Übungsblatt 06

Aufgabe 1

Ihnen sind drei kleine Python-Programme aus der 4. Vorlesung gegeben. Alle erzeugen eine Ausnahme.

Ändern Sie die Programme so ab, dass die Ausnahmen aufgefangen werden und eine Beschreibung des jeweiligen Fehlers ausgegeben wird.

In [ ]:
```python
# a) aus 4.5.2: einfache Generatorfunktion
def simpleGenerator():
    x = 1
    print("Vor dem ersten Yield")
    yield x
    x = 2
    yield x
    x = 3
    yield x
    print("Kein weiteres Yield")

gen_obj = simpleGenerator()

for i in range(10):
    next(gen_obj)
```

In [ ]:
```python
# b) aus 4.5.4: für Funktionswerte einer Parabel
parabel = (x**2 - 2*x + 3 for x in range (-10,10))
print(f"Minimum = {min(parabel)}")
print(f"Maximum = {max(parabel)}")
```

In [ ]:
```python
# c) aus 4.5.4: rekursive Funktion zur Berechnung der Summe 1 bis n
def rekSum(n):
    if n == 1:
        return n
    else:
        return n + rekSum(n-1)

rekSum(3000)
```

Aufgabe 2

Schreiben Sie eine Funktion, die für reelle Zahlen quadratische Gleichungen der Form $x^2 + px + q = 0$ löst, sofern sie lösbar sind. Die Argumente sind `p` und `q`. Zurückgegeben wird die Lösungsmenge als Tupel `(x1, x2)`.

Die Funktion soll Vor- und Nachbedingungen testen und bei Nichterfüllung eine Ausnahme erzeugen:

- Vorbedingung: $(p/2)^2 - q >= 0$

- Nachbedingung: Alle Werte aus der Lösungsmenge `(x1, x2)` sollen die Gleichung $x^2 + px + q = 0$ erfüllen

Hinweis: die quadratische Gleichung hat die Lösungen:

$x1 = -p/2 + \sqrt{(p/2)^2 - q}$

$x2 = -p/2 - \sqrt{(p/2)^2 - q}$

In [ ]:
```
# TODO Aufgabe 2
```

Aufgabe 3

Schreiben Sie eine Funktion, die reelle Ganzzahlen in ihre Primfaktoren zerlegt. Das Argument der Funktion ist `zahl`. Der Rückgabewert ist eine Liste - beginnend mit 1, auch wenn das keine Primzahl ist, mit den Primfaktoren.

Die Funktion soll Vor- und Nachbedingungen testen und bei Nichterfüllung eine Ausnahme erzeugen:

- Vorbedingung: `zahl` ist vom Typ `int` und größer als 0
- Nachbedingung: Das Produkt aller Primzahlen muss wieder `zahl` ergeben

Beispielausgabe:

```
>>> primfak(13)
[1, 13]
>>> primfak(120)
[1, 2, 2, 2, 3, 5]
>>> primfak(-2)
---------------------------------------------------------------------
----
AssertionError                            Traceback (most recent call
last)
```

In [ ]:
```
# TODO Aufgabe 3
```

Aufgabe 4

In der folgenden Aufgabe sehen Sie ein umfangreicheres Paket. Es soll sich hierbei um ein hypothetisches `sound`-Modul handeln:

```
sound
|-- effects
| |-- echo.py
| |-- __init__.py
| |-- reverse.py
| |-- surround.py
|-- filters
| |-- equalizer.py
| |-- __init__.py
```

```
|  |-- karaoke.py
|  |-- vocoder.py
|-- formats
|  |-- aiffread.py
|  |-- aiffwrite.py
|  |-- auread.py
|  |-- auwrite.py
|  |-- __init__.py
|  |-- wavread.py
|  |-- wavwrite.py
|-- __init__.py
```

- Laden und entpacken Sie das zip-File in Ihrem Arbeitsverzeichnis

In [ ]:
```
# Am besten vorher den Kernel restarten!
# 1. TODO Importieren Sie das sound-Modul und kontrollieren Sie, welche Module g
```

Versuchen Sie nun auf das Modul `effects` zuzugreifen. Was passiert und warum?

In [ ]:
```
# 2. TODO Zugriffsversuch auf sound.effects
```

Versuchen Sie nun zuerst das Modul `effects` direkt zu importieren und im Anschluss darauf zuzugreifen.

In [ ]:
```
# 3. TODO effects importieren und darauf zugreifen
```

- Restarten Sie den Kernel, damit die geladenen Module zurückgesetzt werden
- Ändern Sie das Paket/den Paketinhalt so ab, dass das Modul `effect` automatisch beim Import des übergeordneten `sound` -Moduls geladen wird

In [ ]:
```
# 4. TODO Automatischer Import von effects beim Import von sound
```

- Als Nächstes soll beim Import von `effects` automatisch das Unterpaket `filters` und dessen Modul `karaoke` nachgeladen werden
- Restarten Sie zunächst wieder den Kernel
- Ändern Sie das Paket/den Paketinhalt dementsprechend ab
- Testen Sie das geladene Modul, indem Sie `func1()` des `karaoke` -Moduls aufrufen

In [ ]:
```
# 5. TODO Automatischer Import von formats beim Import von effects
```

Zum Schluss wird noch eine andere Möglichkeit zum automatischen Importieren von Unterpaketen betrachtet.

- Restarten Sie den Kernel
- Entpacken Sie das `sound` -Modul erneut und speichern Sie es unter dem Namen `sound2` in Ihrem Arbeitsverzeichnis
- Versuchen Sie "alles" aus dem `sound2` -Modul zu importieren mithilfe des `*` -Operators

Was fällt Ihnen auf?

In [ ]:
```
# 6. TODO alles Importieren mit *-Operator
```

Das automatische Nachladen kann mithilfe eines Paketindex erreicht werden. Dafür muss in der `__init__.py`-Datei des Pakets eine Liste mit dem Namen `__all__` definiert werden.

- Schreiben Sie in die `__init__.py`-Datei des `sound2`-Moduls eine dementsprechende Liste mit den Inhalten: `"formats", "filters", "effects", "foobar"`
- Restarten Sie den Kernel und führen Sie das 6. TODO nochmals aus

Bei erfolgreicher Implementierung werden beim Import via `*`-Operator alle angegebenen Untermodule nachgeladen.

In [ ]:
```
# 7. TODO Definition der __all__-Liste
```