Kryptoprogramme

mit Python

# Überblick

Die Kryptologie findet in unserem Schullehrplan nach einem halbjährigen Programmierblock statt. Damit das Programmierwissen nicht verloren geht, macht es Sinn, dieses in den folgenden Themen immer wieder aufzugreifen. Im Sinne eines Spiralcurriculums soll während des ganzen Schuljahres immer wieder ein Programmierauftrag gestellt werden, damit das Wissen über die Programmiersprache Python aufgefrischt wird.

Die Aufträge haben zum Ziel

* die Python Programmierfähigkeiten zu stärken,
* die Unicode-Tabelle kennenzulernen,
* die im Unterricht besprochenen Verschlüsselungsverfahren zu vertiefen,
* die Sicherheit der besprochenen Verschlüsselungsverfahren zu beurteilen.

# Vorwissen

Die SuS haben sich zu diesem Zeitpunkt bereits ein halbes Jahr mit **Python** beschäftigt. Die Programmierfähigkeiten sind in der Regel noch minimal. Bekannt sind

* Zuweisungen und Variablen
* Datentypen: string und int
* for- und while-Schleifen
* Verzweigungen und Bedingungen
* Listen
* Funktionen mit Parametern

In der Vergangenheit haben die SuS bereits Aufgaben mit Strings gelöst. Konkret haben sie gelernt, dass ein String eine Abfolge von Zeichen darstellt. Sie sollten wissen, dass man über die Zeichen in einem String mit einer for-in-Schleife iterieren kann.

Aus der Kryptologie kennen die SuS die Idee der **Verschiebeverschlüsselung**. Darauf aufbauend soll mit Hilfe von Programmen entschieden werden, dass die Verschiebeverschlüsselung unsicher ist und auch die polyalphabetische Verschlüsselung einfach zu knacken ist. In den Aufträgen wird die Verschiebeverschlüsselung nicht erklärt. Idealerweise wurde diese unmittelbar vor diesen Übungen besprochen.

Aus der Unterstufe kennen die SuS die **ASCII**-Tabelle. Es sollte also eine Vorstellung vorhanden sein, dass jedem Zeichen ein Index in einer Tabelle zugeordnet werden kann und umgekehrt. Im Unterstufenunterricht haben sie Übungen gelöst, bei der eine Binärfolge zu einem ASCII-Wort konvertiert werden sollte und umgekehrt.

# Theorie

## Unicode

Die Unicode-Tabelle codiert Symbole zu einer Dezimalzahl bzw. einer Binärzahl. Python-Strings sind utf-8 codiert.

* Mit ord() kann die Position eines Zeichens in der Tabelle ausgegeben werden.
* Mit chr() kann das Zeichen an der entsprechenden Position in der Tabelle ausgegeben werden.

utf-8 kodiert Zeichen mit variabler Byte-Anzahl. Dabei wird ein Unicode-Zeichen in 1 bis 4 Byte kodiert. Die Codepoints 0 bis 127, die dem ASCII-Zeichensatz entsprechen, werden in einem Byte kodiert, wobei das höchstwertige Bit stets 0 ist. Mit Hilfe des achten Bits kann ein längeres Unicode-Zeichen eingeleitet werden, das sich auf 2, 3 oder 4 Byte erstreckt.

Diese Ideen können mit den SuS mündlich entwickelt und mit der Übung 4.1 abgeschlossen werden.

## Verschiebeverschlüsselung

Der Schlüssel bezeichnet die Anzahl verschobener Stellen in der Unicode-Tabelle (hin zu grösseren Indizes). Die neue Position wird also zu *ord(character) + key*. Durch die Verwendung eines grösseren Schlüsselraums wird das Dechiffrieren ohne Computer zu aufwendig.

Für die Kryptoanalyse reicht es bei genügend kleinen Schlüsseln aus, alle Möglichkeiten von Schlüsseln durchzuprobieren und auf bekannte Wörter zu überprüfen.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Klartext | a | b | … | z | A | B | … | Z |
| Geheimtext |  |  |  | a | b | … | … | … |

## Monoalphabetische Verschlüsselung

Bei der monoalphabetischen Verschlüsselung wird jedes Symbol durch ein beliebiges anderes ersetzt. Das Knacken von Hand ist aufwendig. Der Schlüsselraum ist wesentlich grösser als bei der Verschiebeverschlüsselung. Die Verschlüsselung kann geknackt werden, indem man die Symbolhäufigkeit in der Geheimnachricht mit der Häufigkeit der Buchstaben in der deutschen Sprache vergleicht.

