Name:

Geburtsdatum:

Matrikelnummer:

Klausur
Informatische Werkzeuge in den Geistes- und Sozialwissenschaften 1

17. Februar 2022

	Nur zur Korrektur, bitte freilassen!											
Aufgabe	1.1	1.2	1.3	2.1	3.1	3.2	3.3	4.1	4.2	4.3	Summe	Note
Möglich	2	4	4	8	10	4	6	4	7	11	60	
Erreicht												

Klausurnote: Bonuspunkte: Endnote:

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 (Whitespace)

2 Pkt
2 min

Die Programmiersprache python ist *whitespace-sensitiv*. Erklären Sie kurz (!) das Konzept *Whitespace* und beschreiben Sie zwei Zeichen, die unter den Begriff *Whitespace* fallen.

Lösung: Whitespace ist jedes Zeichen und jede Zeichenfolge, die Platz in einem angezeigten Dokument einnehmen, ohne dabei eine sichtbare Markierung zu verursachen.

Beliebte Whitespace-Zeichen sind zum Beispiel:

```
" "  (Leerzeichen)
"\n" (Zeilenumbruch)
"\t" (Tabulator)
```

Korrektur:

- 1 Punkt für die korrekte Erklärung des Konzeptes
- Je einen halben Punkt für ein korrekt beschriebenes Zeichen (bis zu 1 Punkt).

Aufgabe 1.2 (URIs und URNs und URLs)

4 Pkt
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 URLs (*Uniform Resource Locator*) (wo die Lokalisierung / Ansteuerung im Vordergrund steht) und URNs (*Uniform Resource Names*).

URIs sind eine Obermenge von URLs 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 URLs (und URIs), ISBN-Nummern sind jedoch nur URNs, keine URLs.

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 1.3 (Syntax-Hightlighting) 4 Pkt

In vielen Entwicklungsumgebungen (z.B. in JupyterLab) werden bestimmte Teile von Quellcode farblich hervorgehoben (z.B. in python Schlüsselwörter wie **return** oder **break**). 4 min

Macht dies python zu einer *Plain Text*- oder zu einer *Markup*-Sprache? Begründen Sie Ihre Antwort kurz.

Lösung: python ist eine Plain-Text-Sprache da alles an besonderer Präsentation von der Entwicklungsumgebung kommt und nichts davon im Quelltext selbst beschrieben wird. In einer Markup-Sprache wären die Anweisungen zur Darstellung Teil des Quelltextes.

Korrektur:

- 1 Punkt für eine korrekte Beschreibung des Konzeptes.
- 0.5 Punkte für korrekt beschriebene Zeichen ("Zeilenumbruch" und `"\n"` sind beide zulässig.)

2 Reguläre Ausdrücke

Aufgabe 2.1 (Regulärer Ausdruck für Video-URLs mit Zeitindex) 8 Pkt
Der neue belgische Video-Streaming Service `ToYou.be` möchte eine große Menge Text (die sie im Internet komplett legal gefunden haben) mit einem regulären Ausdruck durchsuchen, um zu erfahren, wie viele von ihren Video-URLs vorkommen. So eine URL besteht für unsere Zwecke aus vier Teilen (in dieser Reihenfolge), und zwar... 8 min

- ... dem String `"http://"` oder `"https://"`
- ... dem String `"www.toyou.be/watch?v="`
- ... genau elf Zeichen für die Id (hier nur Großbuchstaben, Kleinbuchstaben, Ziffern, Unterstrich und Bindestrich)
- ... einer optionalen (!) Zeitangabe. Diese beginnen mit dem String `"&t="`. Danach folgt ein String nach dem Muster `XhYmZs`, wobei `X`, `Y` und `Z` jeweils positive ganze Zahlen (z.B. `14` oder `0`, aber nicht `-100` oder `3.4`) sind.

Hier sehen Sie ein paar Beispiele für diese URLs:

```
http://www.toyou.be/watch?v=dQw4w9WgXcQ
https://www.toyou.be/watch?v=QH2-TGUlwu4
http://www.toyou.be/watch?v=gocwRvLhDf8&t=0h4m57s
https://www.toyou.be/watch?v=K0RWJsZD1kg&t=0h29m53s
https://www.toyou.be/watch?v=NHEaYbDWyQE&t=2h53m59s
```

Geben Sie einen regulären Ausdruck an, der auf Video-URLS wie oben beschrieben matcht. Falls Sie Gruppierungen verwenden, ist es egal ob diese *capturing* oder *non-capturing* sind.

Lösung: Ein solcher Ausdruck wäre zum Beispiel:

```
https?://www\.toyou\.be/watch\?v=[A-Za-z0-9_-]{11}(?:&t=\d+h\d+m\d+s)?
```

Korrektur:

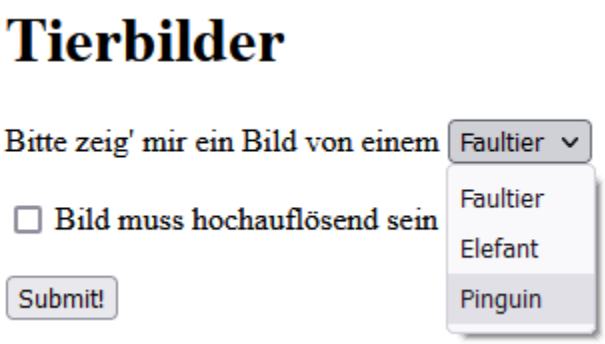
- Je einen Punkt für korrektes Matching der ersten beiden Teile
- Je drei Punkte für korrektes Matching der letzten beiden Teile

Einen Punkt Abzug für Fehler wie z.B. unescapte Punkte oder Fragezeichen. Zwei Punkte Abzug falls Zeitangabe nicht optional.

---

3 Digitale Dokumente

Aufgabe 3.1 (HTML-Seite mit Formular) 10 Pkt
Schreiben Sie eine gültige HTML-Datei, die in einem Browser etwa so dargestellt wird wie in folgendem Bild zu sehen: 10 min



Das Formularelement sollte auf eine weitere Website „display.html“ umleiten, wenn auf den Knopf geklickt wird.

Lösung: Hier ist eine von vielen Möglichkeiten.

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Tierbilder</h1>
        <form action="display.html">
            Bitte zeig' mir ein Bild von einem

            <select>
                <option>Faultier</option>
                <option>Elefant</option>
                <option>Pinguin</option>
            </select>
            <br/><br/>

            <input type="checkbox" id="hq">
            <label for="hq"> Bild muss hochaufloesend sein</label>
            <br/><br/>

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

Korrektur:

- 3 Punkte für das grundlegende HTML-Gerüst (`<html>`, `<body>`, `<h1>`. DOCTYPE ist optional.

- 2 Punkte für ein `<form>`-Element eine korrekte `form action`
- 2 Punkte für ein korrektes `<select>`-Element
- 2 Punkte für ein korrektes `<input type="checkbox»`-Element (statt label kann auch einfach Text verwendet werden)
- 1 Punkt für ein korrektes `<input type=ßubmit»`-Element

Aufgabe 3.2 (CSS Regeln) 4 Pkt
Geben Sie für jede der folgenden Anforderungen eine valide CSS-Regel an: 4 min

1. Geben Sie allen `<p>` und `<h1>`-Elementen mit einer(!) Regel eine rote Schriftfarbe.
2. Geben Sie dem Element mit der ID `"binblau"` eine blaue Hintergrundfarbe.
3. Geben Sie allen `<div>`-Elementen einen Margin von 15px in alle Richtungen.
4. Zentrieren Sie allen Text in Elementem der Klasse `centered` horizontal.

Lösung: Korrekte CSS-Regeln sind:

1. `p, h1 { color : red; }`
2. `#binblau { background—color : blue; }`
3. `div { margin: 15px 15px 15px 15px;}`
4. `.centered { text—align : center; }`

Korrektur:

- 1 Punkt pro korrekter CSS-Regel. Ein halber Punkt bei kleineren Fehlern.

Aufgabe 3.3 (XPath) 6 Pkt
Im Kontext von `XML`-Dokumenten, was versteht man unter einem *XPath*? 6 min

Beschreiben Sie außerdem für jeden der unten angegebenen XPaths, was genau damit erreicht werden kann.

1. `/city/library/book[@lang="de"][314]`
2. `//div[@id="hero"]//img`

Lösung: Die *XML Path Language* ist eine Abfragesprache für XML-Dokumente, ähnlich wie reguläre Ausdrücke für unstrukturierten Text. Ein XPath-Ausdruck adressiert eine bestimmte Menge an Knoten eines XML-Dokuments, das dabei als Baum betrachtet wird.

1. Dieser *XPath* beschreibt einen genauen Pfad durch ein Dokument. Gezielt wird auf den 314-ten `book`-Knoten mit dem Attribut `lang="de"` unter einem `library`-Knoten unter einem `city`-Knoten.

2. Dieser Knoten zielt auf alle `img`-Knoten, die Nachfahren (Kinder, Kindeskinder, ...) eines `div`-Knotens mit der Id `"hero"` sind, egal wo im Dokument sich dieser `div`-Knoten befindet.

Korrektur:

- 2 Punkte für korrekte Beschreibung des Konzepts
- Je 2 Punkte für korrekte Beschreibung eines XPaths

---

4 Programmieren in Python

Aufgabe 4.1 (Python-Verständnisfrage) 4 Pkt
Was gibt das Programm in Abbildung 1 aus, wenn es ausgeführt wird? Warum? 4 min

```
p = q = 1

while True:

    if p < q:
        p = q
        q = q * 2

    elif q < p:
        q = p
        p = p * 5

    else:
        p = p + 5
        q = q + 2

    if (p < 0) or (q < 0):
        break

if p == q:
    print("Swordfish")
else:
    print("Friend")
```

Abbildung 1: Was macht dieses Programm?

Lösung: Es wird nichts ausgegeben. Die Schleife läuft endlos weiter, da weder `p` noch `q` jemals unter 0 fallen können (sie werden nur mit positiven Zahlen multipliziert und addiert, egal was passiert).

Korrektur:

- Zwei Punkte für die Erkenntnis, dass es keinen Output gibt.

- Zwei weitere Punkte für korrekte Beschreibung, warum die Schleife endlos ist.

Aufgabe 4.2 (Listen von Listen) 7 Pkt

Schreiben Sie eine python-Funktion longestString, die aus einer *Liste von Listen von Strings* den längsten String findet und zurück gibt. 7 min

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"
```

Korrektur:

- Zwei Punkte für korrekte Funktionsdefinition (kein Programm)
- Vier Punkte für korrekte Ermittlung des längsten Strings (vergleichbar mit der Musterlösung oder anders!)
- Einen Punkt für korrekte Rückgabe (nicht Printen) des längsten Strings

---

Aufgabe 4.3 (Python-Programmieraufgabe bottle / Farben) 11 Pkt

Geben Sie den python-Quellcode für eine `bottle`-Route an, die auf dem Pfad 11 min

/colour/<red>/<green>/<blue>

ansteuerbar ist. Die Route soll eine valide HTML-Seite zurückgeben, deren `<body>`-Tag leer ist, aber auch mittels CSS die entsprechende Hintergrundfarbe zugewiesen bekommen hat.

Beispiel: Bei Besuch von `/colour/255/0/128` soll eine pinke Website (ohne Text) erscheinen.

Hinweis: Ihnen könnten hierzu die Funktionen int(someString) (zur Umwandlung eines String in einen Integer) und hex(someInteger) (zur Umwandlung eines Integers in einen String, in Hexadezimalschreibweise) aus der Standardbibliothek nützlich erscheinen.

Zu den von hex() produzierten Strings ist es wichtig zu wissen, dass z.B. hex(128) == "0x80", aber z.B. hex(10) == "0xa", nicht "0x0a". Bedenken Sie dies bei Erstellung Ihres Farbwertes für CSS!

Sie können davon ausgehen, dass alle benötigten **import**-Statements bereits vorhanden sind und die Eingaben für die Farben nur Werte zwischen (inklusive) 0 und 255 annehmen werden.

Lösung: Eine Lösung ist diese hier:

```
from bottle import route, run

@route('/colour/<r>/<g>/<b>')
def colour(r,g,b):
    """Displays a web page with the given colour as background"""

    # Convert to strings and lose the leading "0x"
    red = hex(int(r))[2:]
    gre = hex(int(g))[2:]
    blu = hex(int(b))[2:]

    # Pad strings if necessary.
    # hex(10) = 0xa, not 0x0a, but we need six places
    if len(red) == 1:
        red = "0" + red

    if len(gre) == 1:
        gre = "0" + gre

    if len(blu) == 1:
        blu = "0" + blu

    # Concatenate into a CSS-readable colour code
    colourcode = "#" + red + gre + blu

    # Compose HTML
    html = """<!DOCTYPE HTML>
    <html>
        <head>
            <style>
                body { background-color: """ + colourcode + """ }
            </style>
        </head>
        <body></body>
    </html>
    """

    # Return HTML
    return html
```

```
run(host='0.0.0.0', port=32500)
```

Korrektur:

- 4 Punkte für korrekte Definition von Route und Funktion
- 4 Punkte für korrekte Ermittlung des Farbcodes
- 3 Punkte für Rückgabe von HTML in dem der Farbcode korrekt eingebunden wurde.
