Würgeschlange 3 - Lesson 3

Tobias Maschek, Viktor Reusch
https://github.com/jemx/wise1920-python

mit Materialien von Felix Döring, Felix Wittwer https://github.com/fsr/python-lessons

12. November 2019

Python-Kurs

Gliederung

1. Lists und Dicts - Déjà-vu?
2. Funktionen

   Parameterübergabe
3. builtins
4. I/O

Jetzt auch im Neuland

Folien jetzt auch unter `https://github.com/jemx/wise1920-python`

Lists und Dicts - Déjà-vu?

Déjà-vu, I've just been in this place before. — Aufgabe 3-1

- Es ist eine Liste mit Ziffern gegeben: `v = [1, 1, 2, 4, 3, 7, 3]`
- Erstellt ein dict, welches jede Ziffer (von 0 bis 9) als Key besitzt.
- Die Values sollen die absoluten Häufigkeiten der Ziffern in `v` sein.
- **Profis:** Die Values sollen die relativen Häufigkeiten der Ziffern in `v` sein.

Funktionen

Funktionsdefinition

Warum?

- Teilt Code in Abschnitte auf: übersichtlicher und einfacher lesbar
- Erlaubt einfache Kollaboration: jeder schreibt eine Funktion
- Ermöglicht Wiederbenutzung von Code an verschiedenen Stellen
- Macht *Rekursion* möglich

```
1  def square(i):
2      return i * i
3
4  print(square(5))  # 25
5
6  def return_function(return_early):
7      if return_early:
8          return
9      print("Hi")
10
11 return_function(True)
12 return_function(False)  # Hi
```

Scopes? Scopes :(

```
1 i = 5
2
3 def set_i(k):
4     i = k
5     print(i)
6
7 print(i)
8 set_i(42)
9 print(i)
```

Scopes? Scopes :)

```
1  i = 5
2
3  def set_i(k):
4      global i
5      i = k
6      print(i)
7
8  print(i)
9  set_i(42)
10 print(i)
```

Parameterübergabe

Call by Value

- Variablen-Inhalt wird in Parameter **kopiert**.
- Änderung am Parameter haben keinen Effekt auf die aufrufende Variable

```c
#include<stdio.h>

void use(int i) {
    i = 3;
}

int main() {
    int p = 0;
    use(p);
    printf("%d\n", p); // 0
}
```

Call by Reference

- Parameter ist **Referenz** auf die aufrufende Variable.
- Alle Änderungen am Parameter wirken auch auf die Variable.

```
#include <iostream>
using namespace std;

void change(int &i) {
    i = 3;
}

int main() {
    int p = 0;
    change(p);
    cout << p << endl; // 3
}
```

Call by Object

- Parameter hält eine Referenz auf das selbe **Objekt**, wie die aufrufende Variable.
- Änderungen am Objekt haben eine Wirkung.
- Änderungen am Parameter nicht.

```python
def use(i):
    i = 20

def change(l):
    l.append(12)

p = 10
use(p)
t = [10, 11]
change(t)
print(f"unchanged {p}, changed {t}")  # unchanged 10, changed [10, 11, 12]
```

Keyword Arguments

```
def control_flow(name, formal=True):
    if formal:
        print(f"Hallo Herr/Frau {name}!")
    else:
        print(f"Hi {name}")

control_flow("Meier")                # Hallo Herr/Frau Meier!
control_flow("Frank", False)         # Hi Frank
control_flow("Frank", formal=False) # Hi Frank
```

Aufgabe 3-2

1. Schreibe eine *Batman*-Funktion:
   - Die Funktion bekommt eine Zahl übergeben.
   - Die Zahl wird um eins verringert und es wird „Na “ ausgegeben.
   - Dann wird die Funktion rekursiv (mit der verringerten Zahl) aufgrufen.
   - Wenn die Zahl 0 erreicht, wird „Batman“ ausgegeben und die Rekursion beendet.
2. Schreibe eine Begrüßungsfunktion:
   - Die Funktion bekommt **optional** einen Namen übergeben.
   - Der Standard-Wert für den Namen ist „Welt“.
   - Grüße nun die Person mit dem übergebenen Namen.

builtins

set

- kann nur *hashbare* Einträge enthalten - **Hash, kann man das rauchen?**
- enthält jedes Element nur einmal
- mit `my_set.add()` Elemente hinzufügen
- schnellere Überprüfung mit `in` (prüft, ob Element enthalten ist)
- Mögliche Operationen: `<=`, `<`, `>`, `|`, `&`, `-`, $\sim$, $\wedge$
- **ungeordnet**

set - Beispiel

```
s1 = {1, 2, 'string', ('ein', 'tuple')}

2 in s1  # ==> True
'ein' in s1  # ==> False
('ein', 'tuple') in s1  # ==> True
set(('ein', 'tuple'))  # ==> {'ein', 'tuple'}
set(['eine', 'liste'])  # ==> {'liste', 'eine'}

{2} < s1  # ==> True
s1 <= {2}  # ==> False

s2 = {'anderes', 'set'}
s1.add('anderes') # s1 == {1, 2, 'string', ('ein', 'tuple'), 'anderes'}
s1 | s2  # ==> {1, 2, 'string', object, ('ein', 'tuple'), 'set'}
s1 & s2  # ==> {'anderes'}
s2 - s1  # ==> {'set'}
```

input()

```
>>> var = input()
>? Hello, this is my input



>>>var
'Hello, this is my input'
```

enumerate()

```
my_list = ['apple', 'banana', 'grapes', 'pear']
for c, value in enumerate(my_list, 1):
    print(c, value)

# Output:
# 1 apple
# 2 banana
# 3 grapes
# 4 pear
```

sum()

```
numbers = [21, 42, 84]

sum(numbers)

# Output:
# 147
```

Achtung

`sum()` funktioniert nur auf Zahlen (int, long, float).

min()

```
numbers = [21, 42, 84]

min(numbers)

# Output:
# 21
```

Achtung

`min()` funktioniert nur auf Zahlen (int, long, float).

max()

```
numbers = [21, 42, 84]

max(numbers)

# Output:
# 84
```

Achtung

`max()` funktioniert nur auf Zahlen (int, long, float).

round()

```
round(5.76543, 2)
# Output
# 5.77

import math
round(math.pi, 0)
# Output
# 3.0

round(math.pi)
# Output
# 3
```

Achtung

`round()` funktioniert nur auf Zahlen (int, long, float).

filter()

```
1 def even(i):
2     return i % 2 == 0
3
4 l = list(range(10)) # ==> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
5 list(filter(even, l)) # ==> [0, 2, 4, 6, 8]
6
7 [e for e in l if even(e)] # ==> [0, 2, 4, 6, 8]
```

all(), any()

```
l1 = [True, False, True]
all(l1) # ==> False
any(l1) # ==> True

l2 = [True, True]
all(l2) # ==> True
any(l2) # ==> True

l3 = [False, False]
all(l3) # ==> False
any(l3) # ==> False
```

map()

```
l = [-1, 4, -5, 3, 6]

a = list(map(abs, l)) # ==> [1, 4, 5, 3, 6]

def even(i):
    return i % 2 == 0

any(map(even, l)) # ==> True
```

sorted()

```
l = [-1, 4, -5, 3, 6]

sorted(l) # ==> [-5, -1, 3, 4, 6]

s = list("Hallo!") # ==> ['!', 'H', 'a', 'l', 'l', 'o']
sorted(s)
```

Aufgabe 3-3

Schreibe ein Pyhton-Programm, dass

- alle geraden Zahlen von 0 bis 100 aufsummiert, benutze dabei *sum()* und *filter()*.
- erst eine Liste der größe nach sortiert und dann jedes Element quadriert.

```
unsorted = [3, -2, 4, 0, -16] # input

squared = [256, 4, 0, 9, 16] # output
```

I/O

Datei einlesen

```
# hello_world.txt:
#    Hello
#    World
#    !
with open("hello_world.txt") as file:
    for line in file:
        print(line)
# Output:
#    Hello
#
#    World
#
#    !
#
```

with

`with` sorgt dafür, dass die Datei nach dem Ende des Einlese-Blocks wieder geschlossen wird.

Datei schreiben

```python
with open("numbers.txt", mode="w") as file:
    for i in range(3):
        file.write(f"{i}\n")
# numbers.txt:
#   0
#   1
#   2
```

mode

Es gibt verschidene Modi, mit denen man eine Datei öffnen kann:

r	einlesen *(Standard)*
w	schreiben
w+	lesen und schreiben
a	an bestehende Datei anfügen

Aufgabe 3-4

Lese und schreibe eine Datei:

- Lese die Zeilen einer Text-Datei in eine Liste.
- Drehe die Reihenfolge der Elemente in der Liste um.
- Schreibe die umgedrehte Liste der Zeilen in eine neue Datei.