen wir mit Hilfe der os Bibliothek, wie folgt:

```
# gib dem Ordner und der Datei einen Namen
Speicherort = "Speicher-ordner"
gif_name = "Videodatei.gif"
gif_pfad = Speicherort + '/' + gif_name

# Probiere den Ordner zu erstellen (mit 'os' modul)
try:
    os.mkdir(Speicherort)

# Falls ein Fehler auftritt gib eine Fehlermeldung aus
except OSError:
    print ("Der Ordner %s wurde nicht erstellt (z.B. weil er schon existiert)"
                    % Speicherort)

# ansonsten gib eine Erfoglsmeldung aus
else:
    print ("Der Ordner %s wurde erfolgreich erstellt" % Speicherort)
```

In der Fehler-/Erfolgsmeldung sehen Sie das Prozentzeichen, gefolgt von einem s (für 'String') %s was dann durch die Variable ersetzt wird. Z.B. lautet die Erfolgsmeldung hier:
Der Ordner Speicher-ordner wurde erfolgreich erstellt. Diesen erstellten Ordner können Sie dann auch in Ihrem Dateimanager sehen.



4.4 Mache und speichere Videos

Als nächstes greifen wir wieder auf die Kamera zu und machen Videos. Hier machen zwei Sachen Sinn. Entweder man macht Videos für eine gewissen Zeit (z.B. 100 Sekunden, dauerorientiert) oder man macht eine gewisse Anzahl an Bildern (z.B. 50 Bilder, dimensionen orientiert). Da wir ersteres bereits programmiert haben, gehen wir nun der zweiten Variante nach. Diese stellt sicher, dass alle Videos immer die gleiche Anzahl an Zeitpunkten hat. Damit stimmen die Videos in den Dimensionen x, y und t überein.
Neben den bekannten Zeilen vom Programm, kommt auch noch etwas neues und zwar: Diese

```
with imageio.get_writer(gif_pfad, mode='I') as writer:
```

Zeile sorgt dafür, dass wir einen 'Writer' also einen Schreiber für Videodateien erzeugen. Dieser Writer erzeugt dann unsere Videos, indem er alle einzelnen Bilder hintereinander speichert. Er benötigt den Pfad für das Video (gif_pfad) und den Modus, welcher 'I' ist und 'Image' bedeutet, da wir Bilder hineingeben. Heraus kommen dann Videos (hintereinander gehangene Bilder).
Richtig kurz gehalten kann das dann so aussehen:

```
Kamera = cv2.VideoCapture(0)
anzahl_Bilder = 42

with imageio.get_writer(gif_pfad, mode='I') as writer:
    for i in range(anzahl_Bilder):
        success, img = Kamera.read()
        cv2.imshow("Image",img)
        writer.append_data(img)
        cv2.waitKey(1)
```

Wenn Sie diese Zeilen code an die vorherigen (import ... und Ordner erstellen ...) anhängen und laufen lassen, wird ein Video in den Ordner gespeichert. Dieses Video besteht aus 42 Bildern und ist im gif Format. 42 Bilder erreicht man dadurch, dass der Code durch eine 'for' Schleife läuft, die von 0 bis 41 (also 42 Elemente insgesamt) zählt. Denn ausgesprochen bedeutet beispielsweise **for i in range(5):** , dass i in der Liste [0, 1, 2, 3, 4] alle Zahlen einmal annimmt. Bei weiteren

Fragen, googlen Sie nach 'for-Schleife'.
Ihnen fällt beim Anschauen des Videos dann sicher auf, dass die Farben falsch eingestellt sind. Das liegt daran, dass die Farben rot und blau vertauscht sind (probieren Sie es aus). Die Kamera von cv2 arbeitet nämlich mit dem BGR anstelle des RGB Formates! cv2 weiß das und imshow funktioniert entsprechend richtig, aber imageio weiß das nicht und verlangt aber ein RGB Format. Dadurch werden R und B vertauscht. Beheben kann man das, indem man eine weitere Zeile einfügt:

```
with imageio.get_writer(gif_pfad, mode='I') as writer:
    for i in range(anzahl_Bilder):
        success, img = Kamera.read()
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        writer.append_data(imgRGB)
        cv2.waitKey(1)
```

Herzlichen Glückwunsch! Sie wissen nun, wie man auf die Kamera zugreift, wie man Videos mit Timer macht und/oder wie man eine gewisse Anzahl an Videos macht. Außerdem haben Sie gelernt, wie man Ordner erstellt und Dateien darin speichert. Als nächstes werden Sie lernen, wie Sie verschiedene Einstellungen an der Kamera vornehmen können. Aber zuerst noch einmal das vollständige Programm dieses Kapitels:

```
import cv2
import time
import imageio # videos speichern
import os # Verzeichnisse (Ordner) erstellen

# Erstelle Speicherordner und gebe der Datei einen Namen
# beide Infos ergeben den Speicherpfad
Speicherort = "Speicher-ordner"
gif_name = "Videodatei.gif"
gif_pfad = Speicherort + '/' + gif_name

# Probiere den Ordner zu erstellen (mit 'os' modul)
try:
    os.mkdir(Speicherort)

# Falls ein Fehler auftritt gib eine Fehlermeldung aus
except OSError:
    print ("Der Ordner %s wurde nicht erstellt" % Speicherort)

# ansonsten gib eine Erfoglsmeldung aus
else:
    print ("Der Ordner %s wurde erfolgreich erstellt" % Speicherort)
```

```
# initialisiere Kamera
Kamera = cv2.VideoCapture(0)

# gebe an wie viele aufnahmen gemacht werden sollen
anzahl_Bilder = 42

# kommando zum speicher der fotos
with imageio.get_writer(gif_pfad, mode='I') as writer:

        # wiederhole vorgang "anzahl_Bilder" mal
        for i in range(anzahl_Bilder):
                success, img = Kamera.read()

                # Zeige die Bilder im Display an
                cv2.imshow("Image",img)

                # Konvertiere Format von BGR (blau gruen rot)
                # zu RGB (rot gruen blau)
                # und speichere es mit 'writer'
                imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                writer.append_data(imgRGB)

                cv2.waitKey(1)
```

5

SECTION

Kameraeinstellungen verändern

In diesem Abschnitt wollen wir nun Einstellungen an der Kamera vornehmen, die später beim Experimentieren wichtig werden können. Dazu gehören Einstellungen, wie die Sättigung, der Kontrast oder auch die Helligkeit. Bei guten Kameras kann man zusätzlich noch die Belichtungszeit, den Weissabgleich und vieles mehr einstellen. Beginnen wir also. Starten Sie dazu ein neues Dokument und nennen Sie es zum Beispiel: '03_Weitere_Einstellungen.py' und speichern Sie die Datei. Da wir keine neuen Bibliotheken hier benutzen werden, können die import Befehle die gleichen bleiben (os, time, cv2, imageio). Imageio ist wichtig, da wir auch hier wieder Videos speichern wollen.

Wichtig

Dieser Programmcode wird der Unübersichtlichste und wird in den nächsten Kapiteln in einer sogenannten Klasse strukturiert.

5.1 Wiederhole Bekanntes

Wie immer beim Programmieren, kann man alt bewährtes wieder und wieder verwenden. Kopieren Sie daher die Zeilen Code vom vorherigen Skript, welche die Bibliotheken importiert und den Ordner erstellt haben. Den Ordner und die gif Datei können Sie natürlich umbenennen. Ich werde die Datei Weitere_Einstellungen.gif nennen.
Danach können wir die Kamera initialisieren, die Anzahl der Bilder festlegen und die ersten Einstellungen vornehmen.

5.2 Bereite Einstellungen vor

Zwei Einstellungen können wir sofort setzen, da sie sehr klar sind. Diese sind die Bildweite und die Bildhöhe. Gemacht wird das wie folgt: Wir erstellen also ein Objekt mit dem Namen Kamera.

```
# initialisiere Kamera
Kamera = cv2.VideoCapture(2)
anzahl_Bilder = 300

wCam, hCam = 640, 480 # weite und hoehe des Bildes
Kamera.set(3, wCam)
Kamera.set(4, hCam)
```

Diesem Objekt geben wir die Eigenschaft von Weite und Höhe mit Hilfe der set Funktion. Diese nimmt 2 Werte entgegen. Der erste Wert entspricht der Einstellung, wie man es im Troubleshooting[10] oder auf der offiziellen Seite[11] nachlesen kann. 3 und 4 repräsentieren hierbei wie gesagt die Weite und die Höhe des Bildes. Der zweite Inputwert ist der den wir einsetzen, also hier 640 px für die Weite und 480 px für die Höhe.
Alle anderen Werte (wie Helligkeit oder Kontrast) können nicht so intuitiv gesetzt werden wie die Höhe und Weite, weshalb wir Sie Stück für Stück durchprobieren wollen. Dabei lernen Sie, dass man mit Schleifen wunderbar Listen aus Zahlen aber auch aus Wörtern durchlaufen kann.
Bereiten wir also alle Einstellungen vor: Wir geben also die Namen der Modi an, dann den zu-

```
namen = ['Helligkeit', 'Kontrast', 'Saettigung', 'Hue',
                        'Gain', 'Belichtung', 'Weissabgl.']
modi = [10, 11, 12, 13, 14, 15, 17]
initialisierungswerte = [20,20,50,10,0,0,0]
```

gehörigen Zahlenwert (3 = Weite, 4 = Höhe, 10 = Helligkeit, 11 = Kontrast, ...) und letzten Endes einen Initialisierungswert, mit dem wir starten wollen.

Wichtig

Bisher sind diese Werte nur in einer Liste und NOCH NICHT der Kamera überliefert worden. Das passiert erst mit der Kamera.set() Funktion

[10] https://stackoverflow.com/questions/11420748/setting-camera-parameters-in-opencv-python
[11] https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html#gaeb8dd9c89c10a5c63c139bf7c4f5704d

Beginnen wir also damit, in den Videoaufnahme Modus zu gehen und die Kamera zu initialisieren.

```
with imageio.get_writer(gif_pfad, mode='I') as writer:
        # gehe durch alle Modi und Namen
        for modus, name in zip(modi, namen):
                print(modus, name)
                print("Initialisierung")

                # initialisiere die Kamera indem ALLE
                # werte auf standard gesetzt werden
                for m, wert in zip(modi, initialisierungswerte):
                        Kamera.set(m, wert)

                time.sleep(1)
```

Hier passieren mehrere Sachen. Als erstes öffnen wir den writer und starten dann eine Schleife, bei der die Werte modus und name aus den Listen modi und namen angenommen werden, also z.B. (modus, name) ist gleich (11, Kontrast). Das ist praktisch damit wir der Kamera den Zahlenwert übergeben können und uns zum Verständnis den Namen.
Danach beginnt die Initialisierung der Kamera, bei dem wir jedem Modus (Kontrast, Helligkeit, etc.) den Initialisierungswert aus der Initialisierungswert Liste zuordnen wollen. Hier verwenden wir nun tatsächlich **Kamera.set(Modus-Zahl, Initialisierungs-Zahl)** und weißen damit der Kamera die Wertepaare für die Helligkeit (Modus, Wert) = (10, 20), Kontrast = (11, 20), Sättigung = (12, 50) und so weiter zu. Nach der Zuweisung machen wir eine Sekunde lang Pause, damit es benutzerfreundlicher wird.

5.3 Probiere Einstellungen aus

Da nun alle Vorbereitungen getroffen sind, können wir verschiedene Einstellungen ausprobieren. Das ganze machen wir systematisch, indem **immer nur eine** Einstellung (Variable) verändert wird. Das Ziel sollte es sein, einen großen Zahlenbereich abzudecken (also zum Beispiel das Intervall von - 300 bis 300). Außerdem sollten Sprünge vermieden werden (also von 0 bis 300 und von - 300 bis 0 sollte vermieden werden). Auch sollte bei 0 gestartet werden, da manche Einstellungen nicht unter 0 definiert sind und negative Zahlen als maximaler Wert interpretiert werden.
Dieser Ansatz führt zu einer Funktion, die im ersten Drittel des Videos linear von 0 bis 300 in 3-er Schritten zählt. Im zweiten Drittel und dritten Drittel des Videos dann aber von 300 zu minus 300 in erneut 3-er Schritten zählt. Das ganze sieht so aus:

```
# bereits bekannt:
for m, wert in zip(modi, initialisierungswerte):
        Kamera.set(m, wert)
time.sleep(1)

# jetzt neu:

# mache nun pro Modus eine gewissen Anzahl an Bildern uns speicher sie
for i in range(anzahl_Bilder):
        # Teste nun verschiedene Werte durch
        # erhoehe wert im ersten Drittel des Videos (0 bis 300)
        if i < anzahl_Bilder/3:
                einstellungswert = 3 * i

        # fuer den Rest des Videos gehe bis in den anderen
        # Bereich des Einstellungsbereiches (von 300 zu -300)
        else:
                einstellungswert = anzahl_Bilder - 3 * (i- int(anzahl_Bilder/3))

        # gebe den Wert an
        Kamera.set(modus, einstellungswert)
```

Die if-Schleife ist dabei intuitiv, die else-Schleife sollte genauer betrachtet werden. Beim Einsetzen von ein paar Werten von i und bekannter Zahl 'anzahl_Bilder' = 300 sollte das aber kein Problem für Sie darstellen.
Danach werden die Einstellungen mit dem Kamera.set() Befehl auf die Kamera übertragen. Danach folgt der bekannte Abschnitt des Ausgabetextes und des Video konvertierens und speicherns. Als Ausgabetext macht es Sinn, den Einstellungsmodus (Helligkeit, Kontrast, ...), den zugehörigen Wert (von 0 bis 300 bis minus 300) und vielleicht noch andere Parameter, wie die Aufnahmezeit und die Laufvariable i auszugeben. Das ist nun aus dem vorherigen Kapitel bekannt.

Wichtig

Je nachdem, welche Kamera Sie verwenden, kann es sein, dass Sie manche Einstellungen nicht setzen können. Bei meiner Kamera zum Beispiel kann ich die Belichtungszeit nicht automatisch verändern, entsprechend wird Modus 15 bei mir keine Funktion haben, egal welchen Initialisierungswert ich setze.

Das ganze Skript sieht wie folgt aus:

```
import cv2
import time
import imageio
import os

# Erstelle Speicherordner und gebe der Datei einen Namen
Speicherort = "Videoordner"
gif_name = "Weitere_Einstellungen.gif"
gif_pfad = Speicherort + '/' + gif_name

# Ordner erstellen
try:
    os.mkdir(Speicherort)

except OSError:
    print ("Der Ordner %s wurde nicht erstellt" % Speicherort)

else:
    print ("Der Ordner %s wurde erfolgreich erstellt" % Speicherort)
# initialisiere Kamera
Kamera = cv2.VideoCapture(0)
anzahl_Bilder = 300

# einfach zu verstehende Einstellungen
wCam, hCam = 640, 480 # weite und hoehe des Bildes
Kamera.set(3, wCam)
Kamera.set(4, hCam)

# Vorbereitungen
namen = ['Helligkeit', 'Kontrast', 'Saettigung', 'Hue',
                        'Gain', 'Belichtung', 'Weissabgl.']
modi = [10, 11, 12, 13, 14, 15, 17]
initialisierungswerte = [20,20,50,10,0,0,0]

# für Ausgabe der Zeit
startzeit = time.time()
vorherige_Zeit = startzeit
```

```
# Speichere videos mit writer
with imageio.get_writer(gif_pfad, mode='I') as writer:
        # gehe durch alle Modi und Namen
        for modus, name in zip(modi, namen):
                print(modus, name)
                print("Initialisierung")

                # Initialisierung
                for m, wert in zip(modi, initialisierungswerte):
                        Kamera.set(m, wert)
                time.sleep(1)

                # mache nun pro Modus eine gewissen Anzahl an Bildern
                for i in range(anzahl_Bilder):
                        # Teste nun verschiedene Werte durch
                        # erhoehe wert im ersten Drittel des Videos
                        if i < anzahl_Bilder/3:
                                einstellungswert = 3 * i

                        # fuer den Rest des Videos gehe bis in den anderen
                        # Bereich des Einstellungsbereiches
                        else:
                                einstellungswert = anzahl_Bilder - 3 * (
                                                i- int(anzahl_Bilder/3))
                        # gebe den Wert an
                        Kamera.set(modus, einstellungswert)

                        # mache das Foto nun wie gewohnt
                        success, img = Kamera.read()

                        aktuelle_Zeit = time.time()
                        aufnahmedauer = aktuelle_Zeit - vorherige_Zeit

                        text = str(int(i)) + ":  " + str(round(
                                        aktuelle_Zeit - startzeit,3)
                                        ) + " s "  + str(round(
                                        aufnahmedauer,3)) + " s "
                        vorherige_Zeit = aktuelle_Zeit
                        cv2.putText(img, text, (10, 70),
                                cv2.FONT_HERSHEY_PLAIN,
                                        3, (0, 100, 2550), 3)
```

```
# zeige Einstellungen an
text_einstellung = name + ': ' + str(
                    einstellungswert)
cv2.putText(img, text_einstellung, (10, 200),
                        cv2.FONT_HERSHEY_PLAIN,
                        3, (0, 100, 2550), 3)

# Zeige die Bilder im Display an
cv2.imshow("Image",img)

# Konvertiere Format von BGR zu RGB
# und speichere es mit 'writer'
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
writer.append_data(imgRGB)

cv2.waitKey(1)
```

Herzlichen Glückwunsch! Sie wissen nun, wie Sie verschiedene Einstellungen an der Kamera vornehmen können und wie man einer for-Schleife zwei Werte übergibt. Damit sind Sie nun gewappnet um Mikroskopieren zu können. Allerdings wollen Sie ja nicht immer so viel Programmcode schreiben müssen, richtig? Im nächsten Kapitel lernen wir, wie man Programmcode einmal schreibt und ihn dann innerhalb von wenigen Zeilen Code in einem anderen Programm nutzen kann.

6

SECTION

Kameramodul erstellen

In diesem Abschnitt wollen wir nun endlich aufhören alles immer und immer wieder zu schreiben. Wir wollen Code einmal schreiben und dann einfach und schnell in anderen Projekten verwenden können. So wie wir es auch mit den Bibliotheken cv2, time, os und imageio gemacht haben. Wir wollen auch ein Modul schreiben, welches wir importieren können und wir wollen es so schreiben, dass wir es genau wie die bisherigen Bibliotheken nutzen können. Und zwar Name gefolgt von einem Punkt gefolgt von einer Funktion. Beispielsweise time.time() oder auch Kamera = cv2.VideoCapture() mit der Funktion Kamera.set()
Um das zu lernen, werde ich Ihnen die Konzepte 'Modul' und 'Klasse' vorstellen und Minimalbeispiele zeigen. Anschließend schreiben wir unser Kameramodul.

6.1: Achtung

Dieses Kapitel wird das längste Kapitel werden, da neue Konzepte erklärt werden. Diese Konzepte werden Ihnen aber in Zukunft sehr viel Zeit und Nerven sparen. Es lohnt sich also dem Ganzen eine Chance zu geben.
Viel Spaß!

6.1 Lerne das Konzept 'Modul' kennen'

Module sind Codestücke die gut funktionieren und von anderen Programmen genutzt werden können, ohne Sie erneut schreiben zu müssen. Ähnlich wie ein schlaues Buch in einer Bibliothek, kann man man Sie sich beliebig oft ausleihen und für seine Zwecke benutzen. Manchmal muss man nicht einmal verstehen was genau passiert, solange man das Resultat versteht. So gibt es zum Beispiel das Modul Matplotlib, welches wunderbar einfach Funktionsgraphen plotten kann. Der Code an sich ist vielleicht unverständlich, aber ich weiß, dass wenn ich matplotlib.pyplot.plot([0, 9], [0, 45]) eingebe, dass dann eine lineare Funktion angezeigt wird.
Das typische Format eines Moduls besteht aus einem Funktionsteil (am Anfang), einem 'main' Teil (als zweites) und einer if Bedingung (am Ende). Erstellen Sie nun bitte ein Dokument und nennen Sie es 'Rechner.py' oder zumindest ähnlich (bitte keine Bindestriche oder sonstiges) und tippen Sie folgenden Code ein:

```
# Funktion
def Quadratrechner(x):
    return x * x

# Main Teil
def main():
    zahl = 5
    quadratzahl = Quadratrechner(zahl)
    print(str(zahl) + ' zum Quadrat = ' + str(quadratzahl) )

# Bedingung
if __name__ == "__main__":
        main()
```

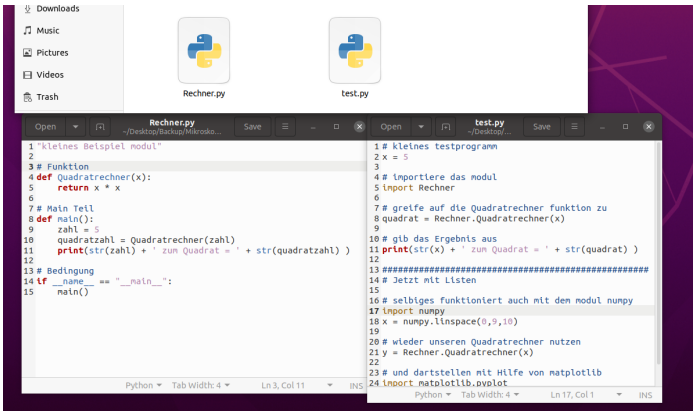
Die ersten Beiden Definitionen sind hoffentlich verständlich in ihrer Funktion. Die if Bedingung erscheint wahrscheinlich erst einmal merkwürdig. Sie macht aber eigentlich nur folgendes: Wenn Sie das Python Skript als solches ausführen, dann wird der main() Teil ausgeführt und die Quadratzahl von 5 ausgegeben. Das kennen Sie bereits. Es ist wie bei einem normalen Programm.
Wenn Sie nun allerdings ein anderes Programm schreiben (erstellen Sie ein Dokument mit dem Namen 'test.py' im selben Ordner wie 'Rechner.py') dann können Sie in dem neuen Programm den Rechner importieren und nutzen. Kleines Beispiel:

```
x = 5
import Rechner
quadrat = Rechner.Quadratrechner(x)
print(str(x) + ' zum Quadrat = ' + str(quadrat) )
```

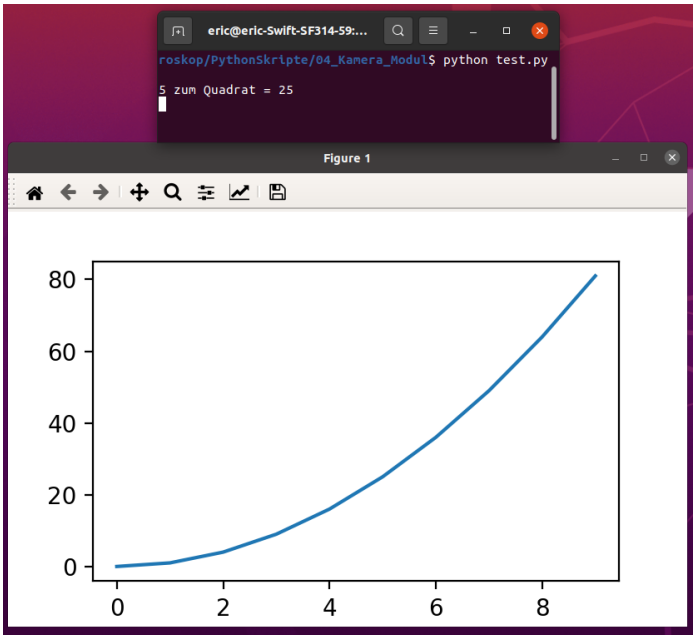
oder noch ein Beispiel indem wir zwei beliebte Module einbinden (numpy - Numerik, matplotlib - Visualisierungen)

```
import numpy
x = numpy.linspace(0,9,10)

# wieder unseren Quadratrechner nutzen
y = Rechner.Quadratrechner(x)

# und dartstellen mit Hilfe von matplotlib
import matplotlib.pyplot
matplotlib.pyplot.plot(x,y)
matplotlib.pyplot.show()
```

Die Dateien sollten im selben Ordner sein und dann kann das ganze so aussehen:



Ich bevorzuge es im normalen Texteditor zu programmieren und die Programme dann mittels der Konsole zu starten. Sie werden wahrscheinlich eher Geany verwenden. Dann wissen Sie ja, wie Sie das Programm ausführen können. Das Resultat sollte zumindest das gleiche sein:



6.2 Lerne das Konzept 'Klasse' kennen'

Nun wissen Sie, wie Sie Übersicht in Ihrem Programm schaffen können... Indem Sie Programmstücke auslagern! Das ist wunderbar. Jetzt wollen wir noch übersicht innerhalb des ausgelagerten Programmes schaffen und die Funktionen innen drin miteinander verknüpfen. Dies führt uns zum sogenannten objektorientiertem Programmieren. Wie der Name bereits sagt, erstellen wir ein Objekt, oder auch Klasse genannt. Dieses Objekt hat verschiedene Eigenschaften und Funktionen. So wollen wir zum Beispiel später das Objekt meine_Kamera erstellen, welches eine Kamera öffnet. Dieser Kamera wollen wir Einstellungen (Helligkeit, Kontrast, ...) zuweisen. Dann soll diese Kamera erst kurze Videos anzeigen, damit wir sehen können, ob die Einstellungen Sinn ergeben und dann wollen wir die Einstellungen entweder wechseln oder ein Video mit guten Einstellungen aufnehmen. Dazu ist es wichtig, die Kameraeinstellungen fest mit den Funktionen Videoansicht und Videoaufnahme zu verknüpfen.
Jetzt aber erstmal ein Minimalbeispiel. Erstellen Sie bitte ein Dokument mit dem Namen guter-Rechner.py und geben Sie folgendes ein:

```
class Rechner():
        def __init__(self, x):
                self.x = x

        def Quadrat(self):
                self.quadrat = self.x * self.x
                return self.quadrat

        def Verdoppeln(self):
                self.doppelt = 2 * self.x
                return self.doppelt

        def Nachfolger(self):
                self.nachfolger = self.x + 1
                return self.nachfolger

        def Darstellen(self):
                import matplotlib.pyplot as plt
                plt.plot(self.x, self.quadrat, label='x * x')
                plt.plot(self.x, self.doppelt, label='2 * x')
                plt.plot(self.x, self.nachfolger, label='x + 1')
                plt.legend()
                plt.show()
```

Auch hier können wir ein Modul draus machen, indem wir wieder einen main Teil und eine if-Bedingung hinzufügen:

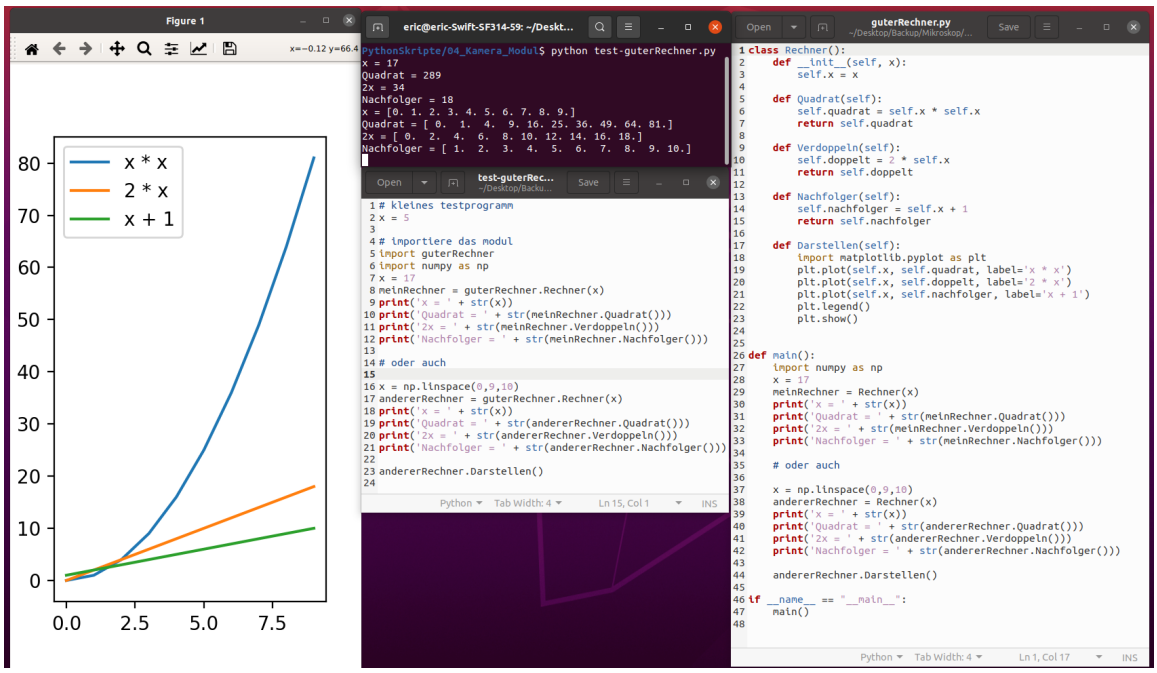
```
def main():
        import numpy as np
        x = 17
        meinRechner = Rechner(x)
        print('x = ' + str(x))
        print('Quadrat = ' + str(meinRechner.Quadrat()))
        print('2x = ' + str(meinRechner.Verdoppeln()))
        print('Nachfolger = ' + str(meinRechner.Nachfolger()))

        # oder auch

        x = np.linspace(0,9,10)
        andererRechner = Rechner(x)
        print('x = ' + str(x))
        print('Quadrat = ' + str(andererRechner.Quadrat()))
        print('2x = ' + str(andererRechner.Verdoppeln()))
        print('Nachfolger = ' + str(andererRechner.Nachfolger()))

        andererRechner.Darstellen()

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

Jetzt können Sie dieses Programm laufen lassen oder ein neues Programm schreiben, in welchem Sie den guten Rechner importieren.
Wie sie vielleicht gemerkt haben, erstellen wir im main Teil zwei verschiedene Objekte. Eins heißt meinRechner und das andere andererRechner. Das eine Objekt kennt die Zahl 17 und das andere Objekt kennt eine Liste von Zahlen. Beide Objekte existieren unabhängig voneinander! In beiden Objekten kann man jedoch auf alle Funktionen der Klasse ohne weitere Probleme zugreifen.
Probieren Sie nun selbst ein neues Programm zu schreiben, was auf dieses Modul zugreift. Dazu importieren Sie den guten Rechner als:

```
import guterRechner
meinRechner = guterRechner.Rechner()
```

Testen Sie sich selbst bevor Sie sich das Bild auf der neuen Seite ansehen.

Herzlichen Glückwunsch! Sie wissen nun, wie Sie Module schreiben und Klassen verwenden können. Das ganze wollen wir jetzt anwenden, indem wir alle unseren bisherigen Skripte in eine einzelne Klasse packen und diese dann in ein Modul einbinden. Das erlaubt es uns, sehr übersichtlich und smart zu arbeiten. Am besten ist allerdings, dass alle Funktionen im Objekt selbst (in der Klasse) miteinander verbunden sind.



6.3 Kameramodul schreiben

Nun ist es an der Zeit das Kameramodul zu schreiben. Da Sie bereits die Konzept 'Modul' und 'Klasse' kennen, wird das folgende einfach für Sie sein. Sie müssen lediglich das Skript 'guter Rechner' ein bisschen umschreiben. Beginnen wir also sofort. Erstellen Sie ein neues Dokument mit dem Namen: 'KameraModul.py' und kopieren Sie folgende Vorlage hinein:

```
import cv2
import time
import imageio
import os

class Kameramodul():
    def __init__(self, kamera=0, wCam=640, hCam=480):

    def kameraeinstellungen(self, Helligkeit=20, Kontrast=20,
                            Saettigung=50, Hue=0, Gain=0,
                            Belichtung=0,Weissabgl = 0):

    def videoansicht(self, laufzeit=5):

    def videoaufnahme(self, Speicherort, gif_name, anzahl_Bilder):

def main():

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

Damit importieren Sie alle wichtigen Bibliotheken, erstellen eine Klasse mit dem Namen Kameramodul, die Sie mit Werten initialisieren (kamera=0, wCam=640 pixel, hCam=480 px sind

'default' Werte, also Werte die die Funktion annimmt, wenn man ihr keine anderen Werte übergibt). Dann haben wir noch die 3 Funktionen 'kameraeinstellungen', 'videoansicht' und 'videoaufnahme'. Diese werden wir gleich mit Inhalt füllen. Falls Ihnen noch weitere Funktionen einfallen (z.B. zur Bildverarbeitung) können Sie auch einfach eine weitere Funktion definieren und hinzufügen. Danach, wie Sie wissen, kommt der main() Teil, den wir zum Entwickeln nutzen, der aber später durch ein anderes Skript ersetzt wird. Schlussendlich kommt dann noch der if name = ... Aufruf, wie wir Ihn aus dem vorigen Abschnitt kennen.
Füllen wir also nun die erste Funktion mit Leben:

```
def __init__(self, kamera=0, wCam=640, hCam=480):
    self.Kamera = cv2.VideoCapture(kamera)

        self.wCam = wCam
        self.hCam = hCam

        self.Kamera.set(3, self.wCam)
        self.Kamera.set(4, self.hCam)
```

Die init Funktion zum initialisieren der Kamera ist wichtig, da wir hier als erstes die Kamera auswählen, mit: cv2.VideoCapture(kamera) und diese Information mit dem 'self.' Aufruf für die ganze Klasse zur Verfügung stellen. Das bedeudet, dass man in jeder Funktion der Klasse (kameraeinstellungen, videoansicht, videaufnahme) ganz einfach auf die Kamera zugreifen kann, indem man self.Kamera eintippt. Ist das nicht fantastisch?
Danach machen wir noch die Einstellungsparameter Höhe und Weite klassenintern aufrufbar mit self.wCam und self.hCam. Das erlaubt es uns, diese Eigenschaft später zu nutzen oder zu verändern. Auch können wir diese Eigenschaften gleich unserer self.Kamera übergeben indem wir den self.Kamera.set(x,y) Befehl nutzen.
Falls Sie es bevorzugen, die Höhe und Weite des Sichtfensters mit in die Funktion 'kameraeinstellungen' einzubauen, dann fühlen Sie sich frei das zu tun.
Kommen wir nun also zu den Kameraeinstellungen. Wie Sie in der Vorlage sehen, werden wir die Helligkeit, den Kontrast, usw. als Einstellmöglichkeiten zur Verfügung stellen. Diese werden mit einem 'default' Wert versehen. D.h. wenn Sie im main() Teil schreiben:

```
meineKamra = Kameramodul() # init
meineKamera.kameraeinstellungen() # Funktion
```

dann wird die Kamera mit den Werten: kamera=0, wCam=640, hCam=480, Helligkeit=20, Kontrast=20, Saettigung=50, usw. initialisiert und eingestellt. (Default bedeutet Voreinstellung)
Die Kameraeinstellungen müssen im nächsten Schritt dann auch an die initialisierte Kamera aus '__init__' übergeben werden. Vorher wollen wir Sie aber noch innerhalb der Klasse zugänglich machen (mit self.):

```
def kameraeinstellungen(self, Helligkeit=20, Kontrast=20,
    Saettigung=50, Hue=0, Gain=0, Belichtung=0,Weissabgl = 0):

        self.Helligkeit = Helligkeit
        self.Kontrast = Kontrast
        self.Saettigung = Saettigung
        self.Hue = Hue
        self.Gain = Gain
        self.Belichtung = Belichtung
        self.Weissabgl = Weissabgl

        modi = [10, 11, 12, 13, 14, 15, 17]
        self.initialisierungswerte = [self.Helligkeit, self.Kontrast,
                        self.Saettigung, self.Hue, self.Gain,
                        self.Belichtung, self.Weissabgl]

        # uebergebe alle werte an die kamera
        for modus, wert in zip(modi, self.initialisierungswerte):
            self.Kamera.set(modus, wert)
```

Das 'zugänglich machen' innerhalb der Klasse (mit self.Eigenschaft) ist für diese Werte optional. Ich nutze es hier, damit Sie a) öfter den 'self.' Befehl sehen können und b) um es beim Speichern des Videos später nutzen zu können, damit die Kameraeinstellungen nicht verloren gehen (das ist beim wissenschaftlichen Arbeiten sehr wichtig!).
Die Kameraeinstellungen sind also jetzt definiert. Wir können nun die 'videoansicht' Funktion schreiben und dann den main() Teil und die 'videoaufnahme'. Beginnen wir mit der Videoansicht, die zum Überprüfen der Kameraeinstellungen und der Bildqualität dienen soll. Diese Funktion wird Ihnen sehr sehr bekannt vorkommen und jetzt kommt's: Es wird das letzte Mal sein, dass Sie diese Funktion schreiben! In allen weiteren Projekten können Sie diese Funktion mit einer Zeile Code nutzen!

```
def videoansicht(self, laufzeit=5):
        startzeit = time.time()
        vorherige_Zeit = startzeit

        schleife = True
        while schleife:
                success, img = self.Kamera.read()

                aktuelle_Zeit = time.time()
                zeit_differenz = aktuelle_Zeit - vorherige_Zeit
                fps = 1/zeit_differenz  # (frames per second)
                vorherige_Zeit = aktuelle_Zeit
```

```
            # zeige Framerate
            text = str(int(fps)) + " FPS, Laufzeit: " + str(int(
                                    time.time() - startzeit)) + " s"
            cv2.putText(img, text, (10, 70), cv2.FONT_HERSHEY_PLAIN,
                                    3, (255, 100, 0), 3)

            # Zeige das Foto an
            cv2.imshow("Image",img)
            cv2.waitKey(1)

            # Gehe aus schleife raus, wenn laufzeit ueberschritten ist
            if time.time() - startzeit >= laufzeit:
                    schleife = False
```

Der default Wert dieser Funktion ist die Laufzeit, welche auf 5 Sekunden gesetzt ist. Wie Sie sehen, wird in dieser Funktion genau ein mal der 'self.' Ausdruck genutzt und zwar, wenn die Kamera das Bild machen soll (success, img = self.Kamera.read()). Auch sehen Sie, dass die Variable 'laufzeit' nicht als 'self.laufzeit' gespeichert wird und somit auch nicht für die ganze Klasse zugänglich ist.
Bisher haben wir nur eingetippt oder kopiert, jetzt wollen wir auch testen! Dazu werden wir den main() Teil schreiben und dann das Programm laufen lassen.

```
def main():
    kamera = 2 # USB Kamera
    wCam, hCam = 1080, 720 # weite und hoehe des Bildes

    # initialisiere Kamera
    USB_Kamera = Kameramodul(kamera, wCam, hCam)

    # Kameraeinstellungen (setze Helligheit und Hue,
    # lasse den Rest als default)
    Helligkeit = 40
    Hue = 150
    USB_Kamera.kameraeinstellungen(Helligkeit=Helligkeit, Hue=Hue)

    # starte ein video, z.B. um zu sehen ob die einstellungen gut sind
    USB_Kamera.videoansicht(laufzeit = 10)
```

Falls alles klappt, sollte sich nun die Kamera öffnen und Sie sollten ein Bild für 10 Sekunden sehen können. Dieses Bild wird im 1080 px, 720 px Format sein (also anders als der default Wert) und mit den Werten 40 und 150 für Helligkeit und Hue (also auch nicht den default Werten. Kontrast, Saettigung, etc. entsprechen den default Werten der Funktion 'kameraeinstellungen'.
Kommen wir nun zum letzten Teil, der Videospeicherung. Hier werden wir, wenn wir mit den Kameraeinstellungen zufrieden sind, ein Video aufnehmen und abspeichern. Der Speichername besteht aus einem selbst gewählten Experimentnamen und zusätzlich noch aus den Kame-

raeinstellungen (da diese in jedem Experiment wichtig für das Protokoll und damit für die Reproduzierbarkeit sind).

```
def videoaufnahme(self, Speicherort, gif_name, anzahl_Bilder):
        speichername = gif_name + "-dim-" + str(self.hCam) + "_px_" + str(
                self.wCam) + "_px-Helligkeit-" + str(
                self.Helligkeit) + "-Kontrast-" + str(
                self.Kontrast)  + "-Saettigung-" + str(
                self.Saettigung)  + "-Hue-" + str(
                self.Hue)   + "-Gain-" + str(
                self.Gain)   + "-Belichtung-" + str(
                1000 * self.Belichtung)   + "_ms-Weissabgl-" + str(
                self.Weissabgl) + ".gif"

        gif_pfad = Speicherort + "/" + speichername

        # Probiere den Ordner zu erstellen
        try:
                os.mkdir(Speicherort)
        # Existiert der Ordner bereits?
        except OSError:
                print ("Der Ordner %s wurde nicht erstellt" % Speicherort)
        else:
                print ("Der Ordner %s wurde erstellt" % Speicherort)

        # kommando zum speichern der fotos
        with imageio.get_writer(gif_pfad, mode='I') as writer:
                # wiederhole vorgang "anzahl_Bilder" mal
                for i in range(anzahl_Bilder):
                        success, img = self.Kamera.read()

                        # Zeige die Bilder im Display an
                        cv2.imshow("Image",img)

                        # Konvertiere Format von BGR zu RGB
                        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                        writer.append_data(imgRGB)
                        cv2.waitKey(1)
```

Hier gibt es viel zu kommentieren. Als erstes fällt Ihnen sicher auf, dass wir mit 'self.XY' auf alle Kameraeinstellungen zurückgreifen können, um sie im Speichernamen zu integrieren. Dann fällt Ihnen bestimmt noch auf, dass keine Zeitmessungen und Bildschirmausgaben stattfinden. Das habe ich der Kürze halber weg gelassen. Fügen Sie es gerne noch ein, wenn Sie wollen. Eine weitere Sache, die Sie vielleicht gesehen haben ist folgendes: speichername = ... str(1000 * self.Belichtung) millisekunden... Das sorgt dafür, dass die Belichtungszeit beim Speichern in millisekunden ausgegeben wird. Der Faktor 1000 bezieht sich hierbei aber nur auf den Ausgabetext

und verändert die Belichtungszeit Variable nicht. Beachten Sie bitte auch, dass die Belichtungszeit nur eingestellt werden kann, wenn Ihre Kamera diese Möglichkeit anbietet. Die mitgelieferte Kamera kann das leider nicht, beziehungsweise habe ich noch nicht herausbekommen wie es gehen sollte.
Der Rest vom Programm ist Ihnen bekannt und Sie können es beliebig erweitern. Jetzt werden wir es im main() Teil noch mit einer Zeile Code aufrufen und dann ist es auch schon geschafft.

```
def main():
    # ...vorheriger Code...

    # Namen vergeben
    Speicherort = "Experiment-01"
        gif_name = "Experimentname"
        anzahl_Bilder = 30

        # eine Zeile code zum Nutzen der Funktion
        USB_Kamera.videoaufnahme(Speicherort, gif_name, anzahl_Bilder)
```

Herzlichen Glückwunsch! Sie wissen nun, wie Sie Module schreiben und Klassen verwenden können. Das ganze können Sie auf ein bestimmtes Gebiet anwenden, indem Sie zum Beispiel ein Kameramodul schreiben. Sie wissen nun, wie man innerhalb von Funktionen Variablen mit dem 'self.XY' Befehl austauscht und wie man über die main() Funktion ein Modul im selbigen Dokument ausführt. Nun werden wir noch schnell lernen, wie man auf das Kameramodul von einem anderen Skript aus zugreift. Sie können sich natürlich auch selbst testen und danach erst auf die Lösung schauen.

6.4 Kameramodul aus anderem Skript aufrufen

Um nun das Kameramodul aufzurufen und auch um ein bisschen die Übersicht zu behalten, erstellen wir einen neuen Ordner, in dem wir das Kameramodul abspeichern und in dem wir ein neues Skript schreiben, mit dem Namen: '05_Benutze_Kamera_Modul.py'. Es ist wichtig, dass beide Skripte im selben Ordner sind (ansonsten wird es ein bisschen umständlicher das Modul zu importieren).
Beginnen wir also im neuen Dokument, das Kameramodul zu importieren. Da Kameramodul so ein langer Name ist können wir es auch unter der Abkürzung kM aufrufen. Sie können es aber auch anders abkürzen oder beim vollen Namen benutzen.

```
import KameraModul as kM

# ruft das KameraModul (kM) auf und geht in class Kameramodul
meine_Kamera = kM.Kameramodul(kamera=2) # USB Kamera

meine_Kamera.kameraeinstellungen() # benutze default

meine_Kamera.videoansicht(5) # 5 sekunden video anzeigen

meine_Kamera.videoaufnahme("Exp_02", "Exp_Name", 50) # Video speichern
```

Nun verstehen Sie hoffentlich, was ich mit Zeitersparnis und Effizienz meine. 5 Zeilen Programm, um eine Kamera aufzurufen, Einstellungen zu wählen, eine 5 Sekunden Videovorschau zu geben und dann 50 Bilder in einem neuen Ordner zu speichern. Ich hoffe ich habe Sie überzeugt, dass Module etwas tolles sind. Auch das Konzept der Klasse ist wunderbar, weil wir auf alle Funktionen mit einem einfachen 'Punkt .' zwischen Kameraobjekt und Funktion zugreifen können (meine_Kamera Punkt kameraeinstellungen oder allgemein meine_Kamera.Funktionsname). Den vollständigen Programmcode finden Sie kommentiert auf GitHub[12].

Herzlichen Glückwunsch! Damit ist dieses Kapitel abgeschlossen und Sie können nun die USB Kamera auseinander bauen und Sie nicht mehr als bloße Kamera, sondern als linsenloses Mikroskop verwenden. Die Theorie dahinter finden Sie im nächsten Kapitel. Danach werden Sie sehen, wie man die Kamera auseinander bauen kann.

[12] GitHubLinkeinfuegen

7

SECTION

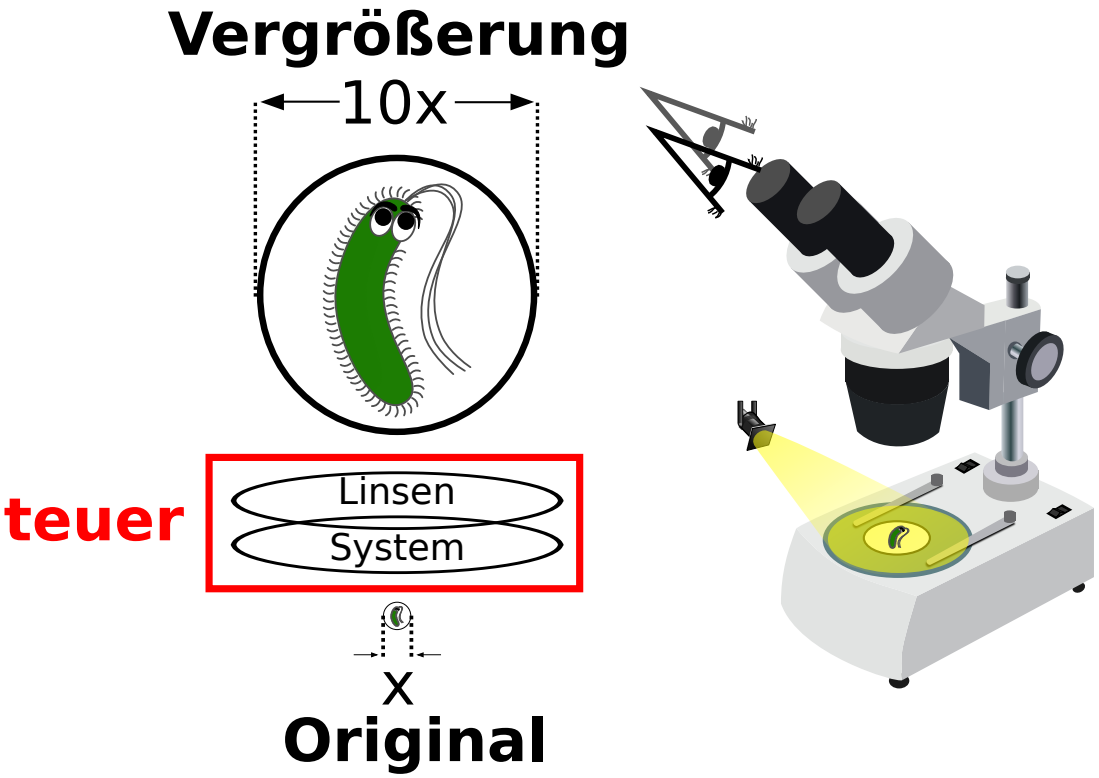
Funktionsweise des Mikroskopes

Das Ziel dieses Projektes ist es, ein Mikroskop ohne Linsen zu bauen. Sie haben sich bestimmt schon gefragt, wozu das gut sein soll. Einer der Hauptgründe ist eindeutig der Preis! In guten Mikroskopen sind zum Teil 10 Tausend Euro teure Linsen oder Linsensysteme verbaut, die sämtliche typischen Linsenfehler[13] bereinigt haben. Dieses Geld kann man sich sparen, wenn man weiß wie. Und das WIE wollen wir hier lernen. Zuerst einmal betrachten wir aber ein typisches optisches Lichtmikroskop.

7.1 Optisches Lichtmikroskop

Die Hauptidee eines Mikroskopes ist es, ein Original Bild zu vergrößern. Typische Vergrößerungen aus meinem Schulunterricht waren 5x, 10x, 25x und manchmal auch 50x oder 100x. Oder ausgesprochen '100-fache Vergrößerung', was genau dem entspricht was es macht. Alle Eigenschaften des Originalbildes werden genau um den Faktor 100 vergrößert. Damit kann man scheinbar unsichtbar kleine Objekte plötzlich doch sichtbar machen.
So kann das (gesunde) menschliche Auge zum Beispiel Strukturen voneinander unterscheiden wenn diese mehr als 0.15 mm voneinander entfernt sind. Also wenn man auf einem Lineal zwischen dem Startpunkt 0 und dem ersten Millimeterstrich, 9 weitere Striche im selben Abstand einfügen würde, dann wären sie tendenziell nicht von Ihnen unterscheidbar. Wenn allerdings nur 5 weitere eingefügt werden, dann (wenn Sie sehr gute Augen haben) sehen Sie die einzelnen Striche getrennt voneinander.

[13] https://www.lernhelfer.de/schuelerlexikon/physik/artikel/abbildungsfehler-bei-linsen#



Nehmen Sie nun eine Sammellinse mit hinreichender Vergrößerung, wie z.B. eine Lupe, dann können Sie plötzlich alle Striche ganz klar sehen. Nehmen Sie nun noch mehr Linsen und verknüpfen diese clever miteinander, dann erreichen Sie Auflösungen wie in einem Mikroskop. 100-fache Vergrößerung erlaubt es dadurch, Strukturen unterscheidbar zu machen, die 0.15 mm / 100 = 0.0015 mm = 1.5 μm klein sind.

Das ist wunderbar für die Mikrobiologie. Allerdings muss man bei der Herstellung dieser Linsen aufpassen. Eine einzelne Linse wird **immer** gewisse Fehler aufweisen, die nicht ohne weiteres behebbar sind. So besteht eine Linse meistens aus einem Material (z.B. Glas oder Plastik). Ein Material hat einen wellenlängenabhängigen Brechungsindex (Wellenlänge ≈ Farbe). Das heißt, dass verschiedene Farben unterschiedlich stark gebrochen/fokussiert werden. Deshalb entstehen immer, bei jeder einzelnen Linse, Bildfehler. Schnell nachzulesen ist dieser Effekt unter Chromatisch Abberation[14]. Dieser Fehler und weitere sind jedoch mit Linsensystemen aus Sammel- und Zerstreuungslinsen behebbar. So konnte chromatische Abberation einer Sammellinse teilweise mit Hilfe einer Zerstreuungslinse behoben werden. Eine kurze interessante Geschichte zum Patent davon finden Sie in der Wikipedia zum Thema Achromat[15]

Nun aber genug zu den teuren Linsensystemen. Ein Mikroskop besteht auch noch aus einem Fuß, Okular, Tubus und vielem mehr. Das macht es sehr groß und schwer. Allerdings ist es auch sehr schön, dass man mit diesem Aufbau gut fokussieren und mit einem beweglichen Objekttisch auch große Objekte untersuchen kann. Ein wichtiger Bestandteil des Lichtmikroskopes ist natürlich die Lichtquelle, die im besten Falle mit verstellbarer Lichtintensität ist.

