Name:

Geburtsdatum:

Matrikelnummer:

Klausur
Informatische Werkzeuge in den Geistes- und Sozialwissenschaften 1

06. Februar 2020

	Nur zur Korrektur, bitte freilassen!										
Aufgabe	1.1	1.2	2.1	2.2	3.1	3.2	4.1	4.2	4.3	Summe	Note
Möglich	4	4	8	4	5	8	5	10	12	60	
Erreicht											

Klausurnote: Bonuspunkte: Endnote:

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 und Verständnis

Aufgabe 1.1 (HTML und XML)

Erklären Sie kurz sowohl die wichtigsten *Gemeinsamkeiten* von und die wichtigstem *Unterschiede* zwischen HTML und XML. Wie verhalten sich die beiden zueinander? — 4 Pkt, 4 min

Lösung:HTML (HyperText Markup Language) ist eine Markup-Sprache, die spezifisch dafür gedacht ist, die Struktur von Webseiten über das Internet zu kommunizieren.

XML (Extensible Markup Language) ist ebenfalls eine Markup-Sprache (oder genauer: eine Familie von solchen), die allerdings nicht auf ein Anwendungsfeld beschränkt ist. Hier kann man die Elementnamen und Attributnamen frei wählen.

Beide benutzen eine geschachtelte, wohlgeklammerte `<tag>`-Paare um eine baumförmige Dokumentenstruktur zu ermöglichen.

Es gibt eine XML-Kompatible Erweiterung von HTML, diese wird XHMTL genannt.

Aufgabe 1.2 (Bäume)

Was ist ein *Baum* im Kontext von Informatik? Welche Bestandteile haben Bäume? Erklären Sie außerdem kurz die Begriffe *Wurzel* und *Blatt*. — 4 Pkt, 4 min

Lösung:Ein Baum ist eine Datenstruktur, die die hierarchische Anordnung von Objekten (genannt: Knoten) ermöglicht. Jedem Knoten K kann dabei eine Menge von Knoten niedriger in der Hierarchie zugeordnet werden. Diese Menge nennen wir die Kinder von K, wobei K deren Elternteil ist.

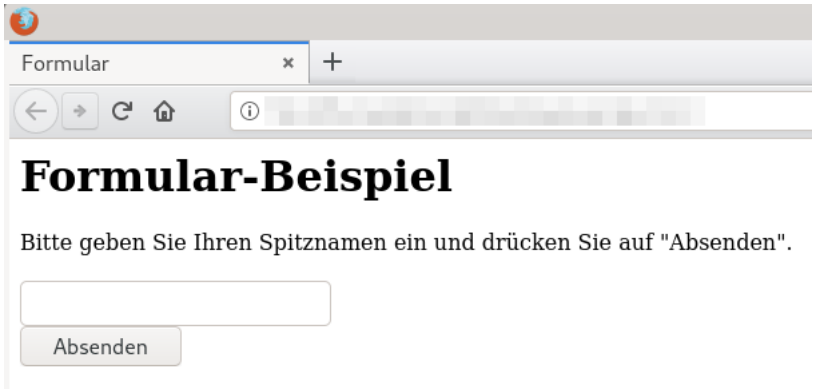
In einem Baum darf kein Knoten mehr als ein Elternteil haben. Die Eltern-Kind-Relation darf außerdem keine Zykel enthalten.

Der Knoten ohne Eltern (pro Baum gibt es nur einen) wird Wurzel genannt. Knoten ohne Kinder sind Blätter.

2 HTML und CSS

Aufgabe 2.1 (HTML-Seite mit Textfeld)

Schreiben Sie eine gültige HTML-Datei, die in einem Browser etwa so dargestellt wird wie in folgendem Bild zu sehen: 8 Pkt, 8 min



Die genaue Art der Überschrift können Sie frei wählen. Das Formularelement muss *keine* besondere Funktionalitäten (wie z.B. eine `page action`) haben. Nur die Elemente sollen alle vorhanden sein.

Lösung:Hier ist die offensichtlichste Lösung. Man beachte das <title> Element. Dies ist im Bild zu sehen und darf nicht vergessen werden. Hier ist eine von vielen Möglichkeiten.

```
<html>
        <head>
                <title>Formular</title>
        </head>
        <body>
                <h1>Formular—Beispiel</h1>
                Bitte geben Sie Ihren Spitznamen ein und dr&uuml;cken Sie auf
                "Absenden".<br><br>
                <form>
                <input type="text" name="nick"><br>
                <input type="submit" value="Absenden">
                </form>
        </body>
<html>
```

Aufgabe 2.2 (CSS für Überschriften)

Geben Sie vier CSS-Regeln an, die folgende Änderungen bewirken, wenn sie in eine HTML-Datei eingebunden werden: 4 Pkt, 4 min

1. Färben Sie alle `<h1>`-Überschriften rot.
2. Zentrieren Sie `<h2>`-Überschriften horizontal mittig auf der Seite.

3. Setzen Sie die Schriftgröße für `<h3>`-Überschriften auf `10pt`.

4. Geben Sie `<h4>`-Überschriften einen `margin` von 35 Pixeln in alle Richtungen.

Lösung:Hier sind die vier Regeln:

```
h1 { color: red; }
h2 { text-align: center; }
h3 { font-size: 10pt; }
h4 { margin: 35px; }
```

---

3 Reguläre Ausdrücke

Aufgabe 3.1 (Regulärer Ausdruck mit Klammerung)
Geben Sie einen regulären Ausdruck an, der auf Ausdrücke matcht, die aus einer Reihe (der Länge 0 oder länger) von beliebig Kleinbuchstaben in genau einem Paar eckiger Klammern (`[]`) bestehen. Dabei sollen ebenfalls keine Vokale (a,e,i,o,u) oder Umlaute (ä,ö,ü,ß) vorkommen. 5 Pkt 5 min

Beispiele: `[]`, `[q]`, `[rhythm]`, `[nrnbrg]`, `[lynx]`, `[fcknzs]`, `[twttr]`

Lösung:Ein solcher Ausdruck wäre zum Beispiel:

```
\[[bcdfghjklmnpqrstvwxyz]*\]
```

Wichtig ist, dass ein paar eckige Klammern escapt werden muss, damit die tatsächlichen eckigen Klammern gematcht werden können.

Aufgabe 3.2 (Regulärer Ausdruck für DOIs)
Um auf digitale Objekte (wie Journals, Berichte, Paper, Artikel, ...) verlässlich verweisen zu können, wurde von der International Organization for Standardization (ISO) im Jahr 2012 ein Standard für *Digital Object Identifiers* (**DOI**) veröffentlicht. 8 Pkt 8 min

Ein **DOI** besteht aus einem Präfix und einem Suffix, die von einem „`/`“ getrennt werden. Das Präfix beginnt (für unsere Zwecke) immer mit dem String „`10.`“ gefolgt von einer ganzen Zahl ≥ 1000. Das Suffix ist beliebig und unterliegt keinerlei Einschränkungen. In der Praxis ist es oft numerisch, der Standard erlaubt allerdings jede (nichtleere) Aneinanderreihung von Zeichen.

In normalen Texten ohne Hyperlinks wird ein **DOI** oft mit dem String „`doi:`“ eingeführt, wie auch in den folgenden Beispielen:

```
doi:10.1000/182
doi:10.20347/WIAS.PREPRINT.2431
doi:10.1016/j.disc.2012.08.018
doi:10.1007/978-3-662-44199-2_5
doi:10.1007/s10817-012-9271-4
```

Geben Sie einen regulären Ausdruck an, der auf textuelle **DOI**-Ausdrücke wie oben beschrieben (also inklusive „`doi:`“) matcht.

Lösung:Ein solcher Ausdruck wäre zum Beispiel:

```
doi:10\.[1-9]\d{3}\d*/.+
```

4 Programmieren in Python

Aufgabe 4.1 (Falsche Route)

Die `bottle`-Route in Abbildung 1 wurde mit der Intention angelegt, eine WebApplication für Addition zur Verfügung zu stellen. So soll zum Beispiel eigentlich an `/plus/37/13` der Wert `50` zurück gegeben werden. Allerdings ist etwas schief gelaufen und stattdessen wird `1337` zurück gegeben. 5 Pkt 5 min

Erklären Sie kurz, welcher Fehler in Abbildung 1 unterlaufen ist und geben Sie eine korrigierte Fassung dieser Route an.

```
@route('/plus/<x>/<y>')
def plus(x,y):
        return y + x
```

Abbildung 1:

Lösung:Das Problem ist, dass die eingegeben Argumente wie Zahlen verwendet werden, obwohl sie in Wirklichkeit Strings sind. Eine korrekte Variante dieser Route wäre zum Beispiel diese:

```
@route('/plus/<x>/<y>')
def plus(x,y):
result = int(x) + int(y)
return str(result)
```

Aufgabe 4.2 (Dateipfade auslesen und Dateien löschen)

Schreiben Sie ein python-Programm, das eine Datei namens `toDelete.txt` öffnet und ausliest. In dieser Datei ist pro Zeile ein Dateipfad notiert, der zu einer Datei führt, die gelöscht werden soll. 10 Pkt 10 min

Ihr Programm soll für alle diese Pfade überprüfen, ob eine Datei mit diesem Pfad tatsächlich existiert. Wenn ja, soll diese Datei gelöscht werden und eine entsprechende Meldung mit **print()** ausgegeben werden. Wenn nicht, dann soll eine entsprechende Fehlermeldung ausgegeben werden.

Hinweis: Die Funktionen os.path.isfile() und os.remove() könnten sich für diese Aufgabe als nützlich erweisen. Denken Sie auch an die nötigen **import** Statements.

Lösung:Hier ist eine mögliche Lösung. Man beachte die Verwendung von .strip(), um überflüssigen Whitespace los zu werden.

```python
import os

# This with-syntax automatically closes the file after use.
with open("toDelete.txt") as delFile:
    for line in delFile:
        # Remove newline at end of line.
        path = line.strip()

        # If corresponding files exists:
        if os.path.isfile(path):
            # Delete file
            os.remove(path)
            print("Deleted",path)
        else:
            print(path, "does not exist!")
```

Aufgabe 4.3 (Bäume mit lxml)

12 Pkt
12 min

Setzen Sie sich genau mit folgendem Schnippsel python-Code auseinander. Es wird die in der Vorlesung besprochene `lxml`-Bibliothek verwendet um iterativ (d.h. in mehreren Durchläufen) eine Baum-Datenstruktur aufzubauen, in der jeder Knoten in seinem .text-Attribut einen String enthält, der einer Zahl entspricht.

```python
from lxml import etree

# Lege die Wurzel des Baumes an
root = etree.Element("node")
root.text = "0"
counter = 1

# Tue genau zwei Mal folgendes:
for _ in range(0,2):

    # Fuer "root" und alle seine Nachfahren tue folgendes:
    for element in root.iter():
        # Wenn der Knoten keine Kinder hat...
        if len(element) == 0:
            # ... fuege diesem Knoten (element) zwei Kinder hinzu ...
            fst_child = etree.SubElement(element, "node")
            snd_child = etree.SubElement(element, "node")

            # ... und gebe allen neuen Kindern den aktuellen counter als text.
            for child in element:
                child.text = str(counter)
                counter = counter + 1
```

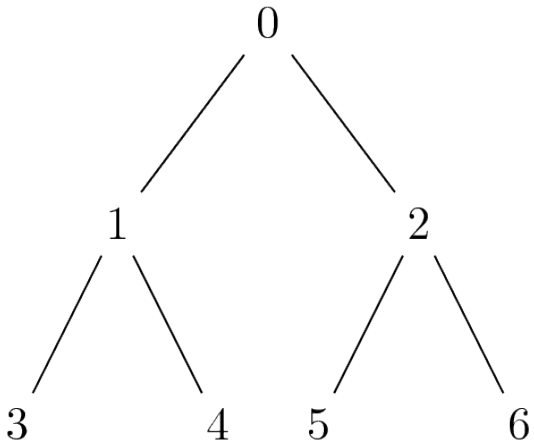
Wird der obige Code ausgeführt, so ist es möglich, den dort generierten Baum mit der Wurzel root wie folgt darzustellen (einmal in XML-Syntax und einmal als Grafik):

```
<node>
        0

        <node>
                1
                <node>3</node>
                <node>4</node>
        </node>

        <node>
                2
                <node>5</node>
                <node>6</node>
        </node>

</node>
```



Eine bildliche Darstellung des links beschriebenen Baumes.

Schreiben Sie nun eine python-Funktion renameNodes, die den Wurzelknoten eines solchen Baums als Argument bekommt. Die Funktion soll alle Knoten in dem Baum besuchen und deren .text-Attribut ersetzen. Dabei soll der neue Wert des Attributs "gerade" sein, wenn der ursprüngliche Wert des Attributs eine gerade Zahl darstellt. Andernfalls soll der neue Wert "ungerade" sein. Bedenken Sie, dass der Wert des .text-Attributs ein String ist, auch wenn er nur Ziffern enthält.

Hinweis: Alle Methoden und Funktionen, die Sie für die Beantwortung dieser Aufgabe benötigen, werden auch im obigen Codeschnippsel verwendet und werden deshalb hier nicht noch einmal aufgezählt.

Lösung:Hier ist eine kommentierte rekursive Lösung dieser Aufgabe:

```
# Funktion definieren
def renameNodes(node):

    # Text in Zahl umwandeln.
    i = int(node.text)

    # Falls gerade...
    if i % 2 == 0:
        # Text-Attribut auf "gerade" setzen.
        node.text = "gerade"
    # ... oder sonst ...
    else:
        node.text = "ungerade"

    # Rekursiver Aufruf auf allen Kindern dieser node.
    for child in node:
        renameNodes(child)
```

Alternativ wäre auch zulässig gewesen, .iter() wie im Beispiel in der Aufgabe zu verwenden:

```
# Funktion definieren
```

```
def renameNodes(start):

    # Tue fuer alle Knoten unter start folgendes:
    for node in start.iter():

        # Text in Zahl umwandeln.
        i = int(node.text)

        # Falls gerade...
        if i % 2 == 0:
            # Text-Attribut auf "gerade" setzen.
            node.text = "gerade"
        # ... oder sonst ...
        else:
            node.text = "ungerade"
```