4 Daten untersuchen

In diesem Kapitel wollen wir die Geodaten ausschließlich untersuchen, also keine Änderungen an ihnen vornehmen oder neue erzeugen. Dies wird im nächsten Kapitel geschehen.

Bei den Untersuchungen dieses Kapitels gehen wir vom Großen ins Kleine. Als Erstes werden die Geodatensätze als Ganzes betrachtet, beginnend von den Arbeitsverzeichnissen über die darin enthaltenen Vektor- und Rasterdaten bis zur inneren Struktur wie Feldnamen oder Ausdehnung. Danach gehen wir in die Features eines Datensatzes selbst. Wir untersuchen den Inhalt der Tabelle im Detail. Wenn die Sachdaten der Tabelle behandelt sind, wenden wir uns als Letztes der Geometriespalte in der Tabelle zu und schauen die geometrischen Eigenschaften der Features an.

Wenn im Folgenden von Datensätzen gesprochen wird, sind vornehmlich Geodatensätze gemeint. Die Informationen dazu gelten in der Regel aber auch für nicht räumliche Datensätze.

4.1 Eigenschaften eines Datensatzes

Wenden wir uns zuerst den Datensätzen als Ganzes zu. Zunächst sollen einige Begriffe geklärt werden. Gegeben seien folgende Datensätze:
A:\Daten\Deutschland.gdb\Verwaltung\Gemeinde
A:\Bundeslaender.shp

Der Datensatz Bundeslaender.shp besteht aus mehreren Dateien (unter anderem Bundeslaender.shp, Bundeslaender.shx, Bundeslaender.dbf).

Dann würden folgende Bezeichnungen verwendet werden.

Bezeichnung	Erklärung	Beispiel(e)
Pfad	Der Weg zum Ordner, in dem eine Datei oder ein Datensatz liegt.	A:\Daten\Deutschland.gdb\Verwaltung A:\
Dateiname	Name der Datei in ihrem Pfad.	Gemeinde Bundeslaender.shp
Basisname	Dateiname ohne die Erweiterung. Dies sind alle Zeichen nach dem ersten Punkt im Dateinamen.	Gemeinde Bundeslaender
Erweiterung	Dateiname ohne Basisnamen. Dies beinhaltet auch den Punkt.	'' [Leerer Text] '.shp'

4.1.1 Inhalte eines Workspace

Der Workspace in ArcGIS kann ein Verzeichnis, eine Geodatenbank oder ein Feature-Dataset sein. Die Auflistung der darin enthaltenen Datensätze erfolgt über verschiedene `listXX()`-Methoden. Das Ergebnis ist eine Liste der Namen der gefundenen Datensätze. Die Auflistung erfolgt immer nur im aktuellen Workspace. Dieser wird durch `ap.env.workspace` definiert.

Bei Schlüsselwörtern ist in *arcpy* Groß- und Kleinschreibung nicht signifikant. In Python ist es generell jedoch üblich, Schlüsselwörter in Großbuchstaben zu schreiben.

Mit Suchmuster besteht die Möglichkeit, über den Namen der Dateien eine Aussage zu treffen. Der Asterisk '*' steht für eine beliebige Zeichenfolge; dies kann auch eine leere Zeichenfolge sein. Das Fragezeichen '?' steht für genau ein beliebiges Zeichen.

Beispiel: Auflisten Workspace

```
>>> import arcpy as ap
>>> ap.env.workspace=r'A:\Daten\Niedersachsen'
>>> ap.ListFeatureClasses('POLYGONE')
[u'landkreise.shp', u'gemeinden.shp', u'gemarkungen.shp', u'fluren.shp']
```

`ListWorkspaces(Suchmuster, Workspace_Typ)`	Listet die **weiteren Workspaces im aktuellen Workspace** auf. Es wird der vollständige Katalogpfad aufgelistet. All: Alle Arten von Workspaces (Vorgabe) Access: Personal-Geodatabases (*.mdb-Dateien) Coverage: Coverage-Verzeichnisse (Verzeichn. mit entspr. Dateien) FileGDB: FileGeodatabases (*.gdb-Verzeichnisse) Folder: Verzeichnisse für Shapefiles SDE: ArcSDE-Datenbanken
`ListFeatureClasses (Suchmuster, Geometrietyp)`	Listet **Vektordatensätze** auf. Für `Geometrietyp` können folgende Schlüsselwörter verwendet werden: *All*: Kein Filter, Alle (Vorgabe) *Point*: Punkt *Polyline:* Linien oder Bögen *Polygon*: Flächen *Annotation*: Annotationen in GDBs *Dimension*: Bemaßungen *Label*: Beschriftungen *Multipatch*: Nur Multipatch *Region*: Region *Arc*: Linien oder Bögen *Tic*: TIC (Coverages)

	Route: Routen *Edge*: Ecken *Line*: Linien oder Bögen *Node*: Knoten *Junction*: Knotenpunkte
`ListRasters(Suchmuster, RasterdatenFormate)`	Listet **Rasterdatensätze** auf. Für `RasterdatenFormate` können folgende Schlüsselwörter verwendet werden: *All*: Alle Rasterdatensätze (Vorgabe) *BMP*: Bitmap *GIF*: Graphic Interchange Format *IMG*: ERDAS IMAGINE *JP2*: JPEG 2000 *JPEG*: Joint Photographics Experts Group *PNG*: Portable Network Graphics *TIF*: Tagged Image File *GRID*: Grid-Dateien
`ListTables(Suchmuster, Tabellentyp)`	Listet die **Tabellen** auf.
`ListDatasets(Suchmuster, Datentyp)`	Listet `Geodatensätze` auf. Für `Datentyp` können folgende Schlüsselwörter verwendet werden: *All*: Alle Datensätze (Vorgabe) *Coverage*: ArcInfo-Coverages *Feature*: Coverage (inVerzeichnissen) oder Dataset (in Geodatenbanken *GeometricNetwork*: geometrische Netzwerke *Mosaic*: Mosaik-Datensätze *Network*: Netzwerk-Datasets *ParcelFabric*: Parcel Fabric (~ amerik. Liegenschaften) *Raster*: Rasterdatensätze *RasterCatalog*: Rasterkataloge *Schematic*: Schematic-Datensätze *Terrain*: Terrain-Datensätze *Tin*: TIN-Oberflächen *Topology*: Topologien *CAD*: CAD-Datensätze[2] (DWG, …?)
`ListFiles(Suchmuster)`	Listet **sämtliche Dateien**, unabhängig von der Anzeige im Inhaltsverzeichnis von ArcGIS Desktop.

[2] Im eigentlichen Abschnitt der erweiterten Hilfe nicht aufgezeichnet, wird jedoch im Tutorial verwendet (http://resources.arcgis.com/de/help/main/10.2/index.html#//00100000002q000000).

Beispiel: „Defekte“ Shape-Datensätze auffinden

Beim Transfer von Shape-Datensätzen geht manchmal eine der benötigten Teil-Dateien verloren. Es fehlt z. B. die Sachdatentabelle (*.dbf) oder die Index-Datei (*.shx). ArcGIS zeigt diese Geodatensätze zwar an, sie lassen sich aber verständlicherweise nicht öffnen. Mit diesem Skript finden Sie solche Datensätze. Zum Testen wurden im Windows Explorer bei drei Shape-Datensätzen in zweien einmal die `.shx` und einmal die `.dbf` in `.xxx` umbenannt.

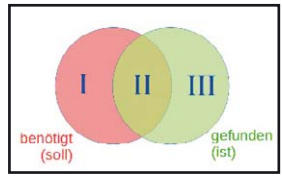
```
# -*- coding: UTF-8 -*-

import arcpy as ap # arcpy als ap laden
from os.path import splitext # Funktion "Split Extension" laden
import string as s # Modul für Zeichnfolgen laden

# Welche Dateierweiterungen werden gebraucht?
noetigeDateien = ['.shp','.shx', '.sbn', '.dbf','.prj']

# Das zu untersuchende Verzeichnis (bitte editieren)
root = r'A:\Daten'
# ArcPy wechselt in das Verzeichnis
ap.env.workspace = root
# TODO: Ist der Workspace tatsächlich ein Verzeichnis?

# Alle Shape-Datensätze listen
shps = ap.ListFeatureClasses()
# Mit jedem Datensatz durchführen
for shp in shps:
    basename = splitext(shp)[0] # Nur der Name der Datei ohne Erweiterung
    # Die Erweiterungen aller mit diesem Basisnamen gefundenen Dateien
    gefundeneDateien= [splitext(f)[1] for f in ap.ListFiles(basename+'.*')]
    # Abgleich der Mengen benötigt und gefunden
    fehlendeDateien = list(set(noetigeDateien) - set(gefundeneDateien))
    weitereDateien = list(set(gefundeneDateien) - set(noetigeDateien))
    if fehlendeDateien:
        print 'Im Datensatz %s fehlen:' %shp
        print '\t', s.join(fehlendeDateien, '\n\t')
        if weitereDateien:
            print u'Zusätzlich gefunden wurden aber:'
            print '\t', s.join(weitereDateien, '\n\t')
```

Codezeile	Erläuterung
4	Lade aus dem Modul `os.path` (Pfad- und Dateistrukturen des Betriebssystems) die Funktion zum Abtrennen der Dateierweiterung (engl. *extension*).
5	Modul für Zeichenfolgen laden.
8	Die Erweiterungen der obligatorisch nötigen Dateien für einen Shape-Geodatensatz als Liste. Die Projektionsdatei (*.prj) ist zwar technisch nicht unbedingt notwendig, aber doch sehr wichtig. Sie können diese Liste anpassen, zum Beispiel die cpg-Datei zur Definition des ASCII-Datensatzes der Sachdaten hinzufügen.
11	Festlegung des zu untersuchenden Verzeichnisses (engl. *folder*). Es muss der absolute Pfad sein.
13	`arcpy` definiert über die Variable `root` nun das Verzeichnis, in dem gearbeitet wird. Dieses Einlesen dauert ebenso lange wie in ArcCatalog, wenn ein Verzeichnis im TOC aufgeblättert wird. Für das Betriebssystem, in Python repräsentiert durch `os.getcwd()` (engl. *cwd = current working directory*), hat sich noch nichts geändert! Es könnte mit `os.chdir(root)` nachgeführt werden.
14	Wie dies zu realisieren ist, wird im Beispiel „Vergleich der Eigenschaften von Arbeitsverzeichnissen“ im Abschnitt 4.1.3.1 gezeigt.
17	Alle Vektordatensätze im Verzeichnis auflisten.
19	Einleitung der Schleife.
20	Der Dateiname wird durch `os.path.splitext` in eine Liste aufgespalten: Erstes Element ist der Basisname, zweites Element die Erweiterung mit dem Punkt davor.
22	Auflistung aller Erweiterungen zu diesem Basisnamen. In der Listenabstraktion wird ‚hinten‘ zunächst nach allen Dateiennamen passend zum Basisnamen aufgelistet. Die einzelnen Elemente dieser Liste werden als `f` an den ‚vorderen‘ Teil übergeben. Dort wird jeder Dateiname in Basisnamen und Erweiterung zerlegt. Nur die Erweiterung wird übernommen und in die Liste eingetragen. 
24	Die fehlenden Erweiterungen (I) sind die nötigen Erweiterungen ohne die gefundenen Erweiterungen. Eine Operation aus der Mengenlehre: Umwandlung der beiden Listen in Mengen. Die Differenzmenge wird gebildet und das Ergebnis wieder in eine Liste zurückverwandelt.
25	Vielleicht wurden Dateien aus Versehen umbenannt. Daher die Auflistung weiterer Erweiterungen aus den gefundenen Erweiterungen ohne die nötigen Erweiterungen (III).

26 + 29 Bei der Verzweigung wird der Umstand genutzt, dass leere Kollektionen (zu denen Listen gehören) als `False` gelten. Die Verzweigung wird somit nur bei vorhandenen Listen für die Mengen I und III genutzt.

27-31 Textausgabe der Ergebnisse. Ergibt im Testdatensatz Meldungen dieser Art:

```
Im Datensatz BRD.shp fehlen:
 .dbf
Zusätzlich gefunden wurden aber:
 .cpg
 .xxx
Im Datensatz DDR.shp fehlen:
 .shx
Zusätzlich gefunden wurden aber:
 .cpg
 .xxx
```

4.1.1.1 Alternativen mit reinem Python

Während eine *list*-Funktion von ArcPy die Daten so betrachtet, wie Sie es unter ArcCatalog gewohnt sind, sehen die Funktionen des reinen Python die Daten in der Weise des Windows Explorers.

In Fällen von Verzeichnissen mit vielen Geodatensätzen kann daher für den ersten Überblick eine der Funktionen aus dem Modul `glob` geeigneter sein, da die Verarbeitungsgeschwindigkeit sich deutlich erhöht.

Bei der Funktion `glob.glob(Suchmuster)` wird das aktuelle Verzeichnis untersucht. Die Funktion `glob.glob1(Verzeichnis, Suchmuster)` hingegen sucht in beliebigen Verzeichnissen. Beim Suchmuster lassen sich mehr Platzhalter nutzen als in arcpy.

Platzhalter	Bedeutung	Beispiel
*	Jede Zeichenfolge.	`*.shp` trifft alle Basisnamen mit der Dateierweiterung shp.
?	Eine Zeichenfolge.	`????.*` alle Dateien, die vier Zeichen im Basisnamen haben.
`[]`	Jedes Zeichen in den eckigen Klammern.	`[ABC]*` alle Dateien, die mit A, B oder C anfangen.
`[..]`	Jedes Zeichen im Bereich zwischen den beiden Zeichen in den eckigen Klammern.	`[A..Z]*` alle Dateien, die mit einem Großbuchstaben beginnen. [0..9]* alle Dateien mit einer Zahl am Anfang.
`[!]`	Jedes Zeichen, das nicht in den eckigen Klammern gelistet ist.	`[!ABC]*` alle Dateien, die nicht mit A, B oder C anfangen.

Beispiel: Nutzung der glob-Funktionen

```
>>> import glob
>>> root = r'A:\Daten'
>>> # glob1-Funktion
...
>>> # Alle TIF-Dateien in root, deren Basisname mit '0','1' oder '2' endet
...
>>> tifs = glob.glob1(root,'*[0..2].tif')
>>> for tif in tifs : # Textausgabe zur Kontrolle
...     print tif
...
srtm_39_02.tif
>>> # glob-Funktion
...
>>> import os      # modul wird gebraucht, um ...
>>> os.chdir(root) # ...das Verzeichnis zu wechseln
>>> # Alle Shape-Dateien im aktuellen Verzeichnis, die nicht mit 'B'
                                                              anfangen
...
>>> shps = glob.glob('[!B]*.shp')
>>> for shp in shps: # Textausgabe zur Kontrolle
...     print shp
...
DDR.shp
```

4.1.1.2 Untersuchen ganzer Verzeichnisstrukturen: `os.walk` und `arcpy.walk`

Im Modul `os` gibt es mit der Funktion `walk` eine Funktion, die sämtliche Verzeichnisse unterhalb eines Einstiegpunkts mit den darin enthaltenen Dateien auflistet. Jedes dieser Listenelemente besteht wiederum aus einer Liste aus drei Elementen: Das erste Element ist der Name des aktuell untersuchten Verzeichnisses. Das zweite Element ist eine Liste mit allen Unterverzeichnissen des aktuellen Verzeichnisses. Das dritte Element ist ebenso eine Liste mit allen Dateien im aktuellen Verzeichnis. Es gibt keinen Parameter für Filter.

```
>>> alles = os.walk(r'A:\Daten\Gemeinden')
>>> for jedes in alles:
...     print jedes
...
('A:\\Daten\\Gemeinden', ['Baden-W\xfcrttemberg', 'Bayern', 'Ber-
lin', 'Brandenburg', 'Bremen', 'Hamburg', 'Hessen', 'Mecklen-
burg-Vorpommern', 'Niedersachsen', 'Saarland', 'Nordrhein-Westfa-
len',
'Rheinland-Pfalz', 'Sachsen', 'Sachsen-Anhalt', 'Th\xfcringen',
'Schleswig-Holstein'], [])
```

```
('A:\\Daten\\Gemeinden\\Baden-W\xfcrttemberg', [], ['Alb-Donau-
Kreis.shp', 'Alb-Donau-Kreis.shx', 'Alb-Donau-Kreis.dbf', 'Alb-Do-
nau-Kreis.CPG', 'Alb-Donau-Kreis.prj', 'Alb-Donau-Kreis.shp.xml',
'Karlsruhe.sbx', 'Konstanz.shp', 'Alb-Donau-Kreis.sbn', 'Alb-Donau-
Kreis.sbx', 'Konstanz.shx',....
```

Eine solche verschachtelte Liste ist am einfachsten über `for`-Schleifen zu verarbeiten, wie es im nachfolgenden Beispiel der anderen *walk*-Funktion gezeigt wird.

Seit ArcGIS 10.1 SP1 gibt es mit `arcpy.da.Walk` eine Funktion, die dieses Verhalten für Geodaten imitiert.

Diese Funktion kann Strukturen innerhalb von Geodatenbanken auflisten. Zudem stellt sie zwei zusätzliche Filter zur Verfügung: datatype und type. Bei `datatype` sind als Schlüsselwörter zulässig: `Any`, `CadDrawing`, `CadastralFabric`, `Container`, `FeatureClass`, `FeatureDataset`, `Geo`, `GeometricNetwork`, `LasDataset`, `Layer`, `Locator`, `Map`, `MosaicDataset`, `PlanarGraph`, `RasterBand`, `RasterCatalog`, `RasterDataset`, `RelationshipClass`, `RepresentationClass`, `SchematicDataset`, `Style`, `Table`, `Terrain`, `Text`, `Tin`, `Tool`, `Toolbox`, `Topology`. Dies entspricht in etwa der Zusammenfassung von `listDatasets`, `listWorkspaces` und `listTables`. Der Filter `type` hingegen ist in etwa die Zusammenfassung von `ListFeatureClasses` und `listRasters`: und behandelt somit das Datenformat. Gültige Schlüsselwörter sind: `ALL` oder `ANY`, `Multipatch`, `Multipoint`, `Point`, `Polygon`, `Polyline`, `BIL`, `BIP`, `BMP`, `BSQ`, `DAT`, `GIF`, `GRID`, `IMG`, `JP2`, `JPG`, `PNG`, `TIF`.

Beide Filter lassen mehrere Schlüsselwörter in Form von Listen zu. Dabei gilt innerhalb der Liste eine ODER-Bedingung, zwischen den beiden Listen aber eine UND-Bedingung.

Beispiel: Eigene Funktion zum rekursiven Auflisten von Geodaten

```
1 def listRecursive(workspace, datatypes='All', types= 'ALL'):
2     """Beschreibung in der Buchdarstellung ausgelassen"""
3     files = []
4     walkResult = ap.da.Walk(workspace, datatype=datatypes, type=types)
5     for pfad, subDirs, dateien in walkResult:
6         for data_name in data_names:
7             files.append(os.path.join(path, data_name))
8     return files
```