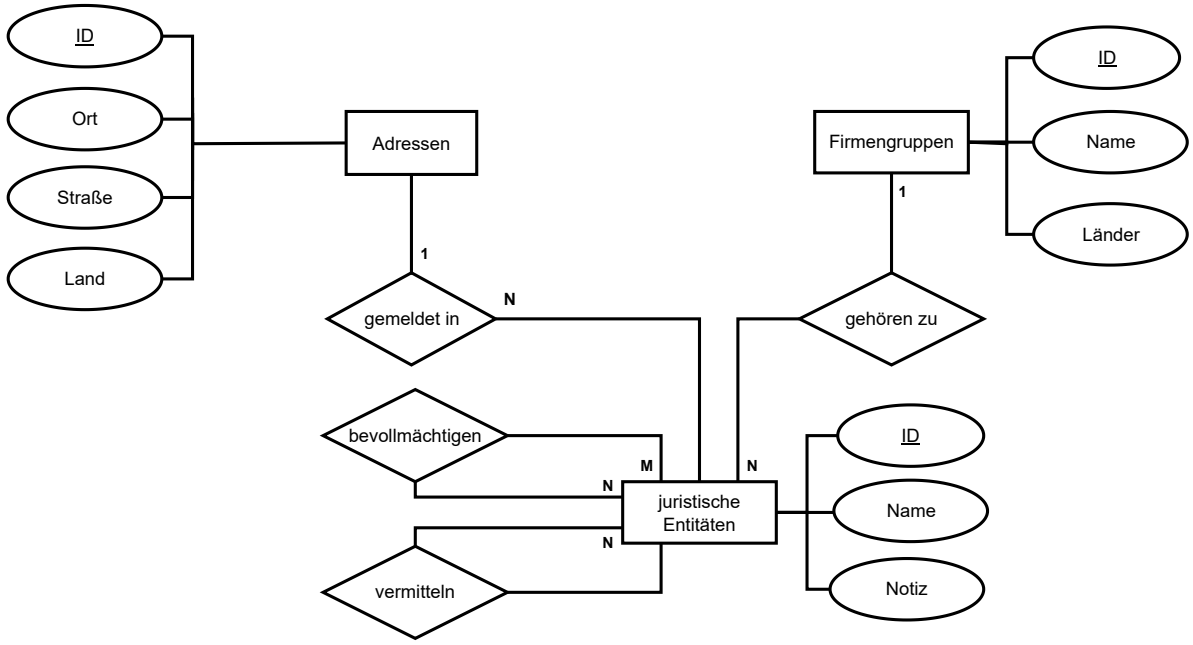
1 Cypher und natürliche Sprache (4 Punkte)

Betrachten Sie folgendes ER-Modell, dass einen Ausschnitt des Graphen des Paradise Papers-Schema modelliert.



Übersetzen Sie die folgenden Cypher Anfragen in natürliche Sprache.

(a)

```
MATCH  (jr:juristische Entitäten) -- (a:Adressen)
WHERE  a.Ort <> 'Sydney' AND a.Land = 'Australien'
RETURN COUNT(DISTINCT jr);
```

(b)

```
MATCH    w = (a:Adressen) -[*]- (f:Firmengruppen)
WITH     LENGTH(w) AS value1, NODES(w) AS value2
WHERE    value1 < 10
RETURN   value2[0].Straße
```

(c)

```
MATCH    (j1:juristische Entitäten) -[b:bevollmächtigen]-
         (j2:juristische Entitäten)
WHERE    j1.Name <> 'Kanzlei Schröder'
WITH     COUNT(DISTINCT j2) AS value, j1
WHERE    value = 5
MATCH    (j1) -- (f:Firmengruppen)
RETURN   f.Länder
ORDER BY f.Name
```

Lösung: 4 Punkte
Vorgesehen sind 1 Punkt für die ersten beiden Anfragen und 2 Punkte für die letzte. Je 0,5 Punkte Abzug für fehlende Details in der Beschreibung.

(a) Die Anzahl an unterschiedlichen juristischen Entitäten, die an einer Adresse gemeldet sind, die sich nicht im Ort Sydney, aber im Land Australien befindet.

(b) Die Straßen der Adressen, die einen Pfad zu einer Firmengruppe aufweisen, dessen Länge kleiner als 10 ist.

(c) Die Länder der Firmengruppen, zu denen eine juristische Entität gehört, deren Name nicht 'Kanzlei Schröder' ist und genau 5 verschiedene andere juristische Entitäten bevollmächtigt hat. Die Ausgabe ist dabei aufsteigend nach dem Name der Firmengruppen sortiert.

2 SQL Injection (6 Punkte)

In dieser Aufgabe betrachten wir ein System, in dem sich Nutzer mit einem Namen, einem Passwort und einer E-Mail registrieren und anmelden können. Der entsprechende Backend Code in Python sieht wie folgt aus:

```
def sign_up(uname, pwd, email):
    if '@' not in email:
        raise Exception("No valid email!")
    # open cursor to perform db operations
    cur = conn.cursor()
    # check if username already exists
    cur.execute("SELECT * FROM users WHERE username=%s;", (uname,))
    if cur.rowcount != 0:
        raise Exception("Username already exists.")
    cur.execute("INSERT INTO users(username, pwd, email) VALUES (%s, %s, %s);", (uname,
     pwd, email))
    cur.close()

def login(uname, pwd):
    # open cursor to perform db operation
    cur = conn.cursor()
    # retrieve password from db
    cur.execute("SELECT pwd FROM users WHERE username=%s;", (uname,))
    if cur.rowcount < 1:
        raise Exception("Username not in database.")
    password = cur.fetchone()[0]
    # check password
    if not (password == pwd):
        #note: failed_attempts is a session variable initialized to 0.
        failed_attempts += 1;
        #user might have forgotten password, ask for email!
        if (failed_attempts > 2):
            #automatically ask the user to reset his/her pwd
            ask_for_email()
            cur.close()
            return False
        raise Exception("Username and password do not match.")
    # login successful
    cur.close()
    return True

def ask_for_email():
    email = input("Pls enter your email: ")
    #check whether mail exists
    cur.execute("SELECT username FROM users WHERE email = '{email}'")
    if cur.rowcount < 1:
        raise Exception("email does not match with any username")
    username = cur.fetchone()[0]
    print("Success! Mail is valid for username " + username + ". Soon, you will receive
     an email with further instructions!")
    send_email_for_reset(email)
```

(a) Gehen Sie davon aus, dass es einen Benutzer mit dem Namen `admin` gibt. Wie können Sie Zugriff zu dessen Passwort erlangen? Erklären Sie Ihr Vorgehen.
Tipp: Können Sie den `UNION`-Operator hierfür verwenden?

(b) Wie kann diese SQL Injection Attacke verhindert werden? Korrigieren Sie den entsprechenden Codeabschnitt.

Lösung:

(a) Vorgesehen sind 5 Punkte, davon 3 für die Idee, zunächst falsche Loginversuche durchzuführen und anschließend eine SQL-Injection über die Mail-Eingabe durchzuführen, und 2 für die konkrete Umsetzung.
Zunächst führen wir drei mal `login('admin', 'randompwd')` aus, um insgesamt drei fehlerhafte Versuche zu bekommen, damit man aufgefordert wird eine E-Mail zum Zurücksetzen des Passwords in `ask_for_email()` einzugeben. Hier übergeben wir nun `not_an_email' UNION SELECT pwd AS username FROM users WHERE username = 'admin';--`. Dabei wird diese E-Mail nicht erneut sanitized, wodurch folgende SQL Query ausgeführt wird:

```
SELECT username
FROM   users
WHERE  email = 'not_an_email'
UNION
SELECT pwd AS username
FROM   users
WHERE  username = 'admin';--'
```

Dadurch, dass die übergebene Mail-Adresse keine richtige Mail-Adresse sein kann (sie enthält kein '@'), wird das obere SELECT kein Tupel zurückliefern, das untere jedoch schon, wodurch insgesamt ein Tupel im Gesamtergebnis sein wird, nämlich das Password. Dieses wird anschließend im Print ausgegeben, wodurch man Zugriff auf den Account des Admins erhält.

(b) Vorgesehen sind 1 Punkt, davon 0,5 Punkte für die Idee und 0,5 Punkte für die konkrete Umsetzung. Das Problem hier ist, dass die übergebene E-Mail-Adresse nicht gesanitized wird. Deshalb kann diese Attacke verhindert werden, indem wir diese sanitizen und die `ask_for_email()` Funktion entsprechend anpassen:

```
def secure_ask_for_email():
    email = input("Pls enter your email: ")
    #check whether mail exists
    cur.execute("SELECT username FROM users WHERE email = %s", (email,))
    if cur.rowcount < 1:
        raise Exception("Username and email do not match")
    username = cur.fetchone()[0]
    print("Success! Mail is valid for username " + username)
    send_email_for_reset(email)
```

3 SQL Injection in Python (6 Punkte)

Im Notebook `SQL Injection.ipynb` wurde gezeigt, wie im gegebenen einfachen Anmeldesystem mittels SQL Injection ein neuer Nutzeraccount angelegt werden kann. Sie sollen nun im beigefügten Notebook `exercise3.ipynb` eine Funktion `inject_password()` schreiben, die mittels SQL Injection das Passwort eines bestehenden Nutzeraccounts in ein beliebiges Passwort ändert. Dabei hat die Funktion die folgenden Parameter:

- `username`: Der Nutzername, dessen Passwort geändert werden soll.
- `password`: Das neue Passwort des Nutzeraccounts.
- `salt`: Der Salt mit dem der Hash des Passworts berechnet werden soll (optional).

Die Funktion soll zunächst über Auswertung einer möglichen Fehlermeldung testen, ob der gegebene Nutzer im System existiert. Falls nicht, so soll auch ein Fehler geworfen werden. Anschließend soll das Passwort des gegeben Nutzers geändert werden. Verlief die Änderung erfolgreich, so soll `True` zurückgegeben werden, andernfalls `False`. Beachten Sie, dass der Salt einem festen Format folgen muss, um von der Datenbank akzeptiert zu werden. Orientieren Sie sich am Vorgehen zur Erstellung eines neuen Nutzeraccounts.

Wir stellen Ihnen außerdem einen Unit-Test zum Prüfen der Funktionalität Ihrer Implementierung zur Verfügung.

4 Data Visualization (4 Punkte)

Im Notebook `Data Visualization.ipynb` wurde gezeigt, wie man mithile von Barplots und Heatmaps verschiedene Daten visualisieren kann. In dieser Aufgabe lernen Sie nun zwei weitere Möglichkeiten kennen: Pie Charts und gestapelte Histogramme. Dabei sollen Sie für jede der beiden Graphen jeweils eine Anfrage im Notebook `exercise4.ipynb` erstellen. Der Datensatz aus dem Notebook wird dabei durch folgendes Relationenschema repräsentiert:

$$
\begin{aligned}
[\text{movies}] &: \{[\underline{\text{id:int}}, \text{name:varchar}, \text{year:int}, \text{rank:float}]\} \\
[\text{directors}] &: \{[\underline{\text{id:int}}, \text{first\_name:varchar}, \text{last\_name:varchar})]\} \\
[\text{movies\_directors}] &: \{[\underline{\text{director\_id:(directors}\rightarrow\text{id)}, \text{movie\_id: (movies}\rightarrow\text{id)}}]\} \\
[\text{movies\_genres}] &: \{[\underline{\text{movie\_id:(movies}\rightarrow\text{id)}, \text{genre:varchar}}]\} \\
[\text{fav\_directors}] &: \{[\underline{\text{director\_id:(directors}\rightarrow\text{id)}}]\}
\end{aligned}
$$

Erstellen Sie nun die folgenden Anfragen:

(a) Pie Chart: Die Verteilung der Filmgenres zwischen 1930 und 1990 (beides exklusive), d.h. der prozentuale Anteil jedes Genres an der Gesamtmenge an vertretenen Genres. Beachten Sie, dass manche Filme mehreren Genres angehören können. (1,5 Punkte)

(b) Gestapeltes Histogramm: Die Anzahl an Filmen, die die favorisierten Regisseure (`fav_directors`) in jedem Genre zwischen 1970 und 2004 (beides inklusive) gedreht haben. (2,5 Punkte)

Gehen Sie dabei für beide Anfragen wie im oben verlinkten Notebook vor, das heißt erstellen Sie zunächst eine (oder mehrere) SQL-Queries, die die benötigten Daten extrahieren und führen Sie diese aus. Bringen Sie die Daten anschließend in eine Form, sodass sie von den entsprechenden Libraries weiterverarbeitet werden können. Weitere Informationen bezüglich den erwarteten Formaten finden Sie im Aufgabennotebook.

