Aufgabenbeschreibungen it4all

1_Zahlen

1. Implementieren Sie die Funktion `ggt(a: int, b -> int) -> int`, die den größten gemeinsamen Teiler zweier Zahlen berechnet!

Angabe: def ggt(a: int, b: int) -> int:

```
pass
```

2. Implementieren Sie die Funktion `factorial(n: int) -> int`, die die Fakultät der Zahl `n` berechnet.

Angabe: def factorial(n: int) -> int:

```
pass
```

3. Implementieren Sie die Funktion `babylonian_root(number: float, count: int) -> float`, die eine Näherung der Wurzel der Zahl `number` in einem iterativen Verfahren nach der Babylonischen Methode berechnet.

Die Formel für die einzelnen Werte x_n ergibt (mit Startwert x_0 = number) sich aus $x_{n+1} = 1/2 \cdot (x_n + x_0 / x_n)$. Es sollen exakt `count` Iterationen berechnet werden.

Es soll eine Exception geworfen werden, falls das Argument `number` keine Zahl (int oder float) oder kleiner oder gleich 0 oder das Argument `count` keine Ganzzahl oder kleiner als 0 ist.

Angabe: def babylonian_root(number: float, count: int) -> float:

```
pass
```

4. Implementieren Sie die Funktion `fibonacci(number: int) -> int`, die die n. Zahl der Fibonacci-Folge berechnet.

Die Werte der Fibonacci-Folge ergeben sich mit den Startwerten $x_0 = x_1 = 1$ nach der Formel $x_n = x_{n-1} + x_{n-2}$.

Angabe: def fibonacci(number: int) -> int: pass

2_Strings

1.
Erstellen Sie ein Skript, das prüft, ob der übergebene String ein Palindrom darstellt! Ein leerer String ist für diese Aufgabe auch ein Palindrom. Ihr Programm soll Groß- und Kleinschreibung ignorieren, d. h. 'a' == 'A'.

Angabe: def is_palindrome(word: str) -> bool:

    pass

2.
Implementieren Sie die Funktion `def format_floating_point_exponential(number: float) -> str`. Diese Funktion soll die übergebene Fließkommazahl in Exponentialschreibweise formatieren. Dazu soll die Zahl in der Form $x = m \cdot 10^e$ mit der Mantisse m und dem Exponenten e normiert werden. Bei der Darstellung sollen die Mantisse und der Exponent dann durch ein `e` getrennt werden. Es sollen keine Nachkommastellen gerundet werden.

Beispiele: $\pi \approx 3.1415 = 3.1415 \cdot 10^0$:= 3.1415e0

Hinweis: Bei mehrfacher Division kann es zu Rundungsfehlern kommen.

Angabe: from math import log10

def format_floating_point_exponential(number: float) -> str:

    pass

3.

Implementieren Sie die Funktion `def file_name_and_ending(filename: str) -> Tuple[str, str]`. Diese Funktion bekommt einen Dateinamen übergeben und soll ihn in Dateinamen und Endung (ohne Punkt) trennen. Dabei sollen auch Spezialfälle beachtet werden:

- Es gibt Dateien ohne Endung (z. B. `config`)
- Dateien, die mit einem Punkt beginnen (z. B. `.gitignore`), sind versteckt und haben keine Dateiendung, sondern nur einen Namen. Der Punkt gehört zum Namen.
- Es gibt Dateien mit mehreren Endungen (z. B. `application.conf.json`)

Angabe: from typing import Tuple

def file_name_and_ending(filename: str) -> Tuple[str, str]:

    pass

4.
Implementieren Sie die Funktion `def name_search(all_names: List[str], fragment: str) -> List[Tuple[str, str]]`. Diese Funktion bekommt eine List von Namen und ein Teil eines Namens übergeben und soll überprüfen, welche der Namen das Fragment beinhalten. Für diese Namen soll jeweils das Präfix und das Suffix als Tuple zurückgegeben werden, durch die sich durch Konkatenation mit dem Fragment der Name bilden lässt.

Beispiel: Für das Fragment `nn` und dem Namen `anna` soll das Tuple `('a', 'a')` zurückgegeben werden.

Angabe: from typing import Tuple, List

def name_search(all_names: List[str], fragment: str) -> List[Tuple[str, str]]:

    pass

5.
Implementieren Sie die Funktion `def three_chinese(line: str, target_vowel: str) -> str`. Diese Funktion soll nach dem Schema des bekannten Kinderliedes 'Drei Chinesen mit nem Kontrabass' im übergebenen String `line` alle Vokale einschließlich der Umlaute ä, ö und ü gegen den angegebenen Vokal `target_vowel` austauschen. Dabei sollen aufeinanderfolgende Vokale auf einen einzigen reduziert werden.

Angabe: def three_chinese(line: str, target_vowel: str) -> str:

    pass

6.
Ziel dieser Funktion ist es, einen Weihnachtsbaum auf der Konsole auszugeben. Dieser hat jeweils eine Stumpfhöhe und eine Kronenhöhe, die jeweils in Zeilen angegeben sind. Die oberste Kronenzeile hat eine Breite von eins, die zweite eine Breite von drei, die dritte von fünf, usw. Sie sind jeweils mittig angeordnet.

