DS 13 – Weitere Übungen

Projekt Verschlüsselung (Teil 1): die Caesar-Chiffre

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.

Die Aufgaben stehen auch im Skript im Abschnitt „Weitere Übungen“; die Lösungen im Lösungsheft.

Aufgabe 1: 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.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

1. P (15) → 20 = U, I (8) → 13 = N, Z (25) → 30, minus 26 ergibt 4 = E, noch einmal Z → E, A (0) → 5 = F. Ergebnis: `UNEEF`.
2. Beim Entschlüsseln zieht man 5 ab: K (10) → 5 = F, J (9) → 4 = E, W (22) → 17 = R, N (13) → 8 = I, J → E, S (18) → 13 = N. Ergebnis: `FERIEN`.

Kontrolle mit der Funktion aus Teilaufgabe B (die Entschlüsselung prüft man, indem man das Ergebnis wieder verschlüsselt):

Lösung

```python
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def verschluessle_buchstabe(buchstabe, schluessel):
    position = alphabet.index(buchstabe)
    neue_position = (position + schluessel) % 26
    return alphabet[neue_position]

def verschluessle_text(text, schluessel):
    ergebnis = ""
    for zeichen in text.upper():
        if zeichen in alphabet:
            neu = verschluessle_buchstabe(zeichen, schluessel)
            ergebnis = ergebnis + neu
        else:
            ergebnis = ergebnis + zeichen
    return ergebnis

print(verschluessle_text("PIZZA", 5))
print(verschluessle_text("FERIEN", 5))
```

```
UNEEF
KJWNJS
```

Lösung

Ausgabe:

```
UNEEF
KJWNJS
```

Lösung

Typischer Fehler beim Z: Position 30 gibt es nicht. Nach Z geht es vorne weiter, deshalb wird 26 abgezogen. Genau das macht im Programm `% 26`.

Aufgabe 2: 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.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

position	schluessel	(position + schluessel) % 26	Buchstabe
24	4	2	C
10	13	23	X
3	30	7	H
25	26	25	Z

Lösung

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
print((24 + 4) % 26, alphabet[(24 + 4) % 26])
print((10 + 13) % 26, alphabet[(10 + 13) % 26])
print((3 + 30) % 26, alphabet[(3 + 30) % 26])
print((25 + 26) % 26, alphabet[(25 + 26) % 26])
```

```
2 C
23 X
7 H
25 Z
```

Lösung

Ausgabe:

```
2 C
23 X
7 H
25 Z
```

Lösung

24 + 4 = 28, und 28 : 26 ergibt 1 Rest **2**. In der letzten Zeile bleibt der Buchstabe gleich: Mit dem Schlüssel 26 läuft man einmal ganz um das Alphabet herum und landet wieder am Start. Der Schlüssel 26 verschlüsselt also gar nicht – er wirkt wie der Schlüssel 0. Ebenso wirkt der Schlüssel 30 wie der Schlüssel 4.

Aufgabe 3: 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))
```

```
UNYYB
HALLO
```

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.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

```python
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def verschluessle_buchstabe(buchstabe, schluessel):
    position = alphabet.index(buchstabe)
    neue_position = (position + schluessel) % 26
    return alphabet[neue_position]

def verschluessle_text(text, schluessel):
    ergebnis = ""
    for zeichen in text.upper():
        if zeichen in alphabet:
            neu = verschluessle_buchstabe(zeichen, schluessel)
            ergebnis = ergebnis + neu
        else:
            ergebnis = ergebnis + zeichen
    return ergebnis

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

```
UNYYB
HALLO
```

Lösung

Ausgabe:

```
UNYYB
HALLO
```

Lösung

1. Die erste Zeile gibt `UNYYB` aus, die zweite wieder `HALLO`.
2. Die zweite Zeile verschiebt jeden Buchstaben zweimal um 13, also insgesamt um 13 + 13 = 26 Stellen. Das ist einmal ganz um das Alphabet herum, und jeder Buchstabe landet wieder an seiner alten Stelle.
3. Nein. Zweimal verschieben ergibt nur dann 26 Stellen, wenn der Schlüssel genau die Hälfte von 26 ist, also 13. Deshalb hat das Verfahren mit dem Schlüssel 13 einen eigenen Namen: **ROT13**. Damit verschlüsselt und entschlüsselt man mit derselben Funktion.

Aufgabe 4: 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`.

Lösung

✅ Lösung

Lösung

```python
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def verschluessle_buchstabe(buchstabe, schluessel):
    position = alphabet.index(buchstabe)
    neue_position = (position + schluessel) % 26
    return alphabet[neue_position]

def verschluessle_text(text, schluessel):
    ergebnis = ""
    for zeichen in text.upper():
        if zeichen in alphabet:
            neu = verschluessle_buchstabe(zeichen, schluessel)
            ergebnis = ergebnis + neu
        else:
            ergebnis = ergebnis + zeichen
    return ergebnis

def fuenfergruppen(text):
    ohne = ""
    for zeichen in text:
        if zeichen != " ":
            ohne = ohne + zeichen
    gruppiert = ""
    for i in range(len(ohne)):
        if i > 0 and i % 5 == 0:
            gruppiert = gruppiert + " "
        gruppiert = gruppiert + ohne[i]
    return gruppiert

print(fuenfergruppen("YWJKKJS ZR FHMY"))
geheim = verschluessle_text("Treffen um acht", 5)
print(fuenfergruppen(geheim))
```

```
YWJKK JSZRF HMY
YWJKK JSZRF HMY
```

Lösung

Ausgabe:

```
YWJKK JSZRF HMY
YWJKK JSZRF HMY
```

Lösung

- **Zeilen 19–22:** Der erste Teil baut den Text ohne Leerzeichen auf, wie im Auftrag *Ohne Leerzeichen* im Abschnitt über Zeichenketten.
- **Zeilen 23–27:** Bei den Indizes 5, 10, 15 … wird **vor** dem Zeichen ein Leerzeichen angehängt. Die Bedingung `i > 0` verhindert ein Leerzeichen ganz am Anfang, denn auch 0 ist durch 5 teilbar.
- **Zeilen 31–32:** Die beiden Funktionen arbeiten zusammen: Erst wird verschlüsselt, dann gruppiert.

Wer die Nachricht entschlüsselt, bekommt `TREFF ENUMA CHT` und muss die Wortgrenzen selbst erraten. Das ist der Preis für die zusätzliche Tarnung.