1.13 Projekt Verschlüsselung (Teil 1): die Caesar-Chiffre

Leitidee

Wie verschickt man eine Nachricht so, dass nur Eingeweihte sie lesen können? Schon der römische Feldherr Julius Caesar soll dafür ein einfaches Verfahren benutzt haben: Jeder Buchstabe wird im Alphabet um ein paar Stellen verschoben. In diesem und im nächsten Abschnitt programmiert ihr dieses Verfahren **im Team**. Dabei braucht ihr fast alles, was ihr bisher gelernt habt: Zeichenketten, Schleifen, Verzweigungen, Funktionen und den Rest-Operator `%`. Ihr lernt außerdem, wie man ein Programmierprojekt plant, die Arbeit aufteilt, testet und die Teile am Ende zusammenfügt.

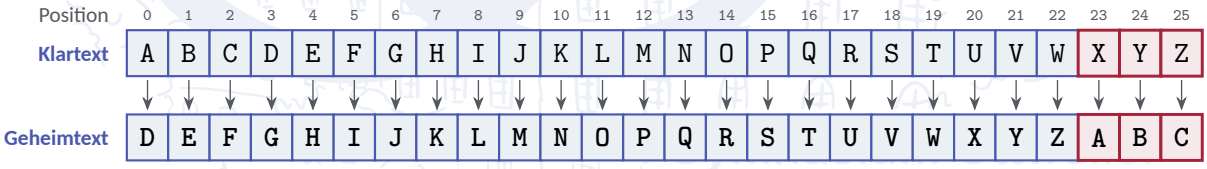
Lernziele

Nach diesem Abschnitt kannst du

- das Verfahren der Caesar-Chiffre beschreiben und von Hand anwenden,
- die Position eines Buchstabens im Alphabet bestimmen und mit `% 26` am Ende des Alphabets vorne weiterzählen,
- Funktionen zum Verschlüsseln eines Buchstabens und eines ganzen Textes implementieren und systematisch testen,
- ein Programmierprojekt im Team planen, Teilaufgaben verteilen und die Ergebnisse zusammenführen.

1.13.1 Wie die Caesar-Chiffre funktioniert

Bei der **Caesar-Chiffre** wird jeder Buchstabe eines Textes im Alphabet um eine feste Anzahl von Stellen nach hinten verschoben. Diese Anzahl ist der **Schlüssel**. Mit dem Schlüssel 3 wird aus A ein D, aus B ein E, aus C ein F und so weiter. Am Ende des Alphabets geht es vorne weiter: Aus X wird A, aus Y wird B, aus Z wird C.



Caesar-Chiffre mit dem Schlüssel 3: Jeder Buchstabe der oberen Reihe wird durch den Buchstaben darunter ersetzt. Am Ende des Alphabets geht es vorne weiter (rot)

So wird aus dem **Klartext** `HALLO` mit dem Schlüssel 3 der **Geheimtext** `KDOOR`: H → K, A → D, L → O, L → O, O → R. Wer den Schlüssel kennt, kann den Geheimtext wieder entschlüsseln, indem er jeden Buchstaben um 3 Stellen **zurück** verschiebt.

Definition: Klartext, Geheimtext, Schlüssel

Beim **Verschlüsseln** wird ein lesbarer **Klartext** mithilfe eines **Schlüssels** in einen **Geheimtext** umgewandelt. Beim **Entschlüsseln** wird aus dem Geheimtext mit dem Schlüssel wieder der Klartext.

Arbeitsauftrag 157: Von Hand verschlüsseln

Benutze die Abbildung oben (Schlüssel 3).

