Beispiel 2 - Python Basics

VU Einführung in Visual Computing, TU Wien, SS 2023 03.03.2023

1 Übersicht

Das zweite Beispiel besteht aus vier Teilen: Im ersten Teil ist es Ziel sich mit Grundlagen von Python und Dokumentation vertraut zu machen und eigene Versionen von built-in `numpy` Funktionen zu implementieren. Im zweiten Teil dieses Übungsbeispiels soll die Theorie bzgl. Graphikpipelines und Objektrepräsentation praktisch umgesetzt werden. Der dritte Teil macht dich mit den grundlegenden Kenntnissen im Umgang mit Bildern und Filtern in Python vertraut und der vierte Teil repräsentiert ein praktisches Anwendungsbeispiel bzgl. der Theorie der Transformationen. Für den Programmablauf wird ein Jupyter-Notebook (`.ipynb`-Dateien) zur Verfügung gestellt, die Funktionen sind in dem jeweiligen Python-Dateien (`.py`) zu implementieren. Das Jupyter-Notebook kann prinzipiell verändert werden und ist für die Bewertung irrelevant, stelle jedoch sicher, dass deine Funktionen ident mit der jeweiligen Spezifikation ist.

Bewertung	
datastructures.py :	5 Punkte
triangles.py :	5 Punkte
images.py :	5 Punkte
transformations.py :	5 Punkte
Maximal erreichbare Punkte :	**20 Punkte**

Abzugeben ist eine ZIP-Datei via TUWEL, die folgende 4 Dateien beinhaltet:
datastructures.py, triangles.py, images.py, transformations.py

1.1 Allgemeine Informationen zur Abgabe

Virtuelle Umgebung In dem Codeframework ist ebenfalls eine virtuelle Umgebung (`.yml`) enthalten, das alle benötigten Bibliotheken installiert. (Wir werden für Beispiel 3 und 5 ebenfalls diese Umgebung verwenden.) Diese kann mittels conda über `conda env create -f FILENAME.yml` installiert werden. Die virtuelle Umgebung muss dann im Editor beim Ausführen der Dateien ausgewählt werden. Bei Problemen recherchiere zuerst im Internet und melde dich bei Bedarf danach im TUWEL Forum.

Es ist sicherzustellen, dass Python beim Ausführen der abgegebenen Dateien keine Fehlermeldungen ausgibt, das Skript abstürzt oder ähnliches. **Wir können keinen fehlerhaften bzw. nicht ausführbaren Code bewerten! Dateinamen, Methodensignaturen und die Bezeichnungen von**

vorgegebenen Variablen dürfen auf keinen Fall verändert werden! Ein Missachten dessen kann dazu führen, dass für Teilaufgaben keine Punkte vergeben werden.
Es liegt in deiner Verantwortung, rechtzeitig **vor der Deadline** zu kontrollieren, ob deine Abgabe funktioniert hat. **Kontrolliere bitte deine Abgabe bzgl. folgender Punkte:**

- Werden die hochgeladenen Dateien angezeigt?
- Kannst du die Dateien herunterladen und öffnen oder im Browser anzeigen?
- Hast du die richtigen Dateien abgegeben?

Sollte der Upload deiner Dateien nicht funktionieren, kannst du uns via TUWEL-Forum kontaktieren oder eine E-Mail an **evc@cg.tuwien.ac.at** senden.

Hinweis: Betrifft nur MacOS M1/M2: Da wir in Aufgabe 3 und 5 ein graphisches Interface zur Verfügung stellen, das auf PyQt6 basiert, installiere miniconda mittels dem Intel-Installer.

1.2 Allgemeine Hinweise

Python-Hilfe verwenden: Um mehr Informationen eines Python-Befehl zu erhalten, tippe `help(Befehl)` in das Kommandozeile einer Pythonkonsole oder des Jupyter-Notebooks.

Debuggen (Visual Studio Code):

- **Breakpoints setzen :** Setze einen Breakpoint mittels Klick auf die Zeile entsprechend 1. Du kannst den Debug-Prozess starten, indem du im Jupyternotebook auf *Debug Cell* klickst.
- **Variablen ausgeben:** In der sogenannten Debug-Console kannst du während des Debuggens Variablen ausgeben, verändern oder Code ausführen.

Internet & Tutorials verwenden: Es gibt jede Menge an Python Tutorials im Internet, speziell wenn es um Debugging geht. Ein Beispiel hierfür wäre **Debugging**[1] von Visual Studio Code.