[14] https://de.wikipedia.org/wiki/Chromatische_Aberration
[15] https://de.wikipedia.org/wiki/Achromat

7.1: Achtung

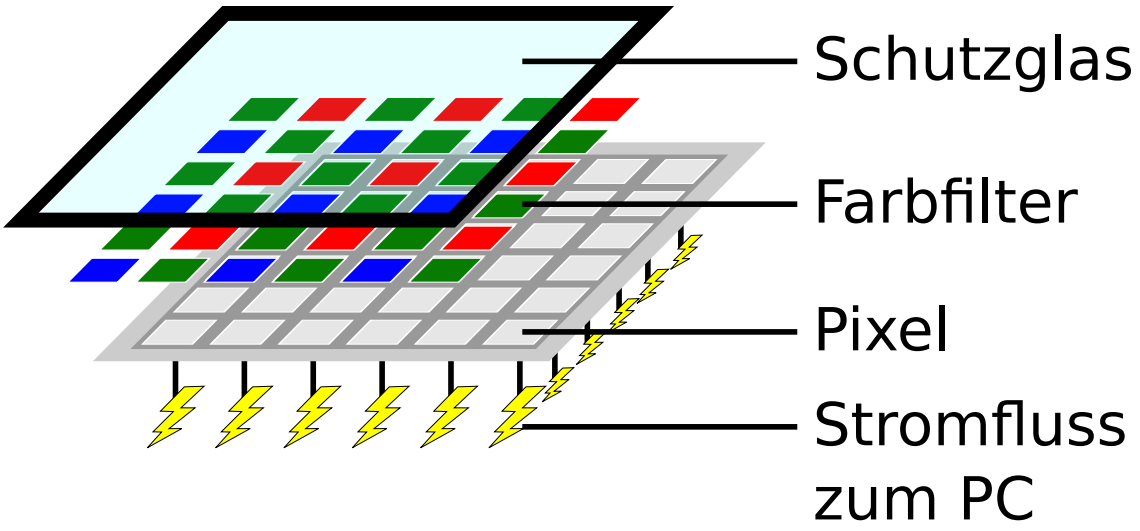
All diese Bauteile kosten Geld. Am teuersten dabei ist das Linsensystem, da es aufwendig berechnet und noch aufwendiger produziert werden muss. Die Idee des linsenlosen Mikroskopes ist es, auf diese teuren Bauteile zu verzichten!

Weitere Informationen

Falls Sie mehr Interesse an der Herstellung von Linsen haben, schauen Sie sich diesen Link an:
Falls Sie mehr Interesse an Linsenfehlern haben, schauen Sie sich diesen Link an:
Falls Sie mehr Interesse an optischer Auflösung und physikalischen Grenzen haben, schauen Sie sich diesen Link an:
Falls Sie wissen wollen, was man in der Natur bis 1.5 μm finden kann, schauen Sie sich diesen Link an:

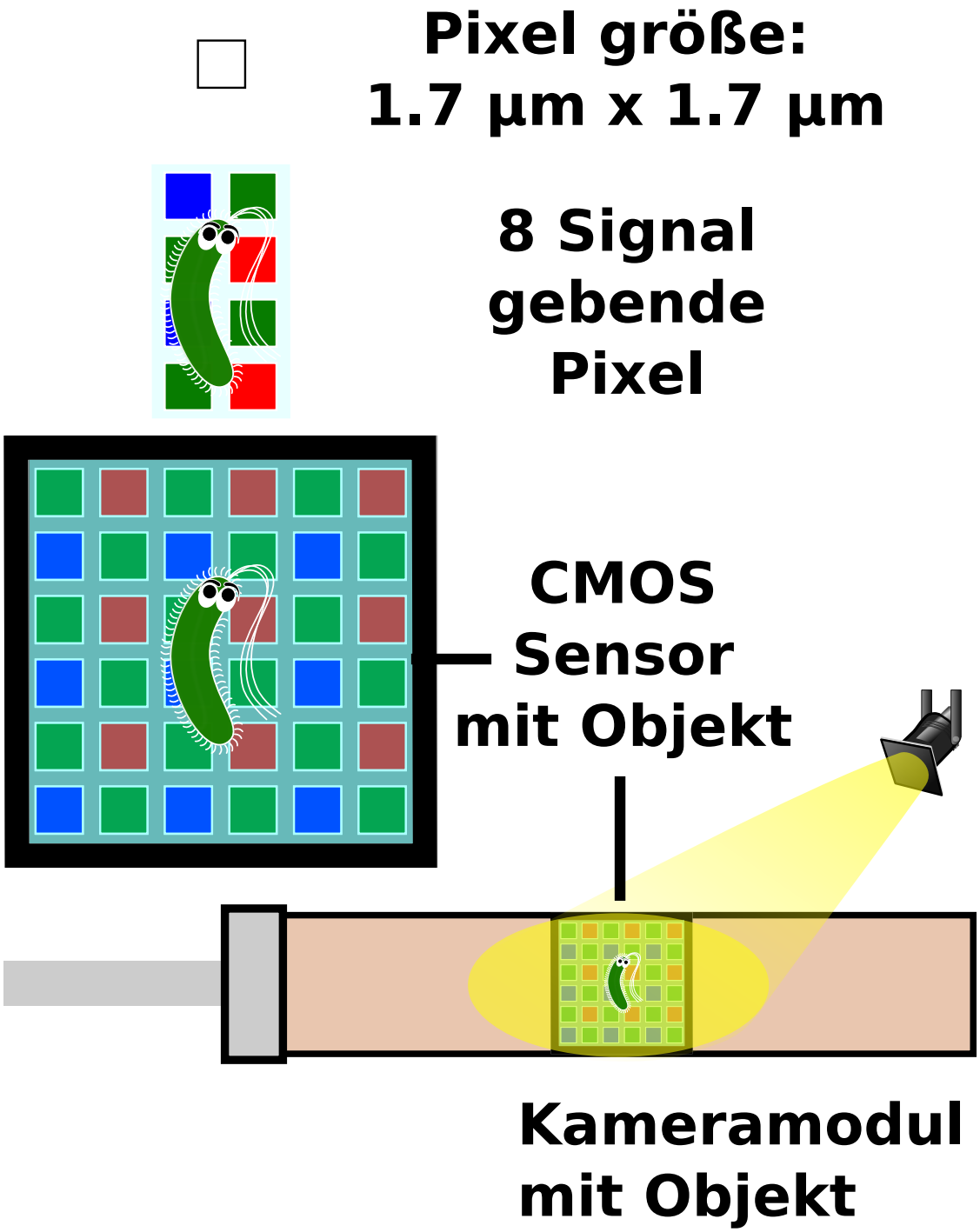
7.2 Linsenloses Mikroskop

Fast jeder Mensch hat heutzutage ein Handy, inklusive Kamera. Diese Kamera hatte damals 1 Megapixel (MP), später waren 2 MP faszinierend und im Jahr 2021 waren bereits über 100 MP = 100 000 000 Pixel pro Kamera möglich. Ein Full HD Fernseher hat 1920 × 1080 Pixel (px) und ein 8K Fernseher 7.680 x 4.320 davon. Doch was bedeutet das eigentlich?
Ein Pixel ist ein Kurzwort für Bildelement (Bild = picture , kurz: pix) und ist die kleinste verbaute Einheit in einem Gerät, die entweder ein Bild produzieren kann (im Fernseher) oder eine Bildinformation aufnehmen kann (in einer Kamera). Eine Bildinformation in diesem Sinne ist meistens eine der Farben **rot**, **grün** oder **blau** oder kurz: RGB
Licht trifft also auf das Gerät (meistens ein CMOS-Sensor[16]), passiert das Schutzglas, danach den Farbfilter (RGB) und landet gefiltert auf dem Pixel. Das Pixel ist nun grob gesagt ein Material, was die Energie des Lichtes in ein elektrisches Signal umwandelt. Siehe hierzu: Photoeffekt[17]



Nun hat eine Handykamera z.B. 4 MP, angeordnet wie in einem Quadrat. Also $\sqrt{4000000}$ $= \sqrt{4 \cdot 1000000} = 2 \cdot 1000$ von diesen Pixeln pro Seitenlänge. Diese sind hinter der Linse auf einer Fläche von circa einem Quadratzentimeter verbaut. Das bedeutet, dass man 2000 px pro 1 cm hat oder $\frac{200\ px}{mm}$. Entsprechend ist ein Pixel 5 μm klein.
Die Kamera die wir hier verwenden hat eine Pixelgröße von 1.75 μm, wie man dem Datenblatt[18] entnehmen kann.
Entsprechend können wir direkt Objekte auf dem Schutzglas platzieren und anfangen zu messen, da wir ohne zusätzliche Linsen auf eine Auflösung bis 1.7 μm kommen. Die Vergrößerung ist dahingehend einfache (1x) Vergrößerung, da wir keine Linse haben und uns das Objekt in Originalgröße anschauen.

[16] https://de.wikipedia.org/wiki/Active_Pixel_Sensor
[17] http://www.abi-physik.de/buch/quantenmechanik/photoeffekt/
[18] https://de.aliexpress.com/item/1005003385168178.html



Nun, wenn das Objekt auf dem Modul platziert ist, sollte man noch einen gleichmäßige, helle Beleuchtung erzeugen, damit man sämtliche Lichteffekte auf das Geringste reduzieren kann. Es reicht eigentlich schon, eine Taschenlampe über der Kamera zu platzieren oder an einem sonnigen Tag draußen zu mikroskopieren.

