3. Rechnerübung: Flussdiagramme und Programme

Aufgabe 1: Flussdiagrammen und Programme herleiten

a) Durchfallquote

Ein Programm zum Berechnen der Durchfallquote einer Klausur funktioniert so: In einem gegebenen Feld `F` stehen alle Noten. Die Index-Variable `n` wird anfangs auf 0 gesetzt. In einer Schleife wird `n` so lange um 1 erhöht, bis es alle Positionen von `F` durchlaufen hat. In jedem Schleifendurchlauf wird geprüft, ob eine Note größer als 4.0 ist. Wenn dies der Fall ist, wird der Durchgefallenenzähler `z` (der anfangs auch 0 war) um 1 erhöht. Nachdem die Schleife beendet ist, wird die Quote als `z` geteilt durch die Anzahl der Elemente von `F` ausgegeben.

i) Entwerfen Sie ein Flussdiagramm für dieses Programm.

ii) Schreiben Sie ein dazu passendes Python Programm.

In [ ]:

b) Element suchen

Ein Programm zum Prüfen, ob eine Liste `L` ein bestimmtes Element `x` enthält, geht so vor: Die Variable `a` durchläuft alle Indizes der Liste. Für jeden Index wird geprüft, ob das `a`-te Element der Liste gleich dem gesuchten `x` ist. Wenn ja, wird auf dem Bildschirm ausgegeben, dass das Element gefunden wurde, und mit dem Kommando `break` wird die `while`-Schleife abgebrochen. Wenn die Schleifenbedingung `a < len(L)` nicht mehr gilt wird auf dem Bildschirm ausgegeben, dass das Element nicht gefunden wurde (das ist dann der `else`-Teil der Schleife).

i) Entwerfen Sie ein Flussdiagramm, das das Programm modelliert.

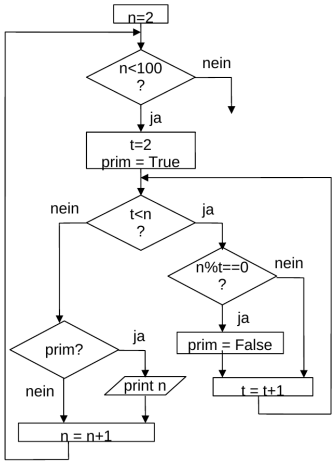
ii) Schreiben Sie ein dazu passendes Python Programm.

In [ ]:

Aufgabe 2: Gegebene Flussdiagramme ausprogrammieren

a) Primzahlen ausgeben

Betrachten Sie das hier dargestellte Flussdiagramm, und setzen Sie es in ein Python Programm um. Das Programm durchläuft in der Variablen `n` alle Zahlen von 2 bis 99 und prüft jeweils für alle `t` von 2 bis `n-1`, ob `t` ein Teiler von `n` ist. Falls nein, bleibt die Wahrheitswertvariable `prim` auf `True`, sonst wird sie auf `False` gesetzt. Nachdem alle in Frage kommenden Teiler `t` getestet wurden und `prim` immer noch `True` ist, wird die aktuelle Zahl `n` auf dem Bildschirm ausgegeben.



In [ ]:

b) Listen auf Gleichheit prüfen

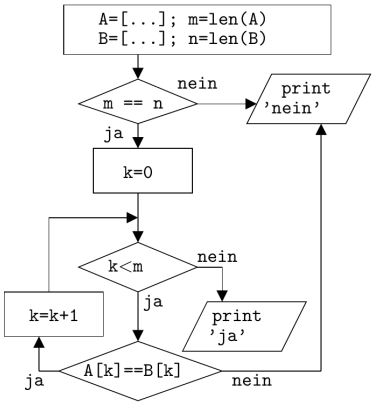
Ein Programm gibt auf dem Bildschirm `ja` oder `nein` aus, je nachdem, ob die beiden Listen `A` und `B` gleich sind oder nicht. Dazu wird zuerst geprüft, ob die Listen gleich lang sind. Wenn nicht, wird "nein" ausgegeben', wenn doch, beginnt eine Schleife mit der Variablen `k`, die bei 0 startet und so lange läuft, wie `k` kleiner als die Länge von `A` (und somit auch kleiner als die Länge von `B`) ist. Für jedes `k` wird verglichen, ob das `k`-te Element von `A` gleich dem `k`-ten Element von `B` ist, und, wenn dem nicht so ist, wird die Schleife mit der Ausgabe "nein" abgebrochen. Wenn die Schleife ohne Abbruch abgelaufen ist, hat es also keine Unstimmigkeiten gegeben und es wird "ja" ausgegeben. Programmieren Sie ein Python Programm, das dem gegebenen Flussiagramm entspricht und testen Sie es mit mehreren Beispielen.

