Name: Vorname:

Geburtsdatum: Matrikelnummer:

Platz:

# Nachklausur
# Informatische Werkzeuge in den Geistes- und Sozialwissenschaften 1

13. April 2023

| | To be used for grading, do not write here | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| prob. | 1.1 | 1.2 | 2.1 | 2.2 | 3.1 | 3.2 | 3.3 | 4.1 | 4.2 | Sum | grade |
| total | 5 | 5 | 6 | 6 | 8 | 6 | 5 | 7 | 12 | 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 (Stellenwertsysteme)** 5 Pkt

Menschen benutzen verschiedene Zahlensysteme. Manche davon sind *Stellenwertsysteme* („Positional Number Systems“). Erklären sie kurz (!), was ein Stellenwertsystem ausmacht. Nennen Sie außerdem zwei Zahlensysteme, die Stellenwertsysteme sind und ein Zahlensystem, welches keins ist. 5 Min

---

*Lösung:* In einem Stellenwertsystem ist der Beitrag einer einzelnen Ziffer zur Gesamtzahl abhängig von der Stelle (i.S.v. Position), an der sich die Ziffer befindet.

Korrekte Antworten für Stellenwertsysteme sind z.B. das Binär-, Oktal-, Dezimal- oder Hexadezimalsystem. Korrekte Antworten für nicht-Stellenwertsysteme sind z.B. das römische oder ägyptische Zahlensystem.

---

**Aufgabe 1.2 (Syntax-Hightlighting)** 5 Pkt

In vielen Entwicklungsumgebungen (z.B. in JupyterLab) werden bestimmte Teile von Quellcode farblich hervorgehoben (z.B. in python Schlüsselwörter wie `return` oder `break`). 5 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 die UDK)** 6 Pkt
6 Min

Die *Universelle Dezimalklassifikation* (kurz: UDK) ist ein System, das weltweit in Bibliotheken zur Klassifizierung menschlichen Wissens benutzt wird. Eine reguläre UDK-Nummer besteht aus Gruppen von 1-3 *Dezimal*ziffern, getrennt durch Punkte zur Leserlichkeit. Je mehr Ziffern, desto präziser die Klassifizierung (so ist "1" z.B. die UDK-Nummer für den gesamten Bereich Philosophie und Psychologie, der Unterbereich Metaphysik hingegen ist "111.5") .

Hier sind ein paar weitere valide Beispiele:

```
599.312.3
316.832
629.7.025.33
37.091.322.7
404
```

Geben Sie einen regulären Ausdruck an, der auf **UDK**-Ausdrücke 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:

```
\d{1,3}(?:\.\d{1,3})*
```

---

**Aufgabe 2.2 (Regulärer Ausdruck für die UDK 2)** 6 Pkt
6 Min

Neben den regulären UDK-Nummern wie in Aufgabe 2.1 sieht der Standard etliche Konstrukte für weitere Informationen vor. Wir beschäftigen uns hier mit dem Konstrukt für Dokumententypen.

Um eine bestimmte Sorte von Dokumenten zu einem Thema zu benennen, wird der UDK-Nummer des Themas eine so genannte „Klammer" angehängt, z.B. `(094)` für Gesetzestexte und juristisches. Diese Klammer enthält (für unsere Zwecke) 2-3 Ziffern, *immer* beginnend mit einer 0.

Hier sind ein paar valide Beispiele:

```
599.312.3(046)
316.832(094)
37.091.322.7(07)
```

Ergänzen Sie Ihren regulären Ausdruck aus Aufgabe 2.1 so, dass nun auch UDK-Ausdrücke mit Dokumententyp wie oben beschrieben matcht. Die Klammer für den Dokumententyp soll dabei optional (!) sein. Falls Sie Gruppierungen verwenden, ist es egal ob diese *capturing* oder *non-capturing* sind.

---

*Hint:* Sie werden in dieser Aufgabe nur für die neu dazugekommenen Teile des regulären Ausdrucks bewertet. Sollten Sie die vorherige Aufgabe nicht bearbeitet haben, dürfen Sie einen Platzhalter benutzen.

---

*Lösung:* Ein solcher Ausdruck wäre zum Beispiel:

```
\d{1,3}(?:\.\d{1,3})*(?:\(0\d{1,2}\))?
```

---

# 3 Digitale Dokumente

**Aufgabe 3.1 (Mentales CSS)** 8 Pkt

8 Min

Gegeben ist der Quelltext der folgenden *HTML* Seite:

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body { background-color: lightblue; }

      h1 {
        text-align: center;
      }

      span { background-color: yellow; }

      div {
        font-weight: bold;
        color: #FF0000;
      }
    </style>
  </head>
  <body>
    <h1 style="font-size:␣5pt;">IWGS</h1>

    <span style="background-color:␣magenta;">
      Lorem ipsum...

      <form>
        <input type="submit" value="Useless␣button!">
      </form>
    </span>
  </body>
</html>
```

Beschreiben Sie, welche Elemente auf der fertig dargestellten *HTML* Seite zu sehen sind, wie sie angeordnet sind und welche Farbe sie haben.

---

*Lösung:* Die Website hat einen hellblauen Hintergrund und beginnt mit einer Überschrift (`<h1></h1>`). Die Überschrift lautet „IWGS“, ist in Schriftgröße 5 und horizontal zentriert.

Unter der Überschrift (nicht mehr horizontal zentriert) befindet sich eine magentane Box (`<span></span>`). In dieser Box steht „Lorem ipsum dolor sit amet...“ .

Darunter ist ein klickbarer Knopf, der mit „Useless Button! “ beschriftet ist.

Kein Element hat den Typ *div*, diese Regeln finden also keine Anwendung.

---

**Aufgabe 3.2 (DOM)** 6 Pkt

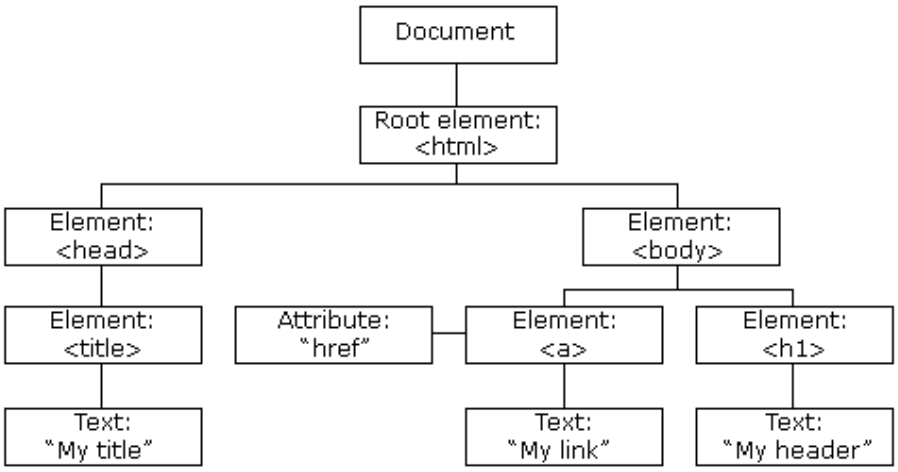
6 Min

Erklären Sie den Begriff des DOM im Kontext von digitalen Dokumenten und welche Rolle das DOM (und darauf aufbauende Programmiersprachen wie Javascript) für interaktive Webseiten spielt.

Skizzieren Sie beispielhaft (kein Quelltext notwendig, Beschreibung genügt), wie Javascript mit dem DOM in Antwort auf ein von der Nutzer*In ausgelöstes Ereignis interagiert.

---

*Lösung:* Das DOM (Document Object Model) ist eine Spezifikation einer Schnittstelle für die Interaktion mit HTML oder XML-Dateien. Diese werden in eine Baumstruktur geparst, in der jeder Knoten einen Teil des Dokuments repräsentiert.




Das DOM bietet eine *standardisierte* (i.e. für alle gängigen Browser gültige) Möglichkeit, mit dafür ausgelegeten Programmiersprachen wie z.B. Javascript, Elemente in HTML- und XML-Dateien zu navigieren, auszulesen, hinzuzufügen, zu löschen oder zu verändern.

Als Beispiel können wir uns vorstellen, dass ein Skript auf einer Website darauf achtet, wenn Nutzer*Innen auf sichtbare Elemente der Website (wie zum Beispiel Überschriften, Links oder Absätze) klicken und die entsprechenden Knoten dann aus dem DOM löscht. So wäre es möglich, nach und nach alle Elemente der Website via Klick zu entfernen.

---

**Aufgabe 3.3 (XPath)** 5 Pkt

Gegeben ist folgender Ausschnitt aus einer XML-Datei zur Katalogisierung von CDs: 5 Min

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
  <CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
```

```
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
  </CD>
  <CD>
    <TITLE>The very best of</TITLE>
    <ARTIST>Cat Stevens</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Island</COMPANY>
    <PRICE>8.90</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>Maggie May</TITLE>
    <ARTIST>Rod Stewart</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Pickwick</COMPANY>
    <PRICE>8.50</PRICE>
    <YEAR>1990</YEAR>
  </CD>
  <CD>
    <TITLE>The dock of the bay</TITLE>
    <ARTIST>Otis Redding</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Stax Records</COMPANY>
    <PRICE>7.90</PRICE>
    <YEAR>1968</YEAR>
  </CD>

  ...
</CATALOG>
```

Geben Sie einen XPath-Ausdruck an, der in der vollständigen Datei (mehr Einträge als hier gezeigt sind) alle Preise von CDs anwählt, die im Jahr 1990 erschienen sind.

---

*Lösung:* Eine mögliche Lösung ist: `//CD[YEAR=1990]/PRICE/text()`, welche explizit den textuellen Inhalt der PRICE-Knoten anwählt. Wir akzeptieren aber auch Lösungen, die auf den ganzen Knoten zeigen.

---

# 4 Programmieren in Python

**Aufgabe 4.1 (Listen von Listen)** 7 Pkt

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

**Beispiel:** Angewendet auf die folgende Liste sollte Ihre Funktion `"elephant"` zurück geben:

```
[[sloth","elephant","cat"],[singing"],["banana","kiwi"]]
```

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

*Hint:* 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 4.2 (Bottle Route zum Testen regulärer Ausdrücke)** 12 Pkt

12 Min

Schreiben Sie eine Route `test_regexes()` für einen `bottle`-Server, die genau ein Argument (String) über die URL annimmt. Dieser String soll gegen eine Liste von regulären Ausdrücken getestet werden, welche erst aus Dateien ausgelesen werden müssen. Im Arbeitsverzeichnis liegen mehrere Dateien, deren Namen alle auf `.regex` enden. Der gesamte Inhalt einer solchen Datei stellt jeweils einen regulären Ausdruck dar.

Ihre Route soll testen, welche dieser regulären Ausdrücke auf den eingelesenen String aus der URL matchen. Am Ende soll eine Liste genau der Dateinamen zurück gegeben werden, die einen passenden Ausdruck enthalten.

*Hint:* Sie können für diese Aufgabe davon ausgehen, dass alle benötigten `import`-Statements bereits vorhanden sind und der Server korrekt gestartet wird. Sie müssen nur den Quelltext für die Route angeben.

*Lösung:* Eine mögliche Lösung ist hier:

```
@route('/test-regexes/<input_string>')
def test_regexes(input_string):
```

```
    # HTML-Skelett (nicht zwingend notwendig)
    html_start = "<html><body><ul>"
    html_elems = ""

    # Gibt alle Dateinamen im aktuellen Verzeichnis.
    for filename in os.listdir():
        # Alles was nicht auf .regex endet ist egal
        if filename.endswith(".regex"):
            # Datei oeffnen und einlesen
            with open(filename) as regex_file:
                regex = regex_file.read()
                # Pruefen, ob regex auf ggb. String passt
                if re.fullmatch(regex, input_string):
                    # Listenelement hinzufuegen
                    html_elems += f"<li>{filename}</li>\n"

    html_end = "</ul></body></html>"
    html_all = html_start + html_elems + html_end

    return html_all
```

---