Name: Vorname:

Geburtsdatum: Matrikelnummer:

Platz:

Klausur
Informatische Werkzeuge in den Geistes- und Sozialwissenschaften 1

15. Februar 2024

	To be used for grading, do not write here												
prob.	1.1	1.2	1.3	1.4	2.1	2.2	3.1	3.2	3.3	4.1	4.2	Sum	grade
total	4	4	4	2	5	4	6	6	12	5	8	60	
reached													

Die „Lösungen“ der Aufgaben in diesem Dokument sollen den Studierenden als Anfangspunkt für die Beantwortung der Aufgaben dienen. Trotz aller Bemühungen kann es zu Unvollständigkeiten 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 Tutor*innen oder auf dem Kursforum und benachrichtigen Sie die Lehrenden; wir werden sie dann gegebenenfalls baldigst korrigieren.

1 Grundlagen und Verständnis

Aufgabe 1.1 (URIs und URNs und URLs) 4 Punkte
4 Min

Was ist der Unterschied zwischen einer *URI*, einer *URL* und einem *URN*? Geben Sie außerdem an, wie diese Konzepte sich zueinander verhalten.

Lösung: Die Abkürzung *URI* steht für *Uniform Resource Identifier*, hier geht es also um die Identifizierung von Ressourcen, anders als bei *URL* (*Uniform Resource Locator*) (wo die Lokalisierung / Ansteuerung im Vordergrund steht) und *URN* (*Uniform Resource Names*).

URIs sind eine Obermenge von *URL* und *URNs*. Jede *URL* und jeder *URN* ist auch eine *URI* aber nicht jede *URI* ist eine *URL* (es könnte auch ein *URN* sein). Alle Adressen von Websites sind *URL* (und *URI*), ISBN-Nummern sind jedoch nur *URN*, keine *URL*.

Korrektur:

- 2 Punkte für die korrekte Beschreibung von URIs als Obermenge von URLs und URNs.
- Je einen Punkt für Erwähnung, dass URLs zur Ressource führen und URNs nicht.

Aufgabe 1.2 (11 in verchiedenen Zahlensystemen) 4 Punkte
4 Min

In Stellenwert-Zahlensystemen (positional number systems) hängt der Wert einer Zahl mit mehr als einer Stelle davon ab, in welchem System wir uns genau befinden.

Geben Sie jeweils den Wert *im Dezimalsystem* an für die Zahlen 11_2 (Binärsystem, Basis 2), 11_{16} (Hexadezimalsystem, Basis 16), 11_8 (Oktalsystem, Basis 8) und 11_1 (Unärsystem, Basis 1) an.

Lösung:

- $11_2 = 3_{10}$
- $11_{16} = 17_{10}$
- $11_8 = 9_{10}$
- $11_1 = 2_{10}$ (wir akzeptieren auch: "ungültig, weil wir nach dem Schema nur die Ziffer 0 in diesem Zahlensystem haben sollten")

Korrektur: Ein Punkt pro korrekter Zahl.

Aufgabe 1.3 (HTML Requests) 4 Punkte
4 Min

Sowohl HTML `GET` Anfragen wie auch HTML `POST` Anfragen können dazu verwendet werden, einem Server Informationen zu übermitteln. Dennoch wird aus gutem Grund zwischen ihnen klar unterschieden. Nennen Sie den Unterschied zwischen `GET` und `POST` Anfragen. Welche Variante würden Sie für das Übertragen eines Passwortes verwenden? Nennen Sie außerdem noch mindestens eine weitere Sorte von HTML Anfragen.

Lösung: Der zentrale Unterschied zwischen `GET` und `POST` Anfragen besteht darin, dass die enthaltenen Informationen bei einer `GET` Anfrage in der URL eingebettet werden und bei einer `Post` Anfrage im Körper ("Body") der Anfrage übertragen werden.

Für die Übertragung eines Passwortes eignet sich eine `POST` Anfrage, da so das Passwort nicht gut leserlich in der URL stehen muss.

Die anderen gültigen Anfragetypen sind: `HEAD`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, und `PATCH`.

Korrektur:

- Zwei Punkte für Benennung von Informationen in URL/Body.
- Einen Punkt für Angabe, dass Passwörter via `POST` versendet werden sollten.
- Einen weiteren Punkt für einen gültigen Typ HTML Anfrage.

Aufgabe 1.4 (Escaping in Python) 2 Punkte
2 Min

Bestimmte Zeichenketten in Python tragen besondere Bedeutungen. So kann zum Beispiel die Zeichenkette `"\n"` dafür genutz werden, einen Zeilenumbruch auszugeben. Allerdings kommt es manchmal auch vor, dass wir die tatsächliche Zeichenkette und nicht ihre besondere Bedeutung benötigen.

Geben Sie einen (!) Python `print`-Befehl an, der folgenden Text inklusive des Zeilenumbruchs und des nicht-Zeilenumbruches ausgibt:

```
NEWLINE:
NOT A NEWLINE:\n
```

Lösung: Gültige Lösung: `print("NEWLINE:\nNOT A NEWLINE:\\n")`

Korrektur: Einen halben Punkt für korrekte Newline, 1.5 Punkte für korrektes Escaping.

---

2 Reguläre Ausdrücke

Aufgabe 2.1 (RegEx für Polynome) 5 Punkte
5 Min

Ein Polynom ist (im Rahmen dieser Aufgabe) eine Reihe von Summanden, voneinander getrennt durch Rechenzeichen, also entweder + oder -. Alle Summanden haben dabei die Form `ax^b`, wobei `a` und `b` jeweils beliebige natürliche Zahlen (also z.B. 7, 0 oder 314...) sein können. Der Faktor `a` kann allerdings auch entfallen.
Beispiele für Polynome:

1. `5x^2+3x^1+0x^0`
2. `x^4-x^3+2x^4`
3. `x^2`

Geben Sie einen regulären Ausdruck für Polynome wie oben beschrieben an. Beachten Sie dabei, dass in einem gültigen Polynom mindestens ein Summand existieren muss und auf den letzten Summand kein Rechenzeichen folgt.

Lösung: Hier ist eine Lösung:

```
(\d*x\^\d+[+-])*(\d*x\^\d+)
```

Aufgabe 2.2 (Reguläre Ausdrücke für IP-Adressen) 4 Punkte
4 Min

Geben Sie einen regulären Ausdruck an, der gegen IPv4-Adressen matcht. Diese Adressen bestehen aus genau vier Blöcken, die durch Punkte voneinander getrennt sind. Jeder Block enthält zwischen einer und drei Ziffern von 0 bis 9.

Beispiele:

- `127.0.0.1`
- `131.188.6.20`
- `203.000.113.000`

Hinweis: Blöcke in tatsächlichen IPv4-Adressen nehmen nur Werte zwischen 0 und 255 an, nicht jede beliebige Kombination aus drei Ziffern wie z.B. 999. Diesen Fakt ignorieren wir in dieser Aufgabe.

Lösung: Ein solcher Ausdruck wäre zum Beispiel:

```
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
```

Wichtig zu beachten ist, dass der Punkt zwischen den Blöcken mit einem Backslash escapt werden muss. Sonst würden auch Ausdrücke wie der folgende zugelassen, weil der . auf alle Zeichen matcht.

---

3 Digitale Dokumente

Aufgabe 3.1 (CSS Grundlagen) 6 Punkte
6 Min

Geben Sie für jede der folgenden Anforderungen eine valide *CSS* Regel an. Jede Regel kann (wenn notwendig) mehrere Attribut/Wert-Paare enthalten, es sollta aber pro Aufgabe nur ein Selektor verwendet werden.

1. Geben Sie allen Hyperlinks eine durchgängige, 2px breite Border und einen Margin von 20px in alle Richtungen.

   Lösung: `a { border: 2px solid black; margin: 20px 20px 20px 20px; }`

   ---

2. Setzen Sie die Schriftgröße des Elements mit der Id `findme` auf 25pt.

   Lösung: `#findme { font-size: 25pt; }`

   ---

3. Setzen Sie sowohl Breite als auch Höhe aller Elemente auf 150px, die *sowohl* der Klasse `limW` *als auch* der Klasse `limH` angehören.

Lösung: `.limH.limW { width: 150px; height: 150px; }`

Aufgabe 3.2 (Javascript und das DOM) 6 Punkte
6 Min

Gegeben ist folgende HTML-Website:

```
<html>
  <body>
    <ul id="getraenke">
      <li>Kaffee</li>
      <li>Tee</li>
    </ul>

    <script>
      const newNode = document.createElement("li");
      const textNode = document.createTextNode("Wasser");
      newNode.appendChild(textNode);

      const list = document.getElementById("getraenke");
      list.insertBefore(newNode, list.children[0]);
    </script>
  </body>
</html>
```

Beschreiben Sie jeweils kurz:

1. Welche interagiert das Javascript-Snippet mit dem DOM und welche Veränderungen daran führt es durch?
2. Was sehen Nutzer*Innen auf der Website *nachdem* das Snippet bereits ausgeführt wurde?

Lösung:

1. Das Javascript-Snippet erstellt eine neues HTML-`<li></li>`-Tag inklusive Textinhalt (`"Wasser"`), ermittelt das DOM-Listenelement `"getraenke"` via dessen eindeutigen `id`-Attributs und fügt diesem DOM-Element vor dem ersten Kind-Element (also als neues erstes Kind-Element) den neu erstellten Tag hinzu.
2. Folglich sehen Nutzer*Innen nach Ausführung des Snippets eine HTML-Liste mit drei Elementen: Wasser, Kaffee und Tee (in dieser Reihenfolge!)

Korrektur:
Drei Punkte für korrekte Beschreibung der DOM-Interaktionen, einer davon konkret für Erwähnung der Baumstruktur des DOM (z.B. Kind/Elter statt nur „fügt ein Element hinzu“etc.). Zwei Punkte für korrekte Beschreibung des Resultats, mit einem Punkt Abzug für falsche Reihenfolge der Listenelemente.

Aufgabe 3.3 (FizzBuzz Template) 12 Punkte
12 Min

Sie treten einen neuen Job beim Unternehmen `FizzBuzz Fizzy Drinks Inc.` an und Ihre erste Aufgabe ist es, „die Website bunter zu machen“.

Schreiben Sie ein `bottle` (bzw. `STPL`) *Template* das eine (nicht-nummerierte) Liste von Produktnamen anzeigt. Sie wissen, dass das Template eine Variable `products` übergeben bekommt. Die Elemente von `products` sind jeweils Tupel aus Produktnummer (`int`) und Produktname (`str`).

Ist die jeweilige Produktnummer ganzzahlig durch drei teilbar, so soll *Rot* für den Produktnamen verwendet werden (Hintergrundfarbe oder Textfarbe, Sie entscheiden!). Ist sie durch fünf teilbar soll *Grün* verwendet werden. Wenn sie sowohl durch drei als auch durch fünf teilbar ist, soll *Blau* verwendet werden. In allen anderen Fällen soll keine Hervorhebung passieren.

Hinweis:

- Gefragt ist nur das Template, Sie brauchen *keinen* Code für den Server.
- Ordentlich aufgerufen soll ihr Template eine komplette (gültige) HTML-Seite zurück geben, nicht nur die Liste selbst.
- In Abwesenheit eines Style-Sheets können Sie auch die `style` Attribute der `<li></li>` Tags für CSS-Regeln verwenden.

Lösung: Hier ist eine mögliche Lösung:

```
<html>
    <body>
    <ul>
        <%
          for product in products:
            idn = product[0]
            nam = product[1]

            change = False

            if (idn % 3 == 0) and (idn % 5 == 0):
              colour = "blue"
              change = True
            elif (idn % 3 == 0):
              colour = "red"
              change = True
            elif (idn % 5 == 0):
              colour = "green"
              change = True
            end
        %>
                    % if change:
                    <li style="background-color:␣{{colour}}">{{nam}}</li>
                    % else:
                    <li>{{nam}}</li>
                    % end
        % end
    </ul>
    </body>
</html>
```

---

4 Programmieren in Python

Aufgabe 4.1 (Password Reverse-Engineering) 5 Punkte

5 Min

Gegeben ist folgende Python-Funktion:

```
def check_passfile():
    with open('passfile.txt') as pfile:
        lines = pfile.read().splitlines()

    phrase = ""
    for i in range(len(lines)):
        phrase = phrase + lines[i][i]

    check = phrase[::-1]
    return check == "OKAY"
```

Geben Sie einen möglichen Inhalt der Datei `passfile.txt` an, sodass die Funktion `check_passfile()` den Wert `True` zurück gibt.

Lösung: Hier ist eine mögliche Lösung:

```
Y
1A
12K
123O
```

Die Idee ist, dass für jede Zeile *i* (angefangen bei Zeile 0) das Zeichen an Stelle *i* dem String hinzugefügt wird. Damit die Funktion `True` zurück gibt muss der String am Ende "YAKO" (umgedrehtes "OKAY") sein, also braucht Zeile 0 Stelle 0 ein Y, Zeile 1 Stelle 1 ein A usw. Zeichen vor und nach diesen Stellen sind egal, es kommt nur auf die Diagonale an.

Aufgabe 4.2 (Hexadezimal-Wörter) 8 Punkte, 8 Min

Zahlensysteme mit Basen > 10 (wie z.B. das Hexadezimalsystem) benutzen häufig Buchstaben als Ziffern (`A` für 10, `B` für 11, ...). Sie interessiert, ob so auch ganze Wörter Zahlen sein können.

Ihnen liegt eine Datei mit dem Namen `word_list.txt` vor, die häufig benutzte Worte enthält, jeweils eines pro Zeile. Wie viele genau oder welcher Sprache ist hier nicht relevant.

Schreiben Sie ein Programm in Python, welches das längste Wort in der Datei ermittelt (und ausgibt), das gleichzeitig auch eine gültige Zahl im Hexadezimalsysten darstellt (Beispiele: "Bad" oder "ADAC").

Hinweis: Für diese Aufgabe kann z.B. die Methode `re.fullmatch(rgx, wrd)` verwendet werden, welche einen Boolean (also `True` oder `False`) zurück gibt.

Lösung: Hier ist eine mögliche Lösung:

```
import re

# Datei einlesen
with open('word_list.txt') as lfile:
    # So haben wir eine Liste von Worte,
    # allerdings ohne den Zeilenumbruch
    words = lfile.read().splitlines()

champion = ""
for word in words:
    # Ist das Wort ein valider Hexcode?
    # Kein 0-9 weil diese nicht in Worten vorkommen.
    hexa = re.fullmatch("[a-fA-F]+", word)
```

```
    # Ist das Wort laenger als der Champion?
    long = len(word) > len(champion)

    # Wenn beides, haben wir einen neuen Champion!
    if hexa and long:
        champion = word

print(champion)
```

Für Neugierige: Angewendet auf eine Liste der 10000 häufigsten englischen Wörter ist das Ergebnis das Wort “decade”. Im Deutschen ist es tatsächlich “ADAC”, oder “Bad” wenn Abkürzungen ausgeschlossen sein sollen.
