1. (3 Punkte) (a) Was bedeutet Memoization? Warum wird eine Rekursion dadurch oft viel schneller?
   (b) Was ist mit dem dp-guess gemeint? Erkläre es an einem Beispiel.
   (c) Bei der Tabulation wird die Tabelle ohne Rekursion gefüllt. In welcher Reihenfolge muss man die Werte berechnen, wenn die Teilprobleme Suffixe `a[i:]` sind?

Lösung:

(a) Jedes berechnete Ergebnis eines Teilproblems wird in einem Dictionary gespeichert. Wird dasselbe Teilproblem noch einmal gebraucht, wird nur nachgeschaut. Ohne Memoization werden dieselben Teilprobleme immer wieder berechnet, die Zahl der Aufrufe wächst oft exponentiell. Mit Memoization wird jedes Teilproblem nur einmal berechnet.

(b) Eine Frage über die Lösung, deren Antwort wir nicht kennen, z.B. beim Frosch: Wie groß ist der erste Sprung? Wir probieren alle möglichen Antworten aus, lösen jeweils den Rest rekursiv und nehmen das Beste.

(c) Rückwärts, von `i = n` (oder `n - 1`) bis `i = 0`. Dann sind die kleineren Teilprobleme mit größerem `i` immer schon berechnet, wenn sie gebraucht werden.

2. (3 Punkte) Ein Frosch sitzt auf der ersten Zahl der Liste `a = [10, 30, 40, 20]` und will zur letzten. Er springt jeweils 1 oder 2 Zahlen weiter, ein Sprung kostet den Unterschied der beiden Zahlen. `dp(i)` sind die minimalen Kosten von der Zahl mit Index `i` bis zum Ende.
   (a) Berechne `dp(3)`, `dp(2)`, `dp(1)` und `dp(0)`.
   (b) Welchen Weg nimmt der Frosch?

Lösung:

(a) `dp(3)` = 0
`dp(2)` = 20
`dp(1)` = 10
`dp(0)` = 30
z.B. `dp(0)` = min(|10 - 30| + dp(1), |10 - 40| + dp(2)) = min(20 + 10, 30 + 20) = 30

(b) über die Indizes 0, 1, 3, also 10, 30, 20

3. (3 Punkte) Ein Räuber will in die Häuser einer Straße einbrechen, darf aber keine zwei benachbarten Häuser ausrauben. Die Werte sind `a = [5, 1, 3, 9, 2, 4]`. `dp(i)` ist der maximale Wert für die Häuser ab Index `i`, mit `dp(n) = 0` und `dp(n-1) = a[n-1]`.
   (a) Berechne `dp(i)` für `i = 6, 5, ..., 0`.
   (b) Welche Häuser werden ausgeraubt?

Lösung:
(a) dp(6) = 0, dp(5) = 4, dp(4) = 4, dp(3) = 13, dp(2) = 13, dp(1) = 14, dp(0) = 18
Dabei ist `dp(i) = max(dp(i+1), a[i] + dp(i+2))`.
(b) die Häuser mit den Indizes 0, 3, 5, also 5 + 9 + 4 = 18

4. (3 Punkte) Es gibt Münzen mit den Werten 1, 3, 4. Der Betrag 6 soll mit möglichst wenigen Münzen bezahlt werden, jeder Wert darf mehrmals vorkommen.
   (a) Wie viele Münzen braucht man, wenn man immer zuerst die größte Münze nimmt, die noch passt?
   (b) `dp(i)` ist die kleinste Anzahl von Münzen für den Betrag `i`. Berechne `dp(0)` bis `dp(6)`.
   (c) Wie bezahlt man 6 mit möglichst wenigen Münzen?

Lösung:
(a) 4 + 1 + 1, also 3 Münzen.
(b) dp(0) = 0, dp(1) = 1, dp(2) = 2, dp(3) = 1, dp(4) = 1, dp(5) = 2, dp(6) = 2
Dabei ist `dp(i) = 1 + min(dp(i-1), dp(i-3), dp(i-4))`, soweit die Beträge nicht negativ sind.
(c) 3 + 3, also 2 Münzen. Das gierige Vorgehen findet die beste Lösung nicht.

5. (3 Punkte) Gegeben sind die Wörter ANANAS und BANANE. `dp(i, j)` ist die Länge der längsten gemeinsamen Teilfolge von `w1[i:]` und `w2[j:]`.
   (a) Wie lautet die Rekursionsgleichung für `dp(i, j)`? Unterscheide, ob `w1[i]` und `w2[j]` gleich sind.
   (b) Gib die Länge und eine längste gemeinsame Teilfolge der beiden Wörter an.

Lösung:
(a) Falls `w1[i] == w2[j]`: `dp(i, j) = 1 + dp(i+1, j+1)`.
Sonst: `dp(i, j) = max(dp(i+1, j), dp(i, j+1))`.
Ist eines der Wörter zu Ende, ist der Wert 0.
(b) Länge 4, z.B. ANAN

6. (5 Punkte) Eine Treppe hat die Stufen 0 bis n-1. Wer auf Stufe `i` steht, zahlt `cost[i]` und steigt dann 1 oder 2 Stufen hoch. Man darf auf Stufe 0 oder Stufe 1 beginnen. Oben ist man, wenn man über die letzte Stufe hinaus ist. Wie viel muss man mindestens bezahlen?

```
cost = [10, 15, 20]                          # Ergebnis 15
cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]  # Ergebnis 6
```

Schreibe eine Funktion `dp(i)` mit Memoization für die minimalen Kosten ab Stufe `i` und berechne damit das Ergebnis.

Lösung:

```
def dp(i):
    if i in memo:
        return memo[i]
    if i >= n:
        return 0
    res = cost[i] + min(dp(i + 1), dp(i + 2))
    memo[i] = res
    return res

for cost in [[10, 15, 20], [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]]:
    n = len(cost)
    memo = {}
    print(min(dp(0), dp(1)))
```

Ausgabe: 15, 6. Der dp-guess ist, ob man 1 oder 2 Stufen hochsteigt.

7. (5 Punkte) Gegeben ist das Rucksackproblem mit folgender Eingabe:

```
wert = [5, 6, 3, 7]
gewicht = [4, 7, 2, 8]
kapazitaet = 15
```

Es wird mit der rekursiven Funktion `opt(k, j)` ohne Memoization gelöst (maximaler Wert bei Kapazität `k` mit den Gegenständen 0 bis `j`).

(a) Schreibe alle `opt(k, j)`-Werte, die dabei berechnet werden, in der Reihenfolge auf, in der sie berechnet werden.

(b) Gib die Indizes der gewählten Gegenstände sowie den Wert und das Gewicht des Rucksacks an.

Lösung:

```
opt(15,0)=5
opt(8,0)=5
opt(15,1)=11
opt(13,0)=5
opt(6,0)=5
opt(13,1)=11
opt(15,2)=14
opt(7,0)=5
opt(0,0)=0
opt(7,1)=6
opt(5,0)=5
opt(5,1)=5
opt(7,2)=8
opt(15,3)=15

Items = 0 2 3
Wert = 15, Gewicht = 14
```

8. (5 Punkte) Gegeben ist das Rucksackproblem mit folgender Eingabe:

```
wert = [10, 12, 8, 20]
gewicht = [1, 2, 3, 4]
kapazitaet = 7
```

Es wird mit dynamischem Programmieren und einer Tabelle gelöst (Zeile `j`: Gegenstände 0 bis `j`, Spalte `k`: Kapazität `k`).

(a) Notiere die Tabelle.

(b) Gib die Indizes der gewählten Gegenstände sowie den Wert und das Gewicht des Rucksacks an.

Lösung:

	0	1	2	3	4	5	6	7
0 !	0	10	10	10	10	10	10	10
1 !	0	10	12	22	22	22	22	22
2 !	0	10	12	22	22	22	30	30
3 !	0	10	12	22	22	30	32	42

```
Items = 0 1 3
Wert = 42, Gewicht = 7
```

9. (3 Punkte) Gegeben ist das Rucksackproblem mit folgender Eingabe:

```
wert = [1, 2, 1, 11, 10, 13, 7, 8, 3, 7, 5, 3]
gewicht = [2, 3, 2, 7, 5, 8, 3, 4, 3, 9, 7, 5]
kapazitaet = 18
```

Beim Lösen mit dynamischem Programmieren ist folgende Tabelle entstanden (Zeile `j`: Gegenstände 0 bis `j`, Spalte `k`: Kapazität `k`):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0 !	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1 !	0	0	1	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3
2 !	0	0	1	2	2	3	3	4	4	4	4	4	4	4	4	4	4	4	4
3 !	0	0	1	2	2	3	3	11	11	12	13	13	14	14	15	15	15	15	15
4 !	0	0	1	2	2	10	10	11	12	12	13	13	21	21	22	23	23	24	24
5 !	0	0	1	2	2	10	10	11	13	13	14	15	21	23	23	24	25	25	26
6 !	0	0	1	7	7	10	10	11	17	17	18	20	21	23	23	28	30	30	31
7 !	0	0	1	7	8	10	10	15	17	18	18	20	25	25	26	28	30	31	31
8 !	0	0	1	7	8	10	10	15	17	18	18	20	25	25	26	28	30	31	31
9 !	0	0	1	7	8	10	10	15	17	18	18	20	25	25	26	28	30	31	31
10 !	0	0	1	7	8	10	10	15	17	18	18	20	25	25	26	28	30	31	31
11 !	0	0	1	7	8	10	10	15	17	18	18	20	25	25	26	28	30	31	31

Gib die Indizes der gewählten Gegenstände sowie den Wert und das Gewicht des Rucksacks an.

Lösung:

```
Items = 3 5 6
Wert = 31, Gewicht = 18
```

Man geht von rechts unten nach oben. Ändert sich der Wert zur Zeile darüber, ist der Gegenstand dabei, und man geht um sein Gewicht nach links.