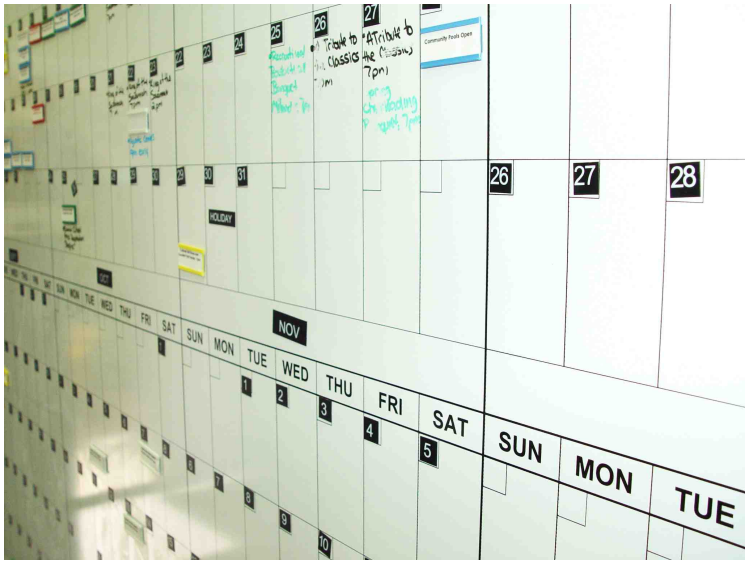
Quick-Start Informatik 2011

Programmieren in Python
Teil 3

Agenda

Rückblick
Rekursion
Klassen & Objekte



Rückblick

1. Listen ([1,2,3,4])
2. Wörterbücher
3. Wiederholtes Ausführen
   - for- & while-Schleifen
4. Eigene Funktionen (def ...)
5. Sichtbarkeit von Variablen (Scope)
6. Dateien lesen und schreiben
7. Turtle Graphics (import turtle)

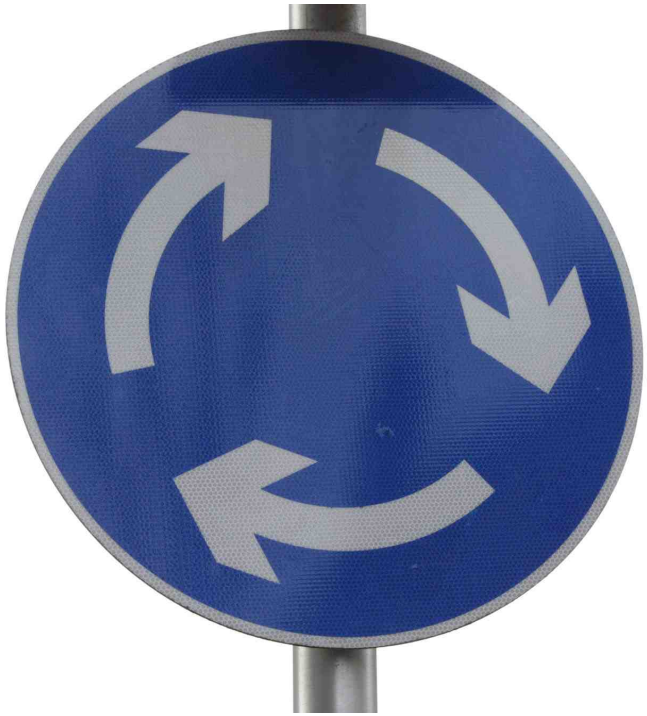
http://www.flickr.com/photos/kyriiee/4637080936

Rekursion

Rekursion

"Um Rekursion zu verstehen,
muss man erstmal Rekursion verstehen."

Prinzip:
Funktion ruft sich selbst auf



Rekursion

Wofür?

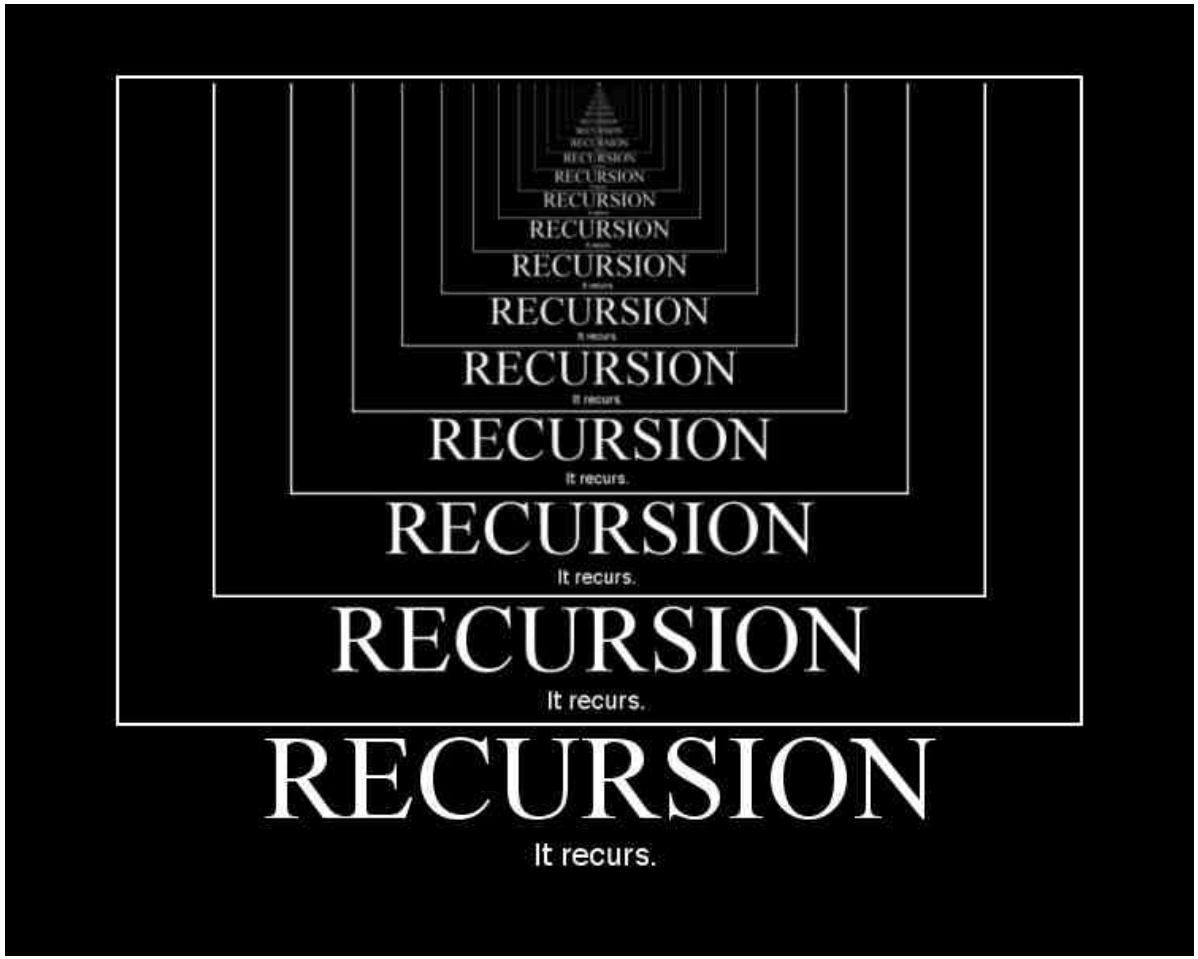
- Programmiertechnik,
  häufig eleganter & kürzer
  als ein iteratives Vorgehen

- Ausblick: Funktionale Programmierung

Rekursion

Wie Endlosrekursion stoppen?

```
def recursion():
    recursion()
```



http://www.flickr.com/photos/torley/2361164281/

Rekursion

Wie Endlosrekursion stoppen?

Wir brauchen eine **Abbruchbedingung**!

Vorgehen allgemein:

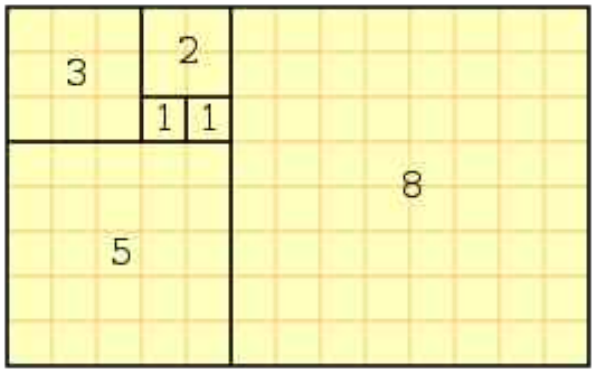
```
def rek_fkt(parameter):
    if BEDINGUNG:
        return ERGEBNIS
    else:
        # mache ggf. etwas...
        return rek_fkt(NEUER_WERT)
```

Abbruchbedingung
Rekursiver Aufruf

Rekursion

Beispiel Fibonacci-Zahlen:
(0, 1, 1, 2, 3, 5, 8, 13, …)

f(n) = f(n-1) + f(n-2)
f(0) = 0, f(1) = 1

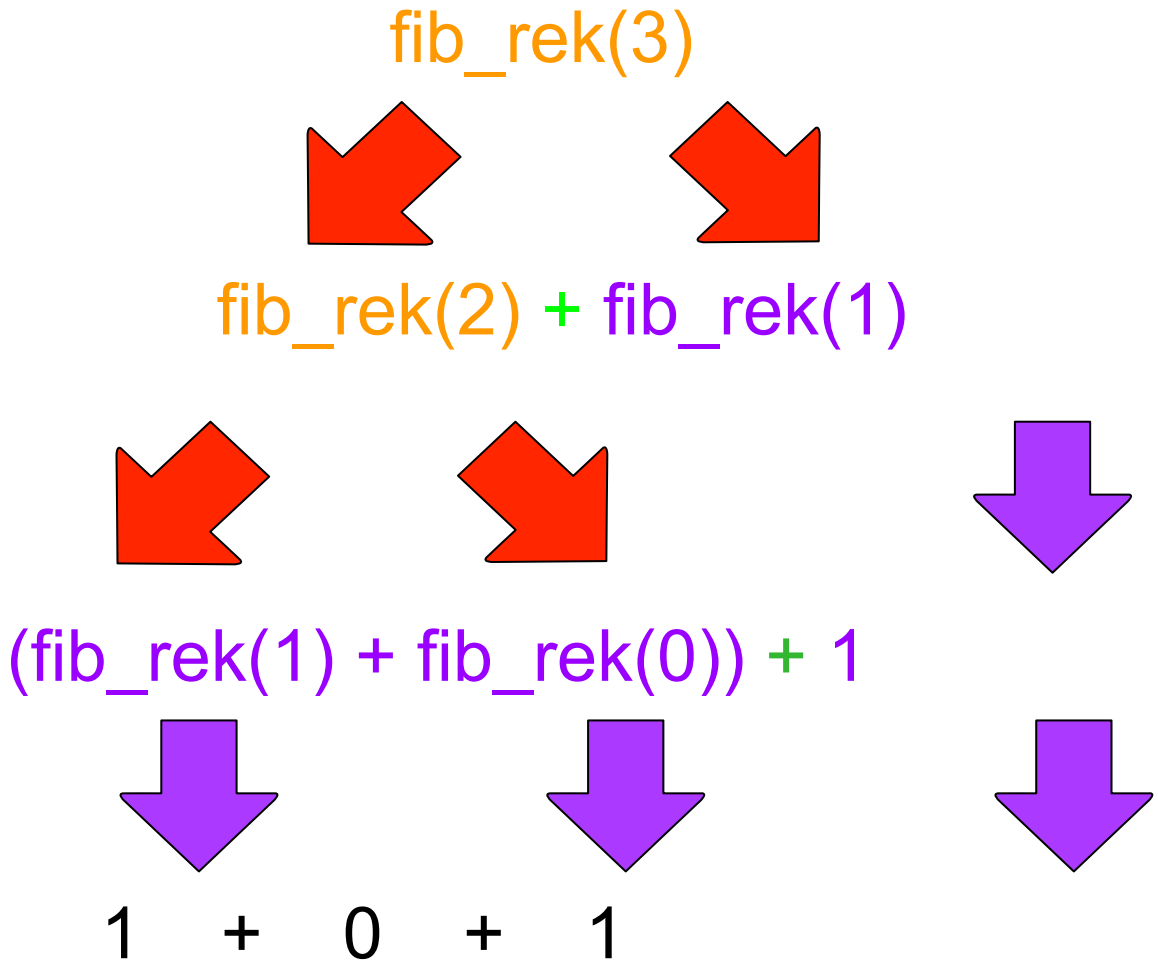


http://de.wikipedia.org/w/index.php?title=Datei:FibonacciBlocks.svg

```
def fib_rek(n):
    if n < 2:      # Abbruchbedingung
        return n
    else:          # Rekursion
        return fib_rek(n-1) + fib_rek(n-2)
```

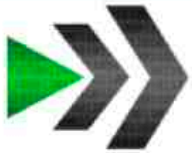
Rekursion

```
def fib_rek(n):
    if n < 2:
        return n
    else:
        return fib_rek(n-1) +
fib_rek(n-2)
```



=> Ergebnis: **2**

Rekursion vs. Iteration



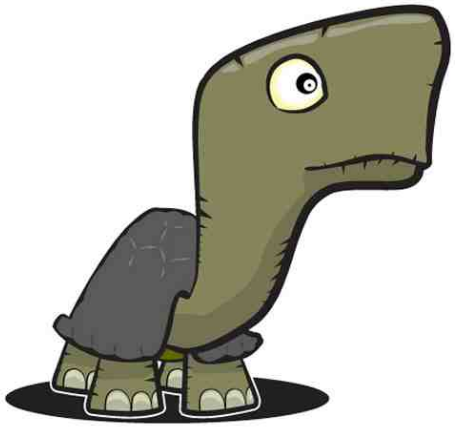
Beispiel Fakultätsfunktion

Iteration	Rekursion

def fak_it(n):
    if n < 2:
        return 1
 	
    result = n
    while n > 1:
        n -= 1
        result *= n
    return result
	def fak_rek(n):
	    if n < 2:
	        return 1
	    else:
	        return n *
	            fak_rek(n-1)

Rekursion & Turtle Graphics

Live Demo!



Klassen & Objekte

Objekte

Wiederholung:

- Objekte können als Literale oder per Konstruktor erstellt werden

```
>>> text = 'cheeseburger'
>>> homer = Turtle()
```

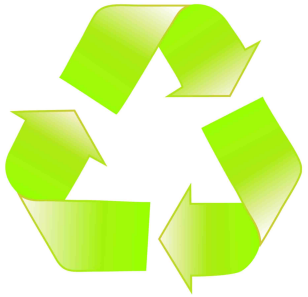
- Auf Objekte können Methoden angewandt werden

```
>>> text.replace('cheeseburger', 'cheezburger')
>>> homer.pencolor('green')
```

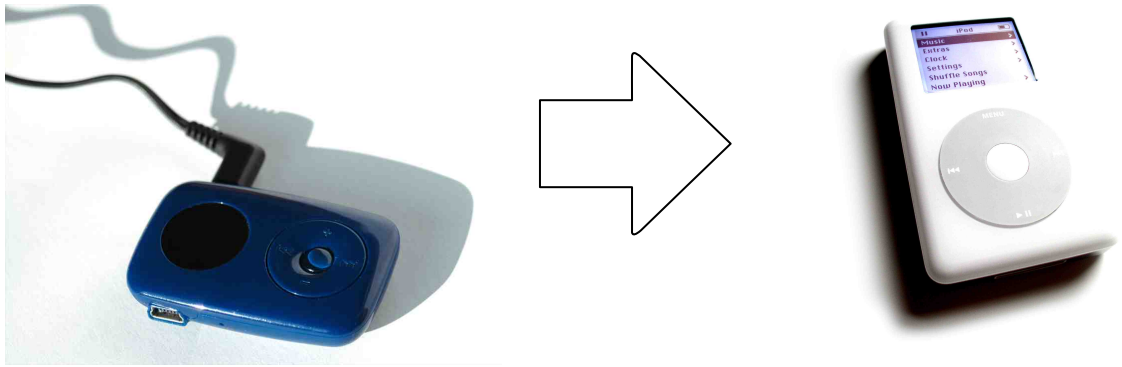
- In Python: Alles ist ein Objekt!

Wofür Klassen?

1. Wiederverwendbarkeit



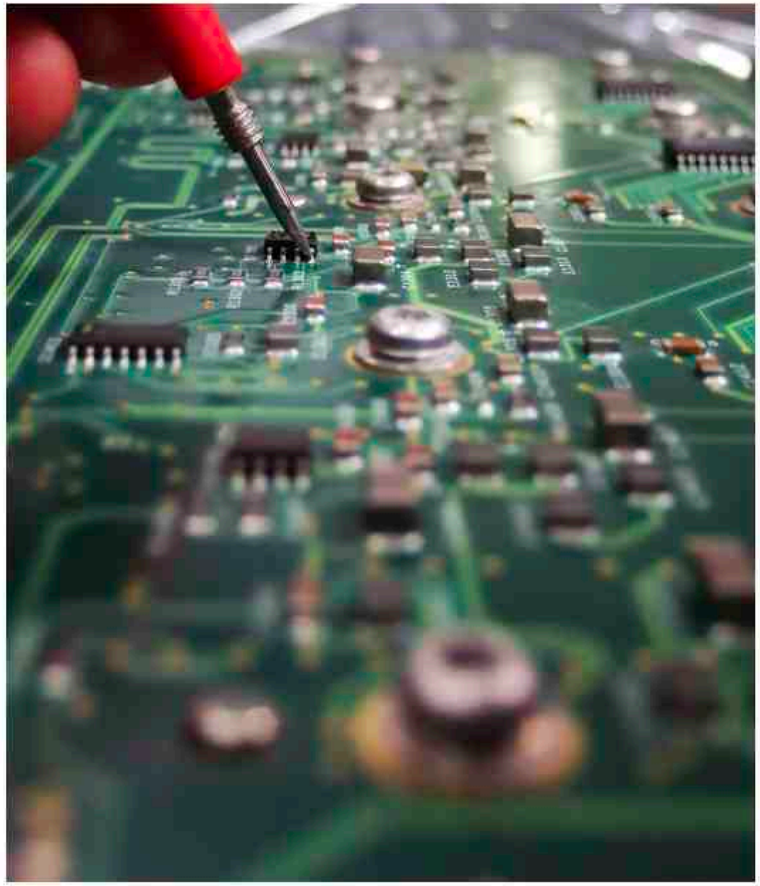
2. Austauschbarkeit



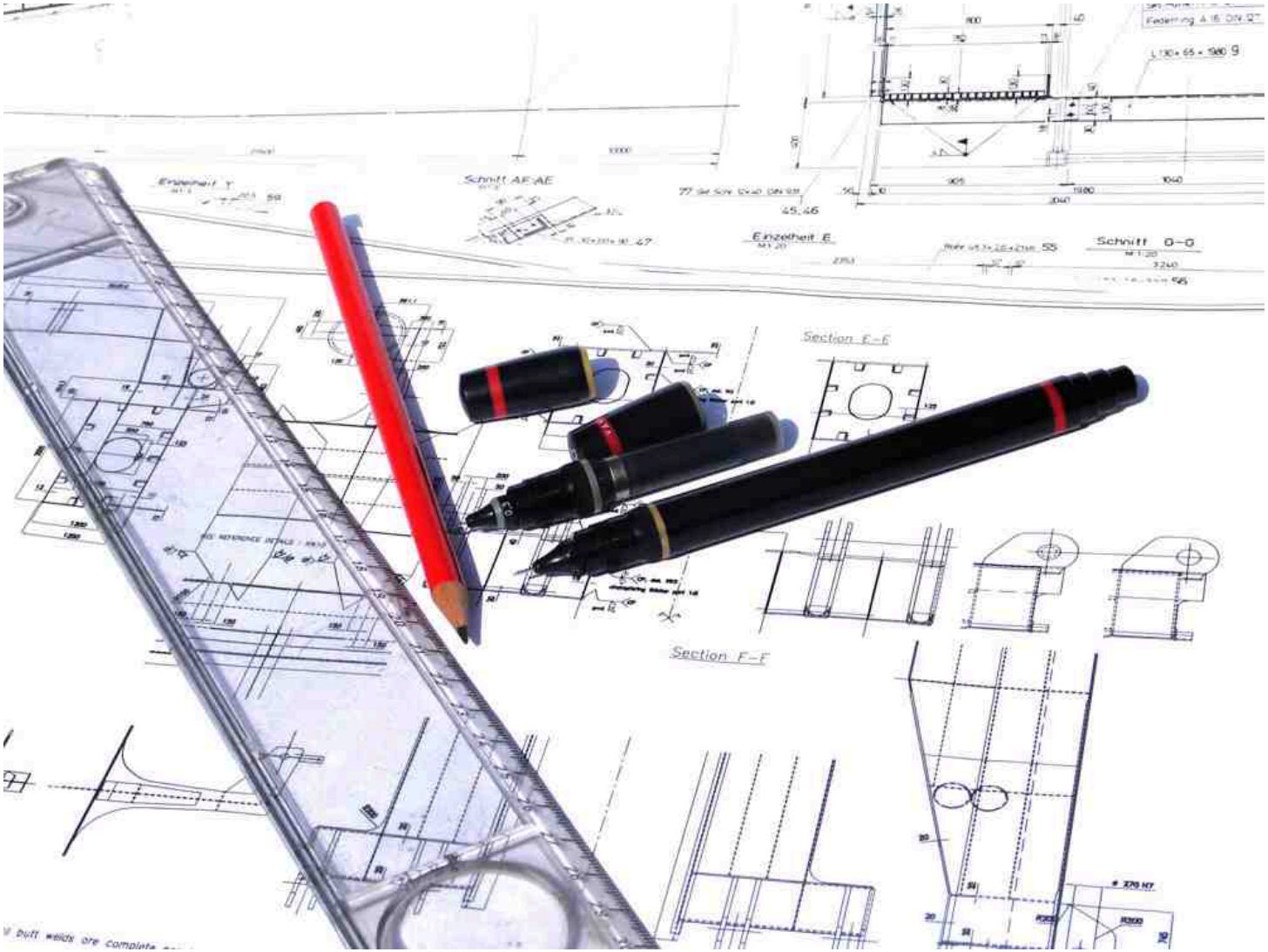
Äußeres & Inneres

Interface

Implementierung



Bauplan eines Objekt: Klasse



Aufbau einer Klasse

Live Demo!

Klassen: Zusammenfassung (1)

```
class Account:
    __number = 0
    __transactions = []
    owner = ''

    def __init__(self, no, owner_name):
        self.__number = no
        self.owner = owner_name

    def deposit(self, value):
        self.__transactions.append(value)

    def withdraw(self, value):
        self.__transactions.append(-value)

    def get_balance(self):
        return sum(self.__transactions)

    def __str__(self):
        return "Kontonummer: {}, Inhaber: {},
                Kontostand: {}".format(self.__number,
                self.owner, self.get_balance())
```

Attribute

Konstruktor

Methoden

Klassen: Zusammenfassung (2)

self

- Macht aus einer Funktion eine Methode
- Zeigt auf das dazugehörige Objekt

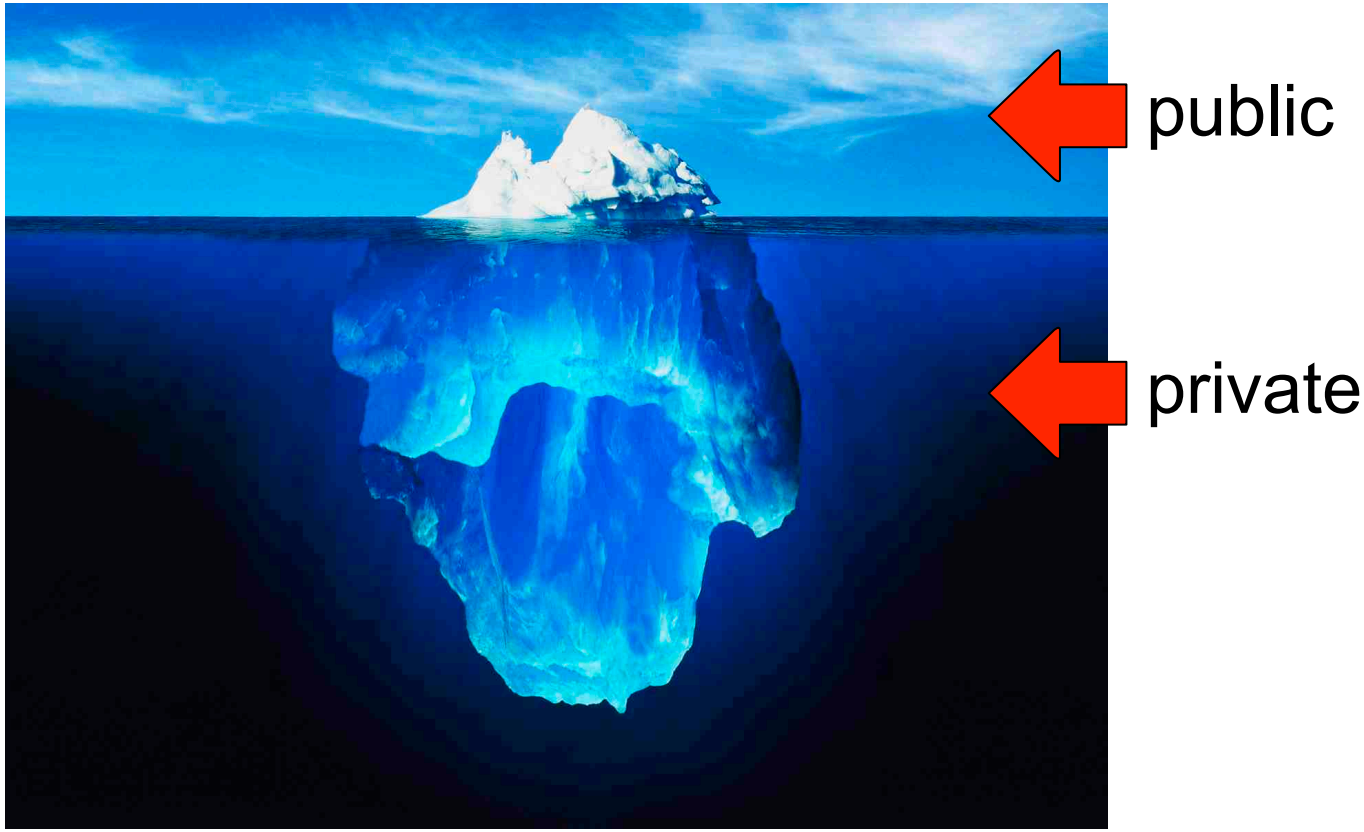
http://www.flickr.com/photos/marleneford/4151554537/

Klassen: Zusammenfassung (3)

Private (vs. Public)

- Attribute & Methoden können als private deklariert werden, indem ihr Name mit einem (doppelten) Unterstrich beginnt

- Wer die Klasse aufschraubt und sich an den Innereien (private Attributen und Methoden) vergeht, verliert die Garantie!



Vielen Dank für die Aufmerksamkeit!

http://www.flickr.com/photos/wwworks/4759535950/