DS 14 – Weitere Übungen

Projekt Verschlüsselung (Teil 2) und Berufe in der Softwareentwicklung

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: Entschlüsseln und Gegenschlüssel ●○○

Der Geheimtext `WGLYPI` wurde mit der Caesar-Chiffre und dem Schlüssel 4 verschlüsselt.

1. Entschlüssele ihn von Hand. Benutze die Positionstabelle aus dem Auftrag *Caesar mit Schlüssel 5* im ersten Projektteil.
2. Gib den Gegenschlüssel an: Mit welchem Schlüssel muss man **verschlüsseln**, um dasselbe Ergebnis zu bekommen wie beim Entschlüsseln mit dem Schlüssel 4?

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

1. Beim Entschlüsseln zieht man 4 ab: W (22) → 18 = S, G (6) → 2 = C, L (11) → 7 = H, Y (24) → 20 = U, P (15) → 11 = L, I (8) → 4 = E. Der Klartext ist `SCHULE`.
2. Der Gegenschlüssel ist 26 − 4 = 22. Vier Schritte zurück landen an derselben Stelle wie 22 Schritte vorwärts.

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("WGLYPI", 22))
print(verschluessle_text("WGLYPI", -4))
```

```
SCHULE
SCHULE
```

Lösung

Ausgabe:

```
SCHULE
```

```
SCHULE
```

Lösung

Beide Aufrufe entschlüsseln: einmal mit dem Gegenschlüssel 22, einmal mit −4. `% 26` sorgt in beiden Fällen dafür, dass das Ergebnis eine gültige Position ist.

Aufgabe 2: Wer macht was? ●○○

In eurem Projekt habt ihr geplant, Schnittstellen festgelegt, programmiert, getestet, dokumentiert und die Sicherheit beurteilt. In einer Softwarefirma übernehmen oft verschiedene Personen diese Aufgaben.

Ordne drei dieser Projektschritte jeweils einem Beruf oder Berufsfeld aus dem Abschnitt *Berufliche Orientierung* zu. Begründe jede Zuordnung in einem Satz.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

Eine mögliche Zuordnung:

- **Programmieren und Testen** – Fachinformatikerin oder Fachinformatiker für Anwendungsentwicklung bzw. Softwareentwicklerin oder Softwareentwickler: Sie schreiben Programme nach den Wünschen der Kundschaft und prüfen sie mit Tests.
- **Sicherheit beurteilen** – IT-Sicherheitsexpertin oder IT-Sicherheitsexperte: Sie untersuchen, ob ein Verfahren angreifbar ist – so wie ihr gezeigt habt, dass man die Caesar-Chiffre durch Ausprobieren knacken kann.
- **Bedienung planen** (zum Beispiel das Menü) – UX/UI-Designerin oder UX/UI-Designer: Sie gestalten, wie Menschen ein Programm bedienen, damit es verständlich ist.

Andere Zuordnungen sind ebenso richtig, wenn die Begründung passt. In kleinen Firmen erledigt oft eine Person mehrere dieser Aufgaben.

Aufgabe 3: Der häufigste Buchstabe verrät den Schlüssel ●●○

Du hast den Geheimtext `NOB OVOPKXD QORD JEW COO` abgefangen. Den Schlüssel kennst du nicht. In deutschen Texten ist **E** meistens der häufigste Buchstabe.

1. Bestimme den häufigsten Buchstaben im Geheimtext. Du kannst von Hand zählen oder die Funktion `zaehle(text, zeichen)` aus dem Auftrag *Buchstaben zählen* im Abschnitt über Zeichenketten benutzen.
2. Nimm an, dass dieser Buchstabe für E steht. Berechne daraus den Schlüssel.
3. Entschlüssele den Text mit `entschluessle_text` und prüfe, ob ein sinnvoller Satz herauskommt.

Deine Antwort:

…

Lösung

✅ **Lösung**

Lösung

1. Der Buchstabe **O** kommt sechsmal vor, jeder andere Buchstabe höchstens zweimal.
2. O hat die Position 14, E die Position 4. Der Schlüssel ist also 14 − 4 = 10.
3. Mit dem Schlüssel 10 ergibt sich ein sinnvoller Satz:

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 entschluessle_text(geheimtext, schluessel):
    return verschluessle_text(geheimtext, 26 - schluessel)

def zaehle(text, zeichen):
    anzahl = 0
    for z in text:
        if z == zeichen:
            anzahl = anzahl + 1
    return anzahl

geheim = "NOB OVOPKXD QORD JEW COO"
print("O:", zaehle(geheim, "O"))
print("D:", zaehle(geheim, "D"))
print(entschluessle_text(geheim, 10))
```

```
O: 6
D: 2
DER ELEFANT GEHT ZUM SEE
```

Lösung

Ausgabe:

```
O: 6
D: 2
DER ELEFANT GEHT ZUM SEE
```

Lösung

Statt alle 26 Schlüssel auszuprobieren, reicht hier **ein** Versuch. Diese Idee heißt **Häufigkeitsanalyse**. Sie zeigt einen zweiten Grund, warum die Caesar-Chiffre unsicher ist: Die Häufigkeit der Buchstaben bleibt beim Verschlüsseln erhalten. Bei sehr kurzen Texten klappt der Trick allerdings nicht immer, weil dort zufällig ein anderer Buchstabe am häufigsten sein kann.

Aufgabe 4: Knacken mit Suchwort ●●●

Die Funktion `brute_force` aus der Erweiterung gibt alle 26 Entschlüsselungen aus. Bei langen Texten ist es mühsam, darin den Klartext zu suchen.

Implementiere eine Funktion `knacke(geheimtext, wort)`, die alle Schlüssel von 0 bis 25 ausprobiert, aber nur die Ergebnisse ausgibt, in denen das Wort `wort` vorkommt (zusammen mit dem Schlüssel). Teste mit dem Geheimtext `KDOOR ZHOW` und dem Suchwort `"WELT"` sowie mit dem Geheimtext aus dem vorigen Auftrag und dem Suchwort `"DER"`.

Tipp: Mit `wort in text` prüfst du, ob ein Textstück in einer Zeichenkette vorkommt (Abschnitt über Zeichenketten). Diese Bedingung setzt du in die Schleife von `brute_force` vor das `print`.

Lösung

✅ Lösung

Lösung

```
1 alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2
3 def verschluessle_buchstabe(buchstabe, schluessel):
4     position = alphabet.index(buchstabe)
5     neue_position = (position + schluessel) % 26
6     return alphabet[neue_position]
7
8 def verschluessle_text(text, schluessel):
9     ergebnis = ""
10    for zeichen in text.upper():
11        if zeichen in alphabet:
12            neu = verschluessle_buchstabe(zeichen, schluessel)
13            ergebnis = ergebnis + neu
14        else:
15            ergebnis = ergebnis + zeichen
16    return ergebnis
17
18 def entschluessle_text(geheimtext, schluessel):
19    return verschluessle_text(geheimtext, 26 - schluessel)
20
21 def knacke(geheimtext, wort):
22    for schluessel in range(26):
23        versuch = entschluessle_text(geheimtext, schluessel)
24        if wort in versuch:
25            print(schluessel, ":", versuch)
26
27 knacke("KDOOR ZHOW", "WELT")
28 knacke("NOB OVOPKXD QORD JEW COO", "DER")
```

```
3 : HALLO WELT
10 : DER ELEFANT GEHT ZUM SEE
```

Lösung

Ausgabe:

```
3 : HALLO WELT
10 : DER ELEFANT GEHT ZUM SEE
```

Lösung

- **Zeile 22:** Wie bei `brute_force` werden alle Schlüssel von 0 bis 25 durchprobiert.
- **Zeile 23:** Das Ergebnis wird in `versuch` gespeichert, weil es zweimal gebraucht wird: für die Prüfung und für die Ausgabe.
- **Zeilen 24–25:** Nur wenn das Suchwort im Versuch vorkommt, wird er ausgegeben.

Das Suchwort muss in Großbuchstaben stehen, weil `verschluessle_text` immer Großbuchstaben liefert. Mit `"Welt"` würde nichts gefunden. Ein Angreifer, der nur vermutet, dass ein häufiges Wort wie „DER“ oder „UND“ im Text steht, findet den Schlüssel so in einem Augenblick.