Guessing Game

Inhaltsverzeichnis

1 Guessing Game

1.0.1 Programmidee

- Computer wählt zufällige Zahl zwischen 1 und 100
- Der Spieler versucht die Zahl zu erraten
  - Er gibt einen Tipp ab
  - Der Computer gibt Feedback (Treffer, zu groß, zu klein)
  - Anzahl der Versuche wird gezählt

1.0.2 Schritt 1 - Zufallszahl auswählen

Wer erinnert sich noch an das Python Modul dafür?

```
import random

number = random.randint(1, 100)
```

1.0.3 Schritt 2 - Text einlesen und als Zahl interpretieren

Welche Funktionen werden dafür benötigt?

```
guess = int(input("Bitte Zahl eingeben: "))
```

ACHTUNG Keinerlei Fehlerbehandlung!

1.0.4 Schritt 3 - Eingabe mit Zufallszahl vergleichen

```
import random

number = random.randint(1, 100)
guess = int(input("Bitte Zahl eingeben:"))

if number != guess:
```

```
    print(f"{guess} war nicht die gesuchte Zahl")
    print(f"Die gesuchte Zahl war {number}!")
else:
    print(f"Treffer! Die gesuchte Zahl war {number}.")
```

1.0.5 Schritt 4 - Mehrere Versuche erlauben

```
import random

number = random.randint(1, 100)
guess = int(input("Bitte Zahl eingeben:"))

while number != guess:
    print(f"{guess} war nicht die gesuchte Zahl")
    guess = int(input("Bitte Zahl eingeben:"))

print(f"Treffer! Die gesuchte Zahl war {number}.")
```

1.0.6 Schritt 5 - Hinweis ausgeben

```
import random

number = random.randint(1, 100)
guess = int(input("Bitte Zahl eingeben:"))

while number != guess:
    if number > guess:
        print(f"Die gesuchte Zahl ist größer als {guess}.")
    else:
        print(f"Die gesuchte Zahl ist kleiner als {guess}.")

    guess = int(input("Bitte Zahl eingeben:"))

print(f"Treffer! Die gesuchte Zahl war {number}.")
```

1.0.7 Schritt 6 - Versuche zählen

```
import random

number = random.randint(1, 100)
guess = int(input("Bitte Zahl eingeben:"))
guesses = 1

while number != guess:
    if number > guess:
        print(f"Die gesuchte Zahl ist größer als {guess}.")
    else:
        print(f"Die gesuchte Zahl ist kleiner als {guess}.")

    guess = int(input("Bitte Zahl eingeben:"))
    guesses += 1

print(f"Treffer nach {guesses} Versuchen! Die gesuchte Zahl war {number}.")
```

1.0.8 Variante 1 - Zufälliges Raten

- Ausprobieren aller Zahlen von 1 bis 100 in zufälliger Reihenfolge
  - Keine Zahl doppelt versuchen
- Ideen?

1.0.9 Variante 1 - Zufälliges Raten

- Hilfsklasse

```
import random

class RandomGuess:
    def __init__(self, minimum, maximum):
        self.guessed = set()
        self.minimum = minimum
        self.maximum = maximum

    def next_guess(self):
        while True:
```

```
            guess = random.randint(self.minimum, self.maximum)
            if guess not in self.guessed:
                self.guessed.add(guess)
                return guess
            if len(self.guessed) > self.maximum - self.minimum:
                raise Exception("out of guesses")
```

1.0.10 Variante 1 - Zufälliges Raten

- Test der Hilfsklasse

```
guessed = list()
guesser = RandomGuess(1, 10)
try:
    while True:
        guessed.append(guesser.next_guess())
except:
    pass

print(sorted(guessed))
```

1.0.11 Variante 1 - Zufälliges Raten

```
import random

guesser = RandomGuess(1, 100)          # <--
number = random.randint(1, 100)
guess = guesser.next_guess()           # <--
guesses = 1

while number != guess:
    if number > guess:
        print(f"Die gesuchte Zahl ist größer als {guess}.")
    else:
        print(f"Die gesuchte Zahl ist kleiner als {guess}.")

    guess = guesser.next_guess()       # <--
    guesses += 1
```

```
print(f"Treffer nach {guesses} Versuchen! Die gesuchte Zahl war {number}.")
```

1.0.12 Variante 2 - Brute-force search

- Alle Möglichkeiten nacheinander ausprobieren

1.0.13 Variante 2 - Brute-force search

```
import random

number = random.randint(1, 100)
guess = 1                                # <--
guesses = 1

while number != guess:
    if number > guess:
        print(f"Die gesuchte Zahl ist größer als {guess}.")
    else:
        print(f"Die gesuchte Zahl ist kleiner als {guess}.")

    guess += 1                           # <--
    guesses += 1

print(f"Treffer nach {guesses} Versuchen! Die gesuchte Zahl war {number}.")
```

1.0.14 Variante 3 - Binary search

- Geschickte Auswahl der Zahlen zur schnellen Verkleinerung des Suchraums.

