

5. Mentorium Algorithmen & XML

- Präzise formulierte Verarbeitungsvorschrift zur Lösung eines Problems (z.B. durch einen Computer)
- Gibt an, wie Eingabedaten schrittweise in Ausgabedaten umgewandelt werden
- Genaue und eindeutige Handlungsanweisung

Welche Arten von Wiederholungen (Schleifen) kennen Sie?

- In Python haben wir zwei Schleifentypen kennen gelernt:
- **while <Bedingung>:**
 - <zu wiederholende Sequenz>
- **for <element> in <liste>:**
 - <Sequenz pro Element>

- Eine Liste kann mehrere Elemente enthalten.
- Der Zugriff auf die Elemente einer Liste erfolgt über Indizes. Die Elemente sind durchnummeriert, beginnend mit 0.
- Mit dem „print“ Befehl kann ein Element einer Liste ausgegeben werden
- Mit `len(Listenname)` wird die Länge der Liste abgefragt
- Listen können neben Strings und Numbers wiederum Listen enthalten

- Sequentielle Suche eines Elements in einer **Queue**
- Binary-Search-Tree-Algorithmus
 - Baum, bei dem die Kind-Knoten links kleiner und rechts größer sind als der aktuelle Knoten
 - Suche nach einem bestimmten Knoten im Baum

- Entwickeln Sie in der Programmiersprache *Python* einen Algorithmus, der auf Basis eines einfachen Nutzerprofils (bestehend aus dem *Aufenthaltort*) solche POI herausfiltert, die zur aktuellen Situation des Nutzers passen.
- D.h. in diesem Fall, den POI der die geringste Entfernung zum Nutzer hat (ein Suchbegriff wird nicht benötigt).
- Der gefundene POI soll dann auf dem Bildschirm ausgegeben werden.

- ***Nutzerprofil:***

- Aufenthaltort: Zeil, Frankfurt am Main

- ***POI Gesamtliste:***

Suchbegriff	Name	Entfernung
Restaurant	Kneipe abc	800 m
Restaurant	Restaurant xyz	1500 m
Kino	Kino 123	500 m

- ***Funktionskopf:***

- def poi_liste(suchbegriff, poi_gesamtliste):

- ...

-

- ***Hinweis:*** Definieren Sie zunächst einen geeigneten Datentyp *Liste* zur Speicherung der *POI Gesamtliste*. Die Entfernungen eines Nutzers zu einem POI wurden bereits berechnet und sind Teil der *POI Gesamtliste*.

```
def poi_liste(poi_gesamtliste):  
    poiMin = poi_gesamtliste[0]  
  
    for poiAktuell in poi_gesamtliste:  
        if poiAktuell[2] < poiMin[2]:  
            poiMin = poiAktuell  
  
    print poiMin[1]  
  
poi_gesamtliste = [ ["Restaurant", "Kneipe abc", 800],  
                    ["Restaurant", "Restaurant xyz", 1500],  
                    ["Kino", "Kino 123", 500] ]  
  
poi_liste(poi_gesamtliste)
```

Funktion definieren
Variable zum Speichern des POIs mit der
#kleinsten Entfernung
Element aus der poi_gesamtliste entnehmen
Auf minimale Entfernung überprüfen
POI mit aktuell kleinster Entfernung speichern
Name des POIs mit final kleinster Entfernung
ausdrucken
POI Gesamtliste generieren (2-dimensional)
Funktionsaufruf

- Beschreiben Sie kurz die Eigenschaften eines XML-Dokuments.
- Einfache und menschenlesbare Syntax (nicht binär)
- Standardisiert
- Selbstbeschreibend durch enthaltene Meta-Beschreibung
- Erweiterbar durch neue Elementbeschreibungen -> anwendungsspezifische Datenmodelle
- Eignet sich zur Datenhaltung

- Eine Document Type Definition (DTD) beschreibt Struktur und Grammatik von XML-Dokumenten.
- Ist vergleichbar mit einer Variablen- / Typendeklaration in einer Programmiersprache
- Es wird definiert, welche Werte in Elementen vorkommen dürfen, damit ein „**gültiges**“ **XMLDokument** entsteht.
- DTD „übersetzt“ also ein XML-Dokument in Datentypen für Datenbanken und erzeugt Regeln für Elemente

- **DTD**
Document Type Definition – beschreibt die formale Struktur eines Dokuments
- **XML Schema**
Alternativer Ansatz mit Erweiterungen der DTD
- **Parser**
Übersetzt XML-Dokument in einen Dokumentenbaum
- **Style Sheet**
Layout-Vorgaben zur Ausgabe von Dokumenten
- **Style-Sheet-Prozessor**
Setzt die Style-Vorgaben um und generiert die Ausgabeseiten

Was ist ein "wohlgeformtes" bzw. "valides" XML Dokument?

- Wegen der eindeutigen, baumartigen Struktur und der Ähnlichkeit zu objektorientierten Systemen erkennt ein Rechner beim Einlesen von XML die Datenstruktur eindeutig.
- Ein „**wohlgeformtes Dokument**“ in XML ist ein syntaktisch korrektes Dokument mit abgeschlossenen, nicht überlappenden Tags und einheitlicher Groß- und Kleinschreibung.

- Zur Verarbeitung benötigt man einen Parser.
- Ein Parser ist eine Software, welche DTDs, Schemas und XML-Dokumente einlesen kann, um dann einer Anwendung Zugriffe auf alle Elemente zu ermöglichen.
- Übliches Vorgehen:
 - Anwendung öffnet XML-Datensatz.
 - Parser liest XML-Dokument und nötige DTDs, Schemas.
 - Parser bietet Anwendung Schnittstellen mit Funktionen wie "ElementeAuflisten()".
 - Anwendung durchsucht mittels der Schnittstellen das Dokument und bearbeitet die Elemente.
 - Anwendung speichert den überarbeiteten XML-Datensatz ab.
- Ergebnis
Ein syntaktisch fehlerfreies und gegen eine DTD geprüftes XML-Dokument wird als "gültig" bzw. "valid" bezeichnet.

- Es gibt zwei verschiedene Arten von Parsern:
 - Document Object Model (DOM)
 - Simple API for XML (SAX)
- DOM-Parser laden alle Elemente in den Speicher und erzeugen eine Baum-Datenstruktur, auf der dann gearbeitet wird.
- SAX-Parser navigieren sich lesend durch ein Dokument, ohne es komplett im Speicher abzulegen.

Vergleich DOM- und SAX-Parser

- SAX kann Dateien beliebiger Größe parsen.
- SAX ist effizient, wenn nur bestimmte Teile interessant sind.
- SAX hat einfaches Handling.

- DOM erlaubt freie Zugriffe und Änderungen am Dokument.
- DOM erzeugt eine vollständige Abbildung des Dokuments.

Einsatzszenarien von DOM und SAX

- DOM-Parser eignen sich gut beim Bearbeiten gesamter Dokumente, z.B. zur Bearbeitung eines gegliederten Textes in einer Textverarbeitung.
- SAX-Parser eignen sich zum schnellen Auffinden von Datensätzen, z.B. der Adressen in einer XMLbasierten Kundendatenbank.

- Erstellen Sie für die Speicherung eines dynamischen Kundenprofils eine DTD und daraufbasierend ein XML Beispieldokument. Es soll dabei der Nutzungszeitpunkt, der aktuelle Aufenthaltsort sowie die persönlichen Daten eines Nutzers erfasst werden. Die persönlichen Daten sind dabei nochmal unterteilt in *Pseudonym*, *Alter*, *Geschlecht* und eine kommaseparierte Liste der *Interessen* des Nutzers.
- ***Beispiel:***
 - Pseudonym: mobilerFritz1380, Alter: 25, Geschlecht: männlich
 - Interessen: Kino, Restaurants, Tennis, ...

- Element-Content:

EMPTY	leeres Element
ANY	beliebiger Inhalt
	Auswahlliste
,	Sequenz
()	Gruppierung
(#PCDATA)	Zeichen- oder Stringdaten

- Kardinalität:

	leer: genau ein Wert nötig
+	mindestens ein Wert
?	Null oder ein Wert
*	Null oder mehr Werte

DTD:

```
<!ELEMENT Benutzerprofil (nutzungszeitpunkt, aufenthaltort, pseudonym, alter,
    geschlecht, interessen) >
<!ELEMENT nutzungszeitpunkt (#PCDATA) >
<!ELEMENT aufenthaltort (#PCDATA) >
<!ELEMENT pseudonym (#PCDATA) >
<!ELEMENT alter (#PCDATA) >
<!ELEMENT geschlecht(männlich | weiblich) >
<!ELEMENT interessen (kino | restaurants | tennis | fußball | schwimmen |
    hockey)* >
```

XML-Dokument:

```
<?xml version="1.0" encoding = "ISO-8859-1" ?>
<Benutzerprofil>
  <nutzungszeitpunkt>21.06.2010</nutzungszeitpunkt>
  <aufenthaltort>Universität</aufenthaltort>
  <pseudonym> mobiler fritz 1380 </pseudonym>
  <alter> 25 </alter>
  <geschlecht>männlich</geschlecht>
  <interessen> kino, restaurants, tennis </interessen>
</Benutzerprofil>
```

Offene Fragen?