8

SECTION

Templates

Definition 8.1

Das ist ein Beispiel wie ich eine Box um einen Text machen kann. Der erste Buchstabe fallt aus irgendeinem Grund weg

Definition 8.2

Das ist ein Beispiel wie ich eine Box um einen Text machen kann. Der erste Buchstabe fallt aus irgendeinem Grund weg

Weitere Informationen

A partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant. Partial derivatives are used in vector calculus and differential geometry.

hier kann ein link oder ein buchverweis eingefügt werden

Wichtig

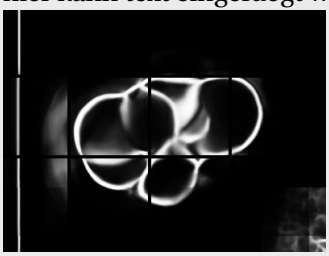
Das ist ein Beispiel wie ich eine Box um einen Text machen kann. Der erste Buchstabe fallt aus irgendeinem Grund weg

Veranschaulichung 8.1

Angular Momentum

Discuss an alternate way of using the conservation of angular momentum for satellite orbits and any other point masses moving in a circle.

hier kann text eingefuegt werden oder auch ein Bild



Veranschaulichung 8.2

Angular Momentum

Discuss an alternate way of using the conservation of angular momentum for satellite orbits and any other point masses moving in a circle.

Beispiel 8.1 Rotational Motion

A block of mass $4m$ is attached to a light string and passes over a pulley with negligible rotational inertia and is wrapped around a vertical pole of radius r. The block is then released from rest, causing the string to unwind and the vertical pole it is wrapped around to rotate. On top of the vertical pole lies a horizontal rod of length $2L$ with block2s of mass m attached to both ends. The rotational inertia of this apparatus is $2mL^2$.

1. What is the tension in the string in terms of the acceleration of the falling block?
2. What is the torque exerted on the rotating pole by the string in terms of the acceleration of the falling block?
3. When the large block has fallen a distance D, what is the instantaneous rotational kinetic energy of the apparatus?

Solution:
Hier kommt text hin

Beispiel 8.2 Rotational Motion

A block of mass $4m$ is attached to a light string and passes over a pulley with negligible rotational inertia and is wrapped around a vertical pole of radius r. The block is then released from rest, causing the string to unwind and the vertical pole it is wrapped around to rotate. On top of the vertical pole lies a horizontal rod of length $2L$ with block2s of mass m attached to both ends. The rotational inertia of this apparatus is $2mL^2$.

1. What is the tension in the string in terms of the acceleration of the falling block?
2. What is the torque exerted on the rotating pole by the string in terms of the acceleration of the falling block?
3. When the large block has fallen a distance D, what is the instantaneous rotational kinetic energy of the apparatus?

Solution:
Hier kommt text hin

$$\vec{r} = \frac{1}{2}\vec{a}t^2 + \vec{v}_0 t + \vec{r}_0 \tag{1}$$

$$\vec{v} = \vec{a}t + \vec{v}_0 \tag{2}$$

$$v^2 = v_0^2 + 2\vec{a}\cdot(\vec{r} - \vec{r}_0) \tag{3}$$

9

SECTION

Code einfügen

Direkt hier in latex schreiben:

```
# importiere "Bibliotheken" die einen Grossteil der Arbeit machen
import cv2 # Zugriff auf Kamera
import time # Zeitmessungen

# suche eine Kamera aus
# (0 is die eingebaute kamera im laptop, 1,2,3,... sind weitere Anschluesse)
Kamera = cv2.VideoCapture(0) # der name Kamera ist nun bereit und kann verandert
                                                        # werden. Zum Beispiel kann die A
                                                        # eingestellt werden

# stelle die Aufloesung in pixel ein
wCam, hCam = 640, 480 # weite und hoehe des Bildes
Kamera.set(3, wCam)
Kamera.set(4, hCam)




# optional wollen wir hier die Anzahl der Bilder pro Sekunde (fps) berechnen.
# Dazu muss die Zeit gemessen werden, die fuer ein Bild gebraucht werden
# Hier wird die Zeit initialisiert.
vorherige_Zeit = 0
```

Listing 1: Example from external file

```
print("Hello World")
```

Listing 2: Example Python code

10

SECTION

FAQs

10.1 Wie installiere ich Anaconda auf meinem Windows/Mac/Linux System?

Eine ausführliche Beschreibung zur Installation finden Sie auf der Seite von Anaconda. Für Linux[19] und MacOS[20] kann man die Schritte genau so durchführen wie sie beschrieben sind. Für Windows empfiehlt es sich, bei Schritt 8, beide Häkchen zu setzen. Der Link zur Seite ist hier: Windows[21].

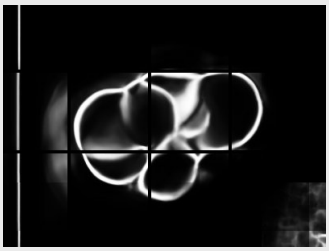
10.2 Wie öffne ich die Konsole?

Veranschaulichung 10.1 Windows - Konsole

Klicken Sie (für gewöhnlich unten links) auf das Windows Symbol und tippen Sie "Konsoleöder cmd"(offiziell Windows-Eingabeaufforderung genannt) ein und öffnen Sie den ersten Treffer. Glückwunsch, Sie sind nun in der Konsole. Da wir diese Konsole oft brauchen werden, bietet es sich an, sie in der Taskleiste anzupinnen.

Falls es immer noch nicht klappt, dann schauen Sie sich unter anderem bitte dieses YouTube Video an: `Platzhalter:MitWindowsKonsolearbeiten`.

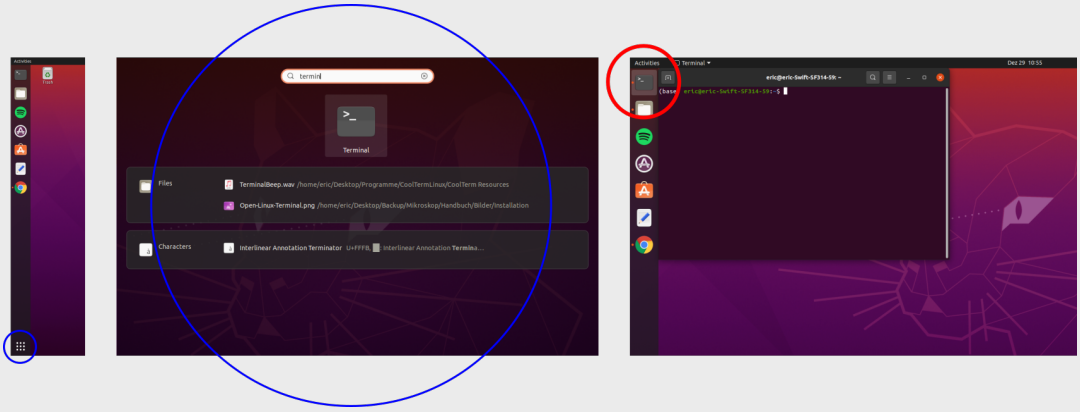
[19] `https://docs.anaconda.com/anaconda/install/linux/`
[20] `https://docs.anaconda.com/anaconda/install/mac-os/`
[21] `https://docs.anaconda.com/anaconda/install/windows/`



Veranschaulichung 10.2 Linux - Terminal

Klicken Sie (für gewöhnlich unten links) auf das Menü - Zeichen (9 Punkte, 'Show Applications', blau markiert) und geben Sie dann in der Suchleiste 'Terminal' ein (auch in blau). Glückwunsch, Sie sind nun in der Konsole. Da wir diese Konsole oft brauchen werden, bietet es sich an, sie in der Taskleiste anzupinnen (rot markiert).

Falls es immer noch nicht klappt, dann schauen Sie sich unter anderem bitte dieses YouTube Video an: `Platzhalter:MitLinuxTerminalarbeiten`.

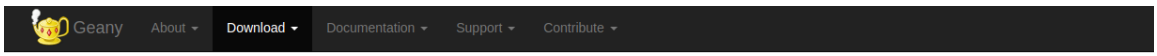


MacOS-Terminal

Da ich keinen Mac Computer zur Verfügung habe, gehen Sie einfach dem Apple Guide nach: `https://support.apple.com/de-de/guide/terminal/apd5265185d-f365-44cb-8b09-71a064a42125/mac` oder versuchen Sie es mit diesem YouTube Video: `https://www.youtube.com/watch?v=aKRYQsKR46I`

10.3 Wie installiere ich Geany?

Geany ist kostenlos hier erhältlich: `https://www.geany.org/download/releases/`
Klicken Sie einfach auf den Windows oder MacOS Link der in der Spalte 'Files' zu finden ist. Bei Windows ist es eine '.exe' Datei und bei Mac eine '.dmg' Datei.



Geany Releases

Distribution	File	GPG Signature	GPG Key
Source (tar.gz)	geany-1.38.tar.gz	geany-1.38.tar.gz.sig (Instructions)	colombanw-pubkey.txt
Source (tar.bz2)	geany-1.38.tar.bz2	geany-1.38.tar.bz2.sig (Instructions)	colombanw-pubkey.txt
Windows (64-bit[1])	geany-1.38_setup.exe	geany-1.38_setup.exe.sig (Instructions)	eht16-pubkey.txt
Mac OSX	geany-1.38_osx-4.dmg	-	-

SECTION

11 Literatur

[1] Academic and Research Computing. *Text Formatting with LaTeX A Tutorial*. NY, April 2007. http://www.rpi.edu/dept/arc/docs/latex/latex-intro.pdf