Stefan Kurtz **Universität Hamburg**
Jeanine Liebold **Zentrum für Bioinformatik**

Programmierung für Naturwissenschaften 1
Wintersemester 2022/2023
Übungen zur Vorlesung: Ausgabe am 14.12.2022

Selbsttest-Aufgaben zum Thema Kontrollstrukturen, Listen, und Funktionen in Python

Um ihre eigenen Fähigkeiten in der Python-Programmierung zu testen, lösen Sie bitte selbständig die folgenden Aufgaben. Nutzen Sie zunächst möglichst nicht den Python-Interpreter, sondern nur Papier und Stift. Sie können Ihre Lösung natürlich später mit dem Python-Interpreter verifizieren. Ihre Lösungen geben Sie nicht ab.

1. Geben Sie bitte an, was das folgende Programmstück ausgibt. Welche Eigenschaft hat die Ausgabe. Was wird in der inneren Schleife berechnet? Wozu dienen die drei Zeilen nach der inneren `for`-Schleife.

```
numbers = [3,4,1,5,0,2]
for i in range(len(numbers)-1):
  min_idx = i
  for j in range(i+1,len(numbers)):
    if numbers[j] < numbers[min_idx]:
      min_idx = j
  tmp = numbers[i]
  numbers[i] = numbers[min_idx]
  numbers[min_idx] = tmp
  print(numbers)
```

2. Beschreiben Sie, was die folgende Funktion berechnet, wenn `n` eine positive ganze Zahl ist und `factors` eine Liste von positiven ganzen Zahlen. Was wird durch die `for`-Schleife nach der Funktionsdefinition berechnet und ausgegeben? Dabei ist `primes_up_to_100` eine Liste der Primzahlen ≤ 100. Sie müssen hier natürlich nicht manuell alle Werte von `n` in der `for`-Schleife ausrechnen.

```
def divisible(n,factors):
  if n == 1:
    return True
  for i in factors:
    if n % i == 0 and divisible(n // i, factors):
      return True
  return False
for n in range(100,150+1):
  if not divisible(n,primes_up_to_100):
    print(n)
```

3. Was berechnet das folgende Programm? Um das zu ermitteln, müssen Sie herausfinden, was jeweils die Funktionen `os.listdir`, `os.path.isdir`, `os.path.getctime` liefern.

Das geht am besten, indem Sie das Programm `bin/pysearch.py` mit den passenden Schlüsselwörtern aufrufen.

```
def collect_file_path_list(file_path_list,directory):
  for entry in os.listdir(directory):
    full_path = '{}/{}'.format(directory,entry)
    if os.path.isdir(full_path):
      collect_file_path_list(file_path_list,full_path)
    else:
      file_path_list.append(full_path)
file_path_list = list()
collect_file_path_list(file_path_list,'.')
chosen_file_path = None
for file_path in file_path_list:
  if chosen_file_path is None or \
     os.path.getctime(chosen_file_path) < \
     os.path.getctime(file_path):
    chosen_file_path = file_path
print(chosen_file_path)
```

4. Geben Sie bitte an, was das folgende Programmstück ausgibt. Beschreiben Sie, was in den Variablen `dataframe` und `line_elements` gespeichert wird. Was müßte man tun, wenn man die Funktion auf eine Datei anwenden möchte, in der in jeder Zeile Tabulator-separierte Werte stehen?

```
def lines2dataframe(lines):
  dataframe = None
  for line in lines:
    line_elements = line.rstrip().split('\t')
    if dataframe is None:
      dataframe = list()
      for key in line_elements:
        dataframe.append((key,list()))
    else:
      for idx, value in enumerate(line_elements):
        dataframe[idx][1].append(value)
  return dataframe
gasverbrauch = ['Woche\t2022\tMean 2018-2021\tMin 2018-2021',
                '1\t1975\t2069\t1763',
                '2\t2239\t2116\t1863',
                '3\t2128\t2073\t1778',
                '4\t2102\t2227\t1752']
print('\n'.join(map(str,lines2dataframe(gasverbrauch))))
```

Aufgabe 8.1 (2 Punkte)

Aus Zeitgründen können wir in der Vorlesung das Thema „Anwendung der in Python verfügbaren Sortiermethoden“ nicht behandeln. Dieser Abschnitt wird daher im Peer-Teaching Format behandelt.

