Testat – Python

Sie ändern vorhandene Code-Fragmente so, dass 10 verschiedene Tests laufen. Sobald alle Tests durchlaufen haben Sie 10 Punkte; je positiv durchlaufenen Test ein Punkt. Tragen Sie Ihren Namen und Matrikelnummer ein und unterschreiben Sie. Geben Sie das Blatt am Ende des Testats ab.

Name: _____	Mat.Nr.: _____	Unterschrift: _____
Benutzername: `bth666`	Passwort: `12345678`	Punkte:

Starten Sie Linux und loggen Sie sich mit o.g. Benutzername/Passwort ein. Starten Sie Visual Studio Code und öffnen Sie den Ordner `~/nethome/Testat`. Ändern Sie nur die Dateien `a1.py`, `a2.py` und `a3.py`. Jede Aufgabe hat Ihren eigenen Unit-Test. Die Tests in den Dateien `test_a1.py`, `test_a2.py` und `test_a3.py` dürfen (normal oder mit dem Unit-Testing in `code`) ausgeführt, aber nicht geändert werden. Die Tests werden vor der Bewertung auf ihren ursprünglichen Zustand zurückgesetzt.

Aufgabe 1. Schreiben Sie in `a1.py` Python Funktionen, die einen Text aus ausschließlich Kleinbuchstaben mit einem aus ausschließlich Kleinbuchstaben bestehenden geheimen Passwort ver- und entschlüsseln. Man läuft zeichenweise durch Text und Passwort gleichzeitig. Wenn man beim letzten Zeichen im Passwort ankommt, beginnt man beim Passwort wieder von vorne. Für den verschlüsselten Text werden die beiden Zeichen verknüpft. Es interessiert nur der Abstand n des Zeichens im Passwort vom Zeichen `a` im Alphabet. Das neue Zeichen ist der n.te Nachfolger des ursprünglichen Zeichens im Text. Wenn man dabei `z` überschreitet, beginnt man wieder bei `a`. Zum Beispiel wird aus `defgh` mit dem Passwort `abc` die chiffrierte Zeichenkette `dfhgi`, da das `d` um 0 (wegen a, Abstand 0) = `d` das `e` um 1 (wegen b, Abstand 1) = `f`, das `f` um 2 (wegen c, Abstand 2) = `h`, das `g` um 0 (wegen a, Abstand 0) = `g`, das `h` um 1 (wegen b, Abstand 1) = `i` verschoben wird. Nutzen Sie `ord` und `chr` für die Berechnung. Implementieren Sie zum Chiffrieren und Dechiffrieren die folgenden Funktionen.

```
def encrypt(text, geheim): ...
def decrypt(chiffriert, geheim): ...
```

Beispiele der Verwendung:

```
>>> a1.encrypt("defgh", "abc")
    "dfhgi"
>>> a1.decrypt(a1.encrypt("defgh", "abc"), "abc")
    "defgh"
>>> a1.encrypt("abcxyz", "ab")
    "accyya"
```

Aufgabe 2. Schreiben Sie in der Datei `a2.py` eine Klasse `UndoList`, die eine Liste ist (Typ `list`). Sie soll sich verhalten wie eine Liste. Es gibt allerdings zusätzlich zu den Listenmethoden eine Methode `undo`, die die letzte Aktion der Methoden `append`, `extend`, `insert`, `remove` sowie das Löschen eines Elements mit **`del`** und das Setzen eines Elements an einem Index rückgängig macht. Es ist am einfachsten sich den Wert der Liste vor einer solchen Aktion in der Instanz zu merken.

Beispiele der Verwendung:

```
>>> ul = UndoList([1,2,3,4,5])
>>> del ul[3]; print(ul); ul.undo(); ul
[1, 2, 3, 5]
[1, 2, 3, 4, 5]
>>> ul.append(6); print(ul); ul.undo(); ul
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5]
>>> ul.extend([6,17,42]); print(ul); ul.undo(); ul
[1, 2, 3, 4, 5, 6, 17, 42]
[1, 2, 3, 4, 5]
>>> ul.insert(0,0); print(ul); ul.undo(); ul
[0, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
>>> ul.remove(4); print(ul); ul.undo(); ul
[1, 2, 3, 5]
[1, 2, 3, 4, 5]
>>> ul[-1]=None; print(ul); ul.undo(); ul
[1, 2, 3, 4, None]
[1, 2, 3, 4, 5]
```

Aufgabe 3. Schreiben Sie in der Datei `a3.py` die folgenden Funktionen, die Generatoren erwarten und zurückgeben.

```
def ith(gen):
    ...
def secondonly(gen):
    ...
```

Sie müssen annehmen, dass der übergebene Parameter `gen` ein Generator ist, der unendlich oft Werte generiert. `ith` gibt überspringt erst eins, dann zwei, dann drei, ... Elemente. `ith` gibt also das erste, das dritte, das sechtse, ... Element zurück. Der Generator `secondonly` gibt nur das zweite Vorkommen des gleichen Wertes von `gen` zurück. Beispiele:

```
>>> list(ith((e for e in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18])))
[1, 3, 6, 10, 15]
>>> list(secondonly((e for e in [1,2,3,4,5,5,4,3,2,1,1,3,5,4,2,5,1,3,2,4])))
[5, 4, 3, 2, 1]
```

Viel Spaß und viel Erfolg!