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

**Programmierung für Naturwissenschaften 1**
**Wintersemester 2022/2023**
**Übungen zur Vorlesung: Ausgabe am 23.11.2022**

Hinweis: In den heutigen Übungen soll ein fachdidaktisches Experiment durchgeführt werden. Dazu wird Dimitri Eckert von der Arbeitsgruppe für Fachdidaktik der Ingenieurswissenschaften an der TU-Hamburg einige kleinere Aufgaben zum Programmverständnis stellen. Dafür sind 15 Minuten vorgesehen. Die Ergebnisse werden wissenschaftlich ausgewertet.

Beachten Sie auch den Hinweis der Woche am Ende des Übungsblatts.

# Selbsttest-Aufgaben zum Thema Kontrollstrukturen, Listen, Strings und reguläre Ausdrücke in Python

Um ihre eigenen Fähigkeiten in der Python-Programmierung zu testen, lösen Sie bitte selbständig die folgenden Aufgaben. Nutzen Sie zunächst 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. Nutzen Sie die ersten 15 Minuten der Übung für diese Aufgaben.

1. Geben Sie bitte an, was das folgende Programmstück ausgibt.

```
n = 0
for i in range(5):
  n = n + i + n
print('n={}'.format(n))
```

2. Geben Sie bitte an, was das folgende Programmstück ausgibt.

```
idx = 0
add_up = 0
for num_list in [[1,2,3],[4,5],[6,7,8],[9]]:
  add_up += num_list[idx]
  idx = (idx + 1) % 3
print('idx={},add_up={}'.format(idx,add_up))
```

3. Geben Sie bitte an, was das folgende Programmstück ausgibt.

```
numbers = [2431130135, 7, 6959708, 7238, 227096408]
fms = '"{{:>{}}}"'.format(1 + int(math.log10(max(numbers))))
for n in numbers:
  print(fms.format(n))
```

4. Geben Sie bitte an, was das folgende Programmstück ausgibt. Beachten Sie, dass `list`, angewendet auf einen String, die Liste der Zeichen des Strings liefert.

```
ls2 = [['a','b','c'],'abc']
print(''.join(ls2[0]) == ls2[1] and ls2[0] == list(ls2[1]))
```

5. Geben Sie bitte an, was das folgende Programmstück ausgibt. Erläutern Sie die Bedeutung der verschiedenen geschweiften Klammern in der Wertzuweisung an `fms`.

```
stars = '*'
for w in range(5,-1,-1):
  fms = '{{:>{}s}}'.format(w + len(stars))
  print(fms.format(stars))
  stars = stars + '**'
```

6. Geben Sie einen regulären Ausdruck an, der auf Zeilen der folgenden Form passt.

```
BASE COUNT      396 a    723 c    677 g    464 t
```

Der erste Buchstabe des Schlüsselwortes `BASE COUNT` steht am Zeilenanfang und der letzte Buchstabe `t` am Zeilenende. Vor den vier Buchstaben, die in dieser Reihenfolge angegeben werden, stehen jeweils nicht negative ganze Zahlen. Die Trennung von Schlüsselworten, Zahlen und Buchstaben erfolgt durch beliebig viele, aber mindestens ein White Space.

7. Geben Sie einen regulären Ausdruck an, der zu Strings mit Datumsangaben der Form

```
Tag. Monat Jahr
```

passt. Dabei gilt:

- Der Tag wird durch ein oder zwei Ziffern angegeben.
- Der Monat ist ein Wort, das mit einem Großbuchstaben beginnt.
- Das Jahr ist eine Zahl, die aus mindestens einer und höchstens vier Ziffern besteht.

Beispiele:

```
25. Jan    1954
3.   July 1952
31.Februar     2000
32. May 100
```

Beachten Sie, dass direkt auf den Tag ein Punkt folgt und dass die drei Teile durch beliebig viele White-Space getrennt werden. Fügen Sie in Ihren regulären Ausdruck Gruppenklammern für die drei Teile eines Datums ein. Sie brauchen in Ihrem regulären Ausdruck bei der Angaben für die Tage nicht die Wertebereiche für den entsprechenden Monat überprüfen. Dieses wird durch die beiden letzten Beispieldaten angedeutet. Ebenso muss der zweite Teil nicht zwingend einen Monat in einer bestimmten Sprache bezeichnen.

8. Sei `r` eine Variable, die eine positive ganze Zahl speichert. Der Wert von `r` ist hier nicht genau angegeben, kann aber beliebig groß sein. Geben Sie eine möglichst kleine obere Schranke für den Wert von `s`, der im folgenden Programm durch die `print`-Anweisung ausgegeben wird, an. Eine obere Schranke ist ein Wert, der nicht erreicht oder überschritten wird.

```
s, t = (0,1)
for i in range(r):
  s += t
  t = t/2
print(s)
```

**Aufgabe 5.1** (8 Punkte)

Sie sind als studentische Hilfskraft im Institut für Zeitgeschichte beschäftigt. Dort hat man eine umfangreiche Sammlung von Podcasts zu Ereignissen u.a. aus den Bereichen Kultur, Politik, Wissenschaft und Wirtschaft gesammelt. Die Titel dieser Podcasts wurden von verschiedenen Mitarbeitern erfasst. Dabei wurde leider kein einheitliches Format für die Angabe des Zeitpunktes eines Ereignisses verwendet. Zudem war bei einigen Ereignissen kein Zeitpunkt mit einer Jahreszahl bekannt. Zusätzlich hat ein Mitarbeiter einige belanglose Aussagen und Gleichungen wahllos in die Datei eingefügt.

Die Sammlung der Titel findet man in einer Datei `events.txt`. In jeder Zeile (abgesehen von den belanglosen Aussagen) wird ein Ereignis beschrieben. Es wurden bereits große Teile eines Python-Programms entwickelt, um eine Datei mit dem beschriebenen Format und insbesondere die Zeitangaben der Ereignisse, falls sie angegeben sind, zu extrahieren. Es fehlt lediglich die Definition von vier regulären Ausdrücken, für die man Sie als Expertin angestellt hat. Ihre Aufgabe ist es also, diese REs zu entwickeln und in das Programm einzufügen. Jeder Ausdruck soll zu einem der vier verschiedenen Formate von Zeitpunkten, die jeweils mindestens eine Jahreszahl enthalten, passen. Damit das Programm in Zukunft weiter entwickelt werden kann, soll es durch Antworten auf detaillierte Fragen dokumentiert werden.

Benennen Sie zunächst die Datei `collect_events_template.py` in `collect_events.py` um. Für die Entwicklung der regulären Ausdrücke steht ein kleiner repräsentativer Testdatensatz in der Datei `events_first20.txt` zur Verfügung. Dort sind Zeilen mit allen vier Formaten sowie Zeilen mit den genannten Ausnahmen vorhanden.

Verschaffen Sie sich zunächst einen Überblick über das Programm in der Datei `collect_events.py` und beantworten Sie die darin gestellten Fragen in einer Datei `antworten.txt`. Aus den Antworten können Sie wichtige Eigenschaften der zu entwickelnden regulären Ausdrücke bzgl. der Gruppenklammern ableiten.

Schauen Sie sich dann die Formate der Zeitpunktangaben im Testdatensatz an und entwickeln Sie passende reguläre Ausdrücke, die jeweils alle entweder am Ende oder am Anfang einer Zeile verankert werden müssen. Beachten Sie dabei Folgendes:

1. bestimmte Zeichen, wie der Punkt oder eine runde Klammer, haben eine spezifische Bedeutung in einem regulären Ausdruck haben, so dass man ein \ voranstellen muss, um das Zeichen selbst zu spezifizieren.
2. Wenn man in einem Formatstring den Platzhalter `{}` verwendet, aber auch geschweifte Klammern platzieren möchte, verwendet man zwei geschweifte Klammern direkt hintereinander.

Priorisieren Sie die regulären Ausdrücke entsprechend der Anzahl der Zeilen, in denen sie auf eine Zeitpunktangabe passen. Die REs müssen in einer Liste definiert werden. Hinweise zu Benennungen finden Sie in der Python-Datei.

Die Verteilung der Häufigkeit der Treffer der einzelnen REs finden Sie in den beiden Dateien mit der Endung `.tsv`. Durch `make test_small` und `test_large` können Sie einzelne Tests durchführen. Der Gesamttest erfolgt durch `make test`.

Punktevergabe:

- je 1 Punkt für einen korrekten regulären Ausdruck.
- $0.5 + 1 + 0.5 + 1 + 0.5 + 0.5 = 4$ Punkte für die korrekten und vollständigen Antworten zum Programm.

**Aufgabe 5.2** (4 Punkte)

In der Vorlesung wurde gezeigt, wie man mit Hilfe von `matplotlib.pyplot` eine Funktion plottet. Dabei wurde mit `ax.plot` ein Line-Plot erzeugt, bei dem benachbarte Punkte auf einer Linie miteinander verbunden werden. Man kann mit `ax.scatter` auch einzelne Punkte zeichnen, die nicht verbunden werden. Dazu muss man wie bei `ax.plot` zwei Listen der gleichen Länge übergeben, eine für die $X$-Koordinaten und eine für die $Y$-Koordinaten der Punkte. Anstatt die Punkte durch Anwendung einer mathematischen Funktion zu erzeugen (wie in der Vorlesung), kann man die Punkte auch aus einer Datei einlesen.

Schreiben Sie ein Python-Programm `plot_coords.py`, das über `sys.argv` genau einen Dateinamen als Argument erhält. Ihr Programm soll die entsprechende Datei öffnen und zeilenweise einlesen. Sie können davon ausgehen, dass in jeder Zeile der Datei genau zwei durch einen Tabulator getrennte nicht negative ganze Zahlen stehen. Die erste Zahl ist jeweils die $X$-Koordinate und die zweite die $Y$-Koordinate eines Punktes. Sie können mit Hilfe von `split` aus der aktuellen Zeile eine Liste von zwei Strings mit den Koordinaten der Punkte erzeugen, siehe *Methods on Strings* in `synopsis.pdf`. Jeder einzelne dieser beiden Strings besteht nur aus Ziffern. Der Ausdruck `int(s)` liefert für einen String `s`, der nur aus Ziffern besteht, die entsprechende ganze Zahl. Sie müssen also auf beide Strings die Funktion `int` anwenden. Die beiden so gewonnenen Zahlen hängen Sie mit `append` an die Liste `x_list` der $X$-Koordinaten bzw. `y_list` der $Y$-Koordinaten.

Nachdem alle Zeilen eingelesen wurden, sind `x_list` und `y_list` komplett und man erzeugt mit `plt.subplots` ein `Figure`- und `Axes`-Objekt, so wie im Beispiel aus der Vorlesung. Die Verwendung von matplotlib mit globalen Objekten, wie Sie häufig auf Internetseiten finden, ist nicht zulässig.

Wählen Sie als Breite und Höhe der Abbildung 3 bzw. 2 Einheiten. Mit dem Aufruf von `ax.set_xticks([])` unterdrücken Sie die Ausgabe von Ticks auf der $X$-Achse. Entsprechendes gilt für die $Y$-Achse. Durch `ax.spines['right'].set_visible(False)` werden Spines am rechten Rand des Axes-Objekts unterdrückt. Entsprechendes gilt für den linken Rand (`'left'`), den oberen Rand (`'top'`) und den unteren Rand (`'bottom'`). Das Zeichnen erfolgt mit `ax.scatter` (siehe oben). Verwenden Sie also als Argumente für `ax.scatter` neben `x_list` und `y_list` (das sind die ersten beiden Argumente) noch den Parameter `s=10` für die Größe der Punkte. Durch `color='black'` werden die Punkte des Scatter-Plots in Schwarz gezeichnet. Als Letztes rufen Sie `fig.savefig('scatter_plot.pdf')` auf, um Ihren Plot auf der Festplatte zu speichern. Vergessen Sie nicht den Magic-String am Anfang Ihres Programms und das Setzen der Ausführungsrechte.

Durch den Aufruf von `make test` wird Ihr Programm mit der Datei `coords.tsv` aufgerufen und es wird geprüft, ob die Datei mit dem oben genannten Namen entstanden ist. Schauen Sie sich die Datei mit einem PDF-Viewer an. Verändern Sie Ihr Programm so, dass das Motiv in einer *passenden* Farbe geplottet wird. Was hier passend ist, dass überlassen wir Ihrem persönlichen Urteil.

Falls auf Ihrem Rechner Python3 ohne `matplotlib` installiert ist, können Sie dieses Modul im Terminal nachinstallieren und zwar durch

```
$ pip3 install matplotlib
```

oder

```
$ pip install matplotlib
```

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Lists*, frame 1-15

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Plotting data using matplotlib*, frame 2-8

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Flow of control*, frame 26-34

Hinweise zur Lösung: `python-slides.pdf`, Abschnitt *Regular expressions*, frame 3, 4, 17, 18

Punktevergabe:

- 2 Punkte für das Einlesen der Koordinaten in zwei Listen
- 2 Punkte für die Erstellung des Scatterplots mit den genannten Befehlen, also insbesondere auch der Verwendung von `fig` und `ax` und einer angemessenen Farbe.

**Bitte die Lösungen zu diesen Aufgaben bis zum 28.11.2022 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken.**

# Hinweis der Woche

Die Datei `synopsis.xml` im Verzeichnis `data` des PfN1-Repositories enthält Informationen zu Python-Klassen und ihren Methoden, entsprechend der Darstellung in diesem Modul. Um Information in dieser Datei zu suchen und formatiert auszugeben, steht Ihnen das Programm `pysearch.py` im entsprechenden Verzeichnis `bin` zur Verfügung. Wenn Sie, wie in den Hinweisen zu Übungsblatt 4 dafür gesorgt haben, dass diese Verzeichnis in Ihrer Pfadliste ist, dann können Sie `pysearch.py` aufrufen und zwar mit der Angabe eines Suchstrings, wie in diesem Beispiel:

```
$ pysearch.py rstrip
rstrip
    s.rstrip()
    remove trailing white spaces from string
$ pysearch.py -d rstrip
rstrip
    s.rstrip()
    l"osche Leerzeichen am Ende von String
```

Sie sehen also, dass bei der Verwendung der Option `-d` eine Beschreibung in Deutsch ausgegeben wird. Wenn es mehrere zum Suchstring passende Einträge gibt, werden alle aufgelistet.

Da Python-Programme zum Teil aus bereits bekannten Programmstücken bestehen, ist `pysearch.py` sehr hilfreich und erspart meist die Suche im Skript. Zudem enthalten die mit copy-paste extrahierten Programmstücke aus der Ausgabe von `pysearch.py` keine Formatierungsartefakte, wie sie bei der Extraktion aus PDF-Dateien auftreten.