07_Uebungsaufgaben_Loesungen

0.1 Vorschläge für Übungsaufgaben

Die folgenden Übungsaufgaben sind ausführlicher als die unterwegs vorgeschlagenen Mini-Übungen. Das Ziel ist es hier, nicht nur einzelne Themen direkt auszuprobieren, sondern vielmehr verschiedene Themen zu kombinieren, um so ein verknüpfendes Verständnis der Lerninhalte zu erreichen.

0.1.1 Arbeiten mit dem Debugger

Finden Sie alle Fehler im folgenden Programm, idealerweise mit einem Debugger aus Ihrer Entwicklungsumgebung:

[1]:
```
"""

Ein Programm, dass leider nicht funktioniert.

Korrigieren Sie die 7 Fehler und schreiben Sie als Kommentar den Fehlertyp
dazu (Syntaxfehler, Laufzeitfehler, logischer Fehler)

"""

def flip_lr (my_list):
    """
    Diese Funktion soll eine Liste umdrehen.
    """

    length = len(my_list)

    # Die umgedrehte Liste
    flipped_list = [None]*length

    for i in range(length)
        flipped_list[i] = my_list[length-i]

    return my_list


def find_smallest_entry (my_list):
```

```
    """
    Diese Funktion soll den kleinsten Eintrag der Liste finden.
    """

    length = len(my_list)

    # Index des kleinsten Eintrags
    i_min = 0

    for i in range(length):
        if (my_list[i_min] < my_list[i]):
            i_min = i

    return i_min


def transpose(my_list_of_lists):
    """
    Diese Funktion gibt das transponierte Äquivalent zurück
    """
    number_of_lines   = len(my_list_of_lists)
    number_of_columns = len(my_list_of_lists[0])
    # Die transponierte Liste
    transposed_list_of_lists = [[0]*number_of_lines]*number_of_columns
    # Befüllen der Einträge
    for i in range(number_of_lines):
      for j in range(number_of_columns):
        transposed_list_of_lists[j][i] = my_list_of_lists[i][j]
    return transposed_list_of_lists



# Wir testen die ersten beiden Funktionen mit einer Besipielliste
list_1 = [5,4,3,2,1,0]
print("Wenn wir die Liste ", list_1,
      " umdrehen, erhalten wir ", flip_rl(list_1))
print("Der kleinste Eintrag der Liste ", list_1,
      " ist ", find_smallest_entry(list_1))
print()

# Wir testen mit einer weiteren Beispielliste
list_2 = [2,1,2,1,2,1,2]
print("Wenn wir die Liste ", list_2,
      " umdrehen, erhalten wir ", flip_lr(list_2))
print("Der kleinste Eintrag der Liste ", list_2,
      " ist ", find_smallest_entry(list_2))
print()
```

```
# Wir testen das Transponieren mit transpose mit einer Besipielliste
list_of_lists = [[1,1],[2,2]]
print("Wenn wir ", list_of_lists,
      "transponieren, erhalten wir", transpose(list_of_lists))
print()

# Wir testen mit einer weiteren Beispielliste
list_of_lists = [[1,0],[0,1],[2,2]]
print("Wenn wir ", list_of_lists,
      "transponieren, erhalten wir", transpose(list_of_lists))
```

```
         File "C:\Users\Robin\AppData\Local\Temp\ipykernel_22508\2091735700.py",␣
↪line 20
       for i in range(length)
                              ^
   SyntaxError: invalid syntax
```

Musterlösung:

[3]:
```
"""

Ein Programm, das leider nicht funktioniert.

Korrigieren Sie die 7 Fehler und schreiben Sie als Kommentar den Fehlertyp
dazu (Syntaxfehler, Laufzeitfehler, Logischer Fehler)

"""

def flip_lr (my_list):
    """
    Diese Funktion soll eine Liste umdrehen.
    """

    length = len(my_list)

    # Die umgedrehte Liste
    flipped_list = [None]*length

    for i in range(length):                                    #1␣
↪Syntaxfehler
        flipped_list[i] = my_list[length-1-i]                  #3␣
↪Laufzeitfehler (& logisch)
```

```
    return flipped_list                                          #4 Logischer␣
 ↪Fehler


def find_smallest_entry (my_list):
    """
    Diese Funktion soll den kleinsten Eintrag der Liste finden.
    """

    length = len(my_list)

    # Index des kleinsten Eintrags
    i_min = 0

    for i in range(length):
        if (my_list[i_min] > my_list[i]):                        #5 Logischer␣
 ↪Fehler
            i_min = i

    return my_list[i_min]                                        #6 Logischer␣
 ↪Fehler


def transpose(my_list_of_lists):
    """
    Diese Funktion gibt das transponierte Äquivalent zurück
    """
    number_of_lines   = len(my_list_of_lists)
    number_of_columns = len(my_list_of_lists[0])
    # Die transponierte Liste
    transposed_list_of_lists=[]                                  #7 Logischer␣
 ↪Fehler
    # Befüllen der Einträge
    for j in range(number_of_columns):
      transposed_list_of_lists.append([])
      for i in range(number_of_lines):
        transposed_list_of_lists[j].append(my_list_of_lists[i][j])
    return transposed_list_of_lists

# Wir testen die ersten beiden Funktionen mit einer Besipielliste
list_1 = [5,4,3,2,1,0]
print("Wenn wir die Liste ", list_1,
      " umdrehen, erhalten wir ", flip_lr(list_1))               #2␣
 ↪Laufzeitfehler
print("Der kleinste Eintrag der Liste ", list_1,
      " ist ", find_smallest_entry(list_1))
print()
```

```
# Wir testen mit einer weiteren Beispielliste
list_2 = [2,1,2,1,2,1,2]
print("Wenn wir die Liste ", list_2,
      " umdrehen, erhalten wir ", flip_lr(list_2))
print("Der kleinste Eintrag der Liste ", list_2,
      " ist ", find_smallest_entry(list_2))
print()

# Wir testen das Transponieren mit transpose mit einer Besipielliste
list_of_lists = [[1,1],[2,2]]
print("Wenn wir ", list_of_lists,
      "transponieren, erhalten wir", transpose(list_of_lists))
print()

# Wir testen mit einer weiteren Beispielliste
list_of_lists = [[1,0],[0,1],[2,2]]
print("Wenn wir ", list_of_lists,
      "transponieren, erhalten wir", transpose(list_of_lists))
```

```
Wenn wir die Liste  [5, 4, 3, 2, 1, 0]  umdrehen, erhalten wir  [0, 1, 2, 3, 4,
5]
Der kleinste Eintrag der Liste  [5, 4, 3, 2, 1, 0]  ist  0

Wenn wir die Liste  [2, 1, 2, 1, 2, 1, 2]  umdrehen, erhalten wir  [2, 1, 2, 1,
2, 1, 2]
Der kleinste Eintrag der Liste  [2, 1, 2, 1, 2, 1, 2]  ist  1

Wenn wir  [[1, 1], [2, 2]] transponieren, erhalten wir [[1, 2], [1, 2]]

Wenn wir  [[1, 0], [0, 1], [2, 2]] transponieren, erhalten wir [[1, 0, 2], [0,
1, 2]]
```

0.1.2 Typen von Exceptions

Diese Aufgabe ist primär eine Rechercheaufgabe. Schreiben Sie dazu eine Funktion error_demo(error_kind), welche als Argument einen Fehlertyp error_kind erhält. Ihre Funktion sollte IndexError, ZeroDivisionError, ImportError, KeyError, NameError, TypeError, UnboundLocalError, AttributeError, ValueError, FileNotFoundError unterstützen, und diese Fehler durch passenden Code auslöst. Dazu soll **nicht** die raise Funktion verwendet werden, sondern fehlerhafter Code von der Art, dass der Pythoninterpreter das Programm mit dem geforderten Fehlertyp abbricht.

Beispiel:

In dem Fall, dass der Parameter error_kind keinem der genannten Fehler entspricht, soll mit der Funktion raise ein ValueError mit der Fehlermeldung "Unknown value for parameter error_kind}" ausgegeben werden. Geben Sie außerdem als Kommentar noch ein Beispiel für einen SyntaxError und einen IndentationError.

[4]:
```
def error_demo(error_kind):
    if error_kind == StopIteration:
        l = []
        it = iter(l)
        next(it)
    elif error_kind == OverflowError:
        import math
        math.exp(1000.0)
    elif error_kind == ZeroDivisionError:
        a = 1/0
    elif error_kind == ImportError:
        import irgendein_kaese
    elif error_kind == IndexError:
        a = []
        a[1]
    elif error_kind == KeyError:
        a = {}
        a['a']
    elif error_kind == NameError:
        unknown_function()
    elif error_kind == UnboundLocalError:
        print(irgendein_kaese)
    elif error_kind == FileNotFoundError:
        open("unknwon_file", 'r')
    elif error_kind == TypeError:
        v2 = 1 + 's'
    elif error_kind == ValueError:
        int("a")
    elif error_kind == AttributeError:
        import math
        math.does_not_exist()
    else:
        raise ValueError("Unknown value for Parameter error_kind!")

error_demo(StopIteration)
error_demo(OverflowError)
error_demo(ZeroDivisionError)
error_demo(ImportError)
error_demo(IndexError)
error_demo(KeyError)
error_demo(NameError)
error_demo(UnboundLocalError)
error_demo(FileNotFoundError)
error_demo(TypeError)
error_demo(ValueError)
error_demo(AttributeError)
error_demo("AndererFehler")
```

```
# Beispiel fuer Syntaxfehler / SyntaxError (Doppelpunkt vergessen)
#if True
#    x = 1

# Beispiel fuer Einrueckungsfehler / IndentationError
#if True:
#    x = 1
#   y = 2
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)

~\AppData\Local\Temp\ipykernel_22508\4037900654.py in <module>
 33         raise ValueError("Unknown value for Parameter error_kind!")
 34
---> 35 error_demo(StopIteration)
 36 error_demo(OverflowError)
 37 error_demo(ZeroDivisionError)


~\AppData\Local\Temp\ipykernel_22508\4037900654.py in␣
↪error_demo(error_kind)
  3         l = []
  4         it = iter(l)
----> 5         next(it)
  6     elif error_kind == OverflowError:
  7         import math


StopIteration:
```

0.2 Impressum

0.2.1 Programmierkurs Python, Dominik Göddeke https://www.ians.uni-stuttgart.de, Universität Stuttgart

Version vom April 2023

Gefördert durch die Stiftung Innovation in der Hochschullehre.

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie.

[ ]: