Einführung in Python

Arne Hüffmeier

Hangman

Nachdem wir das erste Spiel programmiert haben, starten wir mal gleich mit dem Nächsten.

Auch hier fangen wir mal wieder mit den Fragen an.

Wichtige Fragen, die man sich vorher stellen sollte

1. Welche Regeln hat das Spiel?
2. Wie sieht die Eingabe aus?
3. Wie sieht die Ausgabe aus?
4. Was müssen wir uns speichern?
5. Wie ist der Ablauf?

1) Welche Regeln hat das Spiel?

- Das Programm wählt sich ein Wort aus.
- Der Benutzer rät entweder einen Buchstaben oder versucht das Wort zu lösen.
- Wenn der Buchstabe nicht in dem Wort ist oder der Lösungsversuch falsch ist, wird der Rundenzähler reduziert.
- Wenn es keinen Buchstaben mehr zu raten gibt oder der Lösungsversuch richtig ist, hat der Spieler gewonnen.
- Fällt der Rundenzähler auf 0 hat der Spieler verloren.

2) Wie sieht die Eingabe aus?

Das Programm braucht eine Liste von Wörtern.
Der Benutzer muss jede Runde einen Buchstaben oder ein Lösungswort eingeben.

3) Wie sieht die Ausgabe aus?

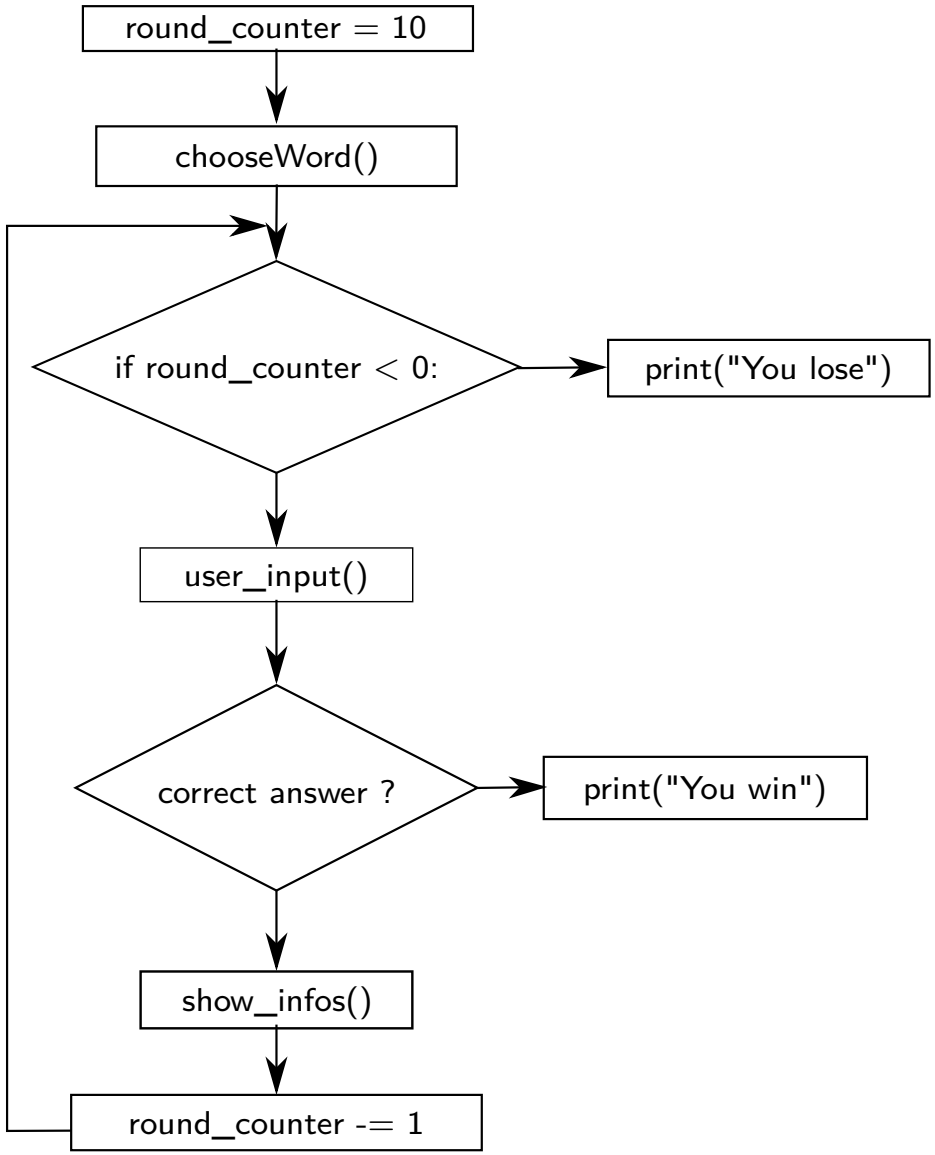
Die Ausgabe können wir mit *print()* realisieren, jedoch brauchen wir mehrere Ausgaben:

1. Rundennummer und Infos zur Eingabe nach jedem Versuch des Spielers
2. „Gewonnen“, wenn der Spieler das Wort richtig geraten hat
3. „Verloren“, wenn der Spieler es nicht geschafft hat, in der vorgegebenen Rundenzahl richtig zu raten

4) Was müssen wir uns speichern?

- das zu erratende Wort
- die schon geratenen Buchstaben
- das zu erratende Wort bei dem die Buchstaben, die schon geraten wurden, angezeigt werden

der Ablauf



Den Ablauf können wir von Mastermind übernehmen.

Zuerst legen wir Variabeln an, mit den Infos die wir speichern müssen.

```
1 word_to_play = []
2 tested_letters = []
3
4 def game_start_and_loop():
5     round_counter = 10
6     correct_word = ""
```

Dann müssen wir noch ein Wort raten.

```
1 word_to_play = []
2 tested_letters = []
3
4 def game_start_and_loop():
5     round_counter = 10
6
7     correct_word = read_file_and_choose_Word()
8     for i in range(0,len(correct_word)):
9         word_to_play.append("_")
```

In dem word_to_play wird der aktuelle Informationsstand gespeichert, den der Spieler hat - also am Anfang die Länge des Wortes.

Zuerst schreiben wir eine Funktion, die sich um die Ausgabe kümmert.

```
1  word_to_play = []
2  tested_letters = []
3
4  def game_start_and_loop():
5      round_counter = 10
6
7      correct_word = read_file_and_choose_Word()
8      for i in range(0,len(correct_word)):
9          word_to_play.append("_")
10
11     while( round_counter != 0):
12         print_game(round_counter)
```

```
1  def print_game(round_counter):
2      print("-"*80)
3      print("Noch",round_counter,"Versuche",end="")
4      print("\t\t\tSchon␣versucht:",end="")
5      for i in tested_letters:
6          print(i+",",end="")
7      print("\n\n\t",end="")
8      for i in word_to_play:
9          print(i+"␣",end="")
10     print("\n\n")
```

Infos über Steuersignal

Symbole	Signal für
\n	Zeilenumbruch.
\t	Tabulator
\b	letztes Zeichen löschen
\r	Zum Anfang der Zeile gehen
\v	Vertikaler Tabulator

Als nächstes fügen wir eine Funktion hinzu, um auf die Eingabe zu reagieren.

```
1  word_to_play = []
2  tested_letters = []
3
4  def game_start_and_loop():
5      round_counter = 10
6
7      correct_word = read_file_and_choose_Word()
8      for i in range(0,len(correct_word)):
9          word_to_play.append("_")
10
11     while( round_counter != 0):
12         print_game(round_counter)
13
14         if (next_round(correct_word) == True):
15             round_counter -= 1
16
17         if(word_to_play.count("_") == 0):
18             break
19
20     if (round_counter == 0):
21         print("Du hast verloren das richtige Wort waere:\n\t ", correct_word)
22     else:
23         print("Du hast \"",correct_word,"\"mit", 10 - round_counter, "fehlern
               erraten und somit gewonnen")
```

```
1  def next_round(correct_word):
2      inp = input("du bist dran:")
3      #Eingabe ueberpruefen
4      if (len(inp) != 1 and len(inp) != len(correct_word)):
5          print("Bitte nur ein Buchstaben eingeben oder ein moegliches
               Loesungswort")
6          return False
7      # Der spieler raet einen Buchstaben
8      if (len(inp) == 1):
9      Der Buchstabe kommt nicht in unserem Loesungswort vor
10     if (correct_word.lower().find((inp.lower())) == -1 and tested_letters.count(
           inp.lower()) == 0):
11             tested_letters.append(inp.lower())
12             tested_letters.sort()
13             return True
14         else:
15             for i in range(0, len(correct_word)):
16                 if (correct_word[i].lower() == inp.lower()):
17                     word_to_play[i] = correct_word[i]
18             return False
19     else:
20         #Der Spiele raet das Loesungswort
21         if (correct_word.lower() == inp.lower()):
22             for i in range(0, len(correct_word)):
23                 word_to_play[i] = correct_word[i]
24             return False
25         else:
26             tested_letters.append(inp.lower())
27     return True
28 ____ ____
```

```
1 from random import randrange
2
3 def read_file_and_choose_Word():
4     f = open("hangman.txt")
5     words = f.readlines()
6     f.close()
7     return  words[randrange(0,len(words))][:-1]
```

Infos zu open

Mit dem Befehl *open* ist es möglich eine Datei zu öffnen, um aus ihr zu lesen oder in die sie zu schreiben.

Bei dem *open* Befehl kann man zusätzlich mit angeben, ob man die Datei nur lesend oder auch schreibend öffnen will.

WICHTIG: Immer, wenn ihr ein *open* benutzt, muss auch ein *close* folgen.

```
1 open("hangman.txt", mode="r") # nur lesen (Wenn man nichts angibt)
2 open("hangman.txt", mode="w") # schreibend
3 open("hangman.txt", mode="a") # auch schreibend aber wenn die Datei schon
4 ____________________________#existiert wird es am ende der Datei angehaengt
```

Fertig?!

```
1 def print_game(roundcounter, correctword):
```

Es ist nicht wirklich schön, dass die Variablen immer übergeben werden müssen.
Zudem ist der Quellcode nicht wirklich übersichtlich.
Um dieses Problem zu lösen, können wir unser Spiel in eine Klasse auslagern.

Was ist eine Klasse?

Wikipedia

Unter einer Klasse (auch Objekttyp genannt) versteht man in der objektorientierten Programmierung ein abstraktes Modell bzw. einen Bauplan für eine Reihe von ähnlichen Objekten.

Eine einfache Klasse in Python.

```
1 class KlassenName():
2
3     def test(self):
4         print("ich bin eine Klasse")
```

Das *class* signalisiert, dass es sich um eine Klasse handelt.
Der *KlassenName* kann durch einen frei gewählten Namen ausgetauscht werden. Jedoch sollte der Name mit einem großen Buchstaben starten.
Die Funktion/Methode in Zeile 4 kennen wir schon. Das Einzige, dass sich dort geändert hat, ist der Parameter *self*.

Wie benutzt man jetzt diese Klasse?

```
1 my_class = KlassenNamen()
2 my_class.test()
```

Bis jetzt fragt ihr euch zurecht warum man das machen sollte. Um das klären zu können, benötigen wir ein etwas umfangreicheres Beispiel.

Beispiel

```
1  class  Beispiel():
2      def __init__(self,text):
3          self.text = text
4
5      def ausgabe(self):
6          print("Ich habe folgeneden Text bekommen", self.text)
7
8  aBeispiel = Beispiel("Klasse A")
9  bBeispiel = Beispiel("Klasse B")
10 aBeispiel.ausgabe()
11 bBeispiel.ausgabe()
```

Was zeigt uns dieses Beispiel?

- In einer Klasse können wir Daten speichern.
- Von einer Klasse können wir mehrere *Instanzen* erzeugen.
- In jeder dieser *Instanzen* können unterschiedliche Daten gespeichert werden.

Gehen wir schrittweise vor.

```
1 class Beispiel()
```

1) Wir definieren eine Klasse mit dem Namen *Beispiel*.

Gehen wir schrittweise vor.

```
1 class Beispiel()
2 ____def __init__(self,text):
```

2) Mit der Funktion `__init__` , die keine Funktion ist, sondern ein Konstruktor, können wir angeben welche Parameter unsere Klasse bekommen muss. (Das *self* muss da stehen)

Gehen wir schrittweise vor.

```
1 class Beispiel()
2     def __init__(self,text):
3         self.text = text
```

3) Innerhalb des Konstruktors werden die Variablen angelegt. Wenn man mit Klassen arbeitet, spricht man häufig von Attributen. Das sind Variablen, die in einer Klasse gespeichert werden.

Gehen wir schrittweise vor.

```
1  class Beispiel()
2      def __init__(self,text):
3          self.text = text
4
5      def ausgabe(self):
6          print("Ich habe folgeneden Text bekommen", self.text)
```

4) Innerhalb der Klassen können Funktionen angelegt werden, die mit den Variablen (Attributen) innerhalb der Klasse rechnen und/oder von außen Informationen bekommen.
(Wenn eine Funktion in einer Klasse ist, wird sie als Methode bezeichnet.)

das *self*

Wenn man eine Klasse erstellt, ist es zwingend nötig, dass der erste Parameter jeder Methode *self* ist. Das *self* ist eigentlich nur ein Zeiger auf sich selbst.

Könnt ihr euch noch an diese Zeile erinnern?

```
1 def printGame(roundcounter, correctword,wordToPlay,testes_letters):
```

Wir bauen jetzt aus unserem Hangman-Game eine Klasse.

1) Der Klassen Kopf

```
1 from random import randrange
2
3 class Hangman():
```

2) Der Konstruktor

```
1     def __init__(self,rounds):
2         self.round_counter = rounds
3         self.corek_tword = ""
4         self.word_to_play = []
5         self.testes_letters = []
```

Alle Variablen die wir benötigen werden hier angelegt.

3) Die read_File Funktion jetzt Methode

```
1     def read_file_and_choose_Word(self):
2         f = open("hangman.txt")
3         words = f.readlines()
4         f.close()
5         self.corekt_word =  words[randrange(0,len(words))][:-1]
```

4) die print Methode

```
1     def printGame(self):
2         print("-"*80)
3         print("Noch",self.round_counter,"Versuche",end="")
4         print("\t\t\tSchon versucht:",end="")
5         for i in self.testes_letters:
6             print(i+",",end="")
7         print("\n\n\t",end="")
8         for i in self.word_to_play:
9             print(i+" ",end="")
10        print("\n\n")
```

5) die nextRound Methode

```
1     def next_round(self):
2         inp = input("du bist dran:")
3         #Eingabe     berprfen
4         if (len(inp) != 1 and len(inp) != len(self.correct_word)):
5             print("Bitte nur einen Buchstaben eingeben oder ein m gliches
                  L sungs wort")
6             return False
7         # Der Spieler raet einen Buchstaben
8         if (len(inp) == 1):
9             #Der Buchstabe kommt nicht in unserem L sungs Wort vor
10            if (self.correct_word.lower().find((inp.lower())) == -1 and self.
                  tested_letters.count(inp.lower()) == 0):
11                self.tested_letters.append(inp.lower())
12                self.tested_letters.sort()
13                return True
14            else:
15                for i in range(0, len(self.correct_word)):
16                    if (self.correct_word[i].lower() == inp.lower()):
17                        self.word_to_play[i] = self.correct_word[i]
18                return False
19        else:
20            #Der Spiele raet das L sungswort
21            if (self.correct_word.lower() == inp.lower()):
22                for i in range(0, len(self.correct_word)):
23                    self.word_to_play[i] = self.correct_word[i]
24                return False
25            else:
26                self.tested_letters.append(inp.lower())
27        return True
```

6) unsere main als Methode

```
1     def game_start_and_loop(self):
2
3         self.read_file_and_choose_Word()
4         for i in range(0,len(self.correct_word)):
5             self.word_to_play.append("_")
6
7         while( self.round_counter != 0):
8             self.print_game()
9
10            if (self.next_round() == True):
11                self.round_counter -= 1
12
13            if(self.word_to_play.count("_") == 0):
14                break
15
16        if (self.round_counter == 0):
17            print("Du hast verloren das richtige Wort waere:\n\t", self.
                  correct_word)
18        else:
19            print("Du hast \"",self.correct_word,"\"in", self.round_counter, "
                  Zuegen erkannt und somit gewonnen")
```

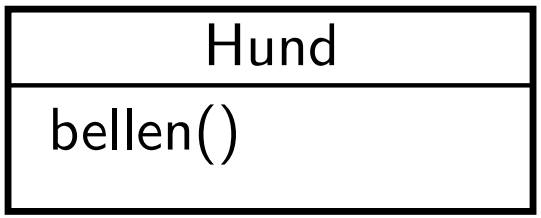
7) So startet man jetzt eine Runde.

```
1 myHangman = Hangman(10)
2 myHangman.game_start_and_loop()
```

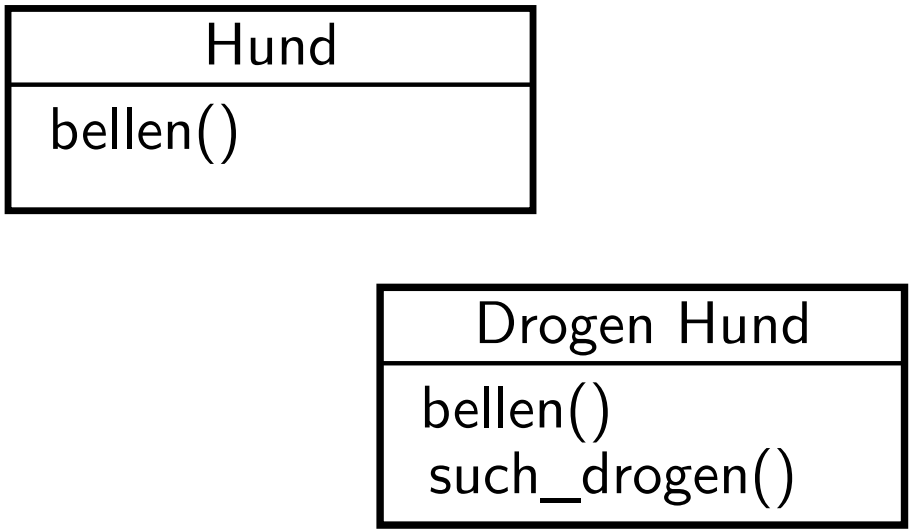
Was hat sich geändert?

- Alle Funktionen wurden eine Ebene weiter eingerückt.
- Die Funktionsparameter wurden auf ein *self* reduziert.
- Innerhalb der Funktionen wurde jeder Variable ein *self.* vorgestellt.

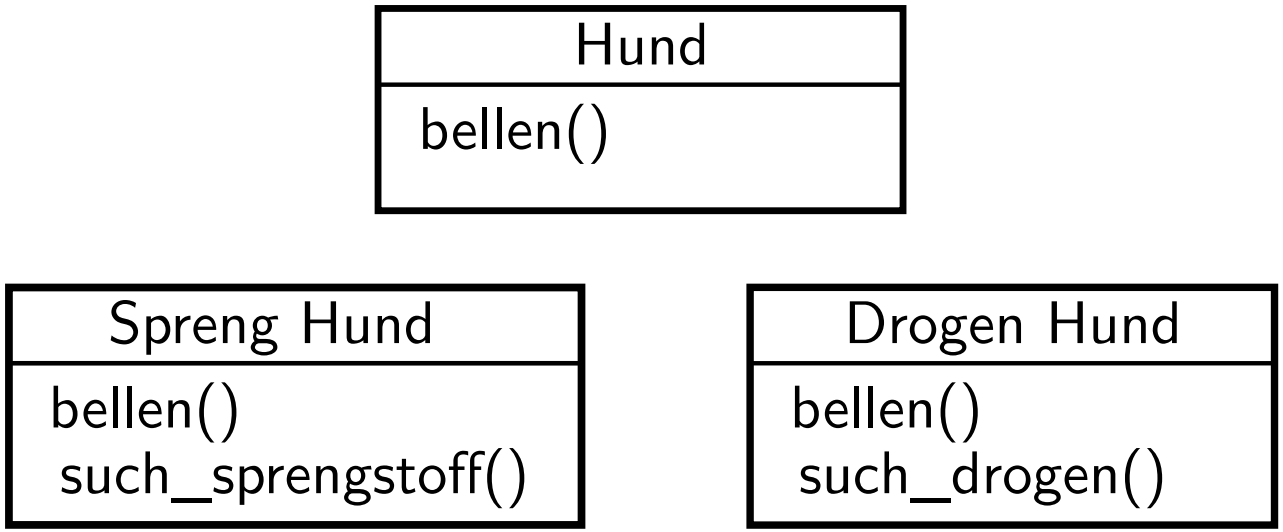
Sonst hat sich an unserem Quellcode nichts geändert.



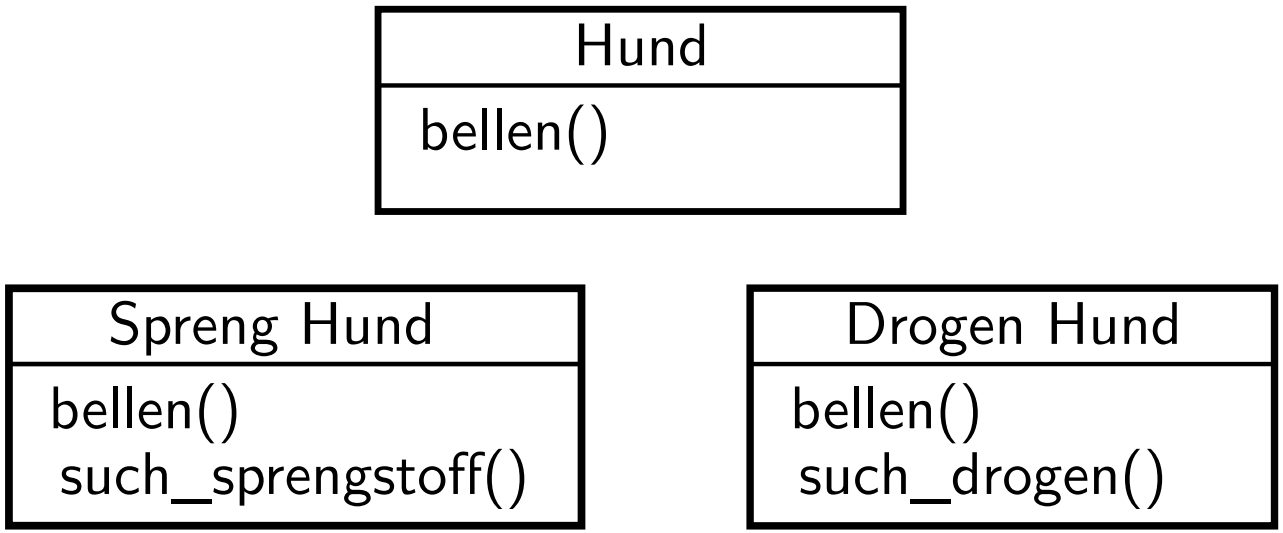
Stellen wir uns vor wir haben eine Klasse Hund. Und dieser Hund kann bellen.



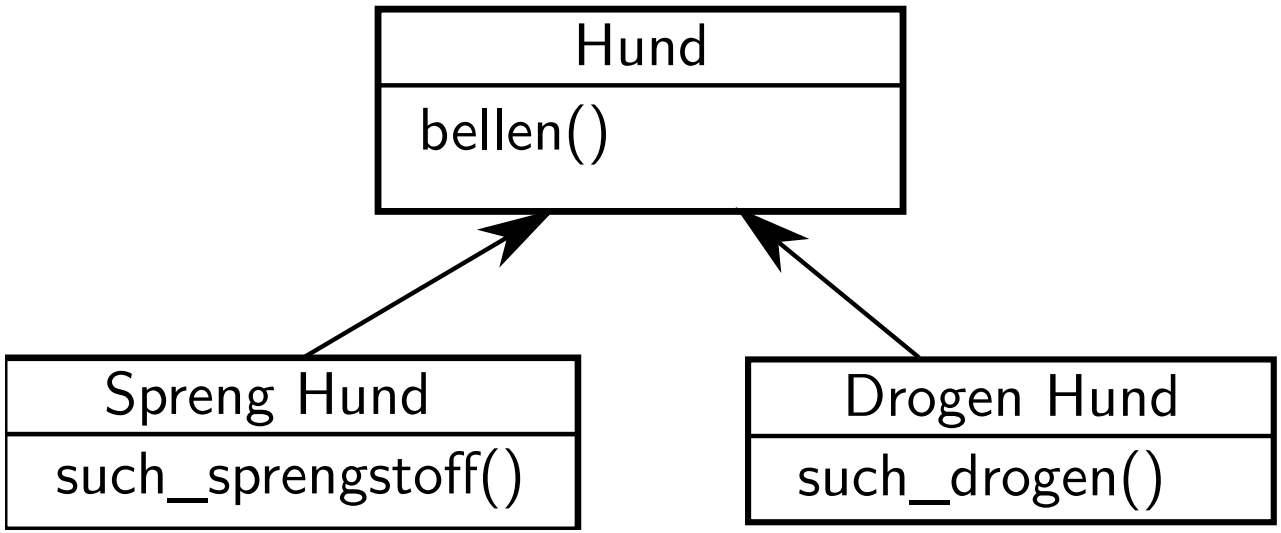
Jetzt benötigen wir noch einen Drogensuchhund und dieser kann natürlich auch bellen, aber zusätzlich noch Drogen suchen.



Zum Schluss bauen wir uns noch einen Sprengstoffsuchhund, der auch bellen kann. Zusätzlich kann er auch Sprengstoffe aufspüren.



Wenn wir für jeden Hund eine Klasse schreiben würden, müssten wir die bellen Methode drei mal schreiben. Wie ihr euch denken könnt ist das nicht sinnvoll.



Besser wäre es, wenn unser Sprenghund und unser Drogenhund von unserem Hund das Bellen "erben" könnten.

Unsere Hund-Klasse sieht folgendermaßen aus:

```
1 class Hund():
2
3     def __init__(self, name):
4         self.name = name
5
6     def bellen(self):
7         print("Wau Wau")
```

und so unser Drogenhund:

```
1 class DrogenHund(Hund):
2
3     def __init__(self, name):
4         self.name = name
5
6     def such_drogen(self):
7         print(self.name, "sucht nach Drogen")
```

zum Schluss noch unser Sprenghund:

```
1  class SprengHund(Hund):
2
3      def __init__(self, name):
4          self.name = name
5
6      def such_sprengstoffe(self):
7          print(self.name, "sucht nach Sprengstoffen")
```

zum ausprobieren benutzen wir:

```
1  hund1 = Hund("waldi")
2  hund2 = DrogenHund("bello")
3  hund3 = SprengHund("bruno")
4
5  hund1.bellen()
6  hund2.bellen()
7  hund3.bellen()
8
9  hund2.such_drogen()
10 hund3.such_sprengstoffe()
```

Wenn man sich jetzt überlegt, dass die Hunde eher "wuff wuff" machen als "wau wau", dann muss man nur an einer Stelle etwas ändern.

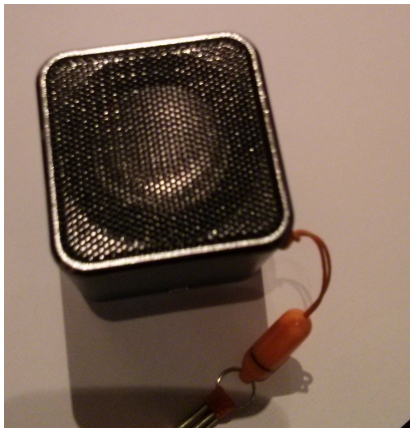
```
1 class Hund():
2
3     def __init__(self, name):
4         self.name = name
5
6     def bellen(self):
7         print("Wuff wuff")
```

Wie geht es jetzt weiter?

Jetzt fragt ihr euch wahrscheinlich (zurecht):

Was mache ich jetzt mit dem Wissen, dass ich mir die letzten Tage erarbeitet habe?
Darauf gibt es meiner Meinung nach nur eine Antwort:
-mehr Programmieren und Probleme lösen-

aus dem Leben eines Programmierers



Ich bin im Besitz von einem dieser kleinen Lautsprecher, die einen Akku eingebaut haben und mit einer Speicherkarte gefüttert werden können.
Leider haben die Erbauer des Gerätes nicht vorgesehen das man in zufälliger Reihenfolge die Lieder abspielen lassen möchte.

Nach etwas Rumprobieren habe ich dann herausgefunden, dass der Player die Lieder einfach nach Reihenfolge der Dateinamen abspielt.

Also habe ich ein Script geschrieben, dass die Lieder in einer zufälligen Reihenfolge durchnummeriert.

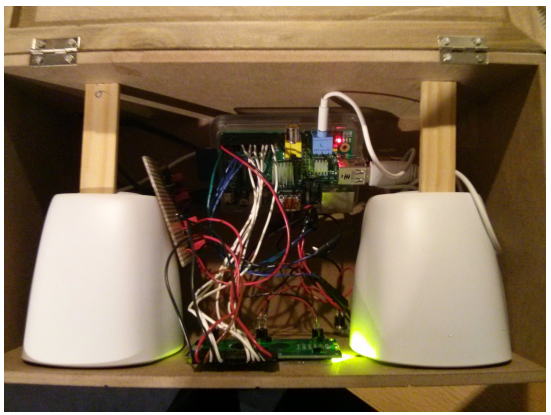
```
#!/usr/bin/env python3
import os
from glob import glob
import random

def _args():
    import argparse
    parser = argparse.ArgumentParser(description="Random script fuer Musicman")
    parser.add_argument('-d', action='store', type=str, dest="dir",
    help = "Ordner der durchmischt werden soll")
    parser.add_argument('-o', action='store', type=int, dest="override",
    help = "vorherige Durchmischung ueberschreiben anzahl der Ziffern angeben")
    return parser.parse_args()

def _getFiles_Unix(args):
    fileList = glob("*.mp3")
    return fileList
```

```
1  def _randomly(fileList, override):
2      count = 1
3      print("Es␣werden␣", len(fileList), "␣Dateien␣durchmischt")
4      while len(fileList) > 0:
5          if len(fileList) != 1:
6              number = random.randint(0, len(fileList) - 1)
7          else:
8              number = 0
9          print(count, "Datei",fileList[number])
10         if override is not None and override > 0:
11             os.rename(fileList[number], ( count + '-' + fileList[number][
                   override + 1:]))
12         else:
13             os.rename(fileList[number], ( count + '-' + fileList[number]))
14         count += 1
15             fileList.remove(fileList[number])
16
17 if __name__ == "__main__":
18     args = _args()
19     os.chdir(args.dir)
20     fileList = _getFiles_Unix(args)
21     _randomly(fileList, args.override)
```

Somit habe ich in weniger als 40 Zeilen meinen Musikwürfel um eine rudimentäre Zufallsfunktion erweitert.

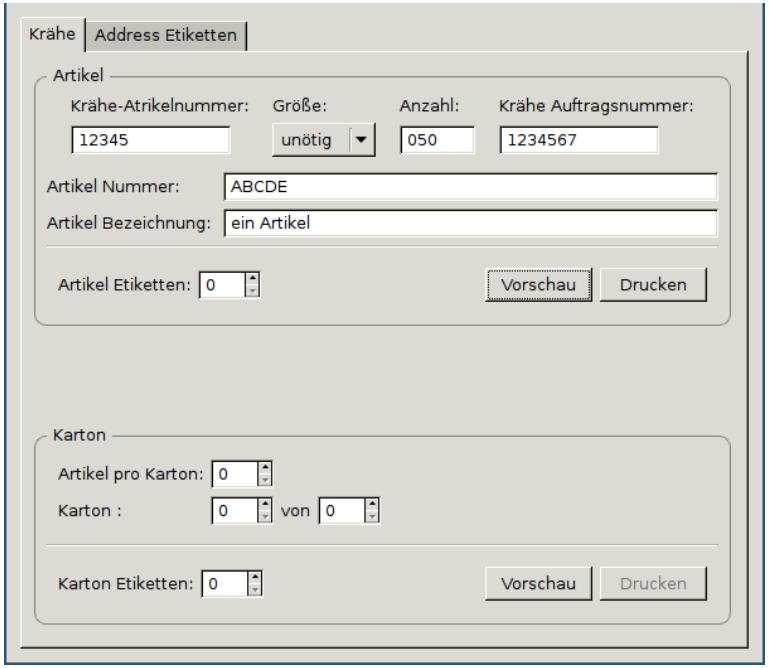
ein Radio Ripper

Da ich ein Fan von Internet Radio bin, aber meine Lieblingsradiosendung nur in der Zeit von 21-23 Uhr gesendet wird, habe ich mir ein Script geschrieben, das die Sendung für mich aufnimmt und in eine mp3 umwandelt. Jetzt kann ich mir die Sendung anhören wann ich will.
Das Script hat 55 Zeilen.



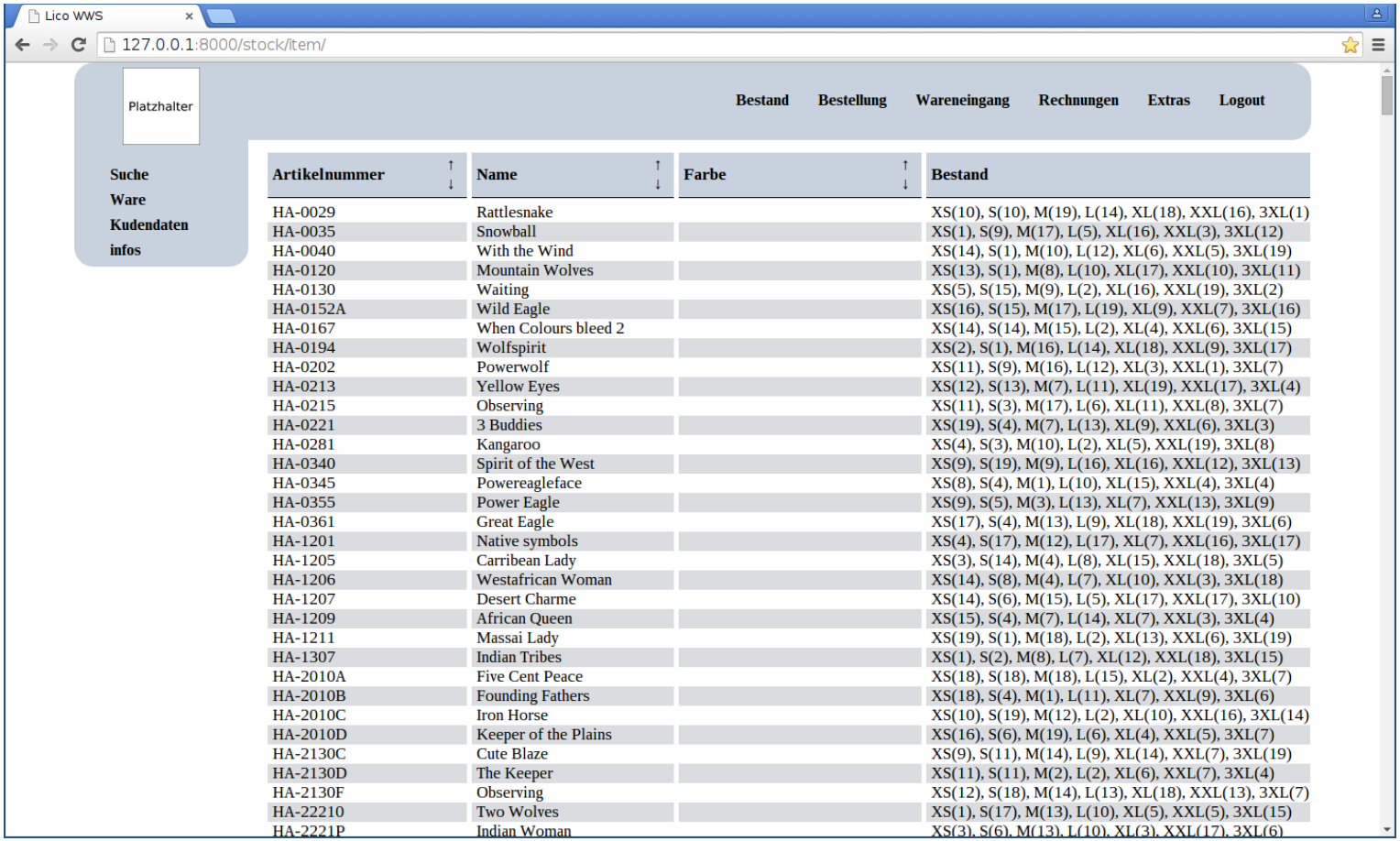
Aus einem Raspberry Pi, einem LCD-Display, einem Wlan-Stick ,PC Lautsprechern, 4 Knöpfen und 140 Zeilen Quell-Code habe ich ein Web-Radio gebaut.

Ettiketten Drucker



Für einen Verwandten habe ich ein Programm geschrieben, dass für ihn Etiketten mit Barcodes generiert.
Benutzt habe ich LaTeX, pyqt4 und 266 Zeilen, die ich selbst geschrieben habe.

Warenwirtschaftssystem



Artikelnummer	Name	Farbe	Bestand
HA-0029	Rattlesnake		XS(10), S(10), M(19), L(14), XL(18), XXL(16), 3XL(1)
HA-0035	Snowball		XS(1), S(9), M(17), L(5), XL(16), XXL(3), 3XL(12)
HA-0040	With the Wind		XS(14), S(1), M(10), L(12), XL(6), XXL(5), 3XL(19)
HA-0120	Mountain Wolves		XS(13), S(1), M(8), L(10), XL(17), XXL(10), 3XL(11)
HA-0130	Waiting		XS(5), S(15), M(9), L(2), XL(16), XXL(19), 3XL(2)
HA-0152A	Wild Eagle		XS(16), S(15), M(17), L(19), XL(9), XXL(7), 3XL(16)
HA-0167	When Colours bleed 2		XS(14), S(14), M(15), L(2), XL(4), XXL(6), 3XL(15)
HA-0194	Wolfspirit		XS(2), S(1), M(16), L(14), XL(18), XXL(9), 3XL(17)
HA-0202	Powerwolf		XS(11), S(9), M(16), L(12), XL(3), XXL(1), 3XL(7)
HA-0213	Yellow Eyes		XS(12), S(13), M(7), L(11), XL(19), XXL(17), 3XL(4)
HA-0215	Observing		XS(11), S(3), M(17), L(6), XL(11), XXL(8), 3XL(7)
HA-0221	3 Buddies		XS(19), S(4), M(7), L(13), XL(9), XXL(6), 3XL(3)
HA-0281	Kangaroo		XS(4), S(3), M(10), L(2), XL(5), XXL(19), 3XL(8)
HA-0340	Spirit of the West		XS(9), S(19), M(9), L(16), XL(16), XXL(12), 3XL(13)
HA-0345	Powereagleface		XS(8), S(4), M(1), L(10), XL(15), XXL(4), 3XL(4)
HA-0355	Power Eagle		XS(9), S(5), M(3), L(13), XL(7), XXL(13), 3XL(9)
HA-0361	Great Eagle		XS(17), S(4), M(13), L(9), XL(18), XXL(19), 3XL(6)
HA-1201	Native symbols		XS(4), S(17), M(12), L(17), XL(7), XXL(16), 3XL(17)
HA-1205	Carribean Lady		XS(3), S(14), M(4), L(8), XL(15), XXL(18), 3XL(5)
HA-1206	Westafrican Woman		XS(14), S(8), M(4), L(7), XL(10), XXL(3), 3XL(18)
HA-1207	Desert Charme		XS(14), S(6), M(15), L(5), XL(17), XXL(17), 3XL(10)
HA-1209	African Queen		XS(15), S(4), M(7), L(14), XL(7), XXL(3), 3XL(4)
HA-1211	Massai Lady		XS(19), S(1), M(18), L(2), XL(13), XXL(6), 3XL(19)
HA-1307	Indian Tribes		XS(1), S(2), M(8), L(7), XL(12), XXL(18), 3XL(15)
HA-2010A	Five Cent Peace		XS(18), S(18), M(18), L(15), XL(2), XXL(4), 3XL(7)
HA-2010B	Founding Fathers		XS(18), S(4), M(1), L(11), XL(7), XXL(9), 3XL(6)
HA-2010C	Iron Horse		XS(10), S(19), M(12), L(2), XL(10), XXL(16), 3XL(14)
HA-2010D	Keeper of the Plains		XS(16), S(6), M(19), L(6), XL(4), XXL(5), 3XL(7)
HA-2130C	Cute Blaze		XS(9), S(11), M(14), L(9), XL(14), XXL(7), 3XL(19)
HA-2130D	The Keeper		XS(11), S(11), M(2), L(2), XL(6), XXL(7), 3XL(4)
HA-2130F	Observing		XS(12), S(18), M(14), L(13), XL(18), XXL(13), 3XL(7)
HA-22210	Two Wolves		XS(1), S(17), M(13), L(10), XL(5), XXL(5), 3XL(15)
HA-2221P	Indian Woman		XS(3), S(6), M(13), L(10), XL(3), XXL(17), 3XL(6)

Für den gleichen Verwandten bin ich dabei, ein Warenwirtschaftssystem zu schreiben, dass komplett im Webbrowser läuft. Aktuell benutze ich das Python-Framework Django und habe bis jetzt 385 Zeilen geschrieben.

Was nützen mir jetzt diese Informationen?

Ihr solltet nur mal eine Idee bekommen, was man alles machen kann. Wichtig ist, dass ihr weiter übt Programme zu schreiben, denn nur so lernt man zu programmieren und natürlich problemorientiert zu denken.

geschafft

Nun kennt ihr die Grundlagen von Python

Viel Spaß im Tutorium!