# Übungsblatt 1 - Cäsarverschlüsselung

In diesem ersten Übungsblatt werden wir uns mit der Cäsarverschlüsselung beschäftigen. Diese ist eine der einfachsten Kodierungen für Text. Die Motivation ist, dass wir eine Nachricht an eine andere Person senden wollen und möchten sicher sein, dass selbst, wenn diese abgefangen wird, der Abhörer sie nicht lesen kann. Dafür haben wir vorher eine Zahl $k$ im Geheimen mit dem Empfänger ausgetauscht. Die Idee ist nun das Alphabet um diese Zahl $k$ zu verschieben und jeden Buchstaben durch sein Gegenstück im neuen Alphabet auszutauschen. Dafür kann man eine sogenannte Cäsar-Scheibe verwenden (s. Abbildung 1).

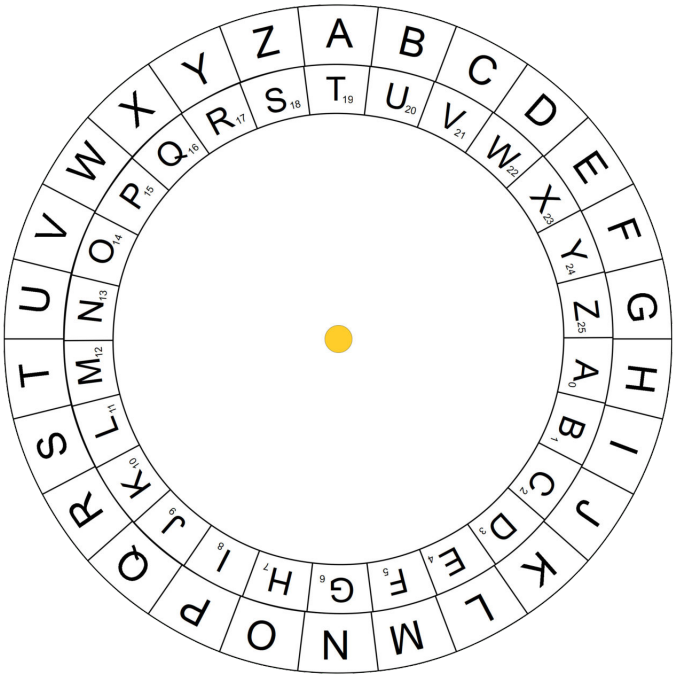



**Abbildung 1:** Cäsar-Scheibe für $k = 19$

Wenn wir jetzt das Wort “Hello” mit $k = 19$ verschlüsseln möchten, tauschen wir jeden Buchstaben auf dem äußeren Ring durch den korrespondierenden Buchstaben auf dem inneren Ring aus. Somit bekommen wir als verschlüsselten Text (auch *Geheimtext* genannt) “axeeh”.
Um nicht zu viele Informationen über den ursprünglichen Text (auch *Klartext* genannt) herzugeben, geben wir den verschlüsselten Text immer in kleinen Buchstaben aus und lassen Leerzeichen sowie Sonderzeichen weg. So wird aus dem Klartext “Hello world!” der Geheimtext “axeehphkew”.

Dieses Verschlüsselungsverfahren wollen wir jetzt in Python umsetzen. Arbeite dafür durch die folgenden Aufgaben.

**Anmerkung:** Man könnte genauso gut den Geheimtext mit nur Großbuchstaben ausgeben. Ich treffe hier die Konvention, dass verschlüsselte Texte in Kleinbuchstaben und entschlüsselte Texte in Großbuchstaben zurückgegeben werden, um diese besser zu unterscheiden.

# Aufgabe 1 - Verschlüsselung

Wir beginnen mit der Verschlüsselung. Wir wollen eine Funktion *casear_encoding* schreiben, welche die folgenden Parameter erhält

- **klartext**: *str*
  Der Klartext als String, welcher die folgenden Zeichen beinhalten kann:
  **a-z**, **A-Z** und alle möglichen **Sonderzeichen**.

- **k**: *int*
  Der Wert $k \in \mathbb{Z}$ der Verschiebung.

und sie soll die folgende Ausgabe erzeugen

- **geheimtex**: *str*
  Der Geheimtext als String, welcher die folgenden Zeichen beinhalten darf: **a-z**.

Es gibt unterschiedliche Möglichkeiten dies nun umzusetzen. Wir werden uns zwei anschauen.

## Aufgabe 1a - Liste

Schreibe die oben definierte Funktion *casear_encoding_liste* mit Hilfe der folgenden Liste:

```
# Buchstabenliste
["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
   "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]
```

Gehe dazu wie folgt vor:

1. Filter den Text, um nur noch Zeichen **a-z** zu haben.
2. Wandel den Wert $k$ in eine Zahl zwischen 0 und 25 um *(optional)*.
3. Verwende geschicktes Zugreifen mit Indizes auf die Buchstabenliste, um zu jedem Buchstaben sein passendes Gegenstück zu finden (eventuell möchtest du das als eine gesonderte Funktion *find_k_letter* auslagern, welche einen Buchstaben $X$ und den Wert $k$ mitbekommt und das Gegenstück $X\_new$ ausgibt).
4. Wandel den Klartext Buchstabe für Buchstabe in den Geheimtext um und gib diesen aus.

*Hinweis 1:* Um den gesamten Text in Großbuchstaben bzw. Kleinbuchstaben zuverwandeln, ohne dabei die Sonderzeichen zu verändern, kannst du die beiden Funktionen *lower* und *upper* recherchieren.

*Hinweis 2:* Du kannst die Buchstabenliste geschickt nutzen, um Sonderzeichen mit *if*-statements und dem “a in B”-Vergleich auszufiltern.

## Aufgabe 1b - ASCII

Es gibt noch einen anderen Weg, welcher keine Buchstabenliste benötigt. Hierfür nutzen wir die Tatsache aus, dass wir jeden Buchstaben durch eine Zahl ausdrücken, Operationen auf dieser Zahl ausführen und sie wieder zurückumwandeln können. Dies werden wir mit dem sogennanten ASCII-Encoding machen. Kurzgesagt, können wir die Großbuchstaben **A-Z** in die Zahlen **65**–**90** umwandeln (für weitere Hintergrundinformationen zur ASCII-Kodierung siehe [2]).

Schreibe die oben definierte Funktion *casear_encoding_ascii* mit Hilfe der beiden Funktionen *chr* und *ord* (recherchiere diese). Gehe dazu wie in Aufgabe 1a vor und ersetze Schritt 1 durch Verwendung des ASCII-Codes eines Zeichens und Schritt 3 durch geschicktes verwenden von Umwandeln eines Buchstaben in ASCII-Code, Ausführung von Addition/Subtraktion und dann Zurückumwandlung in einen Buchstaben.

# Aufgabe 2 - Entschlüsselung

Wir haben nun den Klartext verschlüsselt, der Empfänger kennt bereits den Wert $k$, welchen wir vorher mit ihm/ihr im Geheimen ausgetauscht haben und erhält nun den Geheimtext. Um zu unserem Beispiel aus der Einführung zurückzukommen, der Empfänger kennt $k = 19$ und erhält "axeehphkew". Er/Sie kann nun den Text dekodieren und bekommt "HELLOWORLD".

Schreibe eine Funktion *caesar_decoding*, welche die folgenden Parameter erhält

- **geheimtext**: *str*
  Der Geheimtext als String, welcher die folgenden Zeichen beinhalten kann: **a-z**.
- **k**: *int*
  Der Wert $k \in \mathbb{Z}$ der Verschiebung.

und folgende Ausgabe erzeugt

- **klartext**: *str*
  Der Klartext als String, welcher die folgenden Zeichen beinhalten darf: **A-Z**.

**WICHTIG:** Die Funktion soll keine komplizierten Operationen machen. Sie darf aber die Kodierungsfunktionen aus Aufgabe 1 verwenden.

# Aufgabe 3 - Knacken

Wir haben bereits gesehen, wie wir die Casear-Verschlüsselung nutzen können um Texte zu kodieren und dekodieren. Als nächstes versetzen wir uns die Rolle des Abhörers. Wir haben die Nachricht nun abgefangen, besitzen also den Geheimtext, kennen aber nicht den Wert $k$. Bei der Caesar-Verschlüsselung gibt es nur 26 Möglichkeiten den Text zu verschlüsseln, insofern könnten wir einfach in einer Schleife durch alle möglichen Werte $k$ gehen, den Text mit *caesar_decoding* entschlüsseln, per Hand durchgehen und schauen, welcher Text nicht nur Buchstabensalat ist. Dies ist bei der Cäsar-Verschlüsselung eine mit Aufwand verbundene aber machbare Brute-Force-Entschlüsselung.

