Probeklausur
Informatische Werkzeuge in den Geistes- und Sozialwissenschaften I

16. Januar 2018

The „Lösungen“ der Aufgaben in diesem Dokument sollen den Studierenden als Anfangspunkt für die Beantwortung der Aufgaben dienen. Trotz aller Bemühungen kann es zu unvollständigen oder sogar Fehlern in den „Lösungen“ kommen. Da die Korrektur und Benotung der gestellten Aufgaben niemals lediglich auf einen „Vergleich mit der Musterlösung“ hinausläuft, ist dies auch nicht so schlimm. In jedem Fall sollten die Studierenden die Lösungen nachvollziehen und im Prinzip mittels des Lehrstoffs selbst verifizieren können.

Sollten Sie „Lösungen“ finden, die Sie nicht verstehen oder sogar für fehlerhaft halten diskutieren Sie diese am besten mit den Tutoren oder auf dem Kursforum und benachrichtigen Sie die Lehrenden.

1 Grundlagen & Begrifflichkeiten

Aufgabe 1.1 Erklären Sie die Begriffe *Bit*, *Byte*, *Kilobyte* und *Mebibyte*. Erläuteren Sie außerdem kurz die Beziehungen, die diese Begriffe untereinander haben. — 10 Pkt, 5 min

Lösung:Ein Bit ist die fundamentale Einheit für Information. Bits können entweder den Wert 1 oder 0 annehmen.

Ein Byte sind 8 Bit. Ein Kilobyte sind (Nach SI-Präfix-Nomenklatur) 1000 Byte. Manchmal werden auch 1024 Byte als ein “Kilobyte” bezeichnet. Um hier Präzision in der Sprache zu erlauben wurde der Begriff "Kibibyte"für 1024 Bytes eingeführt. Ein Mebibyte sind 1024 Kibibyte.

Aufgabe 1.2 Erklären Sie kurz die Begriffe *Algorithmus* und *Funktion* und machen Sie den Unterschied zwischen beiden klar. — 10 Pkt, 5 min

Lösung:Eine Funktion ist ein mathematischer Begriff. Funktionen weisen allen Werten, für die sie definiert sind, einen anderen Wert zu. Zum Beispiel weißt die Funktion everse jeder Liste ihre Umkehrung zu.

Ein Algorithmus ist eine Reihe von Anweisungen um ein bestimmtes Problem oder eine Klasse von Problemen zu lösen. Algorithmen können sowohl in Programmiersprachen als auch in natürlicher Sprache (z.B. Kochrezepte) formuliert sein.

Ein Algorithmus ist nicht das gleiche wie eine Funktion. Es gibt mehrere Algorithmen, die die gleiche Funktion berechnen (eventuell mehr oder weniger effizient, aber gleich korrekt) aber jeder Algorithmus berechnet immer nur (maximal) eine Funktion.

Aufgabe 1.3 Erläutern Sie kurz, was es mit der python-Funktion input() auf sich hat. Ist es möglich, Argumente an diese Funktion zu übergeben? Wenn ja: was passiert mit diesen Argumenten? Wenn nein: was passiert, wenn jemand es trotzdem tut? Worauf ist zu achten, wenn Zahlen eingelesen werden sollen? 10 Pkt 5 min

---

Lösung:Mit Hilfe der input()-Funktion ist es möglich, zur Laufzeit des Programms eine Eingabe von den Benutzer'Innen abzufragen (zum Beispiel ob sie ein Haustier haben). Die Eingabe geschieht über das Kommandozeileninterface und wird als String zurück gegeben.

Es ist möglich, Strings als Argument an input() zu übergeben, diese werden dann vor dem Eingabeprompt geprintet.

---

2 Digitale Dokumente

Aufgabe 2.1 (HTML Ping Pong)

Geben Sie den Quellcode für zwei HTML-Seiten (`ping.html` und `pong.html`) an. 20 Pkt

Beide Seiten sollten eine Überschrift, einen kurzen Text und einen Knopf (nicht einfach nur einen Link) enthalten. Beim Klicken des Knopfes soll auf die jeweils andere Seite weiter geleitet werden, so dass durch wiederholtes Klicken zwischen den Seiten hin und her gesprungen werden kann. 10 min

---

Lösung:Der Trick ist, HTML Formulare zu verwenden, und die jeweils andere Seite im action Attribut als Ziel anzugeben.

```
<html>
  <head>
    <title>Ping!</title>
  </head>
  <body>
    <h1>PING!</h1>
    <p>The next logical step would be to click on the Ping! button...</p>
    <form action="pong.html">
      <input type="submit" value="Ping!"/>
    </form>
  </body>
</html>
```

```
<html>
  <head>
    <title>Pong!</title>
  </head>
  <body>
    <h1>PONG!</h1>
    <p>The next logical step would be to click on the Pong! button...</p>
    <form action="ping.html">
```

```
      <input type="submit" value="Pong!"/>
    </form>
  </body>
</html>
```

Aufgabe 2.2 (Cascading Style Sheets)

Geben Sie ein Beispiel für zwei (unterschiedliche) *korrekte* und zwei (underschiedliche) *inkorrekte* CSS-Regeln. 10 Pkt 5 min

Erklären Sie für die letzten beiden Regeln, warum genau diese kein korrektes CSS sind und was verändert werden müsste, damit sie das wären.

Lösung:Beispiele für korrekte CSS-Regeln:

- h1 { text-align: center; }
- body { background-color: #FF9900; }

Beispiele für inorrekte CSS-Regeln:

- p { font-family: verdana font-size: 14pt }
  (Keine Trennung durch Semikola)
- h1 [text-align:]
  (Falsche Klammern, kein Wert zugeordnet)

3 Reguläre Ausdrücke

Aufgabe 3.1 (Reguläre Ausdrücke I)

Geben Sie einen regulären Ausdruck an, der *genau* folgende Strings komplett matcht (also alle diese und keine anderen): 10 Pkt 5 min

cat4, hat4, mat4, bat4, apollo

Lösung:Eine mögliche Lösung wäre der reguläre Ausdruck `^[chmb]at4|apollo$`.

Da wir die RegEx-Abkürzungen für "Anfang des Strings" und "Ende des Strings" (^ und $)verwenden, gehen wir sicher, dass keine anderen Strings außer der explizit aufgeführten (wie z.B. My hat4sunshine is great!) matchen. Der |-Operator erlaubt uns, auch auf den String apollo zu matchen, der das sonst gemeinsame Muster bricht.

Aufgabe 3.2 (Reguläre Ausdrücke II)

Finden Sie einen regulären Ausdruck (Regular Expression), in der üblichen Notation, der gegen alle unten aufgelisteten Positivbeispiele matcht, aber gegen keins der Negativbeispiele. 20 Pkt 10 min

Positivbeispiele:	*Negativbeispiele:*
dogfood	crooked
barefoot	forest
foolish	Atlas
afoot	unfold
footage	palazzi

Hinweis: Die Positivbeispiele müssen hier nicht *komplett* gematcht werden. D.h. Ihr regulärer Ausdruck muss nicht alle Positivbeispiele komplett beschreiben, aber er muss in jedem gefunden werden (Bsp: `"[cm]a"` matcht auf "cat" und "mat", aber nicht auf "hat" oder "golf").

Lösung:Der simpelste (wenn auch nicht der einzige) reguläre Ausdruck, der in allen Positivbeispielen gefunden wird, aber in keinem Negativbeispiel, ist der RegEx `"foo"`.

Komplexere Ausdrücke sind möglich, aber nicht notwendig.

4 Programmieren in Python

Aufgabe 4.1 (Listen von Listen) 30 Pkt 15 min

Schreiben Sie eine python-Funktion longestString, die aus einer *Liste von Listen von Strings* den längsten String findet und zurück gibt.

Beispiel: Angewendet auf die Liste `[["sloth","elephant","cat"],["singing"],["banana","kiwi"]]` sollte Ihre Funktion `"elephant"` zurück geben.

Diese Liste von Listen wird Ihrer Funktion als einziges Argument übergeben werden.

Sie können für diese Aufgabe annehmen, dass alle Längen der Strings in den Listen nur einmal vorkommen (d.h. es gibt keine zwei Strings, die gleich lang sind).

Lösung:Hier ist eine mögliche Lösung mit zwei verschachtelten Schleifen, die über alle Elemente aller Listen iteriert.

```
def longestString(listoflists):
    champion = ""

    # Zwei geschachtelte For-Schleifen
    for liste in listoflists:
        for element in liste:
            if len(element) > len(champion):
                champion = element

    return champion

# Code zum Testen der Funktion (nicht gefragt in der Aufgabe).
testList = [["sloth","elephant","cat"],["singing"],["banana","kiwi"]]
print(longestString(testList)) # Prints "elephant"
```