Für diese Übungsaufgabe muss sich aus jeder Kleingruppe ein Student bzw. eine Studentin anhand der Vorlesungsfolien auf dieses Thema vorbereiten. Sie sollten untereinander frühzeitig klären, wer diese Vorbereitung in dieser Woche übernimmt.

Das in der Vorbereitung erworbene Wissen soll an die anderen Mitglieder der jeweiligen Kleingruppe dadurch weitergegeben werden, dass man zusammen den Inhalt der Folien zum Thema *Sorting using Python's build-in methods*) (siehe `python-slides.pdf`, Section 19, frame 1-9) bespricht. Das kann entweder in den ersten 20-25 Minuten der Übung erfolgen oder zu einem anderen Zeitpunkt außerhalb des Übungstermins.

Der oder die Studierende, die sich vorbereitet hat, kann ggf. bei Unklarheiten Fragen, zumindest zu den Grundbegriffen, beantworten oder Erläuterungen mündlich ergänzen. Falls Sie den in den Folien dargestellten Programmcode ausprobieren möchten, können Sie auf die Materialien zurückgreifen.

Nach Bearbeitung dieser Aufgabe dokumentiert jede Kleingruppe in der Textdatei `bearbeitung.txt` in höchstens 15 Zeilen mit maximal 80 Zeichen pro Zeile ihr Vorgehen bei der Erarbeitung des Themas und ggf. noch bestehende Verständnisfragen oder Hinweise zu Unklarheiten in den Folien. Willkommen sind natürlich auch Bemerkungen zur Lehrform selbst. Es soll nicht der Inhalt der Folien wiedergegeben werden. Selbstverständlich müssen Sie in der Datei auch die üblichen Metadaten angeben.

Aufgabe 8.2 (3 Punkte)

Ein Labormediziner möchte zukünftig Laborbefunde per Email an die beauftragenden Ärzte verschicken. Sie arbeiten bei dem Labormediziner als studentische Hilfskraft und sollen das Vorhaben technisch umsetzen.

Als Ergebnis einer Laboruntersuchung werden Dateien generiert, die die Daten in einem Tabulator-separierten Format enthalten. Hier ist ein Beispiel mit dem Inhalt einer Datei `patient1.tsv`:

```
_EMAILADDRESSE_ mabuse@yahoo.com
_ANSPRACHE_ geehrter Herr Dr.
_NAME_ Mabuse
_PATIENT_ Ihres Patienten Hans Mueller
_MESSUNG_ Anzahl der Leukozyten (weiße Blutkörperchen)
_WERT_ 2000/mikro l
_BEFUND_ zu niedrig
```

Jede Zeile enthält genau einen Bezeichner einer Variablen und einen String mit dem Wert dieser Variable. Sie können davon ausgehen, dass der Bezeichner keine White Spaces enthält und mit einem Unterstrich beginnt und endet. Der Begriff *Variable* meint hier nicht Variablen in einem Python-Programm.

In einer weiteren Datei `email_generisch.txt` steht der Text einer generischen Email. Generisch bedeutet, dass der Text der Email aus festen und variablen Anteilen besteht, wie z.B. hier:

```
To: _EMAILADDRESSE_
From: labormedizin@hamburg.de

Sehr _ANSPRACHE_ _NAME_,

Sie hatten uns Proben _PATIENT_ geschickt.
Die Untersuchungsergebnisse liegen nun vor.
Die _MESSUNG_ ist mit _WERT_ _BEFUND_.

Mit freundlichen Grüßen,

Ihr Labormediziner
```

Die Variablen sind die Bezeichner, die mit dem Unterstrich beginnen und enden, und deren Werte in einer anderen Datei definiert sind.

Benennen Sie die Datei `gen_email_template.py` in `gen_email.py` um. Das Programm wird mit zwei Dateinamen als Argument aufgerufen:

- den Namen einer Datei mit den Daten einer Laboruntersuchung und
- den Namen einer Datei mit dem Text der generischen Email.

Das Programm soll an allen Stellen, an denen in der generischen Email eine Variable steht, den String, entsprechend der Zuordnung aus der ersten Datei, ausgeben. Alle anderen Worte und Leerzeichen aus der generischen Email sollen unverändert zeilenweise mit `print` ausgegeben werden. Beispiel: der Aufruf von

```
./gen_email.py patient1.tsv email_generisch.txt
```

soll den folgenden Text im Terminal ausgeben:

```
To: mabuse@yahoo.com
From: labormedizin@hamburg.de

Sehr geehrter Herr Dr. Mabuse,

Sie hatten uns Proben Ihres Patienten Hans Mueller geschickt.
Die Untersuchungsergebnisse liegen nun vor.
Die Anzahl der Leukozyten (weiße Blutkörperchen) ist mit 2000/mikro l zu niedrig.

Mit freundlichen Grüßen,

Ihr Labormediziner
```

Das Programm soll beide Dateien nur einmal lesen. Sie müssen also zunächst Informationen aus der ersten Datei in geeigneter Form speichern. Das soll in einer Funktion `read_variables` erfolgen. Diese erhält als einzigen Parameter den Namen einer Datei, liest diese zeilenweise ein und liefert die Information in geeigneter Form zurück. In einer zweiten Funktion `substitute_variables` sollen die Substitutionen erfolgen. Dazu erhält diese Funktion als ersten Parameter den von der ersten Funktion gelieferten `return`-Wert und als zweiten Parameter den Namen der Datei mit der generischen E-mail.

Um die Worte und Worttrennzeichen aufzuzählen, verwenden Sie den regulären Ausdruck `r'\w+|\W+'` und die Methode `re.findall`.

In den Materialien finden Sie Dateien mit Testdaten und den erwarteten Ausgaben. Durch `make test` verifizieren Sie die Korrektheit Ihres Programms für diese Testdaten.

Punkteverteilung:

- 1 Punkt für die Speicherung der Daten in einer geeigneten Datenstruktur.
- 1 Punkte für die korrekte Aufzählung aller Worte inklusive der Zeichen, die nicht in Worten vorkommen.
- 1 Punkt für die korrekte Anwendung der Datenstruktur zur Übersetzung und die bestandenen Tests.

Aufgabe 8.3 (4 Punkte)

In dieser Aufgabe geht es um die Generierung von Passwörtern. Einerseits sollen Passwörter sicher sein, d.h. man soll sie nur sehr schwer erraten können. Andererseits soll man sich Passwörter einfach merken können.

Eine häufig verwendete Methode mit diesen Eigenschaften generiert Passwörter entsprechend einer vorgegebenen Struktur aus Ziffern, Satzzeichen und Worten, jeweils mit einer vorgegebenen Anzahl bzw. Länge. In dieser Aufgabe besteht ein *Strukturelement* aus einem der Zeichen `d`, `p` und `w`, jeweils gefolgt von einer positiven ganzen Zahl. So steht `d3` für drei zufällig ausgewählte Ziffern, `p4` steht für 4 zufällig ausgewählte Satzzeichen und `w5` steht für ein zufällig ausgewähltes Wort der Länge 5 aus einer vorgegebenen Liste von Worten. Beachten Sie, dass `p1d0` kein Strukturelement ist, da 0 keine positive ganze Zahl ist.

Beispiel: Der Strukturstring `w4p2d1w5` beschreibt alle Passwörter, die sich in dieser Reihenfolge wie folgt zusammensetzen:

- ein Wort der Länge 4,
- zwei Satzzeichen,
- eine Ziffer,
- ein Wort der Länge 5.

Hier sind einige Passwörter mit dieser Struktur dargestellt. Dabei wird die Wortliste mit insgesamt 23 374 Worten aus dem Material dieser Aufgabe zu Grunde gelegt.

```
meet!‘7rosso
dory]\6clish
sump/^6ethyl
fate+?9wrung
tomb.:9ginny
```

In dieser Aufgabe sollen zunächst vier Funktionen implementiert werden, die in einer Aufgabe des nächsten Übungsblatts in einem lauffähigen Programm verwendet werden sollen. Implementieren Sie in der Datei `pwgen_functions.py` die folgenden Funktionen:

- Die Funktion `structure_string_is_valid(struct_str)` liefert mit einer `return`-Anweisung genau dann `True` zurück, wenn der Strukturstring `struct_str` syntaktisch korrekt geformt ist, d.h. aus einem oder mehreren Strukturelementen (siehe oben) besteht. Verwenden Sie in Ihrer Implementierung einen regulären Ausdruck und die passende Methode, um zu verifizieren, dass der Strukturstring zum reguläre Ausdruck passt. Bedenken Sie dabei, dass der Ausdruck mit `^` und `$` verankert werden muss.
- Die Funktion `structure_elements_list(struct_str)` zerlegt die einzelnen Strukturelemente und liefert mit einer `return`-Anweisung eine Liste von Paaren. Jedes Paar besteht aus einem String mit jeweils genau einem der drei genannten Zeichen und der entsprechenden ganzen Zahl.

  Beispiel: Für den Strukturstring `w4p2d1w5` liefert die Funktion die Liste, die aus den folgenden Elementen besteht: `('w',4), ('p',2), ('d',1), ('w',5).`

  Verwenden Sie zur Implementierung dieser Funktion einen regulären Ausdruck (RE) für ein einzelnes Strukturelement und die passende Funktion aus dem Modul `re`, um die Treffer des RE im Strukturstring aufzuzählen. Aus den Treffern generieren Sie dann die Paare, wie oben

beispielhaft gezeigt. Beachten Sie, dass das zweite Element eines zu generierenden Paares eine ganze Zahl ist, so dass der extrahierte String mit `int` konvertiert werden muss.

- Die Funktion `randstring(alphabet,n)` liefert mit einer `return`-Anweisung einen zufälligen String der Länge `n` über dem Alphabet `alphabet` (repräsentiert als String). Die einzelnen Zeichen des zufälligen Strings sollen mit der gleichen Wahrscheinlichkeit unabhängig voneinander generiert werden. Verwenden Sie dazu die Methode `randint` aus dem Modul `random`, siehe auch `python-slides.pdf`, Abschnitt *Abstract data types and classes*, frame 14-15. Ein zufälliges Zeichen aus `alphabet` berechnet man durch den Ausdruck

  ```
  alphabet[random.randint(0,alpha_size-1)]
  ```

  wobei `alpha_size` die Anzahl der Zeichen in `alphabet` ist. Um einen zufälligen String der Länge `n` zu erhalten, muss man diesen Ausdruck `n` mal auswerten und die gelieferten Zeichen in einem String zusammenfassen.
- Die Funktion `word_dict_get()` öffnet die Datei `wordlist.txt` und liest die darin enthaltenen Worte zeilenweise ein. Mit einer `return`-Anweisung wird ein Dictionary `wd` zurückgeliefert, so dass `wd[m]` die Liste der Worte der Länge `m` aus dieser Datei ist. Falls es in `wordlist.txt` kein Wort einer Länge `m` gibt, dann ist `m` kein Schlüssel in `wd`.

Für diese Funktionen sind in der Datei `pwgen_functions_unit_test.py` sogenannte Unit-Tests implementiert. Durch `make test` werden diese Unit-Tests für Ihre Implementierung der vier Funktionen ausgeführt.

Teilaufgabe: Beschreiben Sie in wenigen Sätzen in der Datei `unittest_doc.txt`, woraus diese Unit-Tests bestehen. Hierzu müssen Sie ggf. recherchieren, was z.B. `assertEqual` bedeutet. Nachdem Sie die Funktionen implementiert haben, verifizieren Sie durch `make test` die Korrektheit Ihrer Implementierung.

Punkteverteilung:

- jeweils 0.5 Punkte für die vier Funktionen
- 1 Punkt für die Beschreibung der Unit-Tests
- 1 Punkt für alle bestandenen Unit-Tests

Aufgabe 8.4 (3 Punkte)

In dieser Aufgabe geht es um das Problem der vollständigen Zerlegung einer positiven ganzen Zahl in Summanden.

Sei n eine positive ganze Zahl und S eine Menge positiver ganzer Zahlen. Eine additive Zerlegung von n bzgl. S ist eine Liste $[s_1, s_2, \ldots, s_k]$ von k Zahlen aus S, mit $k \geq 1$, so dass gilt:

- $s_i \leq s_{i+1}$ für alle i, $1 \leq i \leq k-1$
- $n = \sum_{i=1}^{k} s_i$

D.h. alle Zahlen der Zerlegung stammen aus S und die Liste enthält mindestens ein Element. In der Liste dürfen Zahlen mehrfach vorkommen.

Hier ist eine Tabelle mit additiven Zerlegungen L für gegebene n und S. Falls es keine additive

Zerlegung gibt, ist nichts angegeben.

n	S	L
11	$\{2, 3, 5\}$	$[2, 2, 2, 2, 3]$
32	$\{7, 11, 13\}$	$[7, 7, 7, 11]$
38	$\{7, 11, 13\}$	$[7, 7, 11, 13]$
45	$\{8, 9\}$	$[9, 9, 9, 9, 9]$
47	$\{11, 12, 13, 14\}$	$[11, 12, 12, 12]$
47	$\{13, 14\}$	

Die Aufgabe besteht nun darin zu entscheiden, ob es für gegebene n und S eine additive Zerlegung L von n bzgl. S gibt. Die Konstruktion einer solchen Zerlegung ist Gegenstand folgender Übungsblätter.

Hinweis: Um zu Testen, ob es eine additive Zerlegung gibt, probiert man nacheinander die Zahlen t aus `terms_of_sum` (der Listenrepräsentation von S) aus, indem man t von dem aktuellen Restwert `remain` subtrahiert, falls $t \leq$`remain` ist. Dann löst man rekursiv für die Reste ein kleineres Teilproblem der additiven Zerlegung. Vergessen Sie nicht den Basisfall dieser rekursiven Lösung zu implementieren.

Spielen Sie diesen Ansatz anhand einiger Beispiel durch und überlegen Sie, wie man ihn in Python3 umsetzen könnte.

In einer Datei `splitnumber_exists.py` muss eine rekursive Funktion `splitnumber_exists (number,terms_of_sum)` implementiert werden. Dabei ist `number` die zu zerlegende Zahl und `terms_of_sum` die oben genannte Listenrepräsentation von S. Diese Funktion muss mit einer `return`-Anweisung den Wert `True` liefern, wenn eine additive Zerlegung von `number` bzgl. der Zahlen in `terms_of_sum)` existiert. Sonst muss sie `False` liefern.

Im Material zu dieser Übung finden Sie eine Python3-Datei `splitnumber_exists_unit_test.py`, die Ihr Modul `splitnumber_exists.py` importiert und Ihre Funktion aufruft. Sie müssen sich also in dieser Aufgabe nicht um die Behandlung von Benutzerfehlern kümmern. Durch `make test` werden Tests durchgeführt, die verifizieren, dass Ihre Funktion für die zahlreichen Testfälle die richtigen Ergebnisse liefert.

Punkteverteilung:

- 2 Punkte für korrekte Implementierung von `splitnumber_exists`
- 1 Punkt für erfolgreiche Tests

Bitte die Lösungen zu diesen Aufgaben bis zum 19.12.2022 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken.