1 While-Schleifen

Aufgabe 1

Implementiere eine Funktion `guessing_game()`. Diese Funktion hat keine Parameter. Beim Aufruf fragt sie nach, welche Zahl erraten werden soll. Anschließend wird solange geraten bis diese Zahl eingegeben wurde.

```
>>> guessing_game()
Gib ein welche Zahl erraten werden soll: 4
Gib eine Zahl ein: 1
Diese Zahl ist zu klein!
Gib eine Zahl ein: 6
Diese Zahl ist zu groß!
Gib eine Zahl ein: 4
Richtig
```

Aufgabe 2

Erweitere die Funktion `guessing_game()`, um einen Zähler für die Anzahl der benötigten Versuche. Beim Ende des Spiels soll ausgegeben werden, wie viele Versuche der Spieler benötigt hat.

Aufgabe 3

Implementiere eine Funktion `power_of_two`, die prüft, ob eine nicht negative ganze Zahl eine Zweierpotenz ist.

```
>>> power_of_two(0)
False
>>> power_of_two(1)
True
>>> power_of_two(2)
True
>>> power_of_two(5)
False
>>> power_of_two(16)
True
```

`https://www.codewars.com/kata/534d0a229345375d520006a0/train/python`

Aufgabe 4

Implementiere eine Funktion `halving_sum`, die für eine positive Zahl n die Summe

$$n + n//2 + n//4 + \cdots + 1$$

berechnet. Z.B. gilt `halving_sum(25)` $= 25 + 12 + 6 + 3 + 1 = 47$.

```
>>> halving_sum(25)
47
>>> halving_sum(1)
1
```

https://www.codewars.com/kata/5a58d46cfd56cb4e8600009d/train/python

Aufgabe 5

Implementiere eine Funktion `nb_year`, die berechnet wann eine Population mit gegebenem Anfangsbestand, jährlichem prozentualen Wachstum und jährlicher Zuwanderung eine bestimmte Grenze überschreitet.

Z.B. könnte man eine Kleinstadt mit 1000 Einwohnern, einem jährlichen Wachstum von 2 Prozent und 50 Personen, die jedes Jahr in die Stadt ziehen betrachten und sich fragen wann 1200 Personen in der Stadt wohnen.

1. Jahr $1000 + 1000 * 0.02 + 50 = 1070$ Einwohner
2. Jahr $1070 + 1070 * 0.02 + 50 = 1141$ Einwohner
3. Jahr $1141 + 1141 * 0.02 + 50 = 1213$ Einwohner

Die Grenze ist also nach 3 Jahren erreicht.

Hinweis: Beachte, dass ab dem 2. Jahr abgerundet wurde, da die Einwohnerzahl eine ganze Zahl sein muss.

```
>>> nb_year(1000, 2 , 50 , 1200)
3
```

https://www.codewars.com/kata/563b662a59afc2b5120000c6/train/python

Aufgabe 6

Implementiere eine Funktion `is_square`, die prüft, ob eine natürliche Zahl eine Quadratzahl ist.

```
>>> is_square(0)
True
>>> is_square(1)
True
>>> is_square(2)
False
```

https://www.codewars.com/kata/54c27a33fb7da0db0100040e/train/python

Aufgabe 7

Collatz-Folgen werden nach dem folgenden Prinzip gebildet.

1. Wähle eine beliebige natürliche Zahl n

2.
   - Wenn n gerade ist, wähle als nächstes Folgenglied $\frac{n}{2}$
   - Wenn n ungerade ist, wähle als nächstes Folgenglied $3 * n + 1$
3. Wiederhole Schritt 2 immer wieder bis die Zahl 1 erreicht ist.

Formaler ausgedrückt, wird die folgende Funktion auf ein Folgenglied angewendet um das nächste Folgenglied zu erhalten.

$$f(n) = \begin{cases} n/2 & \text{falls n gerade ist} \\ 3n+1 & \text{falls n ungerade ist} \end{cases}$$

Z.B. erhält man für die Startzahl 5 die folgende Collatz-Folge: $5, 16, 8, 4, 2, 1$

Implementiere eine Funktion `hotpo`, die berechnet wie viele der oben erklärten Berechnungen durchgeführt werden müssen, um von einer eingegeben Startzahl auf die Zahl 1 zu kommen

```
>>> hotpo(1)
0
>>> hotpo(5)
5
```

`https://www.codewars.com/kata/577a6e90d48e51c55e000217/train/python`

Aufgabe 8

Implementiere eine Funktion `collatz`. Diese gibt einen String zurück, der alle Zwischenergebnisse der Collatz-Folge anzeigt, bis 1 erreicht wurde.

```
>>> collatz(1)
'1'
>>> collatz(5)
'5->16->8->4->2->1'
```

`https://www.codewars.com/kata/5286b2e162056fd0cb000c20/python`