Dazu werden mehrere Funktionen implementiert, die (teilweise) aufeinander aufbauen. Daher ist es empfehlenswert, die Funktionen in der hier angegebenen Reihenfolge zu implementieren.

- `def xmas_tree_stub(tree_height: int) -> str:`

  Diese Funktion zeichnet eine Stumpfzeile. Der Buchstabe für den Stumpf soll sich direkt in der Mitte befinden.

  Eine Zeile für einen Baum mit einer Kronenhöhe von 3 sieht folgendermaßen aus:

  ```
  stump_height_3 = '# I #'
  ```

Einfache Version

```
simple_tree: str = """\
#############
#     *     #
#    ***    #
#   *****   #
#  *******  #
# ********* #
#     I     #
#     I     #
#############"""
```

- `def xmas_tree_top_simple(row: int, tree_height: int) -> str:`

  Diese Funktion zeichnet jeweils eine Kronenzeile. Das Argument `row` gibt dabei den Zeilenindex an, `tree_height` die Kronenhöhe. Die Mitte der Krone befindet sich in der Mitte des Strings.

  Eine Beispielrückgabe für die dritte Zeile (Index zwei) eines Baumes mit einer Kronenhöhe von drei sieht beispielsweise folgendermaßen aus:

  ```
  crown_row_3_height_3 = '# ***** #'
  ```

- `def xmas_tree_simple(treetop_height: int, stub_height: int) -> str:`

  Diese Funktion soll einen kompletten Baum zeichnen.

Geschmückte Version

```
design_tree: str = """\
#############
#     *     #
#    *o*    #
#   *J*J*   #
#  *o*o*o*  #
# *J*J*J*J* #
#     I     #
#     I     #
#############"""
```

- `def xmas_tree_top_design(row: int, treetop_height: int) -> str:`

  Diese Funktion soll eine geschmückte Kronenzeile zeichnen. Dabei sollen jeder zweite * jeweils mit einem J bei einem geraden Zeilenindex oder einem o bei einem ungeraden ersetzt werden.

- `def xmas_tree_design(treetop_height: int, stub_height: int) -> str:`

  Diese Funktion zeichnet jeweils einen geschmückten, kompletten Baum.

Angabe:

def xmas_tree_top_simple(row: int, tree_height: int) -> str: pass

```
def xmas_tree_top_design(row: int, tree_height: int) -> str:
    pass

def xmas_tree_stub(h: int) -> str:
    pass

def xmas_tree_simple(treetop_height: int, stub_height: int) -> str:
    pass

def xmas_tree_design(treetop_height: int, stub_height: int) -> str:
    pass
```

7.
Bei der A1Z26-"Verschlüsselung" werden die einzelnen Buchstaben des Alphabets mit dem Wert ihrer Position im Alphabet ersetzt. So wird zum Beispiel aus einem 'a' eine 1, aus einem 'b' eine 2 und aus einem 'z' eine 26.

Anmerkungen:

- Bei einer symmetrischen Verschlüsselung wird der selbe Schlüssel für das Ver- und Entschlüsseln der Nachricht benutzt.
- Bei diesen Aufgaben beschränken wir uns auf Kleinbuchstaben.
- Ein Wort ist im Kontext dieser Aufgabe eine Menge von Kleinbuchstaben, also zum Beispiel "test", "von" oder "azcasflh".

- `def encrypt_letter(letter: str) -> int:`

  Diese Funktion verschlüsselt einen einzelnen Buchstaben.

- `def decrypt_letter(letter: str) -> int:`

  Diese Funktion entschlüsselt einen einzelnen Buchstaben.

- `def encrypt_word(word: str) -> List[int]:`

  Diese Funktion verschlüsselt ein Wort. Der übersichtlichkeit wegen soll das Ergebnis als Liste zurückgegeben werden.

- `def decrypt_word(word: str) -> List[int]:`

  Diese Funktion entschlüsselt ein vorher mit `encrypt_word` verschlüsseltes Wort.

Angabe: from typing import List

ord_a: int = ord('a')

```
def encrypt_letter(letter: str) -> int:
    pass

def decrypt_letter(letter: int) -> str:
    pass

def encrypt_word(word: str) -> List[int]:
    pass

def decrypt_word(word: List[int]) -> str:
    pass
```

8.
Die Ceasar-Chiffre ist ein einfaches Verschlüsselungsverfahren, das, wie der Name schon sagt, von Ceasar erfunden bzw. zumindest benutzt wurde. Dabei wird jeder Buchstabe um einen vorher festgelegten Wert (bei Ceasar selbt z. B. 3) "nach hinten geschoben" (rotiert). Dabei wird zum Beispiel mit dem Parameter 3 aus einem 'a' ein 'd', aus einem 'b' ein 'e' und so weiter. Sollte man über die Grenzen des Alphabets hinausgehen, wird wieder von vorne angefangen. So wird beim Parameter 3 aus einem 'x' ein 'a', aus einem 'y' ein 'b' und aus einem 'z' ein 'c'.

Anmerkungen:

- Bei einer symmetrischen Verschlüsselung wird der selbe Schlüssel für das Ver- und Entschlüsseln der Nachricht benutzt.
- Bei diesen Aufgaben beschränken wir uns auf Kleinbuchstaben.
- Ein Wort ist im Kontext dieser Aufgabe eine Menge von Kleinbuchstaben, also zum Beispiel "test", "von" oder "azcasflh".

Klasse `CeasarCipher`

Implementieren Sie die Klasse `CeararCiphre`. Diese bekommt im Konstruktor eine Ganzzahl übergeben, die die Anzahl an Rotationen darstellt.

Implementieren Sie außerdem folgende Methoden:

- `def crypt_letter(self, lower_letter: str) -> str:`

  Diese Methode ist dafür zuständig, einen einzelnen Buchstaben zu verschlüsseln, also ihn im Alphabet um die angegebene Anzahl nach hinten zu rotieren.

- `def decrypt_letter(self, lower_letter: str) -> str:`

  Diese Methode ist das Komplement zu `crypt_letter` und soll die Verschlüsselung eines einzelnen Buchstaben rückgängig machen.

- `def crypt_word(self, lower_word: str) -> str:`

  Diese Methode soll ein ganzes Wort verschlüsseln, also jeden Buchstaben einzeln.

- `def decrypt_word(self, lower_word: str) -> str:`

  Diese Methode soll ein einzelnes Wort entschlüsseln.

- `def crypt_text(self, lower_text: str) -> str:`

  Diese Methode soll einen ganzen Text verschlüsseln. Dazu sollen zuerst die Sätze anhand der Punkte getrennt werden, danach die Wörter anhand der Leerzeichen, die Wörter verschlüsselt und alles wieder zusammengesetzt werden.

- `def decrypt_text(self, lower_text: str) -> str:`

  Diese Methode soll einen kompletten Text entschlüsseln.

Methode `crack_ceasar`

Um zu testen, wie einfach die Ceasar-Verschlüsselung zu knacken ist, wollen wir einen einfachen Wörterbuchangriff programmieren.

- `def crack_ceasar(encrypted_text: str) -> Optional[str]`

  Die Funktion bekommt einen zu knackenden, verschlüsselten String übergeben. Sie soll ihn mit den möglichen Rotationsanzahlen (1 bis 25) entschlüsseln und überprüfen, ob der Text mit einem der unten angegeben Wörter startet. Falls ja, ist das Knacken (sehr wahrscheinlich) erfolgreich und der entschlüsselte Satz soll zurückgegeben werden. Wird keine passende Konfiguration gefunden, ist das Wörterbuch nicht groß genug und es soll `None` zurückgegeben werden.

  Verwenden Sie folgendes **Wörterbuch**: `ich, du, er, sie, es, wir, ihr, sie, der, die, das`

Angabe:

```
from typing import Optional

class CeasarCipher:

    point_a = ord("a")

    def __init__(self, rounds: int):

        self.rounds: int = rounds
```

```
    def crypt_letter(self, lower_letter: str) -> str:
        pass
    def decrypt_letter(self, lower_letter: str) -> str:
        pass
    def crypt_word(self, lower_word: str) -> str:
        pass
    def decrypt_word(self, lower_word: str) -> str:
        pass
    def crypt_text(self, text_lower: str) -> str:
        pass
    def decrypt_text(self, text_lower: str) -> str:
        pass
word_list = ["ich", "du", "er", "sie", "es", "wir", "ihr", "sie", "der", "die", "das"]
def crack_ceasar(encrypted_text: str) -> Optional[str]:
    pass
```

3_Bedingungen

1. `def calculate_lottery_win(pot: float, win_class: int) -> float:`

Diese Funktion soll den Gewinn in einer fiktiven Lotterie mit den Gewinnstufen 0 bis 5 berechnen. Der Gewinn berechnet sich in Abhängigkeit vom Pot nach folgendem Schlüssel. Bei der Eingabe einer falschen Stufe (negative Zahl, Zahl größer als 5) soll die Gewinnstufe 0 angenommen werden.

- 0: 0.0% vom Pot
- 1: 0.1% vom Pot
- 2: 0.5% vom Pot
- 3: 2.0% vom Pot
- 4: 12.5% vom Pot
- 5: 50.0% vom Pot

Angabe:
def calculate_lottery_win(pot: float, win_class: int) -> float:

pass

2. `def calculate_discount(has_dog: bool, has_cat: bool, has_hamster: bool) -> int:`

Eine neue Tierhandlung möchte zum Kundengewinn Rabatte gewähren. Dazu werden die Kunden gefragt, welche Tiere sie besitzen. Sollte der Kunde einen Hund oder eine Katze haben, bekommt er 5% Rabatt. Hat er einen Hund und eine Katze, bekommt er 8% Rabatt. Hat er einen Hamster, bekommt er - unabhängig von anderen Rabatten - 2% zusätzlich. Implementieren Sie die Funktion, so dass sie für die Kunden die entsprechenden Rabatte berechnet!

Angabe:
def calculate_discount(has_dog: bool, has_cat: bool, has_hamster: bool) -> int:

pass

3. `def greet(hour: int) -> str:`

Diese Funktion soll eine zur Uhrzeit passenden Grußformel zurückgeben. Falls eine ungültige Zeit (kleiner als 0, größer als 24) angegeben wird soll `I do not know this time.` zurückgegeben werden. Zu folgenden Zeiten (Start einschließlich, Ende ausschließlich) sollen folgende Strings zurückgegeben werden:

- 0 bis 6: `Good night`
- 6 bis 12: `Good morning`
- 12 bis 18: `Good afternoon`
- 18 bis 21: `Good evening`

- 21 bis 24: `Good night`

Angabe:
def greet(hour: int) -> str:

    pass

4_Listen

1. Berechnen Sie den Durchschnittswert aller Elemente als Gleitkommazahl in der übergebenen Liste von (Ganz-)Zahlen! Bei einer leeren Liste soll 'None' zurückgegeben werden.

Angabe:

```
from typing import List, Optional

def average(my_list: List[int]) -> Optional[float]:
    pass
```

2. Suchen Sie aus einer Liste von Zeichenketten jeweils die längste heraus! Bei einer leeren Liste soll 'None' zurückgegeben werden. Ignorieren Sie dabei Groß- und Kleinschreibung.

Angabe:

```
from typing import List, Optional

def longest_string(my_list: List[str]) -> Optional[str]:
    pass
```

3. `def filter_greater(vector: List[int], x: int) -> List[int]:`

Diese Funktion soll die Liste `vector` filtern und nur die Zahlen behalten, die größer als die übergebene Zahl `x` sind.

`def count_lower(vector: List[int], x: int) -> int:`

Diese Funktion soll in der Liste `vector` die Anzahl der Zahlen zählen, die kleiner als die übergebene Zahl `x` sind.

`def bank_card_security_value(digits: List[int]) -> int:`

Diese Funktion soll einen fiktiven Algorithmus zur Berechnung eines Sicherheitswertes für Bankkarten berechnen. Dabei wird die erste Zahl in der Liste mit

eins, die zweite mit zwei, etc., multipliziert. Diese Zahlen werden dann addiert und zurückgegeben.

```
def vector_length(vector: List[int]) -> float:
```

Diese Funktion soll die Liste `vector` als Vektor auffassen und dessen euklidische Länge berechnen. Dabei werden alle Einträge quadriert und addiert. Die Wurzel dieser Summe ist die Länge.

```
def vector_add_scalar(vector: List[int], scalar: int) -> List[int]:
```

Diese Funktion soll die Liste als Vektor auffassen und einen neuen Vektor (als Liste)zurückgeben, in dem zu jedem Element im Originalvektor der übergebene Skalarwert `scalar` addiert wurde.

```
def vector_add_vector(vector1: List[int], vector2: List[int]) ->
List[int]:
```

Diese Funktion soll die beiden Listen als Vektoren auffassen und einen neuen Vektor (als Liste) zurückgeben, in dem die beiden Vektoren addiert wurden. Sollten die Längen der beiden Vektoren nicht übereinstimmen, soll eine leere Liste zurückgegeben werden.

```
def flatten_lists(list_of_lists: List[List[int]]) -> List[int]:
```

Diese Funktion bekommt eine Liste von Listen übergeben und soll diese ebnen, d. h., alle Elemente in den Sublisten in eine neue Liste zusammenfügen.

Angabe:

```
from typing import List

def filter_greater(vector: List[int], x: int) -> List[int]:

    pass




def count_lower(vector: List[int], x: int) -> int:

    pass

def bank_card_security_value(digits: List[int]) -> int:

    pass

def vector_length(vector: List[int]) -> float:

    pass
```

```
def vector_add_scalar(vector: List[int], scalar: int) -> List[int]:
    pass

def vector_add_vector(vector1: List[int], vector2: List[int]) -> List[int]:
    pass

def flatten_lists(list_of_lists: List[List[int]]) -> List[int]:
    pass
```

4.

- `def even_indexes(my_list: List[int]) -> List[int]:`

  Diese Funktion soll die Element an geraden Indizes aus der Liste `my_list` zurückgebenen.

- `def reversed_special(my_list: List[int]) -> List[int]:`

  Diese Funktion soll vom vorletzten Element der Liste aus jeweils jedes dritte Element zurückgeben.

- `def first_half(my_list: List[int]) -> List[int]:`

  Diese Funktion soll die erste Hälfte der Liste zurückgeben. Bei ungeraden Längen soll abgerundet werden.

- `def rotate_right(my_list: List[int], count: int) -> List[int]:`

  Diese Funktion soll eine Liste um `count` Umdrehungen nach rechts rotieren. Dazu wird für jede Umdrehung das letzte Element der Liste an den Anfang der Liste gesetzt.

  Hinweis: Diese Funktionalität lässt sich mit zweifachen Slicing der Liste lösen.

Angabe:

```
from typing import List

def even_indexes(my_list: List[int]) -> List[int]:
    pass

def reversed_special(my_list: List[int]) -> List[int]:
    pass

def first_half(my_list: List[int]) -> List[int]:
```

```
    pass

def rotate_right(my_list: List[int], count: int) -> List[int]:

    pass
```

5_Tupel_und_Dicts

1.
Hinweise:

- Sie sollten die Aufgaben zu Listen bereits gelöst haben, da Listen als bekannt vorausgesetzt werden.
- Einige Aufgaben setzen erweitertes Wissen über Strings wie zum Beispiel die Iteration über einzelne Buchstaben oder das Trennen von Strings an bestimmten Zeichen voraus.

Tupel

- `def min_max(my_list: List[int]) -> Tuple[int, int]:`

  Diese Funktion soll aus der Liste von Ganzzahlen das Minimum und das Maximum extrahieren und zurückgeben.

  Hinweise:

  - Benutzen Sie nicht die Funktionen `min()` und `max()`
  - Sie müssen nur einmal über die Liste iterieren

Die folgenden Aufgaben bekommen jeweils eine Liste von Aktien übergeben. Diese Aktien werden als Tuple aus Name (als String) und Wert (in Cent, als Ganzzahl) dargestellt.

- `def account_value(stocks: List[Tuple[str, int]]) -> float:`

  Diese Funktion soll den Gesamtwert aller Aktien in Euro berechnen.

- `def stock_value(stocks: List[Tuple[str, int]], name: str) -> int:`

  Diese Funktion soll den Wert der Aktie mit den Namen `name` aus der Liste heraussuchen. Sollte keine Aktie mit dem Namen existieren, soll -1 zurückgegeben werden.

Angabe:

```
from typing import Tuple, List

def min_max(my_list: List[int]) -> Tuple[int, int]:

    pass

def account_value(stocks: List[Tuple[str, int]]) -> float:

    pass

def stock_value(stocks: List[Tuple[str, int]], name: str) -> int:

    pass
```

2.
Hinweise:

- Sie sollten die Aufgaben zu Listen bereits gelöst haben, da Listen als bekannt vorausgesetzt werden.
- Einige Aufgaben setzen erweitertes Wissen über Strings wie zum Beispiel die Iteration über einzelne Buchstaben oder das Trennen von Strings an bestimmten Zeichen voraus.

Dictionaries

- `def count_char_occurences(my_str: str) -> Dict[str, int]:`

  Diese Funktion bekommt einen String übergeben und soll zählen, wie oft jeder einzelne Buchstabe darin vorkommt. Es soll dabei nicht zwischen Groß- und Kleinschreibung unterschieden werden.

  Hinweis: Wandeln Sie zuerst alle Groß- in Kleinbuchstaben um.

- `def word_position_list(my_str: str) -> Dict[str, List[int]]:`

  Diese Funktion bekommt einen Text übergeben, der nur aus Kleinbuchstaben und Leerzeichen besteht. Sie soll für jedes Wort im Text die Indizes über die Wörter berechnen. Es ist immer mindestens ein Wort im Text.

  Beispiel: Im Text `dies ist ein test` lautet das Ergebnis `{"dies": [0], "ist": [1], "ein": [2], "test": [3]}`

- `def merge_dicts_with_add(dict1: Dict[str, int], dict2: Dict[str, int]) -> Dict[str, int]:`

  Diese Funktion soll aus zwei Dictionaries ein neues machen. Sollte ein Schlüssel in beiden Ausgangsdictionaries vorhanden sein, sollen die Werte im Ergebnisdictionary addiert werden.

Angabe:

```
from typing import Dict, List

def count_char_occurrences(my_str: str) -> Dict[str, int]:

    pass

def word_position_list(my_str: str) -> Dict[str, List[int]]:

    pass

def merge_dicts_with_add(dict1: Dict[str, int], dict2: Dict[str, int]) -> Dict[str, int]:

    pass
```

3.
Hinweise:

- Sie sollten die Aufgaben zu Listen bereits gelöst haben, da Listen als bekannt vorausgesetzt werden.
- Einige Aufgaben setzen erweitertes Wissen über Strings wie zum Beispiel die Iteration über einzelne Buchstaben oder das Trennen von Strings an bestimmten Zeichen voraus.

Tupel und Dictionaries

- `def tuple_list_to_dict(my_list: List[Tuple[str, int]]) -> Dict[str, int]:`

  Diese Funktion soll eine Liste von Tuples in ein Dictionary umwandeln. Dabei soll der erste Eintrag im Tuple als Schlüssel und der zweite Eintrag als Wert verwendet werden. Sollte ein Schlüssel in mehreren Tuples vorkommen, soll der erste Wert verwendet werden.

- `def intersect_dicts(dict1: Dict[str, int], dict2: Dict[str, int]) -> Dict[str, Tuple[int, int]]:`

  Diese Funktion soll die Übereinstimmungen von zwei Dictionaries in ein neues schreiben. Falls ein Schlüssel in beiden Dicts vorhanden ist, sollen die Werte in beiden Ausgangsdicts als Tuple im Resultat stehen.

Angabe:

```
from typing import Dict, List, Tuple

def tuple_list_to_dict(my_list: List[Tuple[str, int]]) -> Dict[str, int]:

  pass

def intersect_dicts(dict1: Dict[str, int], dict2: Dict[str, int]) -> Dict[str, Tuple[int, int]]:

  pass
```

4.
Hinweise:

- Sie sollten die Aufgaben zu Listen bereits gelöst haben, da Listen als bekannt vorausgesetzt werden.
- Einige Aufgaben setzen erweitertes Wissen über Strings wie zum Beispiel die Iteration über einzelne Buchstaben oder das Trennen von Strings an bestimmten Zeichen voraus.

Fitnessprogramm

Der Weihnachtsmann muss seine Rentiere für die lange Reise wieder in Topform bringen. Dafür muss er wissen, welche Tiere Kraftfutter brauchen und welche etwas weniger. Dazu sollen Sie den BMI der Rentiere berechnen.

Der BMI b der Rentiere berechnet sich aus dem Gewicht w_{kg} (in kg) und der Größe h_m (in m):

$$ b = \frac{w_{kg}}{h_m^2} $$

```
from typing import Dict

reindeers: Dict[str, Dict[str, int]] = {
  "Rudolph": {"age_years": 2, "height_cm": 200, "weight_kg": 120},
  "Comet": {"age_years": 1, "height_cm": 180, "weight_kg": 100},
  "Doner": {"age_years": 3, "height_cm": 210, "weight_kg": 90},
  "Blizzen": {"age_years": 4, "height_cm": 190, "weight_kg": 200},
  "Cupid": {"age_years": 2, "height_cm": 192, "weight_kg": 121},
  "Prancer": {"age_years": 4, "height_cm": 215, "weight_kg": 134},
  "Vixen": {"age_years": 6, "height_cm": 230, "weight_kg": 143},
  "Dancer": {"age_years": 1, "height_cm": 176, "weight_kg": 82},
  "Dasher": {"age_years": 5, "height_cm": 197, "weight_kg": 101}
}
```

- `def bmi(height_cm: int, weight_km: int) -> float:`
- `def calculate_bmi(reindeers: Dict[str, Dict[str, int]]) -> Dict[str, float]:`

  Diese Funktion soll in einem neuen Dictionary jeweils den Namen eines Rentiers und dessen BMI zurückgeben.

Angabe:

```
from typing import Dict

def calculate_bmi(height_cm: int, weight_kg: int) -> float:

  pass

def calculate_reindeer_bmis(reindeers: Dict[str, Dict[str, int]]) -> Dict[str, float]:

  pass
```

6_Funktionen

1. Implementieren Sie folgende Funktionen. Diese bekommen jeweils eine Gleitkommazahl übergeben und sollen eine Gleitkommazahl zurückgeben.

- `def celsius_to_fahrenheit(degrees_celsius: float) -> float:`

  Diese Funktion soll eine Temperatur in Celsius mit der Formel $T_F = T_C \cdot 1,8 + 32$ in Fahrenheit umrechnen.

- `def celsius_to_kelvin(degrees_celsius: float) -> float:`

  Diese Funktion soll eine Temperatur in Celsius mit der Formel $T_K = T_C - 273,15$ in Kelvin umrechnen.

- `def kelvin_to_celsius(degrees_kelvin: float) -> float:`

  Diese Funktion soll eine Temparatur in Kelvin mit der Formel $T_C = T_K + 273,15$ in Celsius umwandeln.

- `def kelvin_to_fahrenheit(degrees_kelvin: float) -> float:`

  Diese Funktion soll eine Temperatur in Kelvin mit der Formel $T_F = T_K \cdot 1,8 - 459,67$ in Fahrenheit umwandeln.

- `def fahrenheit_to_celsius(degrees_fahrenheit: float) -> float:`

  Diese Funktion soll eine Temparatur in Fahrenheit mit der Formel $T_C = (T_F -32) \cdot 5 / 9$ in Celsius umwandeln.

- `def fahrenheit_to_kelvin(degrees_fahrenheit: float) -> float:`

  Diese Funktion soll eine Temparatur in Fahrenheit mit der Formel $T_K = (T_F + 459,67) \cdot 5 / 9$ in Kelvin umwandeln.

Angabe:

```
def celsius_to_fahrenheit(degrees_celsius: float) -> float:

    pass

def fahrenheit_to_celsius(degrees_fahrenheit: float) -> float:

    pass

def celsius_to_kelvin(degrees_celsius: float) -> float:

    pass

def kelvin_to_celsius(degrees_kelvin: float) -> float:
```

Kommentiert [SD1]:

```
  pass

def fahrenheit_to_kelvin(degrees_fahrenheit: float) -> float:

  pass

def kelvin_to_fahrenheit(degrees_kelvin: float) -> float:

  pass
```

2.
Gegeben sind folgende Längeneinheiten:

- Yards: `1 yd = 0,9144 m`
- Meilen: `1 mile = 1.609,344 m`
- Seemeile: `1 sm = 1852 m`
- Inch: `1 in = 2.54 cm`

Implementieren Sie (nach obigem Muster) die folgenden Funktionen:

- `def yards_to_meters(length_yards: float) -> float`
- `def meters_to_yards(length_meters: float) -> float`
- `def miles_to_meters(length_miles: float) -> float`
- `def meters_to_miles(length_meters: float) -> float`
- `def seamiles_to_meters(length_sea_miles: float) -> float`
- `def meters_to_seamiles(length_meters: float) -> float`
- `def inches_to_meters(length_inches: float) -> float`
- `def meters_to_inches(length_meters: float) -> float`

Angabe:

```
def yards_to_meters(length_yards: float) -> float:

  pass

def meters_to_yards(length_meters: float) -> float:

  pass

def miles_to_meters(length_miles: float) -> float:

  pass


def meters_to_miles(length_meters: float) -> float:
```

```
    pass

def seamiles_to_meters(length_sea_miles: float) -> float:

    pass

def meters_to_seamiles(length_meters: float) -> float:

    pass

def inches_to_meters(length_inches: float) -> float:

    pass

def meters_to_inches(length_meters: float) -> float:

    pass
```

7_Klassen

1.
Hinweis: Die Kreiszahl π ist in Python im Paket `math` als pi definiert. Es soll eine Klasse Circle erstellt werden, die einen Kreis mit einem Mittelpunkt (als 2-Tuple) und Radius darstellt.

`def __init__(self, center: Tuple[float, float], radius: float)`: Der Konstruktor bekommt den Mittelpunkt und den Radius übergeben und speichert beides unter dem selben Namen ab.

`def __repr__(self) -> str`: Für Debuggingzwecke soll auch die repr - Funktion überschrieben werden. Diese soll den Kreis im Format "Circle(c = (<x>, <y>), r = <radius>)" (ohne spitze Klammern!) als String repräsentieren. Die Zeichen in spitzen Klammern sollen jeweils mit den entsprechenden Werten ersetzt werden.

`def __eq__(self, other) -> bool`: Diese Funktion wird beim Vergleich einer Instanz eines Kreises mit dem Operator == mit einem beliebigen anderen Objekt aufgerufen. Zwei Kreise sollen dann gleich sein, wenn ihr Mittelpunkt und ihr Radius übereinstimmen.

`def area(self) -> float`: Diese Funktion soll die Fläche des Kreises an Hand der Formel $a = r^2 \cdot \pi$ berechnen.

`def perimeter(self) -> float`: Diese Funktion berechnet den Umfang des Kreises mit der Formel $u = 2 \cdot r \cdot \pi$

`def intersects(self, other: 'Circle') -> bool`: Diese Funktion entscheidet, ob sich diese Instanz eines Kreises mit einer anderen Instanz überschneidet. Dies ist der Fall, wenn der (euklidische) Abstand beider Kreise kleiner (oder gleich für diese Aufgabe) ist als die Summe der Radien beider Kreise. Hinweis: Der euklidische Abstand ist (in $\mathbb{R}^2$) folgendermaßen definiert: $d(p, q) = \sqrt{ (q_1 - p_1)^2 + (q_2 - p_2)^2 }$

Angabe:

```
from math import pi




class Circle:

    def __init__(self, center_x: float, center_y: float, radius: float):

        pass

    def area(self) -> float:

        pass
```

2.
Mau-Mau ist ein Kartenspiel für zwei und mehr Spieler, bei dem es darum geht, seine Karten möglichst schnell abzulegen. Die Spieler legen dabei jeweils reihum - falls möglich - eine Karte auf den Stapel. Möglich ist dies, wenn die abzulegende Karte in Kartenwert oder Kartenfarbe mit der obersten offen liegenden Karte übereinstimmt. Auf die Pik 10 darf also entweder eine andere Pik-Karte oder eine andere 10 gelegt werden.

Gegeben sei folgende Implementierung der Klasse `Card`, die eine Spielkarte darstellt:

```
class Card:
  def __init__(self, suit: int, rank: int):
    self.suit: int = suit
    self.rank: int = rank

  def __eq__(self, other: Any) -> bool:
    return isinstance(other, Card) and other.suit == self.suit and
other.rank == self.rank

  def __repr__(self) -> str:
    return "Card({}, {})".format(self.suit, self.rank)
```

Die Attribute `suit` und `rank` stehen für die Farbe und das Bild der Karte und nehmen jeweils Werte von 1 bis 4 beziehungsweise 2 (Karte 2) bis 14 (Ass) an.

Implementierung

Es sollen folgende Methoden implementiert werden:

- `def can_by_played_on(first_card: Card, second_card: Card) -> bool:`

  Diese Funktion bestimmt, ob die erste Karte aufgrund der Regeln von Mau-Mau auf die zweite gelegt werden kann.

- `def playable_cards(current_card: Card, hand_cards: List[Card]) -> List[Card]:`

  Diese Funktion sucht aus einer Menge an Handkarten diejenigen heraus, die auf die gerade oben liegende gelegt werden können.

Angabe:

```
from typing import List, Any

class Card:

  def __init__(self, suit: int, rank: int):

    self.suit: int = suit

    self.rank: int = rank

  def __eq__(self, other: Any) -> bool:
```

```
        return isinstance(other, Card) and other.suit == self.suit and other.rank == self.rank  # pragma: no
cover

    def __repr__(self) -> str:

        return "Card({}, {})".format(self.suit, self.rank)  # pragma: no cover

def can_be_played_on(first_card: Card, second_card: Card) -> bool:

    pass

def playable_cards(card: Card, hand: List[Card]) -> List[Card]:

    pass
```

3.
Hinweise:

- Sie sollten bereits die Aufgaben zu Funktionen, Bedingungen, String, Listen und Tuples bzw. Dicts bearbeitet haben.
- Einige der zu implementierenden Methoden beruhen aufeinander. Es empfiehlt sich daher, diese in der gegebenen Reihenfolge zu implementieren (bis alle Unittests korrekt sind), da so die Wahrscheinlichkeit eines Fehlschlagens eines Unittests aufgrund eines Fehlers in einer aufgerufenen Methode verringert wird.
- Die einfachen Anführungszeichen bei Type hints mit der eigenen Klasse sind erforderlich, da die Klasse in sich selbst nicht direkt als Type hint referenziert werden kann. Sie haben aber ansonsten keine Wirkung.
- Die Funktionen `__add__`, `__sub__` und `__mul__` werden jeweils aufgerufen (siehe Unittests), falls
    - zu einer Instanz mit `+` addiert wird (`__add__`)
    - von einer Instanz mit `-` subtrahiert wird (`__sub__`)
    - eine Instanz mit `*` multipliziert wird (`__mul__`)

Vektor2D

In dieser Aufgabe soll eine Klasse `Vector2D` implementiert werden, die einen zweidimensionalen Vektor repräsentiert.

- `def __init__(self, x: float, y: float):`

  Der Konstruktor der Klasse bekommt die `x` und `y`-Koordinate übergeben und speichert diese unter dem selben Namen ab.

- `def __repr__(self) -> str:`

Die Funktion `__repr__` wird benutzt, um eine Instanz einer Klasse genauer zu beschreiben. Überschreiben Sie daher diese Funktion, so dass sie die Korrdinatenrepräsentation des Vektor `"(x, y)"` zurückgibt.

- `def __eq__(self, other: Any) -> bool:`

  Diese Funktion soll diese Instanz von `Vector2D` mit einem beliebigen anderen Objekt vergleichen. Sollte das andere Objekt kein Vektor sein, schlägt der Vergleich fehl. Ansonsten müssen für einen erfolgreichen Vergleich die `x`- und die `y`-Korrdinate der beiden Vektoren übereinstimmen.

  Mit der Funktion `isinstance(o, t)` können Sie überprüfen, ob ein Objekt `o` eine Instanz einer Klasse `t` ist.

- `def abs(self) -> float:`

  Diese Funktion berechnet die Länge dieser Instanz.

- `def __add__(self, other: 'Vector2D') -> 'Vector2D':`

  Diese Funktion addiert jeweils die beiden Korrdinaten zweier Vektoren und gibt das Resultat als neuen Vektor zurück.

- `def __sub__(self, other: 'Vecto2D') -> 'Vector2D':`

  Diese Funktion subtrahiert den Vektor `other` von dieser Instanz und gibt das Ergebnis als neuen Vektor zurück.

- `def __mul__(self, scalar: float) -> 'Vector2D':`

  Diese Funktion multipliziert (skaliert) diesen Vektor mit dem übergebenen Wert `scalar` und gibt das Ergebnis als neuen Vektor zurück.

- `def dot(self, other: 'Vector2D') -> 'float':`

  Diese Funktion berechnet das Skalarprodukt dieser Instanz und des Vektors `other`, indem sie die Einträge elementweise multipliziert und addiert.

Angabe:

```
from typing import Any




class Vector2D:

    def __init__(self, x: float, y: float):

        pass
```

```
def __repr__(self) -> str:
    pass
def __eq__(self, other: Any) -> bool:
    pass
def abs(self) -> float:
    pass
def __add__(self, other: 'Vector2D') -> 'Vector2D':
    pass
def __sub__(self, other: 'Vector2D') -> 'Vector2D':
    pass
def __mul__(self, scalar: float) -> 'Vector2D':
    pass
def dot(self, other: 'Vector2D') -> float:
    pass
```

8_Unit_Testing

CODE:

```
import copy

import os

from typing import List, Any


def is_number(s: Any) -> bool:

    """checks if a string can be converted to float"""

    try:

        float(s)

        return True

    except ValueError:

        return False


class Table:

    def __init__(self, name: str):

        self.name: str = name

        self.fields: List[str] = []

        self.data: List[List[str]] = []

    def load_from_csv(self, csv_file, delimiter=';'):

        # reset previous attributes

        self.data = []

        self.fields = []

        if not os.path.isfile(csv_file):

            raise Exception("File not found")
```

```
        with open(csv_file, 'r', encoding="utf-8-sig") as f:
            for idx, line in enumerate(f):
                # remove trailing new line, split by delimiter and set to lowercase
                line = line.rstrip("\n").lower()
                line = line.split(delimiter)
                # first line in file is treated as list of fields
                if idx == 0:
                    self.fields = line
                else:
                    self.data.append(line)
        # convert all number strings to floats
        for i, row in enumerate(self.data):
            for j, entry in enumerate(row):
                if is_number(entry):
                    self.data[i][j] = float(entry)
    def copy(self, original_table: 'Table'):
        self.fields = copy.deepcopy(original_table.fields)
        self.data = copy.deepcopy(original_table.data)
    def length(self) -> int:
        return len(self.fields)
    def insert(self, row: List[str]) -> bool:
        pass
```

CSV:

```
Kuerzel;Gebaeude;Groesse
Z6_HS4;Z6;600
Z6_HS1;Z6;200
```