Objektorientiere Programmierung

Datenkapselung und Sichtbarkeit

Prinzip: Datenkapselung

Unter Datenkapselung versteht man den Schutz von Daten bzw. Attributen vor dem unmittelbaren Zugriff. Der Zugriff auf die Daten bzw. Attribute erfolgt nur über entsprechende Methoden, die man auch als Zugriffsmethoden bezeichnet.

Das Prinzip der Datenkapselung wird meist so umgesetzt, dass das Ändern der Attribute nicht direkt erfolgt, sondern über eine Methode, die man als Setter bezeichnet. Der deutsche Fachbegriff ist Änderungsmethode.

Das „Holen“ oder Abfragen eines Wertes bezeichnet man als get. Dementsprechend wird die Methode zum Abfragen eines Wertes als Getter bzeichnet oder deutsch Abfragemethode.

Das Konzept der Setter und Getter - wird in Python allerdings nicht durch Methoden wie in Java erreicht, sondern über Decorator.

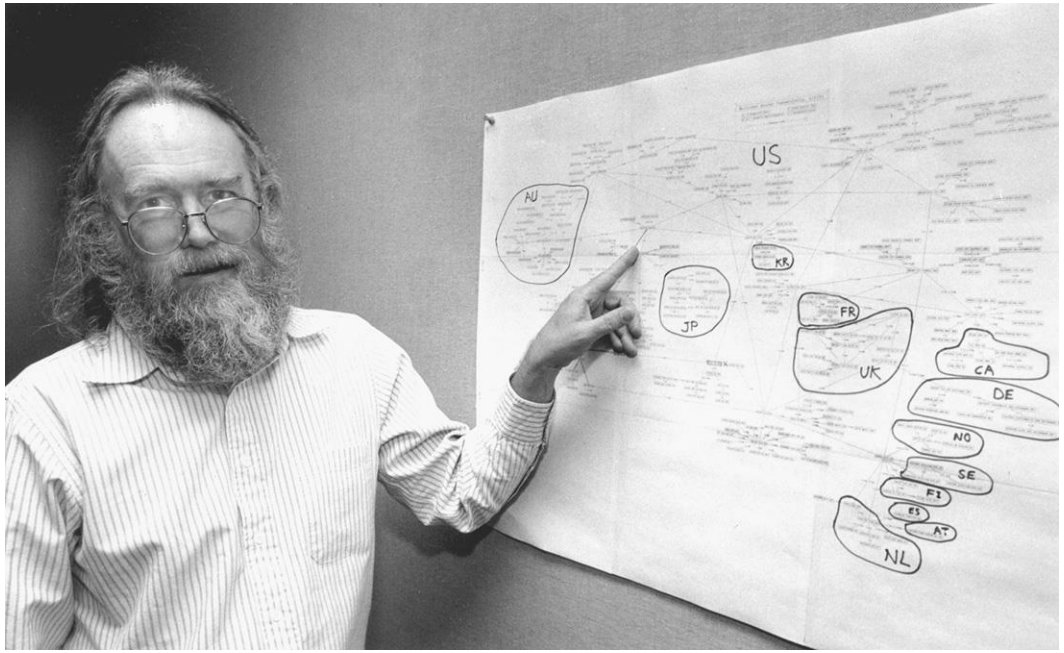
Postels Gesetz

"be conservative in what you do, be liberal in what you accept from others"

"sei streng bei dem was du tust und offen bei dem, was du von anderen akzeptierst"

Robustheitsgrundsatz (Postel's Law)

Jon Postel, Internetpionier



Public Attribut

Attribute in einer Klasse sind **per default** public, also öffentlich.

```
class Person:
    def __init__(name):
        self.name = name

p = Person("John")
p.name = "Pete"
```

Man kann das Attribut name von außen verändern. Das ist nur möglich, weil name eine öffentliche Variable ist.

In Java, PHP oder C++ definiert man solche Variablen mit dem Keyword public.

Public Methode

Genauso, wie Attribute einen Sichtbarkeitsstatus haben, haben auch Methoden einen. Auch Methoden sind in Python per default public, auf sie kann von außen zugegriffen werden.

```
class Person:
    def __init__(name):
        self.name = name

    def say_hello():
        print(self.name)

p = Person("John")
p.say_hello()
```

nicht-öffentliche Attribute

In Python wird Kapselung durch Konventionen erreicht und nicht durch strenge Regeln. Es gibt streng genommen keine protected Attribute wie in Java/C++. Attribute, die mehr einen internen Charakter haben, und nicht für den Zugriff von außen gedacht sind, können mit einem Underscore gepräfixt werden.

Deshalb ist folgendes Vorgehen möglich, aber nicht empfohlen.

```
class Person:
    def __init__(name):
            self.name = name
            self._internal = 342

p = Person("Jean")
p._internal = 42   # nicht empfohlen
```

nicht-öffentliche Attribute

Um die Variable von außen zu ändern, könnte man eine öffentliche Methode bereitstellen.

```
class Person:
    def __init__(name):
            self.name = name
            self._internal = 342

    def set_internal(value):
            self. _internal = value

p = Person("Jean)
p.set_internal(10)
```

Property: managed Attributes

Der property-Dekorator ist eine Möglichkeit, eine Methode von außen so anzusprechen, dass sie sich im Zugriff wie eine Variable verhält. Beim Setzen der Variable über @energy.setter können zum Beispiel Validierungen oder ähnliches vorgenommen werden. Der Zugriff von außebn über den Dot-Operator ändert sich nicht.

```
Class Player:


  def  __init__(self, name, energy):
    self.engery = energy
    self.name = name


   @property
    def energy():
       return self._energy


   @energy.setter
    def energy(value):
       if value < 0:
          raise ValueError("negative energy is not possible")
       self._energy = value


p = new Person()
p.energy = 101
print(p.energy)
```