1. Verschlüssele das Wort `INFORMATIK`:

I	N	F	O	R	M	A	T	I	K

2. Entschlüssele den Geheimtext `ZDVVHU`.
3. Beschreibe in einem Satz, wie du beim Entschlüsseln vorgegangen bist.

Deine Antworten zu 2 und 3:

1.13.2 Das Projekt im Team planen

Ein Programm, an dem mehrere Personen arbeiten, entsteht nicht dadurch, dass alle gleichzeitig drauflos tippen. Professionelle Softwareteams gehen in Schritten vor. Für euer Projekt sieht das so aus:

1. **Verstehen und planen:** Was genau soll das Programm können? Wie löst man das Problem – erst einmal ohne Computer?
2. **Zerlegen und verteilen:** Das Problem wird in Teilaufgaben zerlegt. Für jede Teilaufgabe wird eine Funktion geplant. Jede Person im Team übernimmt eine Teilaufgabe.
3. **Umsetzen und testen:** Jede Person programmiert ihre Funktion und testet sie mit vorher festgelegten Beispielen.
4. **Zusammenführen:** Die Funktionen werden zu einem Programm zusammengesetzt und gemeinsam getestet.
5. **Dokumentieren und vorstellen:** Das Team hält fest, was das Programm kann, wie es aufgebaut ist und was offen geblieben ist.

Damit ihr gleichzeitig an verschiedenen Funktionen arbeiten könnt, müsst ihr **vorher** festlegen, wie jede Funktion heißt, welche Parameter sie bekommt und was sie zurückgibt. Das nennt man die **Schnittstelle** einer Funktion. Wer die Funktion benutzt, muss nur die Schnittstelle kennen – nicht, wie die Funktion innen arbeitet.

Definition: Schnittstelle einer Funktion

Die **Schnittstelle** einer Funktion legt fest, wie sie heißt, welche Parameter sie erwartet und welchen Rückgabewert sie liefert.

Für euer Projekt sind die Schnittstellen schon vorgegeben:

Teil	Funktion mit Parametern	Rückgabewert
A	`verschluessle_buchstabe(buchstabe, schluessel)`	der verschobene Buchstabe
B	`verschluessle_text(text, schluessel)`	der Geheimtext
C	Hauptprogramm (keine Funktion)	– (fragt ab und gibt aus)

Teilaufgabe B benutzt die Funktion aus Teilaufgabe A. Trotzdem kann man B schon programmieren, bevor A fertig ist: Man weiß ja, wie A heißt und was A zurückgibt.

Tipp

Du arbeitest den Abschnitt allein zu Hause durch? Dann übernimm alle Teilaufgaben nacheinander. Der Ablauf bleibt derselbe: erst planen, dann jede Funktion einzeln programmieren und testen, dann zusammenfügen.

Arbeitsauftrag 158: Projektplan

Plant gemeinsam auf Papier, **bevor** ihr programmiert. Beantwortet dazu die drei Fragen:

1. Wie ermittelt man zu einem Buchstaben seine Position im Alphabet?
2. Wie kommt man von der Position zurück zum (verschobenen) Buchstaben?
3. Was passiert am Ende des Alphabets, z. B. bei `Z` mit dem Schlüssel 3?

Verteilt dann die Teilaufgaben. Tragt ein, wer welche Teilaufgabe übernimmt und wer sie testet. Zum Testen tauscht ihr: Niemand testet die eigene Funktion allein.

Teilaufgabe	programmiert von	getestet von	fertig?
A: einen Buchstaben verschlüsseln			
B: einen Text verschlüsseln			
C: interaktives Programm			

Unsere Antworten zu den Fragen 1 bis 3:

1.13.3 Werkzeuge für das Projekt

Das Alphabet als Zeichenkette

Das Alphabet speichert ihr am einfachsten als Zeichenkette. Den Index eines Zeichens findet ihr mit `.index()`. Diese Methode kennst du von den Listen. Sie funktioniert bei Zeichenketten genauso:

```
1 alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2 print(alphabet)
3 print("Position von C:", alphabet.index("C"))
4 print("Buchstabe an Position 5:", alphabet[5])
```

AUSGABE

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Position von C: 2
Buchstabe an Position 5: F
```

- **Zeile 1:** Das Alphabet besteht aus 26 Großbuchstaben. A hat den Index 0, Z den Index 25.
- **Zeile 3:** `alphabet.index("C")` geht vom **Zeichen** zur **Position**: Ergebnis 2.
- **Zeile 4:** `alphabet[5]` geht von der **Position** zum **Zeichen**: Ergebnis F.

Stolperstein

Kommt das gesuchte Zeichen nicht im Alphabet vor – zum Beispiel ein Leerzeichen oder ein Kleinbuchstabe –, meldet `.index()` einen Fehler:

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
print(alphabet.index("a"))
```

FEHLERMELDUNG

```
ValueError: substring not found
```

substring not found heißt „Teilzeichenkette nicht gefunden“. Prüft deshalb vorher mit `zeichen in alphabet`, ob das Zeichen überhaupt ein Großbuchstabe ist.

