# Abgabe

Die Übungen werden in GitHub Classroom bearbeitet und via Moodle abgegeben. Insgesamt sind **maximal 35 Punkte** erreichbar. Bei der Abgabe via Moodle ist es wichtig, dass ihr **den Link zu eurem GitHub Repository** (aus dem URL/Suchfeld) in das Textfeld (Onlinetext) bei der Abgabe kopiert, damit wir eure Abgabe bewerten können. Der Link hat folgendes Format:

`https://github.com/hpi-aiis-pt1-fall25/<übungsname>-<username>`

Wichtig: die Bewertung erfolgt in einem *Testatgespräch*, d.h. ihr müsst eure Lösungen in einem Termin mit einem Tutor oder einer Tutorin besprechen. Auschlaggebend für die Bewertung ist, dass ihr eure Lösungen erklären und Fragen dazu beantworten könnt! Wir werden den Stand des letzten Commit bewerten, der vor der jeweiligen Deadline abgegeben wurde.

**ACHTUNG:** Für diese Übung habt ihr **zwei Wochen bzw. drei Wochen** Zeit. Die Abgabe erfolgt am 23.11.2025 23:59:59 (Aufgabe 1&2) und 30.11.2025 23:59:59 (Aufgabe 3). **Der Umfang ist jedoch auch entsprechend größer. Wir empfehlen euch dringend, frühzeitig mit der Bearbeitung zu beginnen.**

**2er-Gruppen:** In dieser Übung könnt ihr in 2er-Gruppen (den Gruppen eurer Testatgespräche) arbeiten. In diesem Fall kann die Abgabe in einem eurer Repositories erfolgen. Wie immer muss jeder von euch in der Lage sein, die Lösung zu erklären und Fragen dazu zu beantworten.

**MeMe – Die Meme-Meisterschaft:** In langer Tradition der von unserem Lehrstuhl veranstalteten PT1-Vorlesungen gibt es auch dieses Jahr wieder die Meme-Meisterschaft. Teil dieser Übung ist es, dass ihr ein **eigenes** Meme mit HPI-, Studiums- oder Informatikbezug erstellt. Das beste Meme wird nach Abgabe der Übung von einer Jury gekürt und erhält einen Preis. In der Übungssession am 19.11 stellen wir die Regeln noch einmal vor. Ihr seid herzlich eingeladen auch nur zu diesem Teil der Übungssession zu kommen. Pro 2er-Gruppe kann nur ein Meme eingereicht werden, allerdings könnt ihr auch zwei Memes erstellen und mit eurem Tutor oder eurer Tutorin besprechen, welches Meme ihr einreicht.

# Aufgabe 1: Erstmal Einlesen - 10 Punkte (Deadline: 23.11.2025 23:59:59)

In dieser Übung sollt ihr ein Programm schreiben, dass ein gegebenes Meme in ASCII-Art konvertiert. Zuerst müssen wir jedoch die Bilddatei einlesen. Wir werden mit dem leicht lesbaren PPM Format `P3` arbeiten.

In der ersten Zeile steht `P3` als "magic string", um das Format zu identifizieren. In der zweiten Zeile stehen die Dimensionen des Bildes, also die Breite und Höhe in Pixeln (zuerst die Breite, dann die Höhe). In der dritten Zeile steht der maximale Wert, den ein RGB-Wert annehmen kann. Da wir mit 8-Bit Farbtiefe arbeiten, ist dieser Wert 255. In den folgenden Zeilen stehen die RGB-Werte der einzelnen Pixel, wobei jeder Wert durch ein Leerzeichen getrennt ist. Die Reihenfolge der Pixel ist von links nach rechts und von oben nach unten.

Ein Beispiel für ein 5x3 (Breite x Höhe) Pixel Bild, das nur aus roten Pixeln besteht, sieht wie folgt aus:

```
P3
5 3
255
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
255 0 0 255 0 0 255 0 0 255 0 0 255 0 0
```

**Hinweis:** Beachtet, dass sich im PPM Format die Zeilenumbrüche nicht unbedingt nach der Dimension des Bildes richten müssen. Stattdessen kann z.B. auch alle 1024 Werte ein `\n` eingefügt werden, ohne die Dimensionen des Bildes zu beachten. Wir haben für Aufgabe 2 ein Python-Skript geschrieben, das bei der Konvertierung zu PPM die Bilddimensionen beachtet, sodass jede Zeile in der PPM-Datei genau eine Zeile Pixel des Bildes enthält. Nutzt ihr jedoch andere Tools, müsst ihr die Informationen über Breite und Höhe aus der zweiten Zeile des PPM Formates nutzen, um die Datei korrekt einzulesen.

**Die Aufgabe:** Lest die `beispiel.ppm` Datei ein[1] und gebt die RGB-Werte der einzelnen Pixel zeilenweise auf der Kommandozeile aus. Um jedes RGB-Tripel sollen Klammern gesetzt werden, die einzelnen Werte sollen durch Leerzeichen getrennt werden. Jede Zeile in der Ausgabe soll einer Zeile im Bild entsprechen. Ein 5x3 (Breite x Höhe) Bild hat also 3 Zeilen in der Ausgabe, mit je 5 Tripeln für die RGB-Werte pro Pixel. Die Ausgabe soll also wie folgt aussehen:

```
(255 0 0) (255 0 0) (255 0 0) (255 0 0) (255 0 0)
(255 0 0) (255 0 0) (255 0 0) (255 0 0) (255 0 0)
(255 0 0) (255 0 0) (255 0 0) (255 0 0) (255 0 0)
```

Ihr könnt eure Funktion auch an der `beispiel_tricky.ppm` testen. Diese Datei enthält die gleichen Pixel wie `beispiel.ppm`, aber die Zeilenumbrüche sind anders gesetzt. Die Abgabe soll in der Datei `read_ppm.c` erfolgen. Tipp: schaut euch den Inhalt von `beispiel.ppm` und vergleicht ihn mit der Ausgabe eures Programms.

# Aufgabe 2: Memes mal anders - 15 Punkte (Deadline: 23.11.2025 23:59:59)

Nun kommen wir zum eigentlichen Teil der Übung: Das Konvertieren von Memes in ASCII-Art.

## JPG zu PPM konvertieren

In der Regel werden Memes als JPG-Dateien geteilt. Falls eure Input-Datei kein JPG ist, könnt ihr Online-Tools verwenden, um sie zu konvertieren. Ihr habt verschiedene Möglichkeiten eure JPG-Dateien in das PPM-Format zu konvertieren.

1. Ihr verwendet ein Python-Skript von uns, das eine JPG-Datei in PPM umwandelt. Das Skript findet ihr im Repository unter `jpg2ppm.py`. Es kann wie folgt aufgerufen werden:

   ```
   python jpg2ppm.py my_very_cool_meme.jpg
   ```

   und schreibt die RGB-Werte in eine Datei mit dem gleichen Namen, aber der Endung `.ppm`. Im Beispiel ist die Outputdatei also `my_very_cool_meme.ppm`. **Hinweis:** Bevor ihr das Skript ausführt, solltet ihr eine virtuelle Python-Umgebung erstellen und die notwendigen Pakete installieren. Siehe die Hinweise weiter unten.

2. Alternativ könnt ihr auch ImageMagick verwenden, um die Bilder in PPM zu konvertieren. Verwendet dafür folgenden Befehl:

   ```
   convert my_very_cool_meme.jpg -compress none my_very_cool_meme.ppm
   ```

   **Hinweis:** ImageMagick schreibt die RGB-Werte nicht so, dass jede Zeile genau eine Zeile Pixel des Bildes enthält. Stattdessen wird nach einer beliebigen festen Anzahl Werte (z.B. 1024) ein `\n` eingefügt. Beachtet das bei der Implementierung, falls ihr ImageMagick verwendet. Eure Implementierung sollte also sowohl für `beispiel.ppm` als auch für `beispiel_tricky.ppm` aus Aufgabe 1 funktionieren.

#### Wie rufe ich ein Python-Skript auf?

Da wir einige zusätzliche Pakete benötigen, die nicht standardmäßig installiert sind, solltet ihr das Skript in einer virtuellen Umgebung ausführen. Am einfachsten geht das mit venv.
Bevor wir anfangen: `python -V` sollte `Python 3.x` ausgeben. Ist das bei euch nicht der Fall und ihr bekommt stattdessen `Python 2.x`, müsst ihr eventuell `python3` statt `python` verwenden.

Mit `python -m venv .venv` erstellt ihr eine neue virtuelle Umgebung im Ordner `.venv`. Diese könnt ihr mit `source .venv/bin/activate` aktivieren. **Für diese Terminal-Session** benutzt ihr nun die virtuelle Umgebung. Schließt ihr das Terminal, müsst ihr die Umgebung erneut mit `source .venv/bin/activate` aktivieren. Mit `which python` könnt ihr überprüfen, ob ihr die richtige Python-Installation verwendet. Die Ausgabe sollte `<...>/.venv/bin/python` sein.

Nun können wir in der virtuellen Umgebung unsere notwendigen Pakete installieren. Dazu verwenden wir pip, das in Python 3.x standardmäßig installiert ist.

[1] In dem Programmrahmen ist das Einlesen von Dateien bzw. Schreiben in Dateien schon für euch vorgegeben.

```
pip install opencv-python tqdm numpy
```

Nun ist die Installation der Pakete abgeschlossen und wir können das Skript ausführen!

**Hinweis:** Die Dateien im `.venv` sollen nicht in GitHub hochgeladen werden.

### Die ASCII-Konvertierung

Für ASCII-Art werden die RGB-Werte jedes einzelnen Pixels zusammengefasst und mit einem einzelnen ASCII-Zeichen dargestellt. Um ein gutes Ergebnis zu erzielen, sollten sehr helle Pixel (z.B. weiß) mit einem Leerzeichen dargestellt werden, während sehr dunkle Pixel (z.B. schwarz) mit einem ASCII-Zeichen mit hoher Dichte dargestellt werden.

Ihr solltet genügend verschiedene ASCII-Zeichen verwenden, z.B. (von dunkel nach hell):

```
char asciiMap[10] = "@%#ox;:,. "
```

Euer C-Programm soll in der Datei `asciify_meme.c` abgegeben werden und folgende Funktionalität haben:

1. Das kompilierte Programm soll mit einem Argument aufgerufen werden, das den Dateinamen der Eingabedatei im PPM Format enthält. Das kompilierte Programm soll also z.B. mit `./asciify_meme my_very_cool_meme.ppm` aufgerufen werden (hint: `argc` und `argv`).
2. Das Programm soll die Datei einlesen und die RGB-Werte in einem Array speichern. Die Höhe und Breite des Bildes sollen auf der Kommandozeile ausgegeben werden.
3. Das Programm soll die RGB-Werte in ASCII-Art konvertieren und das Ergebnis in der Datei `ascii_meme.txt` speichern.

### MeMe: Die Meme-Meisterschaft

Teil dieser Übung ist es auch, dass ihr ein **eigenes** Meme mit HPI-, Studiums- oder Informatikbezug erstellt. Dieses Meme sollt ihr in ASCII-Art konvertieren und in die resultierende `ascii_meme.txt` in eurem Repository speichern. Außerdem sollt ihr auch das JPG-Original in euer Repository hochladen.

Da für jeden Pixel ein ASCII-Zeichen verwendet wird, kann die Ergebnisdatei sehr groß werden. Um das Ergebnis anzuschauen, könnt ihr die Datei z.B. in einem Browser oder VS Code öffnen und Zoom Out verwenden. Zusätzlich zu den Dateien sollt ihr auch einen **Screenshot** (so gut es geht herausgezoomed, um alles zu sehen) von dem ASCII-Meme in euer Repository hochladen. Diesen sollt ihr `screenshot_ascii_meme.png` oder `screenshot_ascii_meme.jpg` nennen. Ihr könnt den Screenshot auch auf mehrere Bilder aufteilen, falls das ASCII-Meme nicht auf ein Bild passt (fügt dann fortlaufende Nummern an die Dateinamen an).

Am Ende der Übung wird eine Jury das beste Meme küren. Kunibert konnte in einem seiner monatlichen Golfclub-Besuche die Unterstützung eines anonymen Potsdamer Mäzenen gewinnen. Neben Ruhm und Ehre wird es also auch einen Sachpreis geben. Beachtet bitte, dass wir die Memes eventuell in der Vorlesung zeigen werden. Ihr solltet den Inhalt eures Memes also also selbst verantworten können. Weitere Details und Fragen können wir in der Übungssession am 19.11. klären.

