Prof. Stefan Göller
Anna Maiworm

Einführung in die Informatik

WiSe 2025/2026

Hausaufgabenblatt 03, Abgabe am 17.11.25 um 12:00 Uhr

Allgemeine Hinweise

- Wenn die Hausaufgabe exakte Benennungen von Dateien, Funktionen, Variablen, etc. vorgibt, dann **müssen** Sie sich an diese Vorgabe halten. Falls nicht wird Ihre Abgabe nicht oder nur mit Punktabzügen korrigiert.
- Gleiches gilt, wenn die Aufgabe konkrete Anweisungen für input() oder print() vorgibt.
- Jede gefragte Ausgabe muss mit dem Befehl print() ausgegeben werden. Sofern nicht anders vorgegeben, geben Sie nur das geforderte Ergebnis und **keinen** zusätzlichen Text aus.
- Benutzen Sie **keine Python-Imports**, außer dies ist in der Aufgabenstellung explizit erlaubt.
- Beachten Sie die Anweisungen, die ggf. zusätzlich in den jeweiligen Aufgaben gegeben sind.
- Geben Sie für jede Aufgabe eine separate Datei ab. Benennen Sie diese Datei als bXXaY.py, wobei XX die Blattnummer und Y die Aufgabennummer ist.
- Schreiben Sie die Namen aller Gruppenmitglieder als Kommentar in die erste Zeile jeder Datei.
- Programmcode muss möglichst einfach und gut lesbar sein, um die Korrektur zu erleichtern. Code, der nicht oder nur schwer lesbar ist, kann zu Punktabzug führen.

Hausaufgabe 1 (20 Punkte):

Das Ziel dieser Aufgabe ist es die Bestandteile eines Programms zu implementieren, welches ein zufällig generiertes Passwort errät.

Machen Sie sich hierzu erst nochmals mit der *längen-lexikographischen Ordnung* und dem Begriff *Alphabet-String* vertraut, welche in Präsenzaufgabe 1 erklärt werden. Machen Sie sich nochmal klar, dass man durch die längen-lexikographische Ordnung jedem Wort über einem vorgegebenen Alphabet eine eindeutige Position zuordnen kann.

Nutzen Sie zum Lösen der Aufgabe das Template in der Datei b03a1.py und fügen Sie Ihre Lösungen in die vorgegebenen Funktionen ein. Beachten Sie, dass in diesen Funktionen aktuell ein Standardwert ausgegeben wird (0 für Integer, `""` für String). Die `return` Anweisung in den von Ihnen zu schreibenden Funktionen soll und muss also von Ihnen angepasst werden. Das Template enthält außerdem bereits mehrere vorgegebene Funktionen, welche Sie verwenden dürfen wenn es im entsprechenden Aufgabenteil angegeben ist. Außerdem enthält das Template das Hauptprogramm, welches:

1. Den Nutzer nach der Alphabetgröße und der maximalen Passwortlänge fragt. Außerdem wird abgefragt, ob das Passwort mittels lineare Suche und/oder mittels binäre Suche erraten werden soll (geben Sie z.B. `2` ein um beide Suchen auszuführen).
2. Ein geheimes Passwort `Z` erstellt (über einem Alphabet der eingegebenen Größe). Auf die Variable `Z` dürfen Sie **zu keinem Zeitpunkt** direkt **zugreifen**. Alle benötigten Zugriffe geschehen in den jeweils erlaubten vorgegebenen Funktionen.
3. Mittels der von Ihnen geschriebenen Funktionen das Passwort mittels lineare Suche und/oder mittels binäre Suche errät und außerdem ausgibt wie viele Vergleiche benötigt wurden.

Sie dürfen weder die vorgegebenen Funktionen, noch das Hauptprogramm verändern!

Im Folgenden sollen Sie mehrere Funktionen schreiben, um den Passwort-Knacker umzusetzen. Sie dürfen in allen Aufgabenteilen annehmen, dass die Eingaben wie beschrieben sind. Verwenden Sie, wann immer möglich, Ihre zuvor geschriebenen Funktionen, um die aktuelle Aufgabe zu lösen. Die vorgegebenen Funktionen `wort_bzgl_laenge`, `wort`, `kleiner`, `groesser`, `gleich`, sowie das Hauptprogramm dürfen **nicht** verändert werden!

Hinweis: In dieser Aufgabe sollen die Ausgaben der Funktionen nur mit `return` *und nicht mit* `print` *ausgegeben werden. Um ihre einzelnen Funktionen zu testen, schreiben Sie für (a) z.B.* `print(zufallsstring("ABC",6))` *in eine separate Datei im gleichen Verzeichnis. Um Ihre Funktion in der neuen Datei verfügbar zu machen, fügen Sie am Anfang der Datei die Zeile* `from b03a1 import *` *ein.*

(a) (3 Punkte) Eine Funktion `zufallsstring` mit ...

Eingabe: Alphabet-String `A`, Ganzzahl `laenge` ≥ 0

Ausgabe: Ein String, dessen Länge zufällig aus $\{0, ..., \texttt{laenge}\}$ gewählt ist und in dem an jeder Position ein zufällig gewähltes Zeichen aus `A` steht

Hinweis: Verwenden Sie `random.choice(<iterierbares Objekt>)` um einen Zufallswert zu erhalten. Hierbei kann `<iterierbares Objekt>` z.B. ein String oder eine `range`-Anweisung sein.

(b) (1 Punkte) Eine Funktion `alphpos` mit ...

Eingabe: ein Zeichen `a` aus `A`, Alphabet-String `A`

Ausgabe: Position von `a` in `A` (wir beginnen bei 0 zu zählen)

(c) (4 Punkte) Eine Funktion `lexpos` mit ...

Eingabe: String `s` bestehend aus Zeichen aus `A`, Alphabet-String `A`

Ausgabe: Ganzzahl `n` so, dass der String `s` die Position `n` hat unter allen Wörtern der Länge `len(s)` über `A` (wir beginnen bei 0 zu zählen)

Hinweis: Für einen String `s` = $s_0 s_1 \ldots s_{k-1} s_k$, wobei die s_i einzelne Zeichen sind, und wert(s_i) den Wert des Zeichens innerhalb des verwendeten Alphabets `A` (also seine Position im Alphabet-String) beschreibt, kann die von `lexpos` zu bestimmende Position folgendermaßen berechnet werden:

$$wert(\mathtt{s}_k) \cdot |\mathtt{A}|^0 + wert(\mathtt{s}_{k-1}) \cdot |\mathtt{A}|^1 + ... + wert(\mathtt{s}_1) \cdot |\mathtt{A}|^{k-1} + wert(\mathtt{s}_0) \cdot |\mathtt{A}|^k$$

(d) (4 Punkte) Eine Funktion `laengenlexpos` mit ...

Eingabe: String `s` bestehend aus Zeichen aus `A`, Alphabet-String `A`

Ausgabe: die laengenlexikographische Position von `s` bzgl. des Alphabets `A`

Beispiele:

`laengenlexpos("","ABCDEF")` liefert 0
`laengenlexpos("AAA","A")` liefert 3
`laengenlexpos("AABBCCAABBCC","ABC")` liefert 297840
`laengenlexpos("AABBCCAABBCC","ABCD")` liefert 5961135

(e) (1 Punkte) Eine Funktion `trans`, deren Rumpf nur aus **einer Zeile** besteht, mit ...

Eingabe: String `s` bestehend aus Zeichen aus `A`, Alphabet-String `A`, Alphabet-String `B`

Ausgabe: das eindeutiges Wort `w` über dem Alphabet `B`, sodass das Wort `s` über dem Alphabet `A` dieselbe längen-lexikographische Position hat wie das Wort `w` über `B` (wir beginnen jeweils bei 0 zu zählen)

Hinweis: Sie dürfen die vorgegebene Funktion `wort` (Beschreibung in Präsenzaufgabe 1) verwenden. In den folgenden Aufgaben wird `trans` nicht benötigt.

(f) (3 Punkte) Eine Funktion `suche_z_linear`, welche einen Alphabet-String `A` als Eingabe bekommt und mittels linearer Suche das Passwort errät. D.h. die Funktion zählt nacheinander alle möglichen Strings über dem Alphabet `A`, gemäß der längen-lexikographischen Ordnung, auf, bis das richtige Passwort gefunden wurde. Sie dürfen die vorgegebene Funktion `gleich` aufrufen um zu prüfen, ob ein String dem richtigen Passwort entspricht. Außerdem dürfen Sie die vorgegebene Funktion `wort` verwenden. Wurde das korrekte Passwort gefunden, so soll dieses ausgegeben werden.

Hinweis: Die Funktion `suche_z_linear` *bekommt die maximale Passwortlänge nicht als Eingabe. Sie dürfen diese auch nicht in der Funktion abfragen (daher ist die Verwendung der längen-lexikographische Ordnung notwendig).*
Es ist zu erwarten, dass die Suche bei großer Passwortlänge bzw. Alphabetgröße sehr lange läuft, bis sie terminiert. Testen Sie diese Funktion also vor Allem mit kleinen Werten.
Die Funktionen `suche_z_linear` *und* `suche_z_binaer` *(Aufgabenteil (g)) funktionieren nur, wenn zuvor das geheime Passwort* `Z` *festgelegt wurde (zum Beispiel, wenn die Funktionen im Hauptprogramm ausgeführt werden).*

(g) (4 Punkte) Eine Funktion `suche_z_binaer`, welche einen Alphabet-String `A` und eine Ganzzahl `laenge` entgegen nimmt. Das Passwort soll mittels binärer Suche erraten werden. Um dies zu ermöglichen gibt die eingegebene Ganzzahl `laenge` die **maximale** Länge des Passwortes an und in Ihrer Funktion dürfen Sie mittels Aufruf der Funktionen `groesser`, `kleiner` und `gleich` ein Orakel befragen, welches Ihnen zurückmeldet, ob das geheime Passwort größer, kleiner oder gleich dem eingegebenen String ist, bzgl. der längen-lexikographischen Ordnung. Außerdem dürfen Sie die vorgegebene Funktion `wort` verwenden. Wurde das korrekte Passwort gefunden, so soll dieses ausgegeben werden.

Hinweis: Es reicht aus, wenn sie für die rechte Grenze des initialen Suchintervalls einen ausreichend großen Wert nutzen. Sie müssen also nicht den kleinstmöglichen Wert verwenden.

Präsenzaufgaben

Präsenzaufgabe 1:

Im Folgenden sprechen wir über die Wörter *über einem Alphabet* (z.B. $\{A, B, C, D\}$), d.h. die Strings die aus Zeichen des Alphabets bestehen. Im Programm werden wir das verwendete Alphabet als *Alphabet-String* darstellen, d.h. ein String aller Zeichen, in dem keines der Zeichen wiederholt wird (z.B. `"ABCD"`).

Die *lexikographische* Ordnung, welche durch den `<`-Operator auf Strings realisiert wird, ist nicht geeignet um alle Strings über einem Alphabet aufzuzählen, da sie folgende Eigenschaft hat:
Es gibt Strings `x` und `y`, sodass zwischen `x` und `y` unendlich viele andere Werte liegen, d.h. es gibt unendlich viele Strings `z` für die gilt `x < z < y`. Dies gilt zum Beispiel für `x=="A"` und `y=="B"`, denn es ist `"A" < "AA" < "AAA" < "AAAA" < ... < "B"`. Würde man also versuchen alle möglichen Strings nach dieser Ordnung aufzuzählen, so würde man niemals bei `"B"` ankommen.

Betrachten Sie nun die sogenannte *längen-lexikographische* Ordnung, hier dargestellt durch `<<`, welche zwei Strings `x` und `y` auf folgende Weise miteinander vergleicht:

- Zuerst wird die Länge der Strings verglichen:
  Wenn `len(x) < len(y)`, dann ist `x << y`.
  Wenn `len(y) < len(x)`, dann ist `y << x`.
- Funktioniert dies nicht, also falls `len(x) == len(y)`, dann werden die Wörter lexikographisch verglichen, also:
  Wenn `x < y`, dann ist `x << y`.
  Wenn `y < x`, dann ist `y << x`.

Die Aufzählung aller Wörter über einem Alphabet gemäß der längen-lexikographischen Ordnung beginnt also mit dem leeren String (einziger String der Länge 0), dann kommen alle Wörter der Länge 1, dann alle Wörter der Länge 2, usw. Zum Beispiel für die Wörter über dem Alphabet(-String) `"AB"` erhalten wir:

```
"" << "A" << "B" << "AA" << "AB" << "BA" << "BB" << "AAA"
                          << "AAB" << "ABA" << "ABB" << "BAA" << ...
```

Wir gehen davon aus, dass die Zeichen in einem Alphabet-String entsprechend der lexikographischen Ordnung sortiert sind, d.h. verwenden wir Buchstaben, so müssen diese entsprechend der alphabetischen Reihenfolge sortiert sein.