**Definition 1.** Der Begriff **Brute Force**, übersetzt "rohe Gewalt", kommt aus der Informatik und bezeichnet Methoden zur Lösung von Problemstellungen, bei denen systematisch alle denkbaren Möglichkeiten durchprobiert werden, bis die Lösung gefunden ist.

Als stolze Programmierer wollen wir uns aber nicht mit "roher Gewalt" zufrieden geben und suchen nach einer anderen Möglichkeit den Wert für $k$ aus dem Geheimtext herauszufinden. Wir haben bereits bei dem Einführungsbeispiel gesehen, dass bestimmte Eigenschaften des Klartextes erhalten bleiben, wie der doppelte Buchstabe in "Hello" und "axeeh". Außerdem bleibt auch die Häufigkeitsverteilung der Buchstaben gleich, d.h. wenn wir vorher die Verteilung $\{L : 2, E : 1, H : 1, O : 1\}$ haben, hat der Geheimtext die gleichen Werte mit anderen Labels $\{E : 2, X : 1, A : 1, H : 1\}$. Dies versuchen wir jetzt als clevere Abhörer auszunutzen. Gehe davon aus, dass der Geheimtext sehr lang ist, z.Bsp. 300+ Wörter, und auf Deutsch.

Schreibe eine Funktion *caesar_cracking*, welche den folgenden Parameter erhält

- **geheimtext**: *str*
  Der Geheimtext als String, welcher die folgenden Zeichen beinhalten kann: **a-z**. Zudem 300+ Wörter lang und auf Deutsch ist.

und das folgende zurückgibt

- **klartext**: *str*
  Der Klartext als String, welcher die folgenden Zeichen beinhalten darf: **A-Z**.
- **k**: *int*
  Der Wert $k \in \mathbb{Z}$ der Verschiebung.

Gehe dazu wie folgt vor:

1. Nutze die Häufigkeitsverteilung der Buchstaben im Geheimtext aus, um den Wert $k$ herauszufinden.
2. Dekodiere den Geheimtext und gib den Klartext aus.

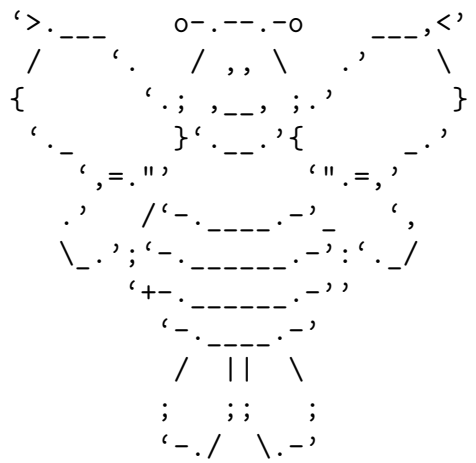
*Hinweis:* Recherchiere die Funktion *Counter* aus der Bibliothek *collections*, um dir ein wenig arbeit beim Finden der Häufigkeitsverteilung zu ersparen.

### Knacken ausprobieren

Um deine Funktion *caesar_cracking* zu testen, suche dir einen Wikipedia Artikel deiner Wahl im Internet heraus und nutze einen der Abschnitte als deinen Klartext, welchen du mit *caesar_encoding* verschlüsseln und dann mit *caesar_cracking* versuchen kannst zu knacken.
Ich verwende dafür immer gerne den Abschnitt "Entstehung und Kontroverse um Urheberschaft" aus dem deutschen Wikipedia Artikel von Batman (siehe [1]).

## Bonus Aufgabe

Schreibe eine Funktion *caesar_encoding_optimal*, welche nur genau einmal über den Klartext iteriert. Hierfür kannst du sowohl den Ansatz aus Aufgabe 1a wie 1b nehmen. Falls deine Kodierungsfunktionen bereits optimiert sind, hier ein Fleißbienchen für dich



## Literatur

[1] *Batman*. Wikipedia, März 2024. URL: https://de.wikipedia.org/wiki/Batman#Entstehung_und_Kontroverse_um_Urheberschaft (besucht am 05. 04. 2024).

[2] KnowHow. *ASCII – Erklärung und Beispiele*. Digital Guide IONOS, Apr. 2022. URL: https://www.ionos.de/digitalguide/server/knowhow/ascii-american-standard-code-for-information-interchange/#:~:text=ASCII%20ist%20ein%207%2DBit,wird%20traditionell%20f%C3%BCr%20Pr%C3%BCfzwecke%20verwendet. (besucht am 05. 04. 2024).