5 Anwendungsbeispiel: Instagram (10 Bonuspunkte)

Social-Media-Plattformen stellen ihren Nutzer*innen eine Vielzahl verschiedener Funktionen zur Verfügung. Zum Beispiel ist es bei Instagram möglich, Fotos und Videos zu posten und andere Fotos/Videos zu liken/kommentieren. Hierbei würde sich folgende Modellierung anbieten:

- Nutzer*innen haben eine NID, einen Namen sowie ein Geburtsdatum.
- Fotos und Videos sind Posts und haben eine PID und ein Veröffentlichungsdatum. Videos haben zudem noch eine Dauer (in Sekunden).
- Posts werden jeweils von einer Nutzer*in gemacht.
- Nutzer*innen können zudem die Posts anderer Nutzer*innen liken und zu beliebigen Zeitpunkten (gegebenenfalls mehrmals) kommentieren.

Basierend auf diesem Schema, bearbeiten Sie die folgenden Aufgaben:

(a) Setzen Sie obiges Schema in ein ER-Modell um. Wählen Sie hierbei sinnvolle Schlüssel, nutzen Sie die Chen-Notation und verwenden Sie Vererbungen, falls möglich. (2 Punkte)

(b) Übersetzen Sie obiges Schema ins relationale Modell. Vereinfachen Sie dieses soweit wie möglich. (2 Punkte)

(c) Übersetzen Sie die folgenden umgangssprachlichen Anfragen in Relationale Algebra:

1. Bestimmen sie die Namen der Nutzer*innen, deren Geburtsdatum vor 2010 liegt und die bereits ein Video gepostet haben, dass zwei Minuten lang ist. (1 Punkt)
2. Bestimmen Sie die Geburtsdaten der Nutzer*innen, die bereits ein Foto geliked haben, sowie genau zwei Kommentare zu ebendiesem Foto veröffentlicht haben. (1 Punkt)

(d) Übersetzen Sie die folgenden umgangssprachlichen Anfragen in SQL. Sie dürfen lediglich in der zweiten Anfrage Unteranfragen verwenden. Vermeiden Sie zudem Duplikate.

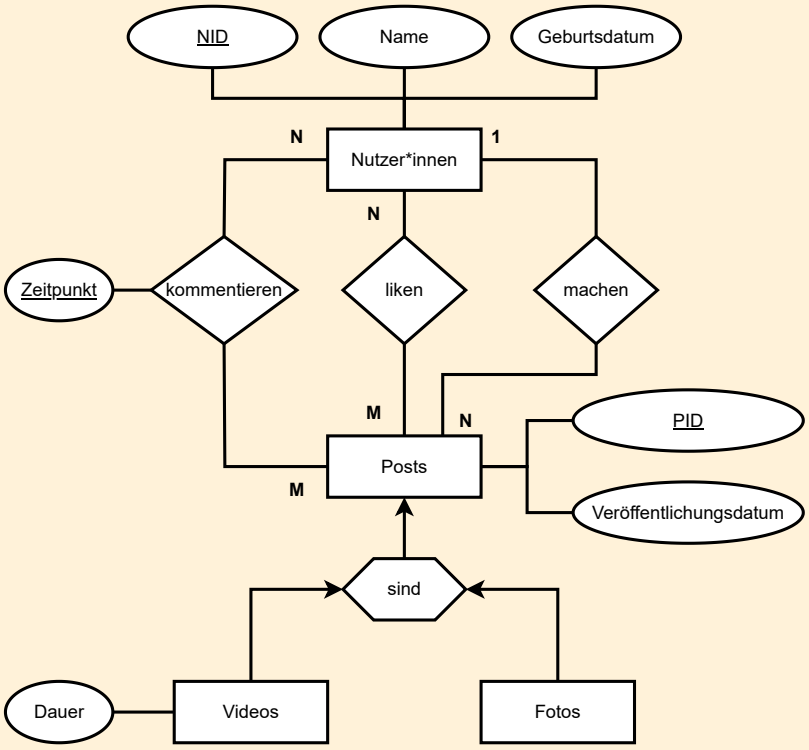
1. Bestimmen Sie die Dauern der Videos, die bereits geliked und kommentiert wurden und vor dem 11.03.2022 veröffentlicht wurden. Ordnen Sie Ihre Ausgabe absteigend nach dem Veröffentlichungsdatum. (1 Punkt)
2. Bestimmen Sie die Namen der Nutzer*innen, die vor dem 17.03.2004 geboren wurden und noch nie einen Post gemacht haben, aber schon unter mindestens drei verschiedenen Posts einen Kommentar hinterlassen haben. Geben Sie lediglich die ersten fünf Tupel aus. (1 Punkt)

(e) Übersetzen Sie folgende SQL Anfrage kanonisch in relationale Algebra und optimieren Sie diese heuristisch mit den aus der Vorlesung bekannten Regeln. Sie müssen lediglich den optimierten Baum angeben. (2 Punkte)

```
SELECT N.Name
FROM   kommentieren k, Nutzer*innen N, Videos V, Posts P
WHERE  N.Geburtsdatum < '2000-09-01'
       AND V.Dauer > 30 AND V.VID = P.PID
       AND P.Veröffentlichungsdatum > '2015-03-05'
       AND k.Post = P.PID AND k.Nutzer*in = N.NID
```

Lösung:

(a) Vorgesehen sind 0,5 Punkte je Stichpunkt.



(b) 0,5 Punkte für die korrekte Umsetzung der Nutzer*innen, 0,5 Punkte für eine korrekte Umsetzung der Posts inklusive der Vererbung mit Videos und Fotos, sowie je 0,5 Punkte für die beiden Relationen.

[Nutzer*innen] : {[NID: int, Name: string, Geburtsdatum: date]}
[Posts] : {[PID:int, Nutzer*in:(Nutzer*innen→NID), Veröffentlichungsdatum: date]}
[Videos] : {[VID:(Posts→PID), Dauer: int]}
[Fotos] : {[FID:(Posts→PID)]}
[liken] : {[Nutzer*in:(Nutzer*innen→NID), Post:(Posts→PID)]}
[kommentieren] : {[Nutzer*in:(Nutzer*innen→NID), Post:(Posts→PID), Zeitpunkt: timestamp]}

(c) Vorgesehen ist ein Punkt pro Anfrage (je 0,5 Punkte Abzug für inkorrekte (Join-)Prädikate oder Aggregate, unterschiedliche Schemata bei Mengenoperationen, nicht-disjunkte Schemata bei Joins, uneindeutige Bezeichner, inkorrekte Bezeichner durch Verwechslung von Umbenennung und Teilergebnisnotation, usw.). Wichtig ist hierbei nur, dass die Anfragen das korrekte Ergebnis produzieren, nicht, dass sie mit der Musterlösung übereinstimmen.

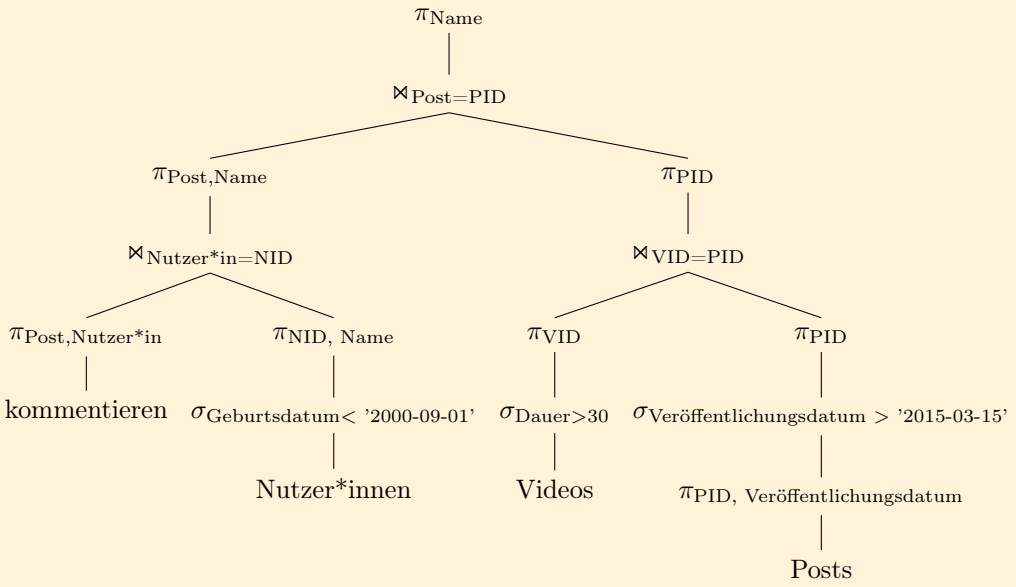
1. $R := ((\sigma_{\text{Geburtsdatum} < \text{'2010-01-01'}} \text{ Nutzer*innen}) \bowtie_{\text{NID} = \text{Nutzer*in}} \text{Posts}) \bowtie_{\text{PID} = \text{VID}} \text{Videos})$
   $\pi_{\text{Name}} (\sigma_{\text{Dauer} = 120} R)$
2. $R_1 := (\rho_{\text{Nutzer*in'}\leftarrow\text{Nutzer*in,Post'}\leftarrow\text{Post}} \text{ liken}) \bowtie_{\text{Nutzer*in'} = \text{Nutzer*in} \wedge \text{Post'} = \text{Post}} \text{kommentieren}$
   $R_2 := \sigma_{\text{count(*)} = 2}(\gamma_{\text{Nutzer*in, Post,count(*)}} (\text{Fotos} \bowtie_{\text{FID} = \text{Post}} R_1))$
   $\pi_{\text{Geburtsdatum}} (\text{Nutzer*innen} \bowtie_{\text{NID=Nutzer*in}} R_2)$

(d) Vorgesehen ist ein Punkt pro Übersetzung (je 0,5 Punkte Abzug für Unteranfragen in SQL, kartesische Produkte in RA, uneindeutige Bezeichner, inkorrekte (Join-)Prädikate oder Aggregate, Zugriff auf durch Gruppierung uneindeutige Attribute im SELECT, nicht-disjunkte Schemata bei Joins der RA, inkorrekte Bezeichner durch Verwechslung von Umbenennung und Teilergebnisnotation der RA, usw.). Grundsätzlich sind sowohl explizite als auch implizite Joins zum Lösen der Aufgaben in Ordnung. Wichtig ist hierbei nur, dass die Anfragen ein äquivalentes Ergebnis produzieren, nicht, dass sie mit der Musterlösung übereinstimmen.

1.
```
SELECT   DISTINCT V.Dauer
FROM     liken l
         JOIN kommentieren k ON l.Post = k.Post
         JOIN Posts P ON P.PID = k.Post
         JOIN Videos V ON V.VID = P.PID
WHERE    P.Veröffentlichungsdatum < '2022-03-11'
ORDER BY P.Veröffentlichungsdatum DESC;
```

2.
```
SELECT   DISTINCT N.Name
FROM     (SELECT DISTINCT NID FROM Nutzer*innen
          EXCEPT
          SELECT DISTINCT Nutzer*in AS NID
          FROM   Posts
         ) AS noPosts
         JOIN
         (SELECT DISTINCT Nutzer*in, Post
          FROM   kommentieren
         ) AS commented ON noPosts.NID = commented.Nutzer*in
         JOIN Nutzer*innen N ON N.NID = noPosts.NID
WHERE    N.Geburtsdatum < '2004-03-17'
GROUP BY noPosts.NID
HAVING   COUNT(*) >= 3
LIMIT    5;
```

(e) Vorgesehen sind 2 Punkte (je 0,5 Punkte Abzug für kartesische Produkte, fehlender Predicate Pushdown, überflüssige Attribute durch fehlende Projektionen, usw.).



Abgabe

Lösungen sind in Teams von 2 bis 3 Studierenden bis zum 14. Juli 2022, 10:15 Uhr über Ihre persönlichen Statusseite im CMS einzureichen. Nutzen Sie hierfür die Team Groupings Funktionalität im CMS.

Ihre Abgabe muss dem folgenden Format entsprechen:

```
abgabe.zip
├── abgabe.pdf
└── jupyter.txt
```

Hierbei enthält `abgabe.pdf` Ihre Lösungen zu Aufgabe 1, 2 und 5 und `jupyter.txt` Ihre Lösung zu Aufgabe 3 und 4. Achten Sie darauf, dass Sie nur die von Ihnen zu ergänzenden Jupyter Zellen so kopieren, dass Einrückung und Formatierung korrekt sind.

Abgaben, die nicht den oben angegeben Vorgaben entsprechen, führen zu Punktabzug. Einzelabgaben werden nicht mehr korrigiert und mit 0 Punkten bewertet.