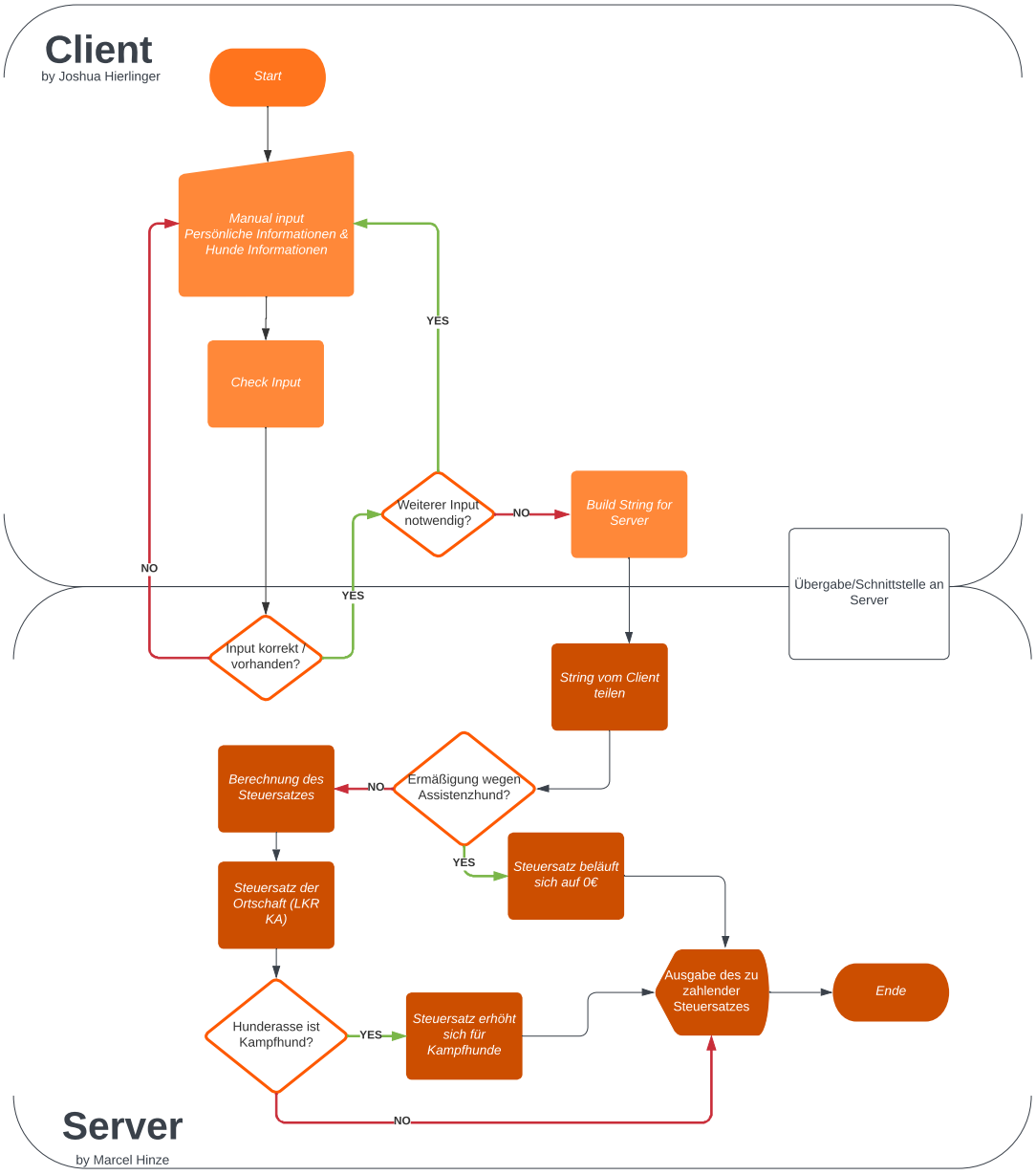
Hund anmelden

Bei dem geschriebenen Code geht es um die Hundeanmeldung (Berechnung und Rückgabe der Hundesteuer). Dabei werden Persönliche Informationen von dem Hundehalter abgefragt, als auch die Informationen des Hundes. Hierbei gibt es Pflichtinformationen und freiwillige Angaben, wie auch im Standardverfahren von einer realen Anmeldung der Gemeinden. Der Code ist nur auf den Landkreis Karlsruhe und insgesamt 20 verschiedene „normal“- und Kampfhundarten beschränkt. Je nach Postleitzahl wird ein unterschiedlicher Steuersatz zugewiesen, daher ist zum Beispiel die PLZ eine Pflichtinformation des Hundehalters und eine genaue zugewiesene Zahl für einen Ort. Assistenzhunde wurden ebenfalls beachtet, welche bei Bestätigung bei der Abfrage nicht Steuerpflichtig sind.



Client

Joshua Hierlinger

Durch „import Server as Server“ wird der die Datei „Server“ importiert, so dass in der Datei auf Funktionen vom Server zugegriffen werden können. „as Server“ deklariert die Variable, welche verwendet wird um diese Datei anzusprechen..

```
import Server as Server
```

Folgend werden globalen Variablen bestimmt. Hiermit können alle Funktionen der Datei auf die Variablen zugreifen und nicht abgeschottet nur eine Funktion. Für die Initialisierung der Variablen ist ein Wert notwendig. „None“ ist hierfür ein Platzhalter.

```
Name = None
Vorname = None
GeburtsdatumHuman = None
Straße = None
Hausnummer = None
Ort = None
PLZ = None
NameDog = None
GeburtsdatumDog = None
Dograsse = None
Ermäßigung = None
```

Die „def buildStringForServer():“ Funktion ist zur Erstellung des Strings für den Server verantwortlich, indem mit einem f“String die Variablen zusammengesetzt werden und durch Semikolons separiert werden. Die Persönlichen Informationen sowie die Hunde Informationen werden zusätzlich auch ein Komma getrennt, so dass eine Unterscheidung möglich ist. Die Auswahl auf einen solchen Autauschstring fiel, da ein Semikolon sowie ein Komma nicht durch die eingaben des Benutzers vorhanden sind. Die Funktion gibt den vollendeten String zurück.

```
def buildStringForServer():
    buildetStiring = f"{Name};{Vorname};{GeburtsdatumHuman};{Straße};{Hausnummer};{Ort};
{PLZ};,{NameDog};{GeburtsdatumDog};{Dograsse};{Ermäßigung}"
    return buildetStiring
```

„getInformations“ ruft die Funktion auf um die Variablen des User abzufragen. Gleichzeitig wird mit „print“ eine für den User übersichtlichere Eingabe ermöglicht, da dieser so definitiv weiß welche Informationen in dem Moment einzugeben sind. In Bezug auf die persönlichen Angaben oder des Hundes.

```
def getInformations():
    print("Persönliche Angaben")
    getName()
    getVorname()
    getBirthHuman()
    print("Wohnanschrift")
    getStreet()
    getStreetNumber()
    getCity()
    getPLZ()
    print("Angaben zum Hund")
    getNameDog()
    getBirthDog()
    getDogRase()
    getErmäßigung()
```

Mit diesen Funktionen wird eine Eingabe des Benutzers erwartet bzw. abgefragt. Z.B.: „Name*: Müller“. Durch „def getInformations“ werden diese nacheinander aufgerufen. Sobald der User auf eine Abfrage eine Eingabe getätigt hat, so wird die nächste abfrage aufgerufen. Gleichzeitig wird durch ein Stern (*) dem User mitgeteilt, dass es sich über ein Pflichtfeld handelt und ausgefüllt werden muss. Beispielsweise ist der der Name des Hundes nicht von Bedeutung und kann dadurch vernachlässigt werden.

```
def getName():
    global Name
    Name = input("Name*: „)

def getVorname():
    global Vorname
    Vorname = input("Vorname*: ")

def getBirthHuman():
    global GeburtsdatumHuman
    GeburtsdatumHuman = input("Geburtsdatum*: ")

def getStreet():
    global Straße
    Straße = input("Straße*: ")

def getStreetNumber():
    global Hausnummer
    Hausnummer = input("Hausnummer*: ")

def getCity():
    global Ort
    Ort = input("Ort*: ")

def getPLZ():
    global PLZ
    PLZ = input("PLZ*: ")

def getNameDog():
    global NameDog
    NameDog = input("Name: ")

def getBirthDog():
    global GeburtsdatumDog
    GeburtsdatumDog = input("Geburtstag*: ")

def getDogRase():
    global Dograsse
    Dograsse = input("Hunderasse*: ")

def getErmäßigung():
    global Ermäßigung
    Ermäßigung = input("Steuerbefreitung/Ermäßigung (Ja/Nein)*: „)
```