Eine wesentliche Erkenntnis, die aus der Übung gewonnen werden kann, ist: Eine Vergrösserung des Schlüsselraums führt nicht zu einer Verbesserung der Sicherheit.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Klartext | a | b | … | z | A | B | … | Z |
| Geheimtext | 🤓 | α | … | 🥳 | % | # | … | / |

# Erläuterungen zu den Aufträgen

Ein Grossteil des Codes ist in den Vorlagen bereits vorhanden. Die SuS sollen die Skripte lediglich ergänzen, sodass sie das Problem lösen können.

## Unicode mit Python

In dieser Übung sollen die SuS die Fähigkeit erhalten, die Position eines Zeichens in der Unicode-Tabelle auszulesen bzw. umgekehrt. Nach dem Ende der Übung sollten folgende Ideen vorhanden sein:

* Unicode als Tabelle int <=> char
* Funktion ord()
* Funktion chr()
* Positionen von a-z
* Positionen von A-Z

Bearbeite das Skript 1\_shift\_cipher\_intro.py und löse die dortigen Programmieraufgaben.

## Verschiebeverschlüsselung – Schlüssel bestimmen

Gegeben ist eine Nachricht und deren Verschlüsselung. Gesucht ist der Verschiebeschlüssel. Als Hilfestellung ist die Entschlüsselungsfunktion vorgegeben. Die SuS sollen also

* die encrypt() Funktion nachvollziehen
* die encrypt () Funktion verwenden, um den Schlüssel zu bestimmen
* mit Hilfe der encrypt () Funktion eine decrypt() Funktion schreiben
* mit dem obigen Schlüssel einen Geheimtext entschlüsseln
1. Bearbeite das Skript 2\_shift\_cipher\_unicode.py und löse die dortigen Programmieraufgaben.
2. Wird das Finden des Schlüssels schwieriger, wenn du mehr Unicode-Symbole zulässt?

Die Aufgabe wird minim schwieriger. Die Ausführung des Programms kann etwas länger dauern. Bei 140'000 möglichen Schlüsseln dauert die Durchführung aber auf jeden Fall nicht lange, d.h. maximal einige Sekunden.

## Verschiebeverschlüsselung knacken

Gegeben ist eine verschlüsselte Nachricht. Durch Ausprobieren soll der Schlüssel < 1000 ausfindig gemacht werden. Dieses Beispiel zeigt auf, dass die Entschlüsselung trivial ist, auch wenn keine Klartextnachricht bekannt ist.

1. Bearbeite das Skript 3\_shift\_cipher\_decrypt.py und löse die dortigen Programmieraufgaben.
2. Wird das Finden des Klartexts schwieriger, wenn du mehr Unicode-Symbole zulässt?

Die Aufgabe wird etwas schwieriger. Man muss maximal 140'000 verschiedene Nachrichten sichten, um eine sinnvolle zu finden. Abhilfe kann man schaffen, wenn man nach bekannten Symbolen filtert, d.h. nach ASCII-Zeichen.

## Monoalphabetische Verschlüsselung knacken

Das vorliegende Programm bietet eine Alternative zu Online-Tools mit Unicode-Smileys. Die SuS sollen die Verschlüsselung knacken, indem sie Emojis durch Buchstaben ersetzten. Die Gross- und Kleinschreibung muss beachtet werden. Als Hinweis wird die Tabelle rechts vorgegeben. Man beachte, dass in der Tabelle Gross- und Kleinbuchstaben nicht unterschieden werden, bei den Emojis schon. Die Aufgabe ist trotzdem lösbar.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

1. Bearbeite das Skript 4\_monoalphabetic\_cipher.py und löse die dortigen Programmieraufgaben.
2. Wird das Finden des Klartexts schwieriger, wenn du mehr Unicode-Symbole zulässt?
3. Die Aufgabe wird nicht schwieriger. Man muss genau die gleiche Aufgabe lösen. Die Symbole sind nur Platzhalter für ein unbekanntes Zeichen.