1.0.15 Variante 3 - Binary search

```
import random

number = random.randint(1, 100)
guess = 50                               # <--
partition_size = 50                      # <--
```

```
guesses = 1

while guesses < 15 and number != guess:
    if number > guess:
        print(f"Die gesuchte Zahl ist größer als {guess}.")
        guess = round(guess + partition_size / 2)
    else:
        print(f"Die gesuchte Zahl ist kleiner als {guess}.")
        guess = round(guess - partition_size / 2)

    partition_size /= 2                         # <--
    guesses += 1

print(f"Treffer nach {guesses} Versuchen! Die gesuchte Zahl war {number}.")
```

2 Fakultät

2.0.1 Fakultät - iterativ

```
def factorial(x):
    if x < 0:
        raise Exception('undefined')
    result = 1
    for i in range(1, x+1):
        result *= i
    return result

print(factorial(5))
```

2.0.2 Fakultät - rekursiv

```
def factorial_recursive(x):
    if x == 0:
        return 1;
    else:
        return x * factorial_recursive(x-1)

print(factorial_recursive(5))
```

3 Fibonacci Zahlen

3.0.1 Fibonacci Zahlen - rekursiv

```
def fib(n):
    if n < 1:
        raise Exception('undefined')
    elif n < 3:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

```
fib(40)
```

3.0.2 Fibonacci Zahlen - Dynamische Programmierung

```
_fib_dyn = {}
def fib_dyn(n):
    if n < 1:
        raise Exception('undefined')
    elif n < 3:
        return 1
    else:
        if n not in _fib_dyn:
            _fib_dyn[n] = fib_dyn(n-1) + fib_dyn(n-2)
        return _fib_dyn[n]
```

```
fib_dyn(40)
```

3.0.3 Fibonacci Zahlen - Vergleich

```
import timeit
timeit.timeit('print(fib(40))', globals={'fib': fib}, number=1)
```

```
_fib_dyn = {}
print(_fib_dyn)
timeit.timeit('print(fib_dyn(40))', globals={'fib_dyn': fib_dyn}, number=1)
```

```
print(_fib_dyn)
```

4 Türme von Hanoi

4.0.1 Türme von Hanoi

- Idee: Bevor man eine Scheibe transportiert werden kann, müssen erst alle Scheiben darüber transportiert werden.
- Vom linken Stapel mit n Scheiben müssen zunächst die oberen n-1 Scheiben auf den mittleren Stapel transportiert werden, damit n Scheiben auf den rechten Stapel transportiert werden können.
- Das kann man rekursiv fortführen:
  - Die oberen n-2 Scheiben müssen dazu zunächst auf den rechten Stapel transportiert werden.
  - Die oberen n-3 Scheiben müssen dazu zunächst auf den mittleren Stapel transportiert werden.

4.0.2 Türme von Hanoi

Ausgangssituation:

```
tower1=[4,3,2,1]
tower2=[]
tower3=[]
```

4.0.3 Türme von Hanoi - manuell

```
tower3.append(tower1.pop())
print(tower1, tower2, tower3)
```

```
tower2.append(tower1.pop())
print(tower1, tower2, tower3)
```

```
tower2.append(tower3.pop())
print(tower1, tower2, tower3)
```

4.0.4 Türme von Hanoi - manuell

```
tower3.append(tower1.pop())
print(tower1, tower2, tower3)
```

```
tower1.append(tower2.pop())
print(tower1, tower2, tower3)
```

```
tower3.append(tower2.pop())
print(tower1, tower2, tower3)
```

```
tower3.append(tower1.pop())
print(tower1, tower2, tower3)
```

4.0.5 Türme von Hanoi

Algorithmus:

```
def hanoi(n, towerA, towerB, towerC):
    """move n discs from tower A to tower C"""
    if n > 0:
        # move the upper n-1 discs from tower A to tower B
        hanoi(n - 1, towerA, towerC, towerB)

        # if there are discs left tower A, move them to tower C
        if len(towerA)>0:
            towerC.append(towerA.pop())

        # move the upper n-1 discs from tower B to tower C
        hanoi(n - 1, towerB, towerA, towerC)
```

```
tower1=[4,3,2,1]
tower2=[]
tower3=[]

print(tower1, tower2, tower3)
hanoi(len(tower1), tower1, tower2, tower3)
print(tower1, tower2, tower3)
```