Am Ende des Alphabets: der Rest-Operator

Verschiebt man X (Position 23) um 3, ergibt sich die Position 26. Die gibt es nicht: `alphabet[26]` führt zu einem `IndexError`. Nach Z soll es wieder bei A weitergehen – wie bei einer Uhr, auf der nach 12 wieder die 1 kommt.

Genau das leistet der **Rest-Operator** `%`, den du von den Operatoren kennst. `zahl % 26` ist der Rest bei der Division durch 26. Für Zahlen von 0 bis 25 ändert sich nichts. Aus 26 wird 0, aus 27 wird 1, aus 28 wird 2:

```
1 for position in range(22, 26):
2     neue_position = (position + 3) % 26
3     print(position, "->", neue_position)
```

AUSGABE

```
22 -> 25
23 -> 0
24 -> 1
25 -> 2
```

W (22) wird zu Z (25). X (23) wird zu A (0), Y zu B, Z zu C – genau wie in der Abbildung oben. Die Klammern in Zeile 2 sind wichtig: Zuerst wird addiert, dann der Rest gebildet.

Merke

Mit `(position + schluessel) % 26` erhält man immer eine gültige Position von 0 bis 25. Am Ende des Alphabets geht es dadurch vorne weiter.

Werkzeug

Es geht auch ohne Alphabet-Zeichenkette. Der Computer speichert jedes Zeichen als Zahl. `ord(zeichen)` liefert die Zahl zu einem Zeichen, `chr(zahl)` das Zeichen zu einer Zahl. Die Großbuchstaben A bis Z haben die Zahlen 65 bis 90, der Reihe nach:

```
1 print(ord("A"), ord("B"), ord("Z"))
2 print(chr(68))
3 print(ord("C") - ord("A"))
```

AUSGABE

```
65 66 90
D
2
```

Die Position im Alphabet ist also `ord(buchstabe) - ord("A")`, und zurück geht es mit `chr(position + ord("A"))`. Entscheidet euch im Team für **einen** der beiden Wege.

1.13.4 Teilaufgabe A: einen Buchstaben verschlüsseln

Tipp

Gute Tests prüfen nicht nur den **Normalfall** (A → D), sondern auch die **Randfälle**: das Ende des Alphabets (X, Y, Z) und besondere Schlüssel wie 0 oder 25. Gerade dort stecken die meisten Fehler.

Arbeitsauftrag 159: Teilaufgabe A

Schreibt eine Funktion `verschluessle_buchstabe(buchstabe, schluessel)`, die **einen** Großbuchstaben um `schluessel` Stellen im Alphabet verschiebt und den neuen Buchstaben zurückgibt. Denkt an den Übertrag am Ende des Alphabets (`% 26`). Ihr könnt mit diesem Gerüst beginnen:

```
1 alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2
3 def verschluessle_buchstabe(buchstabe, schluessel):
4     # Position finden, verschieben, mit % 26 umlaufen lassen
5
6 print(verschluessle_buchstabe("A", 3))   # erwartet: D
7 print(verschluessle_buchstabe("X", 3))   # erwartet: A
8 print(verschluessle_buchstabe("Z", 1))   # erwartet: A
```

Das Gerüst läuft so noch nicht: Steht im Funktionsrumpf nur ein Kommentar, meldet Python einen `IndentationError`, denn Kommentare zählen nicht als Anweisungen. Ersetzt den Kommentar durch eure eingerückten Programmzeilen.

Testet die Funktion mit den Aufrufen in der Tabelle. Tragt ein, was eure Funktion zurückgibt, und ob es stimmt.

Aufruf	erwartet	tatsächlich	✓ oder ✗
`verschluessle_buchstabe("A", 3)`	D		
`verschluessle_buchstabe("X", 3)`	A		
`verschluessle_buchstabe("Z", 1)`	A		
`verschluessle_buchstabe("M", 0)`	M		
`verschluessle_buchstabe("B", 25)`	A		

1.13.5 Teilaufgabe B: einen ganzen Text verschlüsseln

Für Teilaufgabe B braucht ihr ein Muster, das ihr aus dem Abschnitt über Zeichenketten (*Einen neuen Text aufbauen*) kennt. Es funktioniert wie die Summenbildung – nur mit Text statt Zahlen. Man beginnt mit einer **leeren Zeichenkette** `""` und hängt in einer Schleife mit `+` Zeichen an:

```
1 text = "ABC"
2 ergebnis = ""
3 for zeichen in text:
4     ergebnis = ergebnis + zeichen + zeichen
5 print(ergebnis)
```

AUSGABE

```
AABBCC
```

- **Zeile 2:** `ergebnis` beginnt als leere Zeichenkette – so wie `summe` bei 0 beginnt.
- **Zeile 4:** In jedem Durchlauf wird das Zeichen zweimal angehängt. Der neue, längere Text wird wieder in `ergebnis` gespeichert.

Arbeitsauftrag 160: Teilaufgabe B

1. Schreibt eine Funktion `verschluessle_text(text, schluessel)`. Sie soll in einer Schleife jeden Buchstaben eines Textes mit `verschluessle_buchstabe` verschlüsseln und den Geheimtext als neue Zeichenkette zurückgeben. Testet mit `verschluessle_text("HALLO", 3)` (erwartet: `KDOOR`).
2. **Zusatz-Herausforderung:** Was passiert bei Leerzeichen oder Kleinbuchstaben, z. B. bei `"Hallo Welt!"`? Überlegt euch im Team eine Lösung. Zum Beispiel: Leerzeichen und Satzzeichen bleiben unverändert, und der Text wird vorher mit `.upper()` in Großbuchstaben umgewandelt.

1.13.6 Teilaufgabe C: das interaktive Programm

Arbeitsauftrag 161: Teilaufgabe C

Setzt beide Funktionen zu einem vollständigen Programm zusammen. Es soll den Klartext und den Schlüssel mit `input()` abfragen und den Geheimtext ausgeben. Testet das fertige Programm mit mindestens drei verschiedenen Eingaben.

1.13.7 Zusammenführen und dokumentieren

Wenn alle Teilaufgaben fertig sind, fügt ihr sie in **einer** Datei zusammen, und zwar in dieser Reihenfolge: das Alphabet, Funktion A, Funktion B, das Hauptprogramm. Eine Funktion muss definiert sein, bevor sie aufgerufen wird. Testet danach das **ganze** Programm noch einmal mit allen Tests aus den Teilaufgaben.

Ein Programm, das in ein paar Wochen niemand mehr versteht, ist wenig wert. Schreibt deshalb an den Anfang einen kurzen **Kommentar** mit dem Namen des Programms, den Namen des Teams und einem Satz dazu, was das Programm tut. Kommentare beginnen in Python mit `#` und werden beim Ausführen übersprungen.

Arbeitsauftrag 162: Projekttagebuch

Führt ein Projekttagebuch. Tragt am Ende jeder Arbeitsphase ein, was ihr geschafft habt, welche Probleme aufgetreten sind und was als Nächstes zu tun ist. Ihr braucht es im nächsten Abschnitt, um das Projekt fortzusetzen und vorzustellen.

Datum	erledigt	Probleme und wie wir sie gelöst haben	nächster Schritt

1.13.8 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 163: Caesar mit Schlüssel 5 ●○○

Benutze das Alphabet mit Positionen. Ein Tipp: Schreibe dir zuerst die Positionen auf, addiere 5 und rechne bei Ergebnissen ab 26 noch 26 ab.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

1. Verschlüssele das Wort `PIZZA` mit dem Schlüssel 5.
2. Entschlüssele den Geheimtext `KJWNJS`. Er wurde mit dem Schlüssel 5 verschlüsselt.