Schreiben Sie die folgenden Funktionen:

(a) Eine Funktion `wort_bzgl_laenge` mit ...

Eingabe: Ganzzahl `n` ≥ 0, Alphabet-String `A`, Ganzzahl `l` ≥ 0

Ausgabe: das `n`-te Wort aller Wörter der Länge `l`, über dem Alphabet `A`, bzgl. lexikographischer Ordnung (wir beginnen bei 0 zu zählen)

Hinweis: Sie dürfen davon ausgehen, dass `n` maximal die Position des letzten Wortes der Länge `l`, über dem Alphabet `A`, ist.

(b) Eine Funktion `wort` mit ...

Eingabe: Ganzzahl `n` ≥ 0, Alphabet-String `A`

Ausgabe: das `n`-te Wort aller Wörter über dem Alphabet `A`, bzgl. längen-lexikographischer Ordnung

Präsenzaufgabe 2:

Schreiben Sie ein Programm, welche eine vom Nutzer gewählte ganze Zahl (von -64 bis 64) mittels binärer Suche errät. Das Programm soll in jedem Schritt dafür eine Vermutung anstellen, welche Zahl der Nutzer gewählt hat und der Nutzer muss antworten, ob die gewählte Zahl größer, kleiner oder gleich der Vermutung ist.

Beispiel:

```
Überlegen Sie sich eine Ganzzahl aus {-64,...,64}.
Ist die Zahl größer [>], kleiner [<] oder gleich [=] 0? >
Ist die Zahl größer [>], kleiner [<] oder gleich [=] 32? <
Ist die Zahl größer [>], kleiner [<] oder gleich [=] 16? >
Ist die Zahl größer [>], kleiner [<] oder gleich [=] 24? <
Ist die Zahl größer [>], kleiner [<] oder gleich [=] 20? >
Ist die Zahl größer [>], kleiner [<] oder gleich [=] 22? <
Ist die Zahl größer [>], kleiner [<] oder gleich [=] 21? =
Die gesuchte Zahl ist die 21.
```

Präsenzaufgabe 3:

Schreiben Sie ein Programm, welches folgende Funktionen definiert. Sie können davon ausgehen, dass die Funktionen nur mit Ganzzahlen aufgerufen werden:

- Eine Funktion `schaltjahr`, welche eine Ganzzahl (Jahr) entgegen nimmt und prüft, ob es sich um ein Schaltjahr handelt. In diesem Fall soll `True` zurück gegeben werden, sonst `False`.

  Hinweis 1: Ein Jahr ist ein Schaltjahr, wenn es durch 400 teilbar ist oder wenn es durch 4 aber nicht durch 100 teilbar ist. Zum Beispiel waren 2000 und 2004 Schaltjahre, 1900 aber nicht.

- Eine Funktion `tage_im_monat`, welche zwei Ganzzahl (Monat, Jahr) entgegen nimmt und zurückgibt, wie viele Tage der übergebene Monat im übergebenen Jahr hat. Wird als Monat ein Wert eingegeben der nicht im Bereich 1-12 liegt, so soll 0 ausgegeben werden.

- Eine Funktion `datum_check`, welche drei Ganzzahlen entgegen nimmt und prüft, ob diese einem gültigen Datum vom Format Tag/Monat/Jahr entsprechen. Die Funktion soll `True` zurück geben, wenn es sich um ein gültiges Datum handelt und `False` sonst.

  Hinweis 1: Ein Datum ist gültig, wenn der Tag eine positive Ganzzahl ist, die zum Monat passt und der Monat im Bereich 1-12 liegt. Die Jahreszahl wird als vollständig angegeben interpretiert, d.h. 23 entspricht dem Jahr 23 und ***nicht*** *dem Jahr 2023.*

Fordern Sie anschließend drei Eingaben (Tag, Monat, Jahr) vom Nutzer an, rufen Sie `datum_check` auf diesen auf und geben Sie das Ergebnis aus.