2 Basics

In der ersten Aufgabe sollst du dich mit ein paar Grundlagen von Python, sowie der Python-Hilfe vertraut machen und gleichzeitig ein wenig Vektorrechnung bzw. Matrizenrechnung wiederholen. Du wirst dieses Wissen bei den weiteren Beispielen und auch bei den Tests benötigen. Für lineare Algebra und mathematische Funktionen hat sich in Python die Bibliothek `numpy` etabliert.

Öffne die Datei datastructures.ipynb und datastructures.py! Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren in dieser Datei wieder. Die Stellen, an denen du deine Lösung implementieren sollen, sind jeweils mit einem *TODO* in `datastructures.py` markiert. Sei dir bei der

[1] Debug in VSCode: https://code.visualstudio.com/docs/editor/debugging

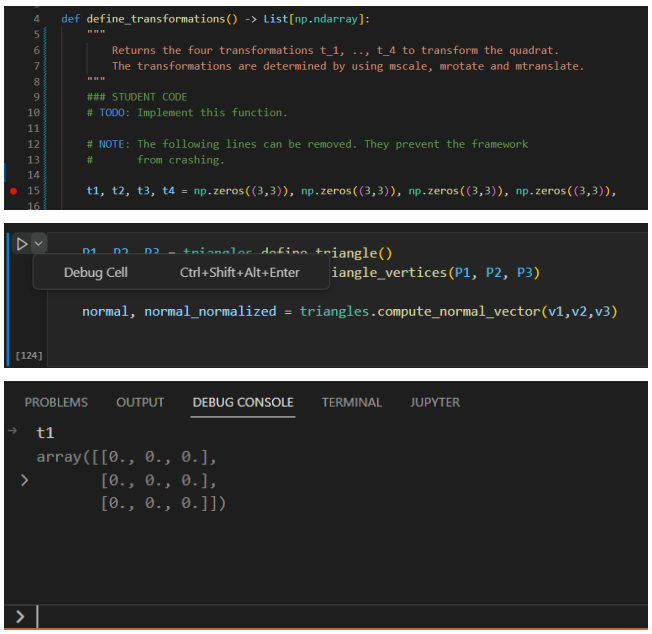


Abbildung 1: Debuggen in Python mittels VSCode

Implementierung der folgenden Aufgaben stets sicher, dass du genau weißt, was in welcher Zeile passiert. **Beim Abgabegespräch müsst du deinen Code im Detail erklären können um Punkt dafür zu bekommen. Auch wenn deine abgegebene Lösung zu 100% richtig ist, bekommst du erst die Punkte, wenn du diese einwandfrei erklären kannst.**

2.1 Basis-Datenstrukturen in Python erzeugen

1. Erzeuge in `define_structures()` mittels `np.array` zwei eindimensionale Vektoren v_1 (1) und v_2 (2) mit jeweils drei Werten, sowie eine Matrix M (3) mit 3 Zeilen und 3 Spalten. Entnehme dabei die Werte für Vektoren und Matrix deiner Matrikelnummer (8-stellig) in folgender Reihenfolge: Angenommen deine Matrikelnummer lautet ABCDEFGH, so sei:

$$v_1 = \begin{bmatrix} D & A & C \end{bmatrix} \tag{1}$$

$$v_2 = \begin{bmatrix} F & B & E \end{bmatrix} \tag{2}$$

$$M = \begin{bmatrix} D & B & C \\ B & G & A \\ E & H & F \end{bmatrix} \tag{3}$$

   Setze für ABCDEFGH die Ziffern deiner Matrikelnummer ein! Editiere die mit *TODO* markierten Zeilen entsprechend. Die Funktion `define_structures` soll eine Liste bestehend aus v_1, v_2, M returnieren.

2. Erzeuge in `sequence(M)` aus der Matrix M eine Sequenz, die von der kleinsten Ziffer deiner Matrikelnummer bis zur größten Ziffer deiner Matrikelnummer im Intervall 0.25 läuft! Verwende für diese Aufgabe die Funktion `np.arange` und die Funktionen `np.min` und `np.max`. **Verwende direkt die Matrix M! Beispiel:** Die Matrikelnummer 01233120 soll folgenden Vektor v_3 (4) erzeugen:

$$v_3 = \begin{bmatrix} 0.0 & 0.25 & 0.5 & 0.75 & 1.0 & 1.25 & 1.5 & 1.75 & 2.0 & 2.25 & 2.0 & 2.25 & 2.5 & 2.75 & 3.0 \end{bmatrix} \quad (4)$$

3. Erzeuge in `matrix(M)` eine Matrix $M_{15\times 9}$ (5) mit 15 Zeilen und 9 Spalten! Die Matrix soll wie ein Schachbrett aufgebaut sein, dessen weiße Felder mit 0 gefüllt sind und dessen schwarze Felder mit der Matrix M gefüllt sind. Entsprechend ist ein weißes bzw. schwarzes Feld 3×3 Zellen groß. Dabei sollen die Ecken des Schachbrettes schwarz sein (d.h. 3×3 Zellen mit der Matrix gefüllt).

   Wichtig: Dieser Teil des Beispiels (das Erstellen der fertigen Schachbrett-Matrix) soll direkt auf die Matrix M zugreifen und dabei maximal 7 Zuweisungsoperationen verwenden. Es dürfen keine Schleifen oder Rekursionen verwendet werden.

$$M_{15\times 9} = \begin{bmatrix} D & B & C & 0 & 0 & 0 & D & B & C \\ B & G & A & 0 & 0 & 0 & B & G & A \\ E & H & F & 0 & 0 & 0 & E & H & F \\ 0 & 0 & 0 & D & B & C & 0 & 0 & 0 \\ 0 & 0 & 0 & B & G & A & 0 & 0 & 0 \\ 0 & 0 & 0 & E & H & F & 0 & 0 & 0 \\ D & B & C & 0 & 0 & 0 & D & B & C \\ B & G & A & 0 & 0 & 0 & B & G & A \\ E & H & F & 0 & 0 & 0 & E & H & F \\ 0 & 0 & 0 & D & B & C & 0 & 0 & 0 \\ 0 & 0 & 0 & B & G & A & 0 & 0 & 0 \\ 0 & 0 & 0 & E & H & F & 0 & 0 & 0 \\ D & B & C & 0 & 0 & 0 & D & B & C \\ B & G & A & 0 & 0 & 0 & B & G & A \\ E & H & F & 0 & 0 & 0 & E & H & F \end{bmatrix} \quad (5)$$

   Als Zuweisungsoperation gilt jeder Befehl, in dem = (ist-gleich) als Zuweisung, und nicht als Vergleich vorkommt. Du kannst dafür die `numpy`-Funktionen `np.stack`, `np.hstack`, `np.vstack` verwenden.

2.2 Implementieren von numpy-Funktionen

Implementiere deine eigene Versionen der Skalarprodukt-, Kreuzprodukt und Vektor/Matrix-Matrixmultiplikationen in `numpy`.

ACHTUNG: Es ist nicht erlaubt, die Funktionen `np.cross`, `np.dot`, `np.multiply`, `np.matmul` oder den '@'-Operator zum Lösen dieser Aufgabe zu verwenden, sie sollen selbst implementiert werden.

Außerdem darf der '*'-Operator nur verwendet werden, um zwei einzelne Werte miteinander zu multiplizieren. Es dürfen zum Lösen dieser Aufgabe Schleifen oder Rekursionen verwendet werden.

Hinweis: Um deine Lösung auf Korrektheit zu überprüfen, kannst du einfach dein Resultat mit dem Resultat vergleichen, das dir die entsprechende `numpy`-Funktion liefert.

1. Implementiere die Funktion `dot_product(`v_1, v_2`)`, die das Skalarprodukt von zwei 3-Element-Vektoren berechnet!
2. Implementiere die Funktion `cross_product(`v_1, v_2`)`, die das Kreuzprodukt zweier 3-Element-Vektoren berechnen soll!
3. Implementiere die Funktion `vector_X_Matrix(`v, M`)`, die einen 3-Element-Vektor von links mit einer 3×3-Matrix multipliziert!
4. Implementiere die Funktion `matrix_X_vector(`M, v`)`, die einen 3-Element-Vektor von rechts mit einer 3×3-Matrix multipliziert!
5. Implementiere die Funktion `matrix_X_matrix(`M_1, M_2`)`, die zwei 3×3-Matrizen miteinander multipliziert! Dabei soll die Matrix M_1 von links auf M_2 multipliziert werden $M_1 \times M_2$.
6. Implementiere die Funktion `matrix_Xc_matrix()`, die zwei 3×3-Matrizen komponentenweise miteinander multipliziert (Hadamard-Produkt)! Das heißt, jeder Wert der Matrix M_1 wird mit jenem Wert der Matrix M_2 multipliziert, der die gleichen Koordinaten hat.

2.3 Hinweise für das Abgabegespräch

Beim Abgabegespräch musst du deinen Code erklären können und genau wissen, was die von dir verwendeten Befehle tun. Du musst auch die Theorie hinter den implementierten Befehlen kennen (z.B. was ist ein Skalarprodukt, wofür wird es verwendet und was sagt es aus).

3 Triangles

Theorie in Vorlesung: *Graphikpipeline + Objektrepräsentation*

In dieser Python-Aufgabe sollst du dich mit den wichtigsten Primitiven in der Computergrafik befassen - den Dreiecken. Dieses Wissen wird bei den weiteren Beispielen und auch beim Test benötigt.

Öffne die Dateien Triangles.ipynb und Triangles.py Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren im Python-File wieder. Die Stellen, an denen du deine Lösung implementieren sollst, sind jeweils mit einem *TODO* markiert.

3.1 Triangles

1. Erzeuge in `define_triangle()` drei Vektoren P_1 (6), P_2 (7) und P_3 (8), welche die drei Punkte eines Dreiecks repräsentieren, mit folgenden Koordinaten:

$$P_1 = \begin{bmatrix}(1+C) & -(1+A) & -(1+E)\end{bmatrix} \tag{6}$$

$$P_2 = \begin{bmatrix}-(1+G) & -(1+B) & (1+H)\end{bmatrix} \tag{7}$$

$$P_3 = \begin{bmatrix}-(1+D) & (1+F) & -(1+B)\end{bmatrix} \tag{8}$$

   Wobei ABCDEFGH wie im Beispiel 2.2-Basics die Matrikelnummer ist.

2. Erzeuge in `define_triangle_vertices`(P_1, P_2, P_3) nun die 3 Seiten-Vektoren des Dreiecks: P_1P_2, P_2P_3 und P_3P_1.
   P_1P_2 soll von P_1 nach P_2 zeigen.
   P_2P_3 soll von P_2 nach P_3 zeigen.
   P_3P_1 soll von P_3 nach P_1 zeigen.

3. Berechne in `compute_lengths`(v_1, v_2, v_3) die Längen der Seiten des Dreiecks. Die Länge eines Vektors wird auch *euklidische Norm* eines Vektors genannt.

   Achtung: Die `numpy`-Funktion *np.linalg.norm* darf zur Lösung dieser Aufgabe nicht verwendet werden, da ihre Funktionalität nachimplementiert werden soll. Du kannst aber deine Ergebnisse mit den Ergebnissen, die dir die Funktion liefern würde, vergleichen.

4. Berechne in `compute_normal_vector`(v_1, v_2, v_3) den Normalvektor der Dreiecksfläche und den normalisierten Normalvektor! Zum Lösen dieser Aufgabe ist es nun auch erlaubt die `numpy`-Funktionen `np.dot`, `np.cross` und `np.linalg.norm` zu verwenden. **Achtung: Ein Normalvektor und ein normalisierter Vektor sind nicht das Gleiche!**

5. Berechne in `compute_triangle_area`(n) die Fläche des Dreiecks mit der Hilfe des Normalvektors n! Zum Lösen dieser Aufgabe ist es nun auch erlaubt die `numpy`-Funktionen `np.dot`, `np.cross` und `np.linalg.norm` zu verwenden.

   Hinweis: Hat die Richtung des Normalvektors hier einen Einfluss?

6. Berechne die drei Winkel des Dreiecks in Grad! Nenne die Winkel α beim Eckpunkt P_1, β bei P_2 und γ bei P_3.

7. Im Jupyter-Notebook werden Eckpunkte, Vektoren als auch der Normalvektor des Dreieck in einem Plot generiert. Du kannst mit Hilfe des Notebooks deinen Code verifizieren.

   Hinweis: Beachte die Richtung der Vektoren! Verwende die `numpy`-Funktion `np.arccos`, um den Arkuskosinus zu berechnen. Achte auf die Einheit der Winkel - `numpy` arbeitet standardmäßig mit der Einheit Radiant! Überprüfe deine Lösung! Ist die Summe deiner drei errechneten Winkel korrekt?

3.2 Hinweise für das Abgabegespräch

Beim Abgabegespräch musst du deinen Code erklären können, und genau wissen, was die von dir verwendeten Befehle tun. Du musst auch die Theorie hinter den nachimplementierten Befehlen kennen.

4 Images

Dieses Beispiel soll grundlegende Kenntnisse im Umgang mit Bildern und Filtern in Python erläutern.

Lade die Dateien Images.py, Images.ipynb und mandrill_color.jpg im Ordner `images`.

Wichtig: Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren im Python-File wieder. Die Stellen, an denen du deine Lösung implementieren sollst, sind jeweils mit einem *TODO* markiert.

4.1 Image Basics

Hinweis: Zum Lösen dieser Beispiele werden Funktionen der Bibliothek **Pillow**[2], wie beispielsweise `Image.open`, und **scipy**[3], z.B. `correlate`, benötigt.

1. Das Bild wird mittels der bereits vorhandenen Funktion `read_img(path)` eingelesen.
2. Konvertiere in der Funktion `convert(img)` das in Punkt 1 mittels der Bibliothek `Pillow` geladene Bild in float-Werte zwischen 0 und 1 (d.h. 0 wird auf 0 abgebildet, 255 wird auf 1 abgebildet). Das Eingangsbild kannst mittels der `numpy`-Funktion `np.array()` in den richtigen Datentyp casten.

   Achtung: Für diese Aufgabe dürfen **keine Schleifen oder Rekursionen** verwendet werden.
3. Vertausche in der Funktion `switch_channels` im Bild aus Punkt 2 den Rot- mit dem Grün-Kanal! Das Ergebnis soll so aussehen wie in Abbildung 2 a) dargestellt.

a) image_swapped b) image_masked

Abbildung 2: Beispiele für image_swapped und image_masked

4. Erstelle in der Funktion `image_mark_green(img)` ein zwei-dimensionales Bild aus binären Werten `True` und `False` in dem jedes Pixel markiert ist, bei dem der Grün-Kanal im Bild aus Punkt 2 größer oder gleich 0.7 ist!

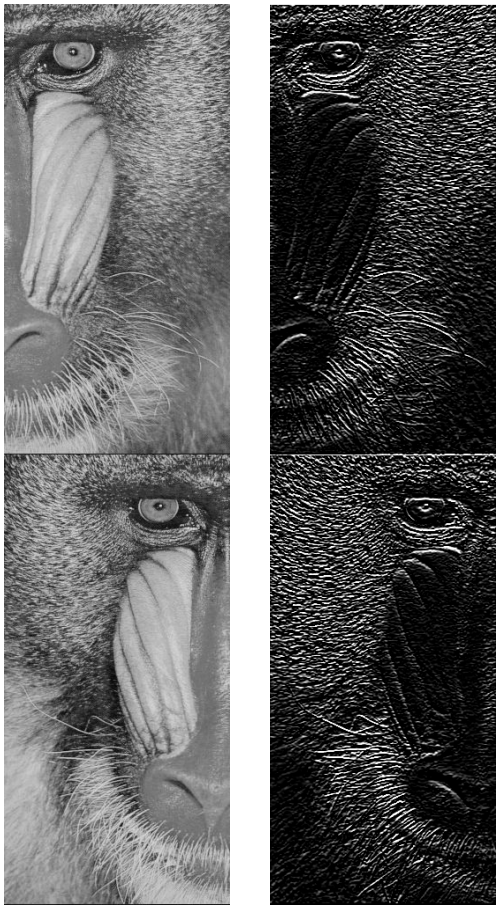
[2]Pillow: https://pillow.readthedocs.io/en/stable/
[3]Scipy: https://scipy.org/

Achtung: Für diese Aufgabe dürfen **keine Schleifen oder Rekursionen** verwendet werden.

5. Setze in der Funktion `image_masked(img, mask)` im Bild aus Punkt 2 alle Pixel auf Schwarz, die im logical-Bild aus Punkt 4 markiert sind (also die Pixel, bei denen der Grün-Kanal im Bild größer-gleich 0.7 ist)! Das Ergebnis soll so aussehen wie in Abbildung 2 b).

   Achtung: Für diese Aufgabe dürfen **keine Schleifen oder Rekursionen** verwendet werden.

6. Wandel in `grayscale(img)` mittels der zur Verfügung gestellten Funktion `utils.rgb2gray(img)` das Bild aus Punkt 2 in ein Graustufenbild um und transformiere dieses in der Funktion `cut_and_reshape(img` von 512×512 nach 1024×256. Dabei soll das Bild in der Mitte zerschnitten werden und der linke Teil unten an den rechten Teil angefügt werden. Das Ergebnis soll so aussehen wie in Abbildung 3 a).



a) image_reshaped b) image_edge

Abbildung 3: Beispiele für image_reshaped und image_edge

4.2 Filter und Faltungen

Theorie in Vorlesungen: *Local Operations, Edge Filtering*

1. Implementiere die Funktion *filter_image(img)*, welche ein RGB-Eingabebild (`img`) mit einem 5×5 Filterkern (`gaussian`) filtert. Der Kernel ist bereits vorhanden und wurde mittels der Funk-

tion `utils.gauss_filter` erstellt. Für alle Pixel, die außerhalb des Bildes liegen, soll [0,0,0] angenommen werden. Das Ergebnis der Funktion soll in etwa so aussehen wie Abbildung 4.

Abbildung 4: image_convoluted

Achtung: Für diese Aufgabe **dürfen Schleifen oder Rekursionen verwendet werden**. Implementiere die Berechnung selbst. Verwende keine Funktionen wie bspw.

- `np.convolve`,
- `scipy.ndimage.convolve` oder
- `scipy.ndimage.correlate`.

Hinweis: Das Ausgabebild muss immer die gleiche Größe haben wie das Eingabebild.

2. Erstelle in `horizontal_edges(img)` ein Kantenbild, in dem nur die horizontalen Kanten des Bildes aus 4.1 Punkt 6 zu sehen sind. Hierzu soll ein Sobel-Filter $\mathbf{G}_{\text{Horizontal}}$ der Form

$$\mathbf{G}_{\text{Horizontal}} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{9}$$

verwendet werden. Verwende zur Berechnung die Funktion `scipy.ndimage.correlate`, Randpixel sollen konstant mit dem Wert 0 angenommen werden. (`mode='constant'`). Das Ergebnisbild soll die gleiche Größe wie das Eingabebild haben und in etwa so aussehen wie Abbildung 3b.

4.3 Hinweise für das Abgabegespräch

Beim Abgabegespräch musst du deinen Code erklären können und genau wissen, was die von dir verwendeten Befehle tun. Du musst auch die Theorie hinter den nachimplementierten Befehlen kennen (z.B. was ist ein Gauß-Filter, wofür wird er verwendet und wie schaut eine Gauß-Kurve aus).

5 Transformationen

Theorie in Vorlesung: *Transformationen*

In diesem Beispiel sollst du dich mit den Grundlagen der Transformationen und Transformationsmatrizen bekannt machen.

Lade die Dateien Transformations.py und Transformations.ipynb! Die folgende Angabe findet sich auch in Form von (englischen) Kommentaren im Python-File wieder. Die Stellen, an denen du deine Lösung implementieren sollst, sind jeweils mit einem *TODO* markiert.

Vorlesungsunterlagen: In den Vorlesungsunterlagen sind weitere hilfreiche Informationen zu den Transformationen zu finden.

5.1 Transformationen und Plotten

1. Implementiere die drei Funktionen `mtranslate(`t_x`, `t_y`)`, `mrotate(angle)` und `mscale(`s_x`,`s_y`)`, welche 3×3 Basis-Transformationsmatrizen zurückgeben sollen. Details zu den Parametern findest du im Kommentar zur jeweiligen Funktion. Hilfreiche Funktionen: `np.sin`, `np.cos`
2. Mit der Funktion `display_vertices(v, title)` kannst du die Vertices plotten, siehe Abbildung 5.

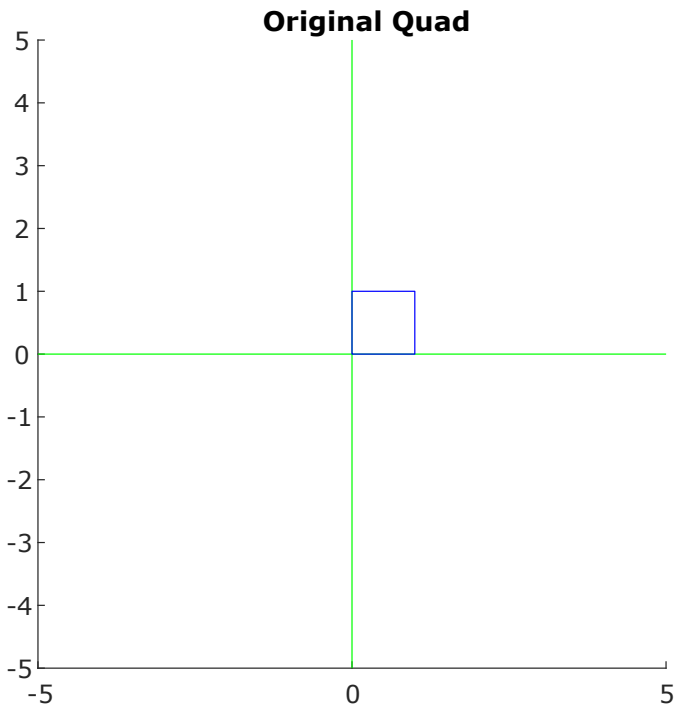


Abbildung 5: Original Quad

3. Vervollständige die Funktion `transform_vertices(v,m)`, welche ein `np.array` von N Vertices v der Dimension $(3 \times N)$ und eine 3×3 Transformationsmatrix als Eingabe nimmt und jeden Vertex mit dieser Matrix transformiert. Wenn diese Methode korrekt implementiert ist, sollte das zweite Bild (Rotated Quad) wie in Abbildung 6 aussehen.

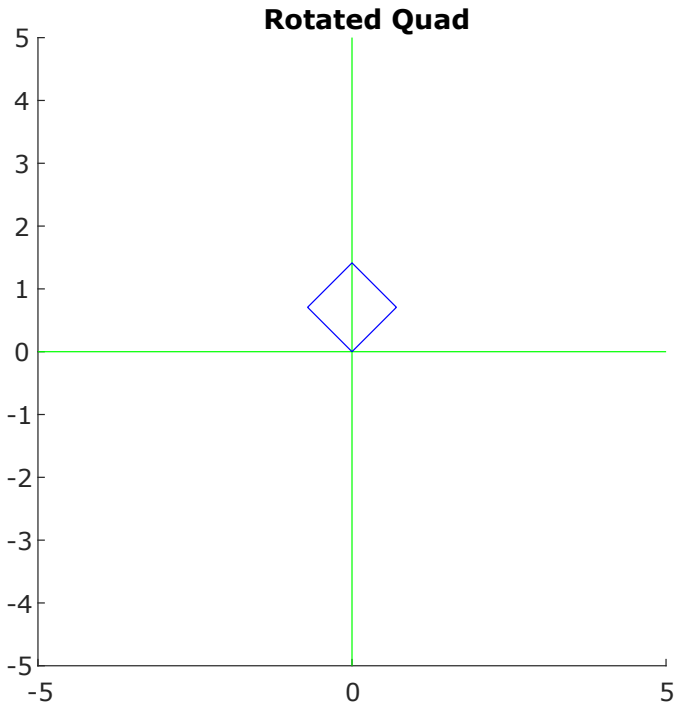


Abbildung 6: Rotated Quad

4. Benutze die zuvor implementierten Funktionen, um die folgenden vier Ziel-Bilder (siehe Abbildungen 7, 8, 9, 10) zu erstellen! Nutze die Matrix-Funktionen aus Punkt 1, um eine geeignete Transformation zu erstellen. Die Transformationen können mit Hilfe des Matrizenmultiplikationsoperator '@' verbunden werden. Die erstellten Transformationen sollen in der Funktion `define_transformations()` definiert werden. **Wichtig:** Die Funktion `transform_vertices(v,m)` darf nur einmal pro Bild aufgerufen werden.

5.2 Hinweise für das Abgabegespräch

Beim Abgabegespräch musst du deinen Code erklären können und genau wissen, was die von dir verwendeten Befehle tun. Du musst auch die Theorie hinter den nachimplementierten Befehlen kennen (z.B. warum braucht man bei 2D Punkten eine 3×3 Matrix)!

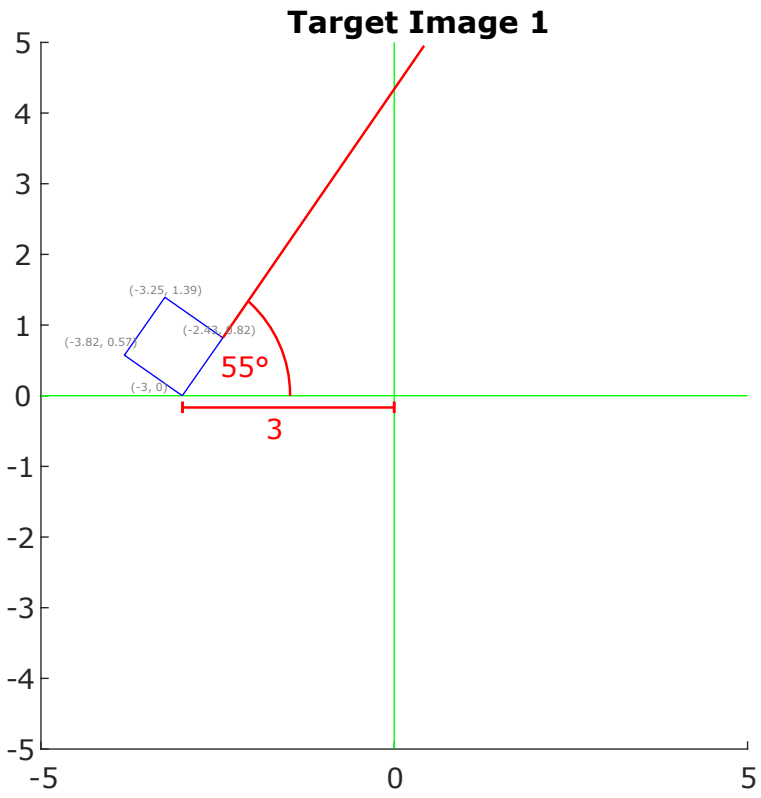


Abbildung 7: Target Image 1

Abbildung 8: Target Image 2

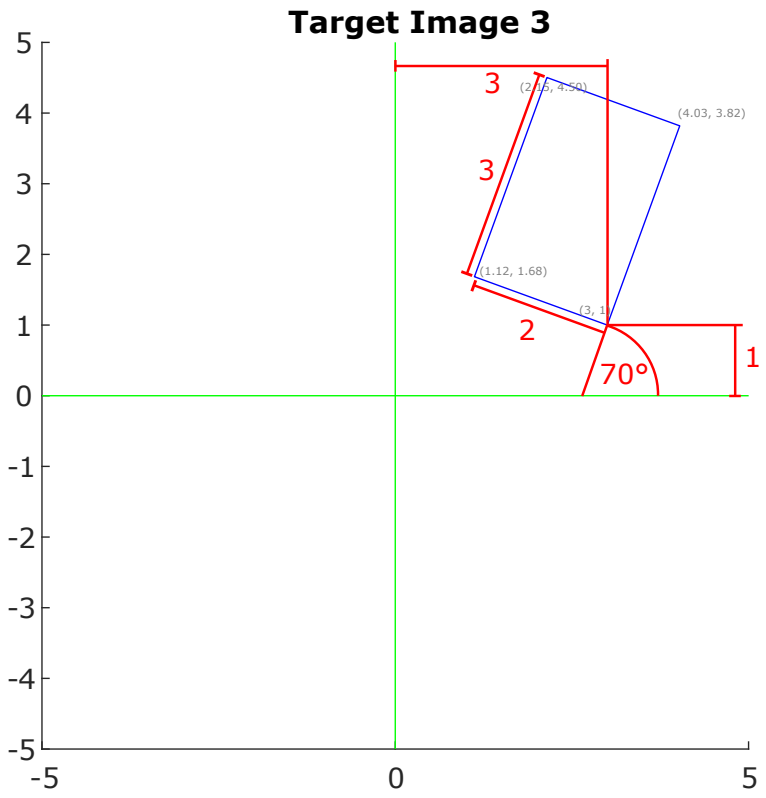


Abbildung 9: Target Image 3

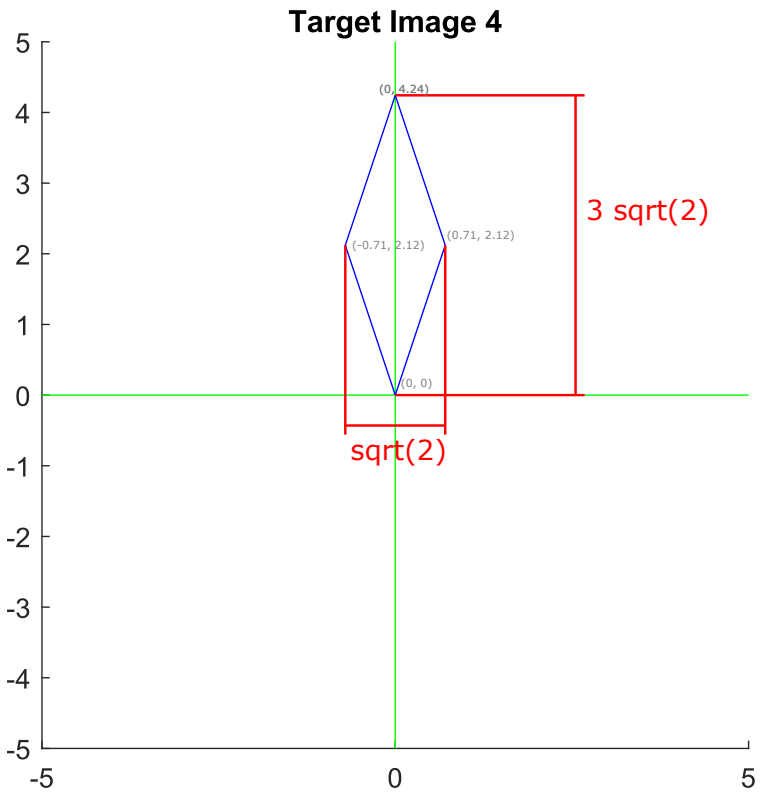


Abbildung 10: Target Image 4