Vorkurs Informatik

Dirk Frettlöh

dfrettloeh@techfak.de

30. Oktober 2020

Hangman

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

Auch hier fangen wir 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 speichern?
5. Was ist der Ablauf?

1) Welche Regeln hat das Spiel?

- Das Programm wählt ein Wort aus.
- Der Benutzer rät entweder einen Buchstaben oder versucht, das Wort zu raten.
- 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.

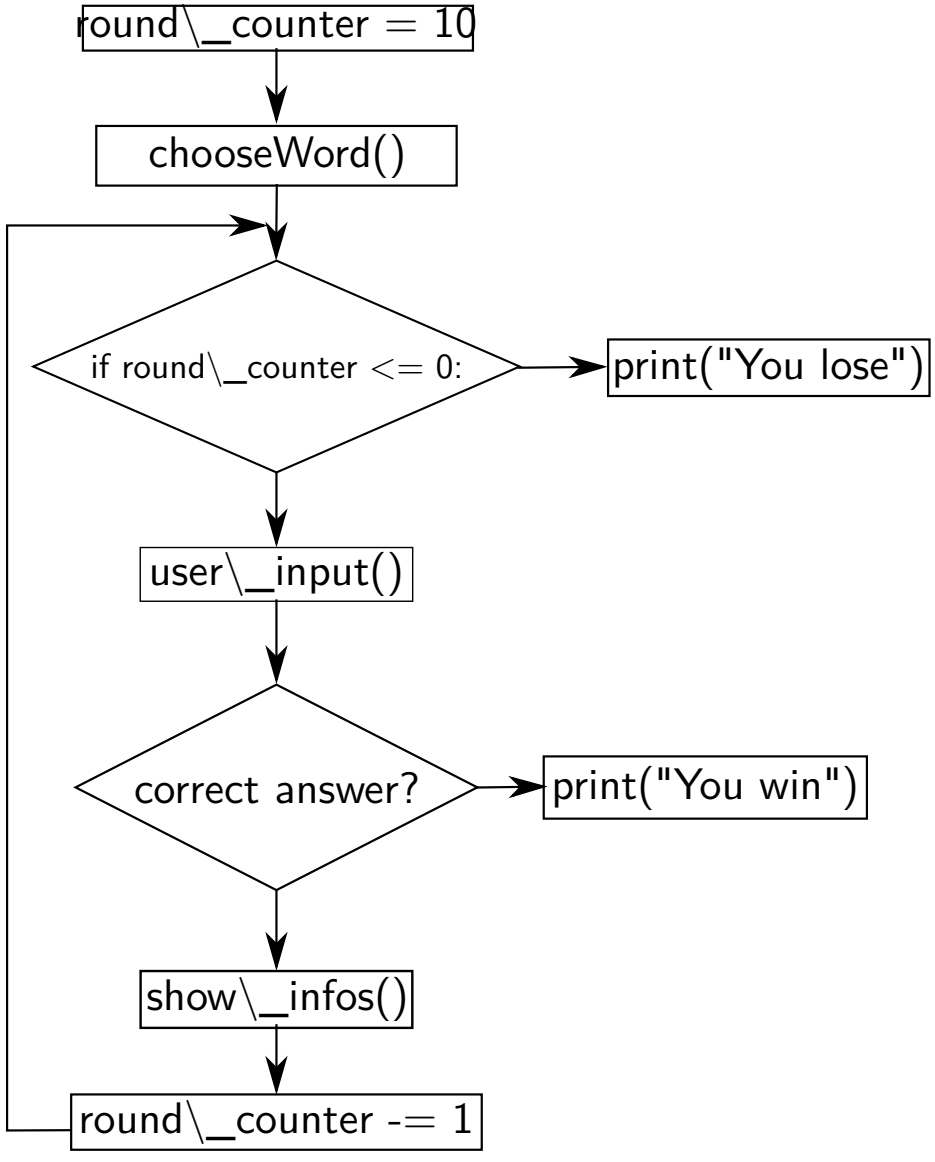
3) Wie sieht die Ausgabe aus?

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

1. die 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 das richtige Wort zu raten,

4) Was müssen wir speichern?

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



Den Ablauf können wir von Mastermind übernehmen.

Zuerst legen wir Variablen an, mit den zu speichernden Informationen.

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

Dann müssen wir noch ein Wort raten.

```
1 tested_letters = []
2
3 def game_start_and_loop():
4     round_counter = 10
5     word_to_play = []
6
7     correct_word = chooseWord()
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 zu Beginn die Länge des Wortes.

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

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

```
1  def show_infos(round_counter, word_to_play):
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")
```

Steuersignale

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,die auf Eingabe reagiert.

```
1  tested_letters = []
2
3  def game_start_and_loop():
4      round_counter = 10
5      word_to_play = []
6
7      correct_word = chooseWord()
8      for (i in range(0, len(correct_word))):
9          word_to_play.append("_")
10
11     while(round_counter != 0):
12         show_infos(round_counter, word_to_play)
13
14         if (next_round(correct_word, word_to_play) == 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")
```

```
def next_round(correct_word, word_to_play):
    inp = input("du bist dran:")
    # Eingabe ueberpruefen
    if (len(inp) != 1 and len(inp) != len(correct_word)):
        print("Bitte nur ein Buchstaben eingeben oder ein moegliches
            Loesungswort")
        return False
    # Der Spieler raet einen Buchstaben
    if (len(inp) == 1):
    Der Buchstabe kommt nicht in unserem Loesungswort vor
    if (correct_word.lower().find((inp.lower())) == -1 and tested_letters.count(
        inp.lower()) == 0):
            tested_letters.append(inp.lower())
            tested_letters.sort()
            return True
        else:
            for i in range(0, len(correct_word)):
                if (correct_word[i].lower() == inp.lower()):
                    word_to_play[i] = correct_word[i]
            return False
    else:
        # Der Spieler raet das Loesungswort
        if (correct_word.lower() == inp.lower()):
            for i in range(0, len(correct_word)):
                word_to_play[i] = correct_word[i]
            return False
        else:
            tested_letters.append(inp.lower())
    return True
```

```
1 from random import randrange
2
3 def chooseWord():
4     with open("hangman.txt") as wordfile:
5         words = wordfile.readlines()
6
7     return  words[randrange(0,len(words))]
```

Informationen zu open

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

Der *with*-Block fängt mögliche Fehler ab und schließt die Datei wieder(!).

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

```
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 show_infos(round_counter, word_to_play):
```

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 beginnen.
Die Funktion/Methode in Zeile 3 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 folgenden 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␣folgenden␣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 show_infos(round_counter, word_to_play):
```

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.correct_word = ""
4         self.word_to_play = []
5         self.testes_letters = []
```

Alle Variablen die wir benötigen, werden hier angelegt.

3) die read_File Funktion jetzt als Methode

```
1      def chooseWord(self):
2          with open("hangman.txt") as wordfile:
3              words = wordfile.readlines()
4
5          self.correct_word =  words[randrange(0,len(words))][:-1]
```

4) die print Methode

```
1     def show_infos(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 ueberpruefen
4         if (len(inp) != 1 and len(inp) != len(self.correct_word)):
5             print("Bitte nur einen Buchstaben eingeben oder ein moegliches
                  Loesungs wort")
6             return False
7         # Der Spieler raet einen Buchstaben
8         if (len(inp) == 1):
9             #Der Buchstabe kommt nicht in unserem Loesungs 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 Loesungswort
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.chooseWord()
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.show_infos()
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")
```

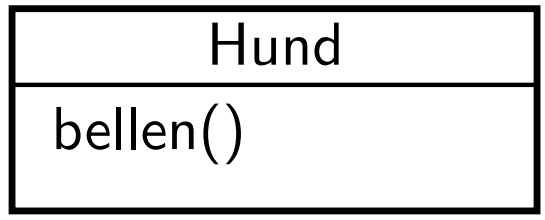
7) So startet man jetzt eine Runde.

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

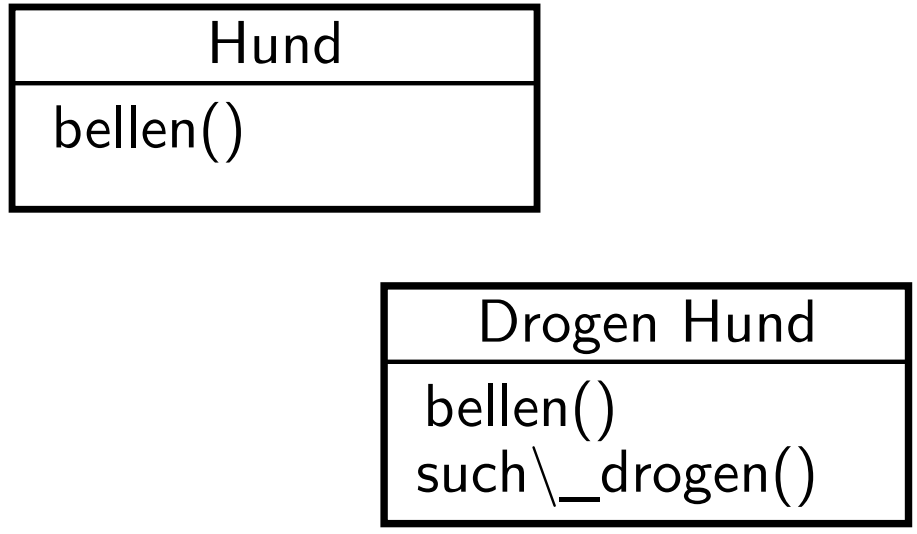
Was hat sich jetzt geändert?

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

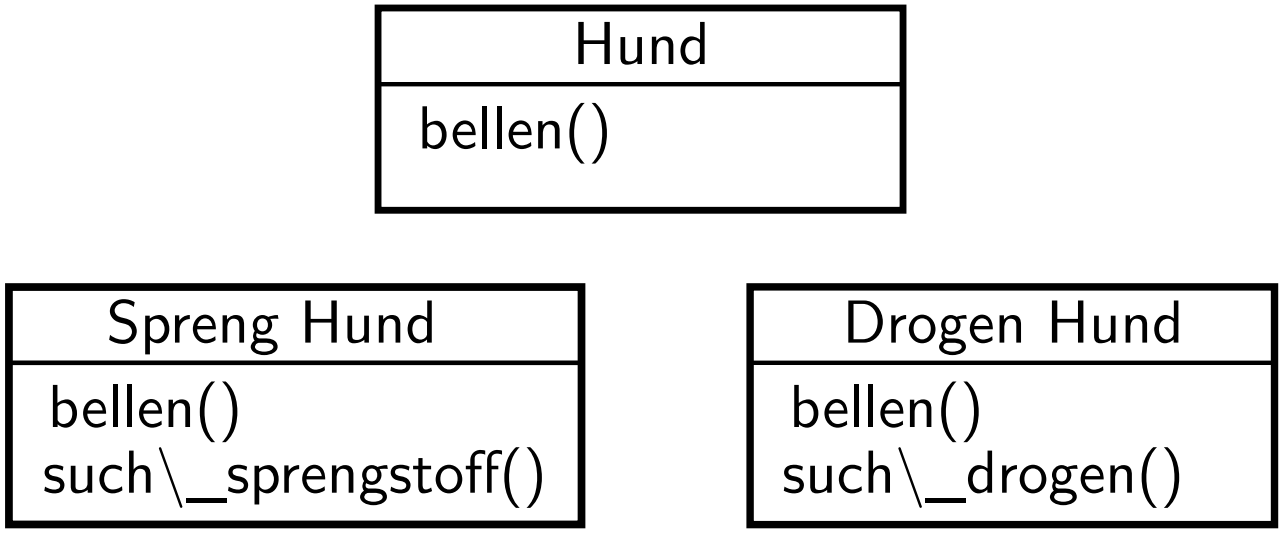
Sonst hat sich an unserem Quellcode nichts geändert.



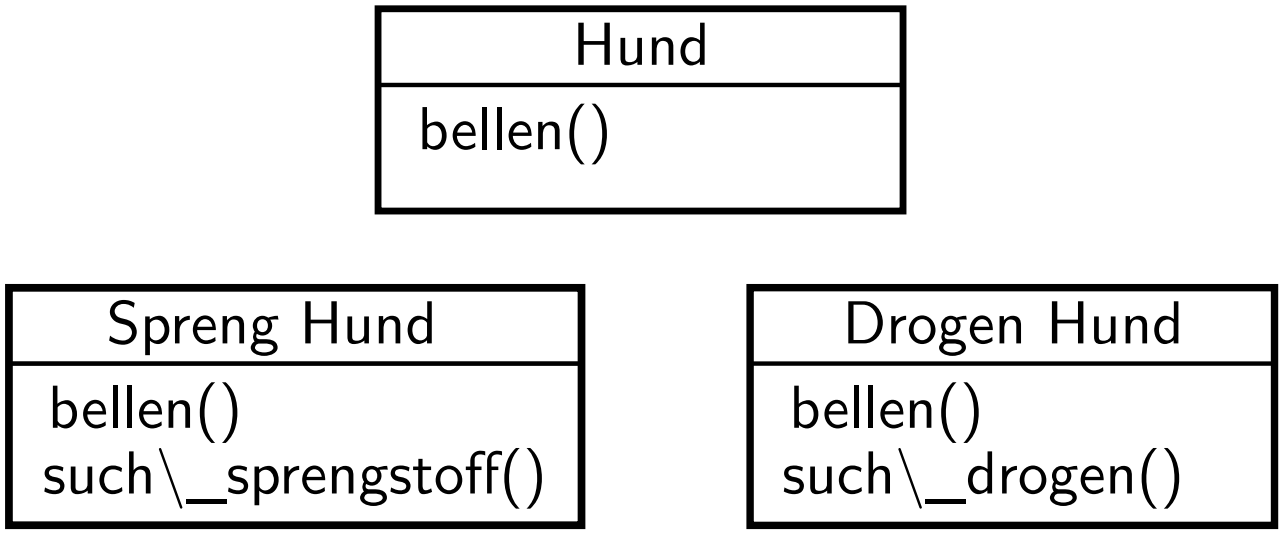
Stellen wir uns vor: Wir haben eine Klasse Hund. Dieser Hund kann bellen.



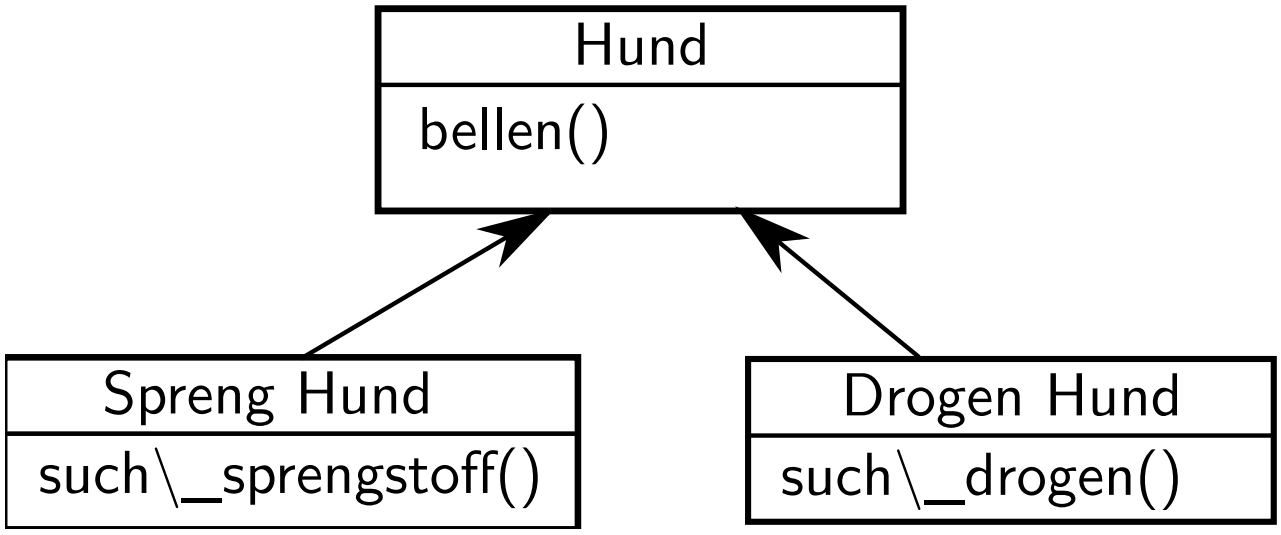
Jetzt benötigen wir noch einen Drogenhund und dieser kann natürlich auch bellen, aber zusätzlich noch Drogen suchen.



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



Wenn wir für jeden Hund eine Klasse schreiben würden, müssten wir die *bellen* Methode dreimal 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")
```

Statische Typen

Normalerweise rät Python den Typ einer Variable, wenn wir sie zuweisen

- `foo = 123` ← int
- `bar = "123"` ← str
- `baz = 123.0` ← float

Das ist extrem vielseitig und flexibel, kann aber auch verwirrend sein und schnell zu Bugs führen (s. Eingabe von Zahlen per `input()` bei Mastermind).

Lösung: statische Typen

Statische Typen

Typdefinition nach Variablenname:
`foo = 123` $\Rightarrow$ `foo : int = 123`

Die Überprüfung übernimmt ein externes Programm: `mypy`

Statische Typen

Standarttypen `int`, `str`, `bool`, `float`, `...` kennt Python bereits.

Datenstrukturen wie z.B. Listen müssen importiert werden:

```
1 from typing import List
2
3 haustiere : List[str] = ["Hund","Katze","Maus"]
```

Statische Typen

Auch in Funktionen funktionieren Typen:

```
1  from typing import List
2
3  def arithmetic_mean (numbers @: List[int]@) @-> float@:
4
5      ergebnis @: float@ = 0
6      for i in numbers:
7
8          ergebnis += i
9
10     ergebnis /= len(numbers)
11
12     return ergebnis
```

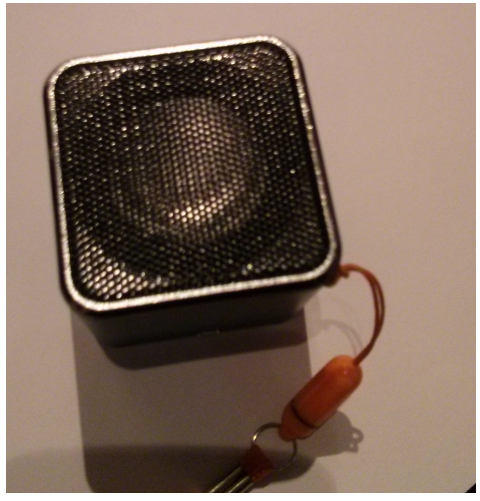
Statische Typen

Eigene Klassen sind auch Typen

```
1
2  class Hund():
3
4      def __init__(self,name:str,groesse:float) -> None:
5          self.name : str = name
6          self.groesse : float = groesse
7
8      def bellen(self) -> None:
9          print('wuff')
10
11 meinHund : Hund = Hund('Markov',32.7)
12 deinHund : Hund = Hund('Blanket',12.2)
```

Anwendungen

(Nicht von mir, von einem echten Tüftler:)

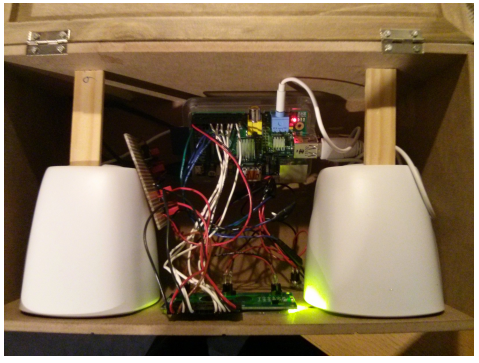


Lautsprecher mit Akku und Speicherkarte. Spielt Lieder in alphabetischer Reihenfolge.

Keine Zufallsreihenfolge ("shuffle" oder "random shuffle") vorgesehen.

Software ist python! Also 40 Zeilen Code ergänzen, Lieder zufällig umbennen (import random).

Derselbe Tüftler:

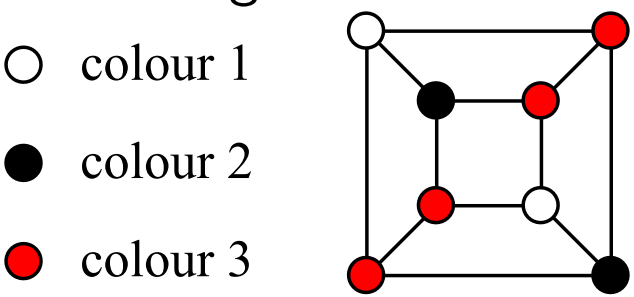


“Aus einem Raspberry Pi, einem LCD Display, einem Wlan-stick, PC Lautsprechern und 4 Knöpfen und 140 Zeilen Python-Code habe ich ein Web-Radio gebaut.”

Ich selbst bin mehr so theoretisch orientiert. Meine Anwendungen:

Vorlesung Kryptographie: ein interaktives “sagemath notebook” benutzt als Sprache python, und verbindet das mit mächtigen Mathewerkzeugen (R für Statistik, numpy, scipy: python-Pakete für Mathe, GAP, Maxima für Algebra...)

Forschung:



$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 2 & 2 & 1 \end{pmatrix}$$

Fragen?

Geschafft

Soviel zu Grundlagen von Python.

Viel Spaß damit!