LIF-Qualifikationskurs
zu
Intelligenten Suchverfahren
mit Python

Objektorientierung mit Python

Python ist objektorientiert

- Ein großer Teil der Sprache ist objektorientiert definiert
- Das hat auch Probleme zur Folge (s.o.):
  - Listen sind Objekte
  - call-by-reference bei Aufrufen
  - Methoden verändern das Objekt dauerhaft
  - Zuweisungen (a=b) erzeugen keine neuen Objekte

Objektorientierung mit Python

Syntax am Beispiel

- Klassenkopf
- Konstruktor

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Objektorientierung mit Python

Syntax am Beispiel

- Klassenkopf

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Objektorientierung mit Python

Syntax am Beispiel

- Klassenkopf mit nachfolgendem Doppelpunkt
- und Einrückung

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Syntax am Beispiel

- Konstruktor ist Methode mit besonderer Kennzeichnung

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Syntax am Beispiel

- Das Objekt selbst wird mit self gekennzeichnet und muss dem Konstruktor als Parameter übergeben werden.

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Syntax am Beispiel

- Auf eine Instanzvariable wird mit self vor dem Namen zugegriffen

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Objektorientierung mit Python

Zugriffe auf Attribute und Methoden von Objekten

- immer mit Punktnotation

```
class Zaehler:
    def __init__(self):
        self.stand = 0
```

Eine prozedurale Methode (der Instanzen)

- Einbau der Zaehle()-Methode und ...

```
class Zaehler:
    def __init__(self):
        self.stand = 0
    def Zaehle(self):
        self.stand += 1
    def ZeigeStand(self):
        return self.stand
```

Eine funktionale Methode (der Instanzen)

- … Einbau der ZeigeStand()-Methode

```
class Zaehler:
    def __init__(self):
        self.stand = 0
    def Zaehle(self):
        self.stand += 1
    def ZeigeStand(self):
        return self.stand
```

Vererbung

- (nicht überraschend) durch Angabe der vererbenden Klasse hinter dem Klassennamen in Klammern

```
class ZyklischerZaehler(Zaehler):
    def __init__(self, zyklusLaenge):
```

Vererbung

- Der Aufruf des Konstruktors der vererbenden Klasse ist zwingend notwendig!

```
class ZyklischerZaehler(Zaehler):
    def __init__(self, zyklusLaenge):
        Zaehler.__init__(self)
```

Objektorientierung mit Python

ACHTUNG:
Python kennt Mehrfachvererbung!

Vererbung

- (nicht überraschend) durch Angabe der vererbenden Klasse hinter dem Klassennamen in Klammern

```
class ErbtVonZwei(Erste_Klasse , ZweiteKlasse):
    def __init__(self):
        Erste_Klasse.__init__(self)
        Zweite_Klasse.__init__(self)
```