## Aufgabe 3: Mini-Memes - 10 Punkte (Deadline: 30.11.2025 23:59:59)

In dieser Aufgabe sollt ihr auf euer Programm aus Aufgabe 1 aufbauen und (in einer neuen Datei) ein Programm schreiben, das das Meme vor dem Konvertieren in ASCII-Art um einen beliebigen Faktor schrumpft. Für den Schrumpffaktor 2 würde ein 16x8 Pixel Bild zu einem 8x4 Pixel Bild schrumpfen. Hierbei sollen die RGB-Werte aller Pixel, die zu einem Pixel im verkleinerten Bild gehören, gemittelt werden.

Beim Schrumpfen kann es passieren, dass die Breite oder Höhe des Bildes nicht durch den Schrumpffaktor teilbar sind. In diesem Fall sollt ihr den übrigbleibenden Rest ignorieren. Beispiel: Bei einem 16x8 (Breite x Höhe) Bild mit einem Schrumpffaktor von 3 sind sowohl Breite als auch Höhe nicht durch den Schrumpffaktor teilbar. Es gilt `16 % 3 = 1` und `8 % 3 = 2`. Es werden also die letzte Spalte (da bei der Breite ein Rest von 1 anfällt) und die letzten beiden Zeilen (da bei der Höhe ein Rest von 2 anfällt) ignoriert.

In dieser Aufgabe sollt ihr auch die **geschrumpfte** Höhe und Breite des Bildes (als Ganzzahl ohne den Rest) zusätzlich zu den originalen Werte auf der Kommandozeile ausgeben. Im Beispiel mit einem 16x8 Bild und Schrumpffaktor 3 wäre die Ausgabe für die geschrumpften Werte also `Höhe: 5, Breite: 2`.

Der Schrumpffaktor soll via `scanf` eingelesen werden. Das Programm soll dann die Datei `ascii_meme_shrunk.txt` erstellen, die das geschrumpfte Meme in ASCII-Form enthält. Die Abgabe soll in der Datei `shrink_meme.c` erfolgen.

Ihr sollt wieder die `ascii_meme_shrunk.txt` Datei in GitHub hochladen. Außerdem sollt ihr einen **Screenshot** von dem geschrumpften ASCII-Meme in euer Repository hochladen. Diesen sollt ihr `screenshot_shrunk_ascii_meme.png` oder `screenshot_shrunk_ascii_meme.jpg` nennen. Hierfür könnt ihr einen beliebigen Schrumpffaktor verwenden.

**Bonus - 1 Punkt** Falls ihr eine bessere Methode findet, um die RGB-Werte zu zusammenzufassen, könnt ihr einen Bonuspunkt erhalten. Beschreibt eure Methode und begründet, warum sie besser ist in einem Kommentar im Code und verwendet das Stichwort `BONUS-RGB-Schrumpfen`.

**Bonus - 2 Punkte** Erweitert euer Programm so, dass es auch den Pfad zu einer JPG-Datei entgegennehmen kann. Das Programm soll dann das Python-Skript oder ImageMagick *aus C heraus* aufrufen um die JPG-Datei nach PPM zu konvertieren und mit dem Ergebnis weiterarbeiten. Es soll aber weiterhin auch mit PPM Dateien als Eingabe funktionieren (hint: Dateiendung). Beschreibt eure Methode kurz in einem Kommentar im Code und verwendet das Stichwort `BONUS-Python-Skript`.

## Checkliste

Insgesamt solltet ihr folgende Dateien bearbeiten oder in euer Repository hochladen:

**Aufgabe 1**:

- `read_ppm.c`

**Aufgabe 2**:

- `asciify_meme.c`
- `ascii_meme.txt`
- `screenshot_ascii_meme.png` oder `screenshot_ascii_meme.jpg` (oder `screenshot_ascii_meme1.png`, `screenshot_ascii_meme2.png`, ...)
- `my_very_cool_meme.jpg` (euer eigenes Meme mit beliebigem Dateinamen, wir müssen es aber finden können)
- `my_very_cool_meme.ppm` (das PPM-Format eures eigenen Memes)

**Aufgabe 3**:

- `shrink_meme.c`
- `ascii_meme_shrunk.txt`
- `screenshot_shrunk_ascii_meme.png` oder `screenshot_shrunk_ascii_meme.jpg`

**Deadline**: 23.11.2025 23:59:59 (Aufgabe 1&2) und 30.11.2025 23:59:59 (Aufgabe 3), **Bearbeitung** in GitHub Classroom, **Abgabe** via Moodle.