Objektorientierung mit Python

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
```

Objektorientierung mit Python

Syntax am Beispiel

- Konstruktor
- ist eine Methode mit besonderer Kennzeichnung

```
class Zaehler:

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

Objektorientierung mit Python

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
```

Objektorientierung mit Python

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
```

Objektorientierung mit Python

Instanz-Methoden

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

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

Objektorientierung mit Python

Kapselung

- Private Attribute durch Angabe von zwei Unterstreichungsstrichen vor dem Namen

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

Objektorientierung mit Python

Vererbung

- zunächst ohne große Überraschungen durch Angabe der vererbenden Klasse hinter dem Klassenamen in Klammern

```
class ZyklischerZaehler(Zaehler):

    def __init__(self, zyklusLaenge):
```

Objektorientierung mit Python

Vererbung

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

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

Objektorientierung mit Python

Vererbung

- Das selbe Problem wie bei Java:
- Ein Zugriff auf die geerbte private Variable ist [offiziell] nicht möglich

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

    def Zaehle(self):
        self.__stand += 1
```

geht nicht

Objektorientierung mit Python

Vererbung

- Ein Zugriff auf die geerbte private Variable ist bei Python trotzdem möglich.

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

    def Zaehle(self):
        self._Zaehler__stand += 1
```

geht aber!

Objektorientierung mit Python

Vererbung

- Eine vollständige Zaehle()-Methode von ZyklischerZaehler könnte also so aussehen:

```
class ZyklischerZaehler(Zaehler):
    ...
    def Zaehle(self):
        self._Zaehler__stand += 1
        if self.ZeigeStand() == self.zyklus:
            self._Zaehler__stand = 0
```