Durch diesen „try und expcept“-Block werden Fehler im Programm abgefangen. Sollte ein Error während der Ausführung entstehen, so greift dieser Block und der im except enthaltene Code wird ausgeführt. Hier wird der Ablauf des Programmes „gesetzt“ zunächst werden die Informationen abgefragt, dann der String gebildet und der gebildete String dem Server übergeben, welcher dann im „serverHandler“ den weiteren Ablauf des Servers regelt.

```
try:
    getInformations()
```

```
    serverString = buildStringForServer()
    print(Server.serverHandler(serverString))
except:
 print("An exception occurred.")
```

Server

Marcel Hinze

Die Bibliothek von „datetime“ wird importiert und der Klasse bereitgestellt.

```
import datetime
```

„dogdict“ und „citydict“ sind Dictonarys, bei denen den Werten auf der linken Seite ein Wert zugewiesen wird (rechts). Dies ist nötig um z.B. zu unterscheiden welche Hunderasse ein Kampfhund ist und welche nicht - Oder um die Steuersätze nach Ort unterscheiden zu können (nach PLZ - weil eindeutige Zuordnung)

```
dogdict = {
    "american staffordshire terrier": True,
    "pitbull terrier": True,
    "bullterrier": True,
    "bullmastiff": True,
    "staffordshire bullterrier": True,
    "cane corso": True,
    "dogo argentino": True,
    "bordeaux dogge": True,
    "fila brasileiro": True,
    "mastin espanol": True,
    "französische bulldogge": False,
    "labrador": False,
    "chihuahua": False,
    "australian shepherd": False,
    "rottweiler": False,
    "border collie": False,
    "golden retriever": False,
    "rhodesian ridgeback": False,
    "mops": False,
    "berner sennenhund": False
}

citydict = {
    75015: 108,
    76359: 96,
    69254: 78,
    76275: 96,
    76287: 90,
    76337: 108,
    76307: 66,
    76327: 72,
    75045: 108,
    76356: 96,
    76297: 84,
    76344: 84,
    76351: 72,
    76707: 72,
    76676: 60,
    76689: 69,
    76646: 96,
    75053: 48,
    75038: 90,
    76703: 87,
```

```
    76698: 61,
    76707: 48,
    68753: 96,
    76661: 96,
    76709: 72,
    76669: 90,
    76684: 75,
    75059: 72
}
```

„serverHandler(clientString):“ wird vom Client als zentrale Funktion und als einzige Schnittstelle aufgerufen. Hier wird der zusammengesetzte String vom Client beim Funktionenaufruf als Parameter mitgegeben.

```
def serverHandler(clientString):
    dogInformations = splitStringFromClient(clientString, True)
    personalInformation = splitStringFromClient(clientString, False)
```

Hier geht es um die Steuerbefreiung für einen Assistenzhund. Sobald die gegebene Antwort des Users Ja auf die Frage der Steuerbefreiung ist, beträgt die Hundesteuer Null Euro. Wird Nein als eingäbe getätigt, so werden die Funktionen „getRegionTax“, „getDogRaceSeperation“ und „getDoxTax“ aufgerufen. Schlussendlich gibt die Funktion die Hundesteuer an den Server zurück, um den String im Aufruf zu vollenden.

```
    if(dogInformations[3].lower() == "nein"):
        regionTax = getRegionTax(personalInformation)
        dogRaceSeperation = getDogRaceSeperation(dogInformations)
        dogTax = getDogTax(dogRaceSeperation, regionTax)
    else:
        dogTax = 0
    return dogTax
```

„splitStringFromClient(clientString, state):“ trennt die jeweiligen Informationen des Strings vom Client von einander und speichert diese im Array („seperations“ & dog- und personalInformation). Zunächst wird beim Komma getrennt, folgend bei den Semikolons. Diese Methode wird Zweimals aufgerufen und der Parameter state (Boolean) wechselt hierbei um beide Arrays in der serverHandler Funktion zu erhalten.

```
def splitStringFromClient(clientString, state):
    try:
        seperations = clientString.split(',')
        if(state):
            dogInformation = seperations[1].split(';')
            return dogInformation
        else:
            personalInformation = seperations[0].split(';')
            return personalInformation
    except:
        print("Error in SplitStringFormClient“)
```

Es wird aus dem Dictionary „citydict“ die entsprechende PLZ ausgewählt und der Steuersatz zurückgegeben.

```
def getRegionTax(personalInformations):
    regionTax = citydict[int(personalInformations[6])]
    return regionTax
```

Hierbei wird der Unterschied zwischen den Hunderassen geprüft. Falls es sich um einen Kampfhund (dogRaceSeperation == True) handelt, wird die Steuer mal fünf gerechnet, falls nicht bleibt die Steuer gleich hoch wie der regionale Steuersatz ist.

```
def getDogTax(dogRaceSeperation,regionTax):
    dogTax = None
    if(dogRaceSeperation == True):
       dogTax = int(regionTax) * 5
    else:
        dogTax = int(regionTax)
    if(dogTax != None):
        return dogTax
    else:
        return "Error in DogTax"
```

Ruft ab, welche Hunderasse ein Kampfhund ist und welcher nicht, durch die Abfrage in dogDict.

```
def getDogRaceSeperation(dogInformations):
    dogRaceSeperation = dogdict[dogInformations[2]]
    return dogRaceSeperation
```

Mit „checkDate(value):“ wird festgelegt, ob sich die Geburtsdaten der Eingaben im Format von TT.MM.YYYY angegeben wurde (Europäische Datumssystem). Ebenfalls werden „Edgecases“ abgebrüht wie z.B. ob das Datum, welches angegeben wurde in der Zukunft liegt oder den Februar, welche z.B. keinen 30. Tag im Monat hat. Als Rückgabe wird ein Fehlertext sowie ein Boolean zurückgegeben, welcher für dem Client von Bedeutung ist. Ist die Eingabe korrekt, so wird ein leerer String und True zurückgegeben.

```
def checkDate(value):
    try:
        day, month, year = map(int, value.split('.'))
        geburtstag_obj = datetime.date(year, month, day)

        try:
            date = value.strip()
            datetime.datetime.strptime(date, "%d.%m.%Y")
        except ValueError:
            return "Ungueltiges Datumsformat. Bitte geben Sie den Geburtstag im Format TT.MM.JJJJ
            ein.“, False

        if geburtstag_obj >= datetime.date.today():
        return "Ungueltiges Datumsformat. Der Geburtstag muss in der Vergangenheit liegen.“, False
```

Überprüft, ob das Datum in der Vergangenheit liegt.

```
      if month == 2 and day > 28:
          leap_year = (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
          if (day == 29 and not leap_year) or (day > 29):
            return "Ungueltiges Datumsformat. Bitte geben Sie ein gueltiges Geburtsdatum ein.“, False
        return "", True

    except:
        return "Bei der Datumsüberprüfung lief etwas schief. Bitte wenden Sie sich an einen
Admin“,False
```

Hier werden die Eingaben des Benutzers geprüft. Zunächst wird geprüft ob die jeweilige Eingabe Notwenig ist durch den Parameter „nessesary“. Folgend wird geprüft ob die Eingabe etwas

enthält. Darauf wird validiert, welcher Eingabe geprüft werden soll durch den Parameter „nameCheck“, da eine unterschiedliche Prüfung je nach Eingabe notwendig ist. Rückgabewerte sind dann True oder False, welche im Client verwendet wird um eine Fehlermeldung anzuzeigen und eine weitere Eingabe zu erwarten oder die nächste Eingabe abzufragen.

```
def checkInput(value,nessesary, nameCheck):
    if(nessesary):
        if(value == None or value == ''):
            return False
        else:
            if("Geburtstag" in nameCheck):
                result = checkDate(value)
                return result
            elif(nameCheck == "Hunderasse"):
                if(value in dogdict):
                    return True
                else:
                    return False
            elif(nameCheck == "Ermaessigung"):
                if(value.lower() == "ja" or value.lower() == "nein"):
                    return True
                else:
                    return False
            elif(nameCheck == "PLZ"):
                if(value.isnumeric and int(value.strip()) in citydict):
                    return True
                else:
                    return False
            elif(nameCheck == "Hausnummer"):
                if(value.isnumeric()):
                    return True
                else:
                    return False
            else:
                return True
    else:
        if(value != None):
            return True
        else:
            return False
```