Meine Lösungen:

Arbeitsauftrag 164: Rechnen mit % 26 ●○○

Berechne zu jeder Zeile die neue Position `(position + schluessel) % 26` und den Buchstaben, der dort steht. Benutze die Positionstabelle aus dem vorigen Auftrag.

`position`	`schluessel`	`(position + schluessel) % 26`	**Buchstabe**
24	4		
10	13		
3	30		
25	26		

Was fällt dir in der letzten Zeile auf? Erkläre.

Meine Erklärung zur letzten Zeile:

Arbeitsauftrag 165: Zweimal verschlüsseln ●●○

Die Funktionen `verschluessle_buchstabe` und `verschluessle_text` (verbesserte Fassung aus Teilaufgabe B) stehen vor diesen beiden Zeilen:

```
print(verschluessle_text("HALLO", 13))
print(verschluessle_text(verschluessle_text("HALLO", 13), 13))
```

1. Gib an, was die beiden Zeilen ausgeben.
2. Erkläre, warum in der zweiten Zeile wieder der Klartext herauskommt.
3. Gibt es einen weiteren Schlüssel zwischen 1 und 25 mit dieser Eigenschaft? Begründe.

Meine Antworten:

Gymnasium Ottweiler

Arbeitsauftrag 166: Geheimtext in Fünfergruppen ●●●

Früher hat man Geheimtexte ohne Leerzeichen und in Gruppen zu je fünf Buchstaben aufgeschrieben. So verrät die Länge der Wörter nichts mehr über den Klartext.

1. Implementiere eine Funktion `fuenfergruppen(text)`. Sie entfernt zuerst alle Leerzeichen und fügt dann nach je fünf Zeichen ein Leerzeichen ein. Aus `"YWJKKJS ZR FHMY"` wird so `"YWJKK JSZRF HMY"`.
2. Teste die Funktion zusammen mit `verschluessle_text`: Verschlüssele „Treffen um acht“ mit dem Schlüssel 5 und gib das Ergebnis in Fünfergruppen aus.

Tipp

Zerlege die Aufgabe in zwei Schritte. Schritt 1: Baue einen neuen Text ohne Leerzeichen auf (Zeichen nur anhängen, wenn es kein `" "` ist). Schritt 2: Laufe über die **Indizes** dieses Textes. Vor jedem Zeichen, dessen Index durch 5 teilbar ist (außer beim Index 0), hängst du zuerst ein Leerzeichen an. „Durch 5 teilbar“ prüfst du mit `i % 5 == 0`.

1.13.9 Das Wichtigste in Kürze

- Die **Caesar-Chiffre** verschiebt jeden Buchstaben um eine feste Anzahl von Stellen (den **Schlüssel**) im Alphabet. Aus dem **Klartext** wird der **Geheimtext**.
- Die Position eines Buchstabens ist sein Index in `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"` (oder `ord(buchstabe) - ord("A")`).
- `(position + schluessel) % 26` sorgt dafür, dass es nach Z wieder bei A weitergeht.
- Einen neuen Text baut man auf, indem man mit `""` beginnt und in einer Schleife Zeichen anhängt.
- Im Team zerlegt man das Problem in Teilaufgaben, legt vorher die **Schnittstellen** der Funktionen fest, testet jede Funktion mit Normal- und Randfällen und führt die Teile am Ende zusammen.

Im nächsten Abschnitt entwickelt ihr die passende Entschlüsselung und schließt das Projekt ab.

Selbstcheck

- ☐ Ich kann einen kurzen Text von Hand mit der Caesar-Chiffre ver- und entschlüsseln.
- ☐ Ich kann erklären, wozu `% 26` bei der Caesar-Chiffre gebraucht wird.
- ☐ Ich kann die Funktionen `verschluessle_buchstabe` und `verschluessle_text` programmieren.
- ☐ Ich kann eine Funktion mit Normal- und Randfällen testen und das Ergebnis in einer Testtabelle festhalten.
- ☐ Ich kann erklären, was die Schnittstelle einer Funktion ist und warum sie für die Arbeit im Team wichtig ist.

Gymnasium Ottweiler