

Informatik I: Einführung in die Programmierung

11. Veränderliche Daten

Albert-Ludwigs-Universität Freiburg



UNI
FREIBURG

Prof. Dr. Peter Thiemann
2. Dezember 2025

1 Veränderliche Daten



UNI
FREIBURG

Veränderli-
che Daten

Identität und
Gleichheit

Parameter
mit Stan-
dardwert und
Namen

■ Identität und Gleichheit

- Bisher haben wir Objