Hardware designen

Software entwickeln

Netzwerke managen

Jäger Alexander

HTL Anichstraße | Anichstraße 26-28, 6020 Innsbruck

JSON und WEB-APIs

Fachspezifische SoftwaretechniK, 4. Klasse

# Einleitung

Dieses Skriptum gibt einen Einblick, wie man Informationen von einem Webserver erhalten kann. Die heutzutage verwendeten Technologien hierfür sind oft:

## http- Protokoll

Das Protokoll ist die Art und Weise wie die Kommunikation zwischen Client und Server abläuft. Hier hat sich in den vergangenen Jahren ein Standard etabliert, welcher eigentlich von fast allen Clients und Servern verwendet wird: http – **H**yper**T**ext **T**ransfer **P**rotokoll. Dieses Protokoll regelt, wie Daten (oder im Falle von http Text) vom Client zum Server gelangen und wieder retour.

## json-Format

Das Format ist die Art und Weise, wie die Daten aussehen, welche vom Server zum Client übertragen werden. In aktuellen Anwendungen wird fast immer das json-Format verwendet (**J**ava**S**cript-**O**bject-**N**otation). Es bietet die Möglichkeit die Daten strukturiert zu übertragen. Trotzdem ist json sparsam – das heißt, es müssen nicht viele zusätzliche Daten für die Struktur zu den eigentlichen Daten hinzugefügt werden.

## Erster Happen

Gleich zu Beginn ein Beispiel: Damit versteht man im Überblick, wie so eine Daten-Abfrage an einen Web-Server funktioniert.

Dir ist langweilig und du suchst Ideen für eine Aktivität. Du weißt, dass es eine Website gibt, welche dir einen Tipp gibt, was du tun könntest: <http://www.boredapi.com/> (Du kannst diese URL ruhig in deinen Browser eingeben).

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungDamit du die Daten für eine Aktivität erhältst, muss du allerdings die URL (Uniform Resource Locator, also Einheitlicher-Ressourcen-Positionsgeber) <http://www.boredapi.com/api/activity/> verwenden. Sobald du diese URL in der Suchleiste des Chrome eingibst, erhältst du json-formatierte Daten, welche die Aktivität enthält (siehe Abbildung 1, gelbe Umrahmung).

Abbildung 1: Abfrage einer Aktivität über den Browse Chrome

Wenn du die URL im Browser Firefox eingibst, dann wird dir die Antwort vom Server bereits übersichtlich dargestellt (siehe Ein Bild, das Text enthält.

Automatisch generierte BeschreibungAbbildung 2, gelbe Umrahmung). Besonders, wenn man die Daten strukturiert sehen möchte, hilft die Darstellung im Firefox.

Abbildung 2: Aktivität vom Webserver im Firefox

### Was ist passiert?

Durch das Eingeben der URL in die Suchleiste des Browsers, wurde über das http-Protokoll eine Anfrage an den Webserver mit der URL <http://www.boredapi.com/> gesendet. Der Server hat die Anfrage ausgewertet und dazu eine Antwort „berechnet“. Diese Antwort hat er wieder an den Browser zurückgeschickt. Und der Browser zeigt dir dann die erhaltene Antwort. Firefox stellt die Antwort strukturiert dar, Chrome zeigt den Text, der vom Server gesendet wurde.

Jedes Mal, wenn du Anfrage erneut sendest (Enter in der Suchleiste mit der bereits vorhanden URL <http://www.boredapi.com/api/activity/>) wiederholt sich der Vorgang. Du kannst das überprüfen – die solltest jedes Mal eine neue Aktivität erhalten.

## Auswertung

Besonders im Chrome ist das Ergebnis der Anfrage sehr unübersichtlich. Warum man das trotzdem braucht? Nun ja, json-Daten kann ein Computerprogramm (z.B. Python-Skript, etc.) auswerten und die Daten weiterverwenden. Diese Schnittstellen werden deshalb auch API genannt, **A**pplication **P**rogramming **I**nterface.

# Datenaustausch über das Internet

Das Internet besteht aus vielen Servern, welche miteinander kommunizieren. Jeder dieser Server stellt verschiedene Dienste (Services) zur Verfügung. Rechner (wie z.B. dein Laptop) können mit diesen Diensten kommunizieren und somit Daten dorthin senden oder von dort erhalten.

## client-server-modellClient und Server

Abbildung 3: Anfrage und Antwort zwischen Client und Server  
(Quelle: https://blog.zeta-producer.com/client/)

Prinzipiell sind solche Dienste immer auf einem Server (oder mehreren Servern) lokalisiert. Das heißt, dass wenn man die Adresse des Servers und des Dienstes kennt, dann kann man zu diesem eine Kommunikation aufbauen. Bei einer solchen Kommunikation unterscheidet man zwischen:

* Server: Der Server stellt einen Dienst (ein Service) bereit. Dieser Dienst kann von einem Client (deutsch: Kunde) genutzt werden. Normalerweise bietet ein Server seinen Dienst nicht nur für einen Client zur Verfügung, sondern vielen. So können z.B. viele Webbrowser (Clients) auf einen Webserver mit einer entsprechenden Website (z.B. <https://orf.at>) zugreifen
* Client: Ein Client (Kunde) kann einen Dienst bei einem Server nutzten. In der Regel nutzen mehrere unterschiedliche Clients die Dienste eines Servers.

Im Gegensatz zum Client-Server-Modell existiert das Peer-to-Peer Model (Gleich zu Gleichen). Hier gibt es nicht die Unterscheidung der einzelnen Beteiligten. In einem Peer-to-Peer-Netzwerk sind alle Computer gleichberechtigte Kommunikationspartner.

## Kommunikation

Die Kommunikation im Internet funktioniert über Nachrichten. Dabei wird zwischen Anfrage (Request) und Antwort (Response) unterschieden:

* Request: Wird die Nachricht vom Client an den Server bezeichnet. Der Anfang einer Kommunikation im Internet ist, dass der Client eine Anfrage an den Server stellt.
* Response: Wird die Antwort des Servers auf diese Anfrage bezeichnet. Der Server sendet also eine Nachricht und beantwortet somit die Anfrage des Clients.

Jede Nachricht besteht aus zwei Teilen:

* Kopf (Header): Beinhaltet Informationen über den Nachrichte nrum, z.B. welche Daten dieser beinhaltet und wie diese Daten gelesen werden können
* Rumpf (Body): Enthält die Nutzdaten, also die eigentlichen Informationen

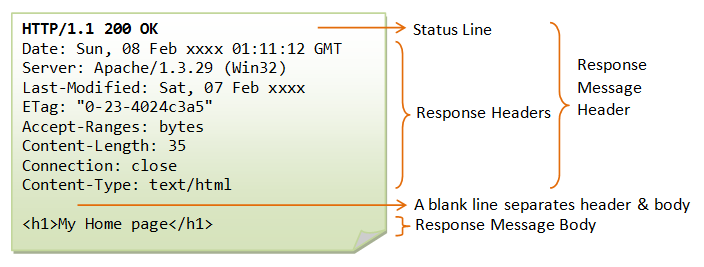


Abbildung 4: Nachrichtenkopf und -rumpf einer Antwort eines HTML-Web-Service  
(Quelle: https://documentation.help/DogeTool-HTTP-Requests-vt/http\_request.htm)

Im Internet werden Nachrichten über das HTTP (Hypertext-Transfer-Protokoll) versendet. Es ist mit der Erweiterung HTTPS (mit Verschlüsselung) eigentlich der Standard für den Austausch zwischen Services und Clients im Internet.

### HTTP

HTTP ist ein Standard, welcher schon 1991 definiert wurde. Die aktuelle Version ist HTTP/3. Das Protokoll ist zustandslos, das heißt, dass Informationen von früheren Anfragen keine Relevanz für eine neue Anfrage haben. Das ist sehr hilfreich, weil dadurch am Server nichts über die Kommunikation gespeichert werden muss.

HTTP wird bei jeder Anfrage verwendet, welche von einem Browser abgesetzt wird. Im Detail wird die URL (siehe URL auf Seite 5) in die Adressleiste des Browsers eingegeben. Der Browser sendet somit an die angegebene Adresse eine Anfrage (Request), welche dann der Server an der Adresse beantwortet (Response). In der Regel ist die Antwort ein HTML-Quelltext, welchen der Browser grafisch darstellt. Dieser Sachverhalt ist in Abbildung 5 dargestellt.

Wenn nun der Benutzer vor dem Computer auf einen Link auf dieser Seite klickt, dann erstellt der Browser wiederum einen Request an den Webserver und versendet ihn. Der Webserver beantwortet die Anfrage und der Browser erhält einen neuen HTML-Quelltext, welchen er grafisch darstellt.

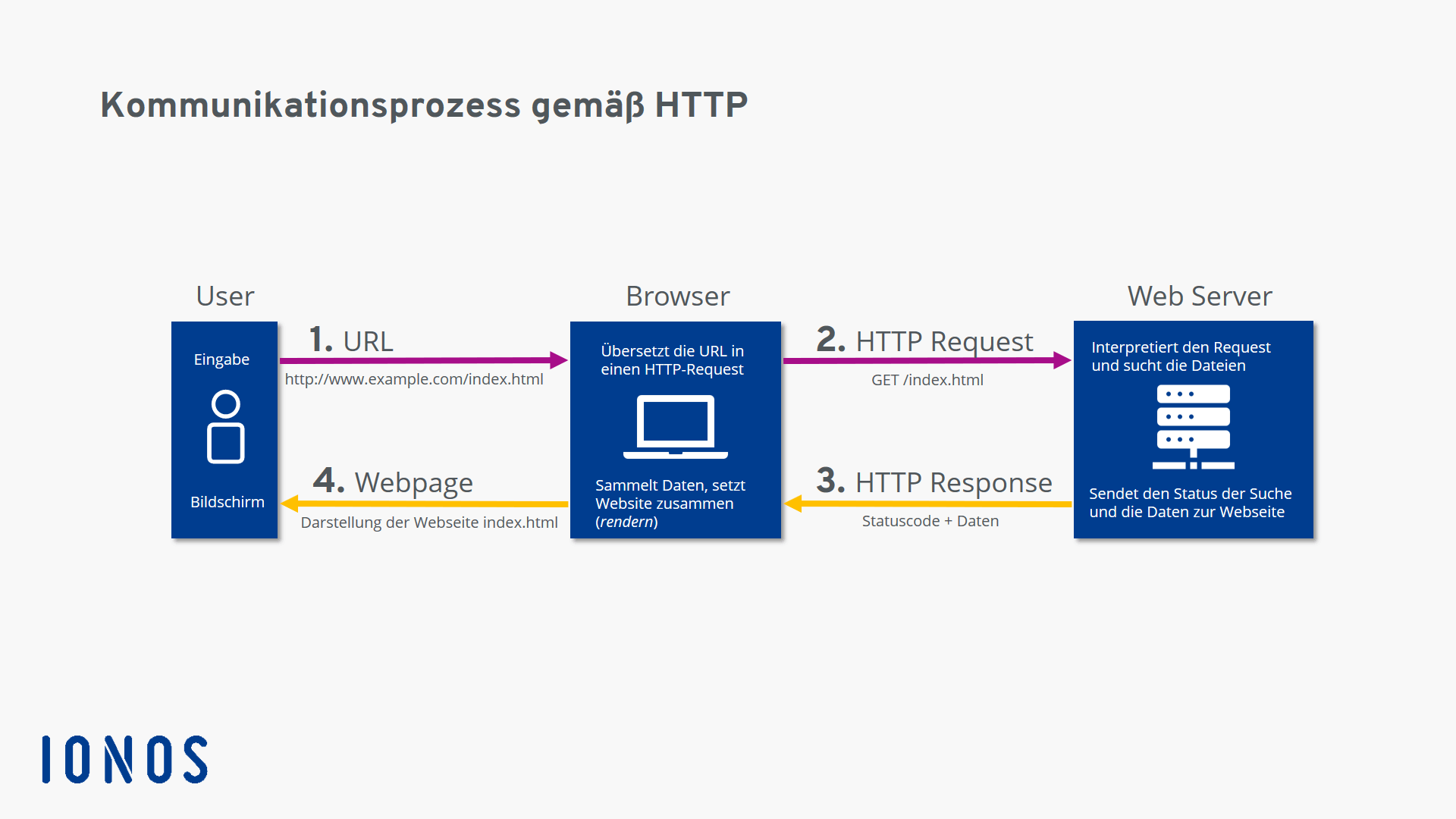


Abbildung 5: Kommunikation zwischen Browser und Webserver über http  
(Quelle: https://www.ionos.de/digitalguide/hosting/hosting-technik/was-ist-http/)

### URL

Abbildung 6: Aufbau einer URL  
(Quelle: https://ahrefs.com/blog/de/seo-friendly-urls/)

Damit klar ist, an welchen Server eine Anfrage übermittelt werden soll, gibt es die sogenannte URL (**U**niform **R**esource **L**ocator, also Einheitlicher Ressourcenzeiger). Die URL besteht aus vielen Teilen und ist nicht nur für die Kommunikation mittels http geeignet. In Abbildung 6 ist der Aufbau einer URL beschrieben. Neben der Angabe des Servers (Top-Level-Domain, Domain, Subdomain) und dem Pfad zum Service (Subfolder, Pfad) wird auch das Kommunikationsprotokoll angegeben.

Schema/Protokoll

Um der Anfrage an einen Service individuelle Werte mitgeben zu können, gibt es zwei Möglichkeiten:

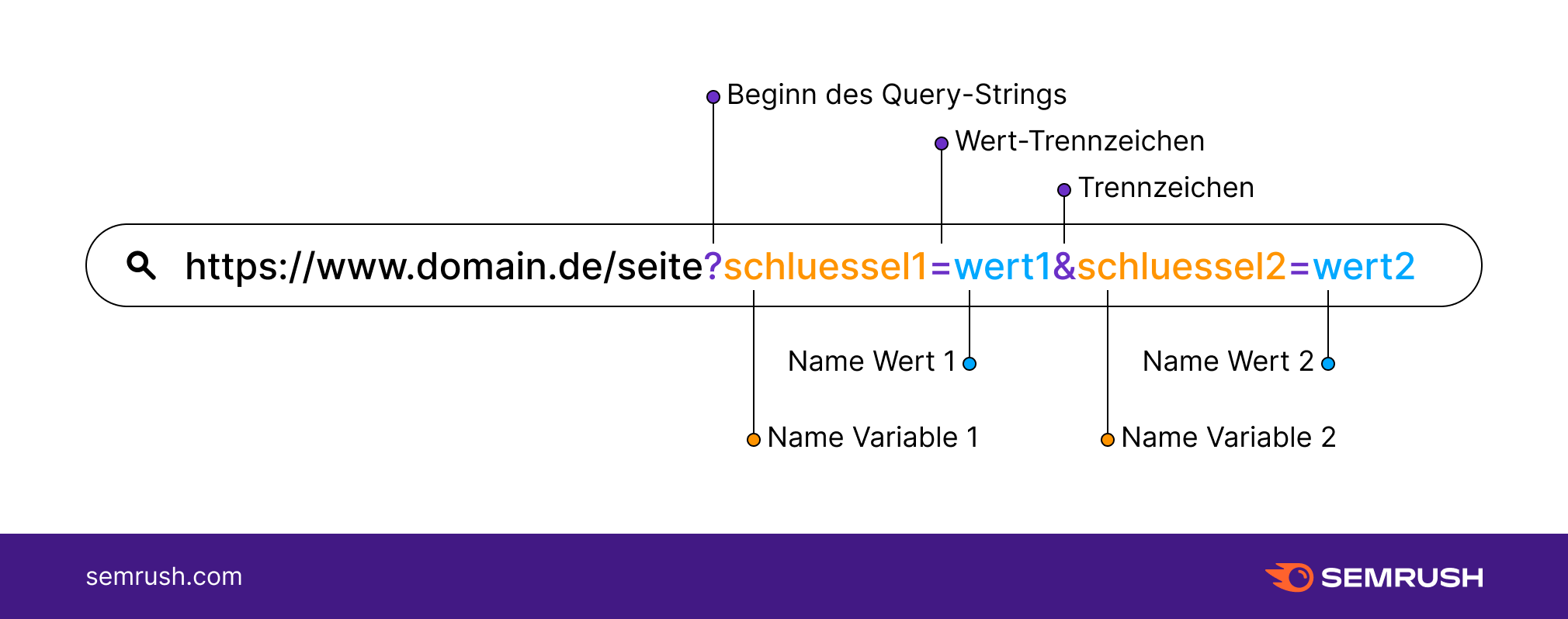
* Parameter in der URL zu verwenden (siehe Parametrisierte Anfragen an einen Webservice auf Seite 5)
* Daten im Rumpf des Requests mitzugeben. Im Browser ist das so nicht möglich. Es kann aber in einem Programm (z.B. Python-Skript, C-Programm) entsprechend programmiert werden.

## Parametrisierte Anfragen an einen Webservice

Anfragen an einen Internet-WebServer sind oft statisch, also ohne, dass bestimmte Werte angegeben werden, z.B. der Aufruf von <https://www.orf.at> ist immer gleich. Wenn jedoch Webservices abgefragt werden, dann ist es oft nötig, die Anfrage an den Service zusätzlich zu parametrisieren. Das heißt, dass z.B. eine Auflistung eingeschränkt wird, oder Daten für einen ganz bestimmtes Objekt gesucht werden.

Eine solche Möglichkeit bieten URL-Parameter. Diese können einfach nach der Angabe des Webservice (siehe Abbildung 6) hinzugefügt werden. Dieser Teil der URL wird auch Query-String (also Anfrage-Text) genannt.

Prinzipiell werden die Parameter in einer URL mittels Fragezeichen (?) eingeleitet. Danach finden sich Paare von Parameternamen und dessen Parameterwert. Diese Paare werden durch ein kaufmännisches Und (&) getrennt. Das Trennzeichen für Parameter und Wert ist das Gleichheitszeichen (=).



Parameter

Parameter

Abbildung 7: Query-String einer URL mit Parametern

Wie die Parameter für einen Service heißen, ist in der Spezifikation des Service angegeben. Weiters ist in dieser Spezifikation auch angegeben, was die Werte für die Parameter für Auswirkungen auf die Anfrage und somit auf die Antwort haben.

### Beispiel

Als Beispiel kann folgende URL verwendet werden, welche zu einem Service gehört, wo Cocktail-Rezepte abgefragt werden können. Bei dieser URL ist der Parameter „s“ verwendet worden mit dem Wert „margarita“: http://www.thecocktaildb.com/api/json/v1/1/search.php?s=margarita

Bei dem Service http://www.thecocktaildb.com/api/json/v1/1/search.php gibt es laut Dokumentation mehrere unterschiedliche Parameter. Einer davon ist *s*. Der Wert, welcher für diesen Parameter angegeben wird, schränkt das Ergebnis dermaßen ein, dass in der Antwort des Service nur mehr Cocktails zu finden sind, welche im Namen den Wert des Parameters haben. Also werden bei obiger Anfrage nur Margarithas zurückgegeben.

Eine Beschreibung aller möglicher Parameter für den Service findet man unter: https://www.thecocktaildb.com/api.php.

# JSON – JavaScript Object Notation

Wenn Daten von einem Computer (z.B. Client) zu einem anderen (z.B. Server) übertragen werden, dann muss beiden Computern bekannt sein, wie die Daten formatiert sind. Das heißt, in welcher Reihenfolge, welche Daten gesendet werden und wie diese zu lesen sind (z.B. werden Zahlen binär übertragen oder als Text). Hier kommt nun JSON ins Spiel, weil es eine gewisse Formatierung und Struktur über die Daten aufweist. JSON ist Text, welcher Daten und deren Struktur beschreibt.

JSON wird zur Übertragung und zum Speichern strukturierter Daten eingesetzt. Es dient als Datenformat bei der Datenübertragung (Serialisierung). Insbesondere bei Webanwendungen und mobilen Apps wird es in Verbindung mit JavaScript, Ajax oder WebSockets zum Übertragen von Daten zwischen dem Client und dem Server häufig genutzt (<https://de.wikipedia.org/wiki/JavaScript_Object_Notation>, 16.08.2022).

## Serialisierung

Die Umwandlung von Daten aus dem Speicher in eine Form, wo diese übertragen werden können, nennt man Serialisierung. Die Daten werden also so hintereinander gesetzt, dass sie nacheinander als Datenstrom versendet werden. Am anderen Ende der Leitung werden die Daten dann stückweise (oder auch gesammelt) empfangen und wieder im Speicher „zusammengebaut“. Dieser Vorgang wird Deserialisierung genannt.

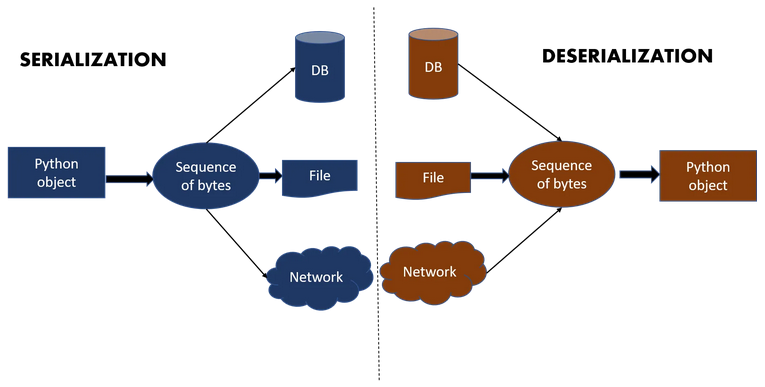


Abbildung 8: Serialisierung und Deserialisierung eines Objektes  
(<https://pythonwife.com/data-serialization-and-deserialization-in-python/>, 16.8.2022)

Wie in Abbildung 8 ersichtlich, werden auch beim Speichern von Daten auf einem Datenträger oder in einer Datenbank die Daten zuerst serialisiert. Beim Laden werden die serialisierten Daten dann wieder deserialisiert.

Das Format eines Datenstromes ist normalerweise ein Programmiersprachen-unabhängiges Format, welches auch zwischen Programmen unterschiedlicher Programmiersprachen und Computern mit unterschiedlichen Betriebssystemen ausgetauscht werden kann.

Die wichtigsten Arten von Serialisierung sind:

* Binäre Serialisierung: Die Daten werden binär hintereinander gereiht. Es ist das kompakteste Format, da keine zusätzlichen Daten für die Struktur benötigt werden. Bei der Deserialisierung muss jedoch die Struktur der Daten bekannt sein, damit die Daten wieder korrekt hergestellt werden können. Die Daten sind normalerweise für Menschen Ein Bild, das Text enthält.

  Automatisch generierte Beschreibungschwierig zu lesen. Dieses Format wird am häufigsten für das Speichern in Dateien verwendet.

Abbildung 9: Daten im Binärformat

* Ein Bild, das Text enthält.

  Automatisch generierte BeschreibungSerialisierung nach XML (E**x**tensible **M**arkup **L**anguage): Das XML -Format kann zu den eigentlichen Daten sehr viel zusätzliche Strukturdaten enthalten. Das kann so weit gehen, dass die gesamte Struktur inklusive Einschränkungen der Werte und deren Abhängigkeiten beschrieben sind. Das Format benötigt neben den eigentlichen Daten viel Struktur-Information. Damit sind Daten in XML meist viel größer als die ursprünglichen Daten. Dafür sind die Daten und die Struktur sehr gut für Menschen zu lesen.

Abbildung 10: Daten im XML-format

* Ein Bild, das Text enthält.

  Automatisch generierte BeschreibungSerialisierung nach JSON: JSON hat sich aus der JavaScript-Schreibweise von Objekten und deren Eigenschaften entwickelt. Inzwischen bieten aber alle gängigen Programmiersprachen einfache Funktionen zum (De-)serialisieren von Daten nach JSON an. Den Daten wird eine einfache Struktur verpasst. Das Format ist relativ kompakt, kann aber noch gut von Menschen gelesen werden. Bei der Arbeit im Internet ist es das mit Abstand am häufigsten verwendete Format.

Abbildung 11: Daten im JSON-Format

## Aufbau von JSON

Wie bereits oben erwähnt erlaubt es JSON die Struktur von Daten darzustellen. Hierfür gibt es drei unterschiedliche Möglichkeiten:

* Eigenschaften: Eigenschaften haben sowohl einen Namen, einen Wert, als auch einen Datentyp
* Objekte: Sie sind eine Zusammenfassung von mehreren Eigenschaften
* Listen: Sind eine Zusammenfassung mehrerer Werte oder Objekte

Leerzeichen in der Struktur des JSON sind nicht von Bedeutung, innerhalb von Eigenschaftennamen und Zeichenketten jedoch schon.

### Objekte und Eigenschaften

Prinzipiell werden bei JSON sogenannt Objekte betrachtet. Objekte werden durch geschwungene Klammern dargestellt. Diese haben Eigenschaften, welche jeweils einen Namen und einen Wert aufweisen. Für den Wert einer Eigenschaft gibt es sechs unterschiedliche Datentypen. Der Name der Eigenschaft muss innerhalb des Objektes eindeutig sein.

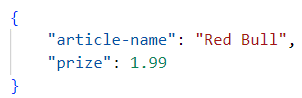


Abbildung 12: JSON-Objekt mit zwei Eigenschaften

Das Beispiel in Abbildung 12 enthält ein Objekt mit zwei Eigenschaften. Die Eigenschaft *article-name* hat den Wert *Red Bull* (Datentyp: Zeichenkette), die Eigenschaft *prize* hat den Wert 1.99 (Datentyp: Zahl).

Die geschwungenen Klammern um die Eigenschaften zeigen, dass dies ein Objekt ist. Eigenschaften werden durch einen Beistrich getrennt, wobei der Beistrich vor einer geschwungenen Klammer weggelassen wird.

#### Datentypen

Abbildung 13: Eigenschaften mit allen möglichen Datentypen

Eigenschaften eines Objektes haben immer einen Namen und einen Wert. Der Wert kann dabei einem der folgenden Datentypen entsprechen:

* Nullwert: wird durch das Schlüsselwort null dargestellt
* Boolescher Wert: Schlüsselwörter true und false – es werden keine Anführungszeichen verwendet
* Zahl: positive oder negative Zahlen möglich, als Dezimalzeichen ist der Punkt zu verwenden
* Zeichenkette: Zur Begrenzung werden sowohl am Anfang als auch am Ende doppelte Anführungszeichen verwendet
* Liste: Ist eine Aufzählung von Elementen, welche durch einen Beistrich getrennt sind. Die Elemente können vom gleichen oder von unterschiedlichen Typen sein, auch Objekte sind möglich. Leere Listen sind zulässig
* Objekt: Eine Eigenschaft kann als Wert wieder ein Objekt aufweisen. Somit können Objekte andere Objekte (wieder mit Eigenschaften) beinhalten.

Für Datumswerte oder Zeitwerte gibt es keinen eigenen Datentyp. Diese werden nach einem bestimmten Schema in Text umgewandelt (Details dazu siehe <https://de.wikipedia.org/wiki/ISO_8601#Darstellung>).

### Unterobjekte und Listen

Die Hauptmöglichkeiten bei JSON Strukturen in die Daten zu bringen, sind Unterobjekte und Listen. In diesem Abschnitt sollen diese deshalb etwas genauer betrachtet werden.

#### Unterobjekte

Unterobjekte sind Werte für Eigenschaften, welche wieder Objekte sind (siehe Abbildung 15). Sie bieten die Möglichkeit einer zusätzlichen Strukturierung der Daten:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 14: Der Wert der Eigenschaft *address* besteht aus einem Objekt, welches Straße, Stadt und Postleitzahl vereint

Theoretisch könnten die Eigenschaften auch alle in einem Objekt versammelt werden. Dadurch würden die logischen Zusammenhänge allerdings verloren gehen. Außerdem wird das JSON viel unübersichtlicher.

Wenn JSON-Objekte mittels Python ausgewertet werden, macht es einen Unterschied, ob Eigenschaften in Unterobjekten gesammelt sind, oder nicht.

#### Listen

Listen (auch Arrays genannt) werden in JSON häufig genutzt. Sie bieten die Möglichkeit, mehrere Werte zusammenzufassen:

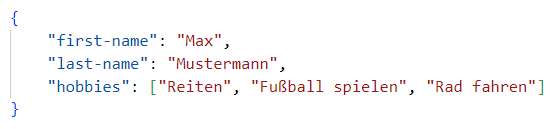


Abbildung 15: Beispiel für eine einfache Liste

In Abbildung 15 sind Werte des Datentyps Zeichenkette in der Liste der Hobbys zusammengefasst. Abbildung 16 zeigt eine Liste aus zwei Objekten. In diesem Fall haben die Objekte die selben Eigenschaften (natürlich mit unterschiedlichen Werten) – das muss nicht sein. Ob die Elemente von Listen in einer Zeile oder in mehre Zeilen aufgeteilt werden ist für die Verarbeitung unerheblich.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 16: Beispiel für eine Liste aus Objekten

Die einzelnen Werte werden durch einen Beistrich getrennt.

## Beispiele

Alle JSON-Dateien, JSON-Texte, etc. stellen ein Objekt dar. Deshalb sind sie immer von geschwungenen Klammern umschlossen (Teilweise aus: https://www.guru99.com/json-tutorial-example.html)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 17: JSON-Beispiel für eine Liste von drei elektronischen Büchern mit jeweils einer Programmiersprache und der Edition

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung 18: JSON-Beispiel für Studenten. In diesem Fall gibt es eine Liste mit zwei Studenten, welche jeweils eine ID eine Vor- und einen Nachnamen haben

Ein Bild, das Text, Whiteboard enthält.

Automatisch generierte Beschreibung

Abbildung 19: JSON-Beispiel mit Unterobjekt in einem Unterobjekt (city), einem leeren Objekt (wife) und einer Liste von Objekten (children)

# Webrequests mit Python

Wie bereits oben besprochen wird JSON hauptsächlich verwendet, um Daten von einem Web-Service zu erhalten. In diesem Abschnitt soll nun erklärt werden, wie man mithilfe von Python eine Anfrage an einen Webserver stellt. Anschließend wird gezeigt, wie man die Antwort auswerten kann.

Um mit Python Anfragen an einen Webserver stellen zu können, braucht man folgende Dinge:

* URL zum Service (Web-Server und Pfad), welcher die Anfrage beantwortet. Es werden auch die möglichen URL-Parameter benötigt, welche bei der Anfrage nötig/zulässig sind
* Beschreibung der Antwort des Service: Wie sieht das JSON aus, welches der Service zurücksendet
* Natürlich eine Python-Installation mit dem Package requests installiert

## Packages

Um mit Python Anfragen an einen Web-Service stellen zu können, wird das Package requests benötigt. Es stellt Funktionen zur Verfügung, mit welchem Anfragen erstellt und ausgewertet werden können. In diesem Skriptum beschäftigen wir uns nur mit der Funktion get, im package requests gibt es aber auch die anderen HTTP-Methoden (wie z.B. POST oder DELETE).

Wie gewohnt kann das Package requests über pip installiert werden:

pip install requests

Möchte man JSON-Daten aus einer Datei laden oder in eine Datei speichern, so ist das Package json zu verwenden. Dieses ist in der Standard-Bibliothek von Python vorhanden.

## Beispiel

Folgendes Beispiel baut auf dem Eingangs-Beispiel auf. Wir stellen dieselbe Anfrage mit Python und werten das Ergebnis aus. Der Source-Code dazu sieht wie folgt aus:

import requests

# Senden der Anfrage an den Web-Server und erhalten der Antwort

response = requests.get("http://www.boredapi.com/api/activity")

# Umwandeln der Antwort in Python-Objekte

result = response.json()

# Ausgabe der Aktivität

print(result["activity"])

Source-Code 1: einfache Abfrage einer Web-API und Auswertung des Ergebnis

Bereits wenige Zeilen Code ermöglichen es in Python eine Abfrage an einen Server zu stellen (siehe Source-Code 1): Nachdem die Bibliothek requests importiert wurde, kann mit Hilfe der Funktion get() eine Anfrage an einen Server gestellt werden. Das Argument der Funktion get() ist dabei die URL des Web-Service. Der Rückgabewert der Funktion ist bereits die Antwort des Service als Python-Objekt (inkl. HTTP-Statuscode, etc.).

Damit man die Antwort gut verarbeiten kann, wird der JSON-Text, welchen man als Antwort erhält, in Python-Objekte umgewandelt. Das passiert in der dritten Code-Zeile mittels response.json(). Wie genau der JSON-Text in Python-Objekte umgewandelt werden, wird weiter unten erklärt.

In der letzten Zeile des Codes wird dann der Wert ausgegeben, welcher sich in der Eigenschaft activity des JSON-Objektes befindet.

## JSON in Python-Objekte

Um mit JSON in Python einfach arbeiten zu können, wird die Antwort einer Web-Service-Anfrage nicht als Text verarbeitet, sondern zuerst in Python-Objekte umgewandelt. Diese Objekte können dann mittels normalen Python-Ablaufstrukturen behandelt werden.

Um die Umwandlung von JSON-Text in Python-Objekte verstehen zu können, ist ein Wissen über Listen und Dictionaries Voraussetzung.

Für alle kommenden Beispiele wird der JSON-Text in Abbildung 20 als Grundlage verwendet. Er wurde wie oben beschrieben in Python-Objekte umgewandelt. Damit kann auf die Objekte, deren Eigenschaften, auf Listen und Unterobjekte mit Python-Code zugegriffen werden. Jeweils in der Variable result ist das Basis-Objekt gespeichert.

Ein Bild, das Text, Whiteboard enthält.

Automatisch generierte Beschreibung

Abbildung 20: JSON-Grundlage für Beispiele

### Eigenschaften

Wir beginnen mit dem einfachsten Fall: Der Zugriff auf einzelne Eigenschaften eines Objektes. Ein JSON-Objekt wird in Python als Dictionary abgebildet. Auf die Eigenschaften kann über den Namen der Eigenschaft zugegriffen werden:

print(result["first-name"]) # Ausgabe: Max

print(result["address"]) # Ausgabe: Werte des Objektes der Eigenschaft address

Source-Code 2: Zugriff auf Werte von Eigenschaften eines Objektes

Auch auf Eigenschaften eines Unterobjektes kann zugegriffen werden. Dazu kann das Unterobjekt in eine eigene Variable gelegt werden. Anschließend wird auf die gewünschte Eigenschaft das Unterobjektes über das Objekt in der neuen Variable zugegriffen.

ad = result["address"] # Wert der Eigenschaft address wird der Variable ad zugewiesen

print(ad["street"])    # Ausgabe: Bahnhofstraße 99

Source-Code 3: Zugriff auf Eigenschaften eines Unterobjektes

Verkürzt kann obiger Code auch so geschrieben werden:

# Zugriff auf die Eigenschaft street im Objekt der Eigenschaft addresse

print(result["address"]["street"])

Achtung: Die Schlüsselwerte in den Dictionaries muss genau mit jener im JSON-zusammenstimmen. Sonst gibt es einen Fehler.

### Listen

Sind in JSON-Objekten Listen vorhanden, so werden diese auch in Python als List-Objekte behandelt. Dadurch ergeben sich folgende Möglichkeiten, um auf die Elemente der Listen zuzugreifen:

* Index des Listenelementes
* Iteration über alle Listenelement mittels for-Schleife

Als Beispiel werden im folgenden die Kinder von Max Mustermann ausgegeben (siehe Abbildung 20). Der Index der Liste beginnt bei 0, so wie überall in Python.

print(result["children"][0]) # Ausgabe des ersten Kindes

print(result["children"][1]) # Ausgabe des zweiten Kindes

print(result["children"][2]) # Ausgabe des dritten Kindes

# Iteration über alle Elemente der Liste in der Eigenschaft children

for child in result["children"]:

    print(child) # Ausgabe des jeweiligen Kindes

Source-Code 4: Zugriff auf die Elemente einer Liste

Die Iteration über die Listen-Elemente hat den Vorteil, dass diese flexibler bei unterschiedlichen Antworten des Web-Service ist. Antwortet dieses bei Anfragen mit unterschiedlichen Personen, mit einer unterschiedlichen Anzahl an Kindern, so ist die Auswertung mit der zweiten Variante weiterhin möglich.

## Vorgangsweise

Die Herangehensweise, um eine Abfrage bei einem Web-Service zu erstellen, ist eigentlich immer dieselbe. Wichtig ist dabei, dass man sich die Dokumentation der Web-Service-API gut durchliest und dort vorhandene Beispiele nutzt.

* Auswahl eines Web-Service

Im Internet sind unzählig viele APIs zu finden. Auch gängige Dienste, wie Spotify, Twitter, etc. bieten entsprechende Schnittstellen an. Diese sind aber meist erst für erfahrene Personen gut anzusprechen. Eine Auswahl von Services und APIs sind im nächsten Abschnitt angeführt.

* Schnittstelle registrieren

Für einige Web-Services braucht man Zugangsdaten. Zumeist funktioniert das so, dass man von der Webseite einen Schlüssel (z.B. api-key) oder ein Personal Access Token (pat) bekommt. Diese Zusatzinformation ist dann bei jeder Anfrage als URL-Parameter mitzugeben. Wie man an diese Information kommt, ist Web-Service spezifisch und wird meist in der Dokumentation erklärt.

* Schnittstellen-Beschreibung lesen

Sehr, sehr wichtig ist das genaue Studium der Web-Service-API. Alle Web-Service bieten eine Beschreibung an, mit welchen URL-Parametern das Service aufgerufen werden kann und wie der JSON-Text der Antwort aussieht.

* Abfrage der Schnittstelle in einem Python-Skript versuchen

Wenn man mit dem Python-Skript beginnt, ist der erste Schritt, einmal eine sinnvolle Antwort vom Service mit Daten zu bekommen. Sollte an der Anfrage etwas nicht stimmen, so senden die meisten Web-Services einen Fehler in der Antwort und einen http-Statuscode der nicht 200 ist. Mit dem Skript in Source-Code 5 kannst du überprüfen, welchen Statuscode deine Anfrage bringt

* Daten der Schnittstelle auswerten und darstellen

War die Anfrage erfolgreich, so ist der nächste Schritt, dass man sich das JSON genau ansieht. Nach der Umwandlung in ein Python-Objekt können die Daten wie oben beschrieben ausgewertet werden.

import requests

# Hier die URL des Web-Service angeben, für welche die Anfrage getestet werden soll

url = "http://www.omdbapi.com/?t=Men+in+black"

# Anfrage an den Web-Service stellen

response = requests.get(url)

# Hier findet die Überprüfung statt, ob die Anfrage erfolgreich war

if response.status\_code == 200:

    print(f"Anfrage an {url} erfolgreich.")

else:

    print(f"Anfrage an {url} nicht erfolgreich:{response.status\_code}\n{response.text}")

Source-Code 5: Test-Code für eine Web-Service-API

## APIs zum Spielen

Folgende Web-Services bieten APIs an, welche über Python angesprochen werden können. Sie dienen zum Üben und ausprobieren der gelernten Inhalte. Die ersten APIs sind recht einfach abzufragen. Die Schwierigkeit steigt jedoch nach unten hin (stand 17.08.2022):

* Was ich tun kann: <https://www.boredapi.com/api/activity>

Sehr einfache API, welche auch ohne URL-Parameter verwendet werden kann. Sehr einfache Antwort ohne Unterobjekte und Listen.

* Open Movie Database: <http://www.omdbapi.com/?t=Men+in+black>

Einfache API mit URL-Parametern. Website bietet die Möglichkeit URLs mit Parametern zu generieren. Es wird ein API-Key als Parameter benötigt, welcher kostenlos angefordert werden kann.

* Cocktail-Rezepte: <https://www.thecocktaildb.com/api.php>

Normale API, welche durch URL-Parameter beeinflusst werden kann. Das Ergebnis enthält Listen mit Objekten

* Wetter-Daten: <https://openweathermap.org/api>

Anspruchsvollere API, muss mit URL-Parametern gesteuert werden. Ergebnis mit Unterobjekten und Listen mit Objekten. Kostenlose Registrierung notwendig.

* Spotify: <https://developer.spotify.com/documentation/web-api/>

Anspruchsvolle API, welche mit URL-Parametern zu steuern ist. Authentifizierung über OAuth nötig. Sehr flexibel und umfangreich

* Weitere APIs, die erkundet werden können:  
  <https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/>

# Arbeiten mit json

Das JSON-Format kann nicht nur dazu verwendet werden, um Daten zu einem Server zu senden oder von ihm zu empfangen. Das Format kann auch dazu genutzt werden, um Daten strukturiert in eine Datei zu schreiben und sie anschließend wieder zu lesen.

## Bibliothek

In der Standardbibliothek existiert das Modul json, das Python-Instanzen ins JSON-Format serialisieren bzw. aus dem JSON-Format erstellen kann. Das Modul json bietet die Funktionen dump, dumps bzw. load und loads an, um Daten zu speichern bzw. zu laden:

* dump(…): Speichert ein Python-Objekt als JSON-Text in eine Datei
* dumps(…): Generiert aus einem Python-Objekt den JSON-Text, welcher in eine Variable gespeichert werden kann
* load(…): Lädt aus einer Datei den JSON-Text und wandelt diesen in ein Python-Objekt
* loads(…): Lädt ein Python-Objekt aus einer Variable, welche einen gültigen JSON-Text beinhaltet

## Speichern von Listen und Dictionaries

Ein JSON-Objekt wird in Python als Dictionary abgebildet. Auf die Eigenschaften kann über den Namen der Eigenschaft zugegriffen werden. Umgekehrt muss aus einer Instanz (Objekt) ein Dictionary gemacht werden, bevor es als JSON-Datei gespeichert werden kann. Listen, Dictionaries und normale Werte können ohne Konvertierung als JSON gespeichert/geladen werden.

Um die Lesbarkeit der entstehenden JSON-Dateien zu erhöhen, kann man den json-Funktionen noch die Parameter indent und separators mitgeben. z.B:

json.dump(meine\_daten, file\_name, indent=2)

Source-Code 6: JSON-Dateien formatiert speichern

In Source-Code 7 werden Daten aus einer Liste und einem Dictionary als JSON-Text in die Datei „meine\_daten.json“ gespeichert. Wenn der Code ausprobiert wird, dann befindet sich nachher die Datei „meine\_daten.json“ im Verzeichnis wo die Daten gepseichert sind.

'''

daten in json speichern

'''

import json

my\_list = [1,2,3,"numbers"]

my\_dict = {

    1:"one",

    2:"two",

    3:"three"

}

my\_data = [my\_list, my\_dict]

with open("meine\_daten.json","w") as f:

    json.dump(my\_data, f)

Source-Code 7: Speichern von Dictionary und Liste als JSON in einer Datei

## Objekt in JSON speichern

Wird gewünscht, ein Objekt einer Klasse als JSON in eine Datei zu speichern, so muss die Klasse ein Kriterium erfüllen: Entgegen den normalen Regeln der objektorientierten Programmierung müssen die Attribute einer solchen Klasse *public* sein. Diese Art von Klasse ist eine spezielle, serialisierbare Klasse, welche auch Datenklasse genannt wird. Normalerweise werden solche Klassen auch nur zum Serialisieren und Deserialisieren von Daten verwendet.

Python bietet eine spezielle Möglichkeit um auf alle Attribute und deren Werte eines Objektes als Dictionary zuzugreifen. Hierfür gibt es das *Magic Attribute* \_\_dict\_\_. Somit können die Werte einfach wie oben in die Datei gespeichert werden:

'''

Objekt in json speichern

'''

import json

# Datenklasse, deshalb sind die Attribute public und nicht private

class Student:

    def \_\_init\_\_(self, matr, firstname, lastname):

        self.matr = matr

        self.firstname = firstname

        self.lastname = lastname

# Erzeugen der Instanz der Klasse Student

rudi = Student("1234", "Rudi", "Mair")

# Speichern der Daten des Objektes mithilfe des magischen Attributes \_\_dict\_\_

with open("mein\_objekt.json", "w") as f:

    json.dump(rudi.\_\_dict\_\_, f, indent="  ")

Source-Code 8: Speichern eines Objektes in eine JSON-Datei

Sind mehrere Objekte in einer Liste zusammengefasst, kann man mittels einer List-Comprehension helfen. (siehe Source-Code 9)

import json

# Datenklasse, deshalb sind die Attribute public und nicht private

class Student:

    def \_\_init\_\_(self, matr, firstname, lastname):

        self.matr = matr

        self.firstname = firstname

        self.lastname = lastname

# Erzeugen der Instanz der Klasse Student

rudi = Student(1234, "Rudi", "Mair")

caro = Student(9876, "Caroline", "Hofer")

alex = Student(3210, "Alexander", "Kogler")

# Liste mit allen Studenten

students = [rudi, caro, alex]

# Dictionaries mithilfe einer List-Comprehension erhalten

students\_for\_json = [student.\_\_dict\_\_ for student in students]

# Speichern der Studenten-Daten

with open("studenten.json", "w") as f:

    json.dump(students\_for\_json, f, indent="  ")

Source-Code 9: Speichern von mehreren Objekten in einer JSON-Datei

Wird obiger Code ausgeführt, so findet man in der Datei „studenten.json“ folgenden Inhalt:

[

  {

    "matr": 1234,

    "firstname": "Rudi",

    "lastname": "Mair"

  },

  {

    "matr": 9876,

    "firstname": "Caroline",

    "lastname": "Hofer"

  },

  {

    "matr": 3210,

    "firstname": "Alexander",

    "lastname": "Kogler"

  }

]

Das Laden der Werte aus einer JSON-Datei in eine Objekt-Struktur ist dann schon nicht mehr so einfach. Vor allem dann, wenn das Objekt Attribute-Werte mit weiteren Objekten beinhaltet. Hier hilft dann eine Suche im Internet, um eine Lösung für das Problem zu finden.

## JSON aus Datei lesen

Daten aus einer JSON-Datei können einfach eingelesen werden. Es entsteht dabei immer eine Struktur aus Python-Dictionaries, so wie das auch beim Erhalten von Antworten auf Web-Requests der Fall ist. (siehe JSON in Python-Objekte auf Seite 15). Im Source-Code 10 ist ein Beispiel, wie Daten in eine Datei gespeichert werden und anschließend wieder geladen.

'''

daten in json speichern

'''

import json

# Erstellen der Daten

my\_list = [1,2,3,"numbers"]

my\_dict = {

    1:"one",

    2:"two",

    3:"three"

}

my\_data = [my\_list, my\_dict]

# Speichern der Daten in die Datei "meine\_daten.json"

with open("meine\_daten.json", "w") as f:

    json.dump(my\_data, f)

# Laden der Daten aus der Datei "meine\_daten.json"

# Die Daten sind dann als Listen/Dicionary-Struktur in der Variable x1 zu finden

with open("meine\_daten.json", "r") as f:

    x1 = json.load(f)

# Ausgabe der geladenen Daten

print(f"{x1[0]=}")

print(f"{x1[1]=}")

Source-Code 10: Laden von JSON-Daten aus einer Datei

## JSON-Text

Neben der Möglichkeit Daten aus Python in JSON-Dateien zu speichern und von dort zu laden, können die Daten direkt in JSON-Text umgewandelt werden. Ebenso kann JSON-Text in einer Variablen in eine Python-Struktur geladen werden. Source-Code 11zeigt, wie das geht:

'''

daten in json konvertieren und wieder retour

'''

import json

# Erstellen der Daten

my\_list = ["Ich", "gehe", "gerne", "in", "die", "HTL"]

my\_dict = {

    1:"Max",

    2:"Selin",

    3:"Claudia"

}

my\_data = [my\_list, my\_dict]

# Umwandeln der Daten in einen JSON-Text

text = json.dumps(my\_data)

print("Werte umgewandelt in einen JSON-Text")

print(text)

print(f"text[0] stellt in diesem Fall das erste Zeichen des Textes dar, also: {text[0]}")

# Umwandeln des JSON-Textes wieder in eine Python-Struktur aus Listen und Dictionaries

data\_from\_json = json.loads(text)

print("Werte wieder in einer Python-Struktur")

print(data\_from\_json)

print(f"data\_from\_json[0] stellt in diesem Fall dens ersten Eintrag in der Liste dar, also: {data\_from\_json[0]}")

Source-Code 11: Umwandlen von Python-Listen/Dictionaries in JSON-Text und wieder retour

# Selbstüberprüfung

Welche Zeichen werden in einem JSON für die Syntax verwendet, um? Kreuze an

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ! | 🞏 | , | 🞏 | = | 🞏 | { } | 🞏 | ( ) | 🞏 |
| " | 🞏 | ? | 🞏 | ; | 🞏 | [ ] | 🞏 | : | 🞏 |

Wie lautet die URL für ein Aufruf an eine Web-Api, wenn dem Server zwei Parameter übergeben werden sollen? Die Parameter sind: name mit dem Wert Roland und alter mit dem Wert 5? Vervollständige folgende URL:

http://api.name-search.com/request?name=roland&alter=5

Die URL für einen Web-API Aufruf lautet <https://www.party-games.com/api/random>. Schreibe ein vollständiges Programm, welches eine Anfrage an diese Web-API-URL durchführt. Das Ergebnis der Anfrage soll anschließend so umgewandelt werden, dass mittels Python auf die einzelnen Elemente zugegriffen werden kann. Eine Auswertung ist nicht nötig. (3P)

import requests  
  
url = "https://www.party-games.com/api/random"  
  
data = requests.get(url).json()  
  
print(data)

Die Antwort auf einen Request an die URL <http://www.lehrer-wuerfelt.at/api?count=3> bringt folgende Antwort. Das Ergebnis wurde so umgewandelt, dass mittels Python darauf zugegriffen werden kann – die json-Struktur ist in der Variable result\_json zu finden.

{

"tries": [

{

"index": 1,

"result": 3,

"image": "http://www.lehrer-wuerfelt.at/image/drei.png"

},

{

"index": 2,

"result": 5,

"image": "http://www.lehrer-wuerfelt.at/image/fuenf.png"

},

{

"index": 3,

"result": 2,

"image": "http://www.lehrer-wuerfelt.at/image/zwei.png"

}

]

}

Was wird auf der Python-Console ausgegeben, wenn der folgende Befehl ausgeführt wird?

print(result\_json["tries"][1]["image"])

http://www.lehrer-wuerfelt.at/image/fuenf.png

Schreibe Code für das obige Beispiel, um alle gewürfelten Zahlen des Ergebnisses auf der Console auszugeben. Beachte, dass das Ergebnis des API-Requests je nach Aufruf variieren kann:

print("\n".join([str(trie["result"]) for trie in result\_json["tries"]]))

Benenne die Teile einer URL:

Timeline

Description automatically generated

Wie können an einen Webservice Daten für eine Anfrage mitgegeben werden?

Kann innerhalb des Pfades angegeben werden, oder als Query String (?name=daniel&age=17)

Wie funktioniert die Kommunikation zwischen Client und Webserver?

Über requests and responses

Was wird mit den Begriffen Serialisierung und Deserialisierung gemeint?

Daten werden serialisiert, wenn diese in byte umgewandelt werden. Deserialisierung ist die Rückabwicklung dieses Prozesses.

Was ist ein Query-String, wozu wird er verwendet und wie ist er aufgebaut?

Wird verwendet um Informationen/Argumente an eine Request mitzugeben. (?name=daniel&age=17)

Gib den Code an, wie eine Instanz einer Python-Klasse in einen JSON-Text serialisiert werden kann.

Entweder kann eine Instanz mittels pickle.dumps() serialisiert werden oder mittels json.dumps().