Aufgabe 1:

```python
# version 3.8.0
import os
import urllib.request
import random
import gzip
import re
import pprint
import sqlite3

pp = pprint.PrettyPrinter(indent=2)

filePath = "./ftpSourceView.txt";

fileContent = open(filePath).readlines();

sourceUrl = fileContent[0].split()[1]
print("URL: " + sourceUrl)

unzippedFiles = []


def get_data(nr_files):
    pp.pprint(">>>>> STARTING DOWNLOAD <<<<<")
    toReturn = []
    dead_files = 0
    for i in range(nr_files):
        data = {}
        fullUrl = sourceUrl + fileContent[2:][random.randint(0, len(fileContent) - 2)].split('"')[1]
        try:
            content = gzip.GzipFile(fileobj=urllib.request.urlopen(fullUrl)).readlines()
            matched_header = re.search(r"HEADER\s*([/\w\s-]*)\d{2}-\w{3}-\d{2}\s*(\d*\w*)", content[0
            data["header"] = matched_header.group(1).strip()
            data["pdbid"] = matched_header.group(2)
            data["X"] = 0
            data["Y"] = 0
            data["Z"] = 0
            data["title"] = []
            for line_src in content[1:]:
                line = line_src.decode()
                match = re.match(r"TITLE\s\d?\s?(.*)", line)
                if match != None:
                    data["title"].append(match.group(1).strip())
                elif line.startswith("ATOM"):
```

```
                    split = line.split();
                    data["X"] += float(split[6])
                    data["Y"] += float(split[7])
                    data["Z"] += float(split[8])
                else:
                    match = re.match(r".*CHAIN:\s(.*);", line)
                    if match != None:
                        data["chain"] = match.group(1)
            data["title"] = " ".join(data["title"])
            pp.pprint(str(((i + 1) / nr_files) * 100) + "% completed")
            toReturn.append(data)
        except:
            pp.pprint("got a faulty file, continuing ...")
            dead_files += 1
    pp.pprint(">>>>> DONE <<<<<")
    pp.pprint("!! Encountered " + str(dead_files) + " files.")
    return toReturn

def insert_data(data, database_file):
    pp.pprint(">>>>> STARTING DB INSERT <<<<<")
    con = sqlite3.connect(database_file)
    cur = con.cursor()
    for i, element in enumerate(data):
        cur.execute('insert into proteins (' +
                    'header, title, pbid, chain, X, Y, Z' +
                    ') VALUES (?,?,?,?,?,?,?)',
                    [element["header"], element["title"], element["pdbid"],
                     element["chain"], element["X"], element["Y"], element["Z"]])
        pp.pprint(str(((i + 1) / len(data)) * 100) + "% completed")
        con.commit()
    con.close()
    pp.pprint(">>>>> DONE WITH DB INSERT <<<<<")

def setup_db(database_file):
    con = sqlite3.connect(database_file)

    cur = con.cursor()

    cur.execute('create table if not exists proteins (' +
                'id integer primary key autoincrement,' +
                'header text,' +
                'title text,' +
                'pbid text,' +
                'chain text,' +
                'X float,' +
                'Y float,' +
```

```
90                  'Z float' +
91                  ')')
92      con.commit()
93      con.close()
94
95  database_file = "./protein.db"
96
97  setup_db(database_file)
98  insert_data(get_data(10), database_file)
```

SQL script:

```
1  # version 3.8.0
2  import sqlite3
3  import pprint
4
5  pp = pprint.PrettyPrinter(indent=2)
6
7  con = sqlite3.connect('./protein.db')
8
9  cur = con.cursor()
10
11 result = cur.execute("select * from proteins where pbid like ('%5%')")
12
13 for line in result.fetchall():
14     pp.pprint(line)
15
16 con.commit()
17 con.close()
```

Aufgabe 2:

```
# version 3.8.0
import urllib.request
import re
import pprint

PP = pprint.PrettyPrinter(indent=1)

BASE_URL = "https://www.icd-code.de/icd/code/"
STARTING_URL = "ICD-10-GM.html"
LIST_OF_IDS = ["H", "J"]


def build_tree(url, ids, base):
    tree = {}
    file = urllib.request.urlopen(url)
    for j in file.readlines():
        for i in ids:
            identification = re.search(r"<a href=\"(" + i + r"\d\d-" + i +
                                       r"\d\d.html)\">(\w\d\d-\w\d\d)<\/a><\/td><td>(.+?)<a",
                                       str(j))
            if identification is None:
                continue
            key = identification.group(2)
            tree[key] = {
                "link": base + identification.group(1),
                "detail": base + identification.group(3),
                "more": parse_second_level(i, base, base + identification.group(1))
            }
    PP.pprint(tree)

# def parse_first_level(url, ids, base, tree):


def parse_second_level(cur_id, base, link):
    tree = {}
    for line in urllib.request.urlopen(link).readlines():
        matches = re.findall(r"<a href=\"(" + cur_id +
                             r"\d\d-\w\d\d.html)\">(\w\d\d-\w\d\d)<\/a>(.+?)(<br\/>|<\/td>)",
                             str(line))
        if matches is None:
            continue
        for match in matches:
            key = match[1]
```

```
44              sublink = base + match[0]
45              tree[key] = {
46                  "link": sublink,
47                  "detail": match[2].strip(),
48                  "more": parse_third_level(cur_id, base, sublink)
49              }
50      return tree
51
52
53  def parse_third_level(cur_id, base, link):
54      tree = {}
55      for line in urllib.request.urlopen(link).readlines():
56          matches = re.findall(r"<a href=\"(" + cur_id +
57                               r"\d\d(?:.-\*?)?.html)\">(" + cur_id +
58                               r"\d\d(?:.-\*?)?)<\/a>(.+?)(<br\/>|<\/td>)", str(line))
59          if matches is None:
60              continue
61          for match in matches:
62              key = match[1]
63              sublink = base + match[0]
64              tree[key] = {
65                  "link": sublink,
66                  "detail": match[2].strip(),
67                  "more": parse_fourth_level(cur_id, base, sublink)
68              }
69      return tree
70
71
72  def parse_fourth_level(cur_id, base, link):
73      tree = {}
74      for line in urllib.request.urlopen(link).readlines():
75          matches = re.findall(r"<tr><td valign=\"top\"><div class=\"code_bottom\">(" + cur_id +
76                               r"\d\d\.\d\d?(?:-?\*?)?)<\/div><\/td><td colspan=\"2\">(.+?)<\/td><\/tr
77                               str(line))
78          if matches is None:
79              continue
80          for match in matches:
81              key = match[0]
82              tree[key] = match[1].strip()
83      return tree
84
85
86  build_tree(BASE_URL + STARTING_URL, LIST_OF_IDS, BASE_URL)
```

Aufgabe 2:

```python
# version 3.8.0
import re
import pprint

pp = pprint.PrettyPrinter(indent=2)

emailFile = open("./imgtSpeciesGeneInformation.txt", "r")

# pattern = re.compile(r"((\w*\s?)*);((\w*\d*-?)*);")

res_dict = {}
avail_gen = []
line_nr = 0

for i in emailFile.readlines()[1:]:
    mo = re.search(r"((\w*\s*)*);((\w|\d|-|/)*);.*", i)
    line_nr += 1
    if mo != None:
        if len(mo.group(3)) < 4:
            continue
        if mo.group(1) not in res_dict:
            res_dict[mo.group(1)] = {}
        if mo.group(3)[3] not in res_dict[mo.group(1)]:
            res_dict[mo.group(1)][mo.group(3)[3]] = 0
        res_dict[mo.group(1)][mo.group(3)[3]] += 1
        avail_gen.append(mo.group(3)[3])

avail_gen = set(avail_gen)

for g in avail_gen:
    for k in res_dict:
        if g not in res_dict[k]:
            res_dict[k][g] = 0

# TODO: final calculation missing

pp.pprint(res_dict)

emailFile.close()
```