Einmal mit

```
A=[7,22,1,5,11,14,3]; B=[7,22,1,5,11,14,3]
```

Einmal mit

```
A=[7,22,1,5,11,14,3]; B=[7,22,1,6,11,14,3]
```



In [ ]:

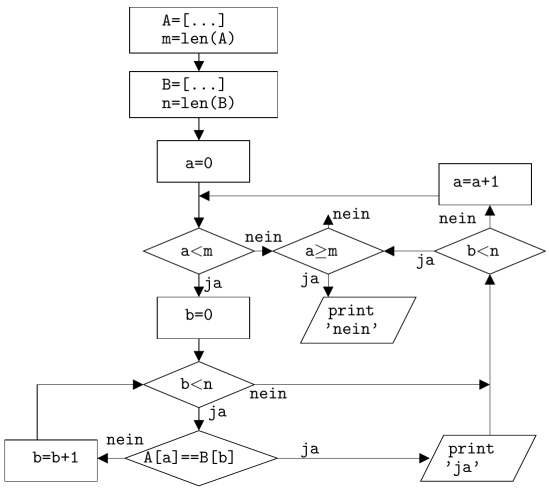
c) Prüfen ob Wert in beiden Listen vorkommt

Ein Programm prüft, ob die zwei Listen `A` und `B` ein gemeinsames Element haben. Wenn dem so ist, wird "ja" sonst "nein" ausgegeben. Dazu läuft eine Variable `a` die Indizes der Liste `A` und eine Variable `b` die Indizes der Liste `B`. Die `b`-Schleife ist in die `a`-Schleife eingebettet. Für jedes `a` durchläuft `b` die komplette Liste `B`. Für jede Kombination von `a` und `b` wird geprüft, ob das `a`-te Element von `A` gleich dem `b`-ten von `B` ist. Wenn ja, dann wird nach Ausgabe von "ja" die Schleife abgebrochen. Damit auch die äußere Schleife abgebrochen wird, muss jetzt der Abbruch der inneren Schleife detektiert werden (erkennbar daran, dass `b` nicht bis zum Ende gelaufen ist) und in diesem Fall auch die äußere Schleife abbrechen. Nach Abbruch der äußeren Schleife muss der Text "nein" nur dann ausgegeben werden, wenn die Schleifen nicht abgebrochen worden waren, erkennbar daran, dass `a` nicht mehr kleiner `m` ist (also bis zum Ende durchgelaufen ist).

Testen Sie Ihr Programm u.a. mit den Listen

```
A=[7,22,1,5,11,14,3]; B=[6,2,6,23,13,11,4]
```

aber auch mal mit zwei Listen, die keine gemeinsamen Elemente enthalten.



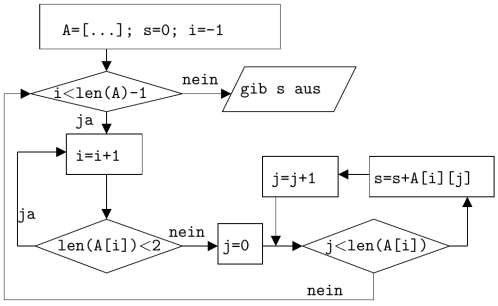
In [ ]:

d) Summe aller Elemente von mindestens zweielementigen Teillisten

Ein Programm berechnet die Summe aller Zahlen in allen mindestens zweielementigen Teillisten der Liste `A`. Im nebenstehenden Flussdiagramm ist zu sehen, dass die Variable `i` durch alle Indizes von `A` läuft. Dass `i` mit `-1` initialisiert wird ist okay, weil es *vor* der Verwendung als Index erhöht wird. Für jedes `i` gibt es zwei Möglichkeiten, die in der Bedingung `len(A[i])<2` behandelt werden: Entweder wird der Schleifenrumpf der `i`-Schleife übersprungen und gleich zum nächsten `i` übergegangen, oder in einer inneren Schleife werden alle Elemente von `A[i]` zur Summe `s` hinzuaddiert. Wenn `i` durchgelaufen ist, wird `s` ausgegeben. Basteln Sie ein zum Flussdiagramm passendes Python-Programm (und verdrehen Sie nicht die Bedingungen).

Testen Sie Ihr Programm u.a. mit

```
A=[[2,3],[1,5,5],[4],[6,4],[2],[2,1]]
```



In [ ]:

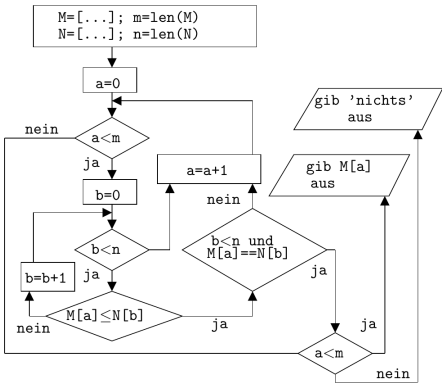
e) Gemeinsames Element in sortierten Listen suchen

Ein Programm zum Prüfen, ob in zwei sortierten Listen `M` und `N` ein Element in beiden Listen auftaucht, geht so vor: Die Variable `a` durchläuft alle Indizes der Liste `M` (beginnend bei 0 und läuft solange wie `a<m`, wobei `m` die Länge der Liste `M` ist). Entsprechend läuft `b` durch alle Indizes der Liste `N` (also so lange wie `b<n` ist). Wenn allerdings das `b`-te Element der Liste `N` größer gleich dem `a`-ten Element der Liste `M` ist, wird die `b`-Schleife abgebrochen. Wenn das `a`-te Element der Liste `M` gleich dem `b`-ten Element der Liste `N` ist, wird die `a`-Schleife abgebrochen. Schließlich wird durch Prüfen, ob `a<m` ist, festgestellt, ob die `a`-Schleife abgebrochen wurde oder nicht, und entsprechend das Element, das in beiden Listen vorkommt, ausgegeben oder der Text, dass nichts in beiden Listen vorkommt.

Das Flussdiagramm stellt dieses Programm dar. Setzten Sie es in Python um. Testen Sie Ihr Programm u.a. mit den Listen

```
M=[1,3,5,7,11,14,22]; N=[2,6,8,11,12,13,23]
```

aber auch mal mit zwei Listen (aufsteigend sortiert), die keine gemeinsamen Elemente enthalten.



In [ ]:

Aufgabe 3: Ein Hangman-Spiel

Die Regeln

Ein Spieler tippt ein Wort (verdeckt) ein. Der andere Spieler muss es erraten. Dazu gibt er einzelne Buchstaben ein. Für die Buchstaben, des zu ratenden Wortes, die er schon eingegeben hat (einmalige Eingabe eines mehrfach vorkommenden Buchstabens genügt) bekommt er angezeigt, wo sie sich befinden. Für die nicht geratenen Buchstaben erscheint nur ein Unterstrich (also z.B. `TA_TAT__`, wenn `T` und `A` von `TASTATUR` schon geraten sind). Für jeden eingegebenen Buchstaben, der im Wort nicht vorkommt, wird ein Fehler gezählt und ein Galgen weiter gemalt. Nach 7 Fehlern ist der Galgen fertig und das Spiel verloren. Wenn das Wort vorzeitig erraten wurde, ist das Spiel gewonnen.

Die Programmidee

Das zu ratende Wort wird in der Variablen `Wort` gespeichert. Die Variable `Fehler` zählt die Fehler, und die Variable `Treffer` zählt die Treffer. Solange weniger als 7 Fehler und weniger Treffer als Buchstaben im Wort (in `n` gespeichert) gemacht wurden, wird der Spieler aufgefordert, einen Buchstaben zu raten. Dazu wird ihm angezeigt, was er bisher geraten hat, indem die Variable `Geraten` (Format s.o.) ausgegeben wird. Der geratene Buchstabe wird in der Variablen `b` gehalten. Nach der Eingabe von `b` läuft eine Schleifenvariable `i` von 0 bis zum Index des letzten Buchstabens des Wortes. Jedes Mal, wenn `Wort[i]` gleich `b` ist aber `Geraten[i]` nicht, wird ein Treffer gezählt und die entsprechende Stelle in Geraten von `_` auf den geratenen Buchstaben `b` geändert. In der Variablen getroffen (Wahrheitswert) wird festgehalten, ob es wenigstens einen Treffer gab. Wenn ein Buchstabe keinen Treffer hat (bzw. schon mal geraten wurde) wird ein Bild des

Galgens ausgegeben und der Wert von `Fehler` um 1 erhöht. Wenn irgendwann der 7. Fehler gemacht worden ist, oder `Treffer == n` ist, wird das Programm nach Ausgabe einer Erfolgs- oder Versagensmeldung beendet.

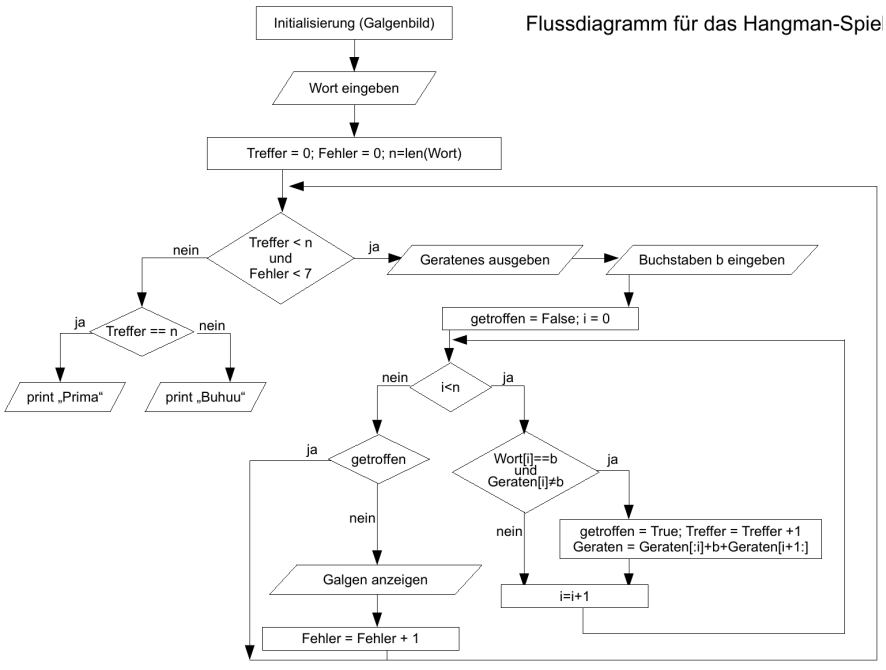
Ein paar Tipps:

- Mit `n=len(Wort)` können Sie die Länge (Anzahl Buchstaben) eines Wortes bestimmen und in `n` speichern.
- Mit `Geraten = "____________________"[0:n]` können Sie ein Wort der Länge `n` bestehend aus nur Unterstrichen erzeugen (vgl. Listen).
- Mit `getpass.getpass(...)` statt `input(...)` kann man Eingaben wie bei Passwörtern machen, so dass sie nicht sichtbar erscheinen. Dazu muss am Anfang des Programms `import getpass` stehen. Das ist unten aber schon vorgegeben und muss Sie nicht weiter interessieren.
- Mit `Geraten=Geraten[:i]+b+Geraten[i+1:]` können Sie das `i`-te Zeichen in `Geraten` durch den Wert von `b` ersetzen.
- Mit `for i in range(7): print(Galgen[i][Fehler-1])` kann man den Galgen für die entsprechende Fehleranzahl ausgeben lassen. Dazu sind die Balken-Texte schon vorgegeben.

Wenn Sie einen Teil des Programms nicht gleich hinkriegen, ersetzen Sie ihn durch einen "Dummy" und machen dort weiter wo Sie es können.

Die Aufgabe:

Betrachten Sie das folgende Flussdiagramm zum oben beschriebenen Programmentwurf, oder - noch besser - malen Sie ein eigenes und vergleichen es danach mit der Vorgabe. Setzen Sie das Flussdiagramm in Python um, und reichern Sie es ggf. hier und da durch zusätzliche informative Ausgaben an. Wenn das Programm fertig ist, spielen Sie damit herum.



In [1]:

```
import getpass

# Die Variable Galgen enthaelt eine Liste von Listen, so dass
# Galgen[0] die oberste Zeile aller Galgendarstellungen enthaelt,
# Galgen[1] die zweitoberste usw. und Galgen[6] die unterste
# Zeile. Galgen[i][Fehler] enthaelt dann die i-te Zeile des Galgens
# der bei der gegebenen Anzahl Fehler dargestellt werden soll.
# Allerdings fangen wir bei 0 an zu zaehlen, so dass Galgen[3][2]
# die viertoberste Zeile des Galgens nach 3 Fehlern ist.

Galgen=[['     ','     ','___  ','___  ','___  ',' ___  ',' ___  '],
        ['     ','   | ','|   | ','|  | ','|  | ','|  | ','|  | '],
        ['     ','   | ','|   | ','O  | ','O  | ',' O  | ','_O_ | '],
        ['     ','   | ','|   | ','|  | ','|  | ','|  | ','|  | '],
        ['     ','   | ','|   | ','   | ','   | ','/|  | ','/|  | '],
        [' ___ ',' __| ',' __| ',' __| ',' __| ',' ___| ',' ___| '],
        ['|   |','|   |','|   |','|   |','|   |','|    |','|    |']]

# Ein paar Saetze, damit die Spieler wissen, was Sache ist:

print("Der Spieler, der raet, schaue bitte kurz weg.")
print("Der andere Spieler gebe ein Suchwort ein.")



# Wir setzten n=len(Wort) um in Zukunft einfach nur noch n
# statt len(Wort) verwenden zu koennen. Ist halt etwas kuerzer.



# Die Variable Geraten wird anfangs mit lauter Unterstrichen
# initialisiert (so viele, wie das Suchword Buchstaben hat,
# also n). Die Konstruktion [0:n] funktioniert genauso wie
# bei Listen. Bei Texten ist damit gemeint, "vom nullten bis
# zum n-ten (excl.) Zeichen".



# Treffer und Fehler anfangs auf 0. Spaeter werden die dann
# hochgezaehlt. Ein Treffer zaehlt jeden getroffenen Buchstaben.
# Buchstaben, die mehrfach vorkommen, zaehlen dann eben mehrere
# Treffern, so dass n Treffer bedeutet: Alles erraten.
# Jeder falsch eingegebene Buchstabe zaehlt einen Fehler.




# Das Raten dauert so lange, wie weniger als 7 Fehler gemacht
# worden sind und weniger Treffer gelandet wurden, als das
# Wort Buchstaben hat.



    # Weil das Programm nett sein will, zeigt es dem Spieler
    # an, was der bisher geraten hat. Nicht geratene Zeichen
    # erscheinen als Unterstrich. Spaeter wird dann jeder
    # Treffer durch den entsprechenden Buchstaben ersetzt.
```

```
    # Der Spieler wird aufgefordert, einen Buchstaben einzugeben.
    # raw_input statt input, damit Texte ohne Anfuehrungsstriche
    # eingegeben werden koennen.



    # Die Wahrheitswert-Variable getroffen wird festhalten, ob
    # der eingegebene Buchstabe (mind. )ein Treffer war oder nicht.



    # In einer Schleife lassen wir i bei 0 beginnend wetzen,
    # bis alle Buchstabenpositionen des Wortes abgearbeitet sind,
    # also so lange wie i<n ist.




        # Fuer jedes i pruefen wir, ob der i-te Buchstabe im
        # Suchwort gleich dem eingegebenen Buchstaben b ist, aber
        # nicht schon vorher eingegeben worden ist, was daran
        # erkennbar waere, dass auch in Geraten[i] dieser Buchstabe
        # schon steht (und nicht mehr der Unterstrich).



            # Wenn der Buchstabe ein Treffer ist, wir dies in der
            # Variablen getroffen festgehalten und der Trefferzaehler
            # wird um 1 erhoeht.




            # Geraten[:i] sind alle Zeichen von Geraten bis zum
            # i-ten (excl.). Und Geraten[i+1:] sind alle Zeichen
            # von Geraten ab dem i+1-ten (incl.). Mit + kann man
            # Texte hintereinanderketten.






    # So, alle Buchstaben des Wortes sind jetzt mit dem geratenen
    # Buchstaben verglichen worden, und falls kein Treffer dabei
    # war, geben wir die naechste Galgendarstellung aus.



        # Diese Schleife legt in g alle Elemente der Listenvariable Galge
        # die selbst ja auch Listen sind. Von jedem Element g wird dann n
        # der Teil gedruckt, der fuer die angegebene Fehlerzahl benoetigt



        # Nicht vergessen: Fehler hochzaehlen, damit bei 7 Schluss ist.



# Wenn wir hier landen, ist die aeussere Schleife auch fertig.
```

```
# Jetzt muessen wir nur noch pruefen, warum, also ob Treffer == n
# oder Fehler == 7 die Abbruchursache war. Entsprechend geben wir
# dann eine Erfolgsmeldung aus, oder loesen das Raetsel auf.
```

```
Der Spieler, der raet, schaue bitte kurz weg.
Der andere Spieler gebe ein Suchwort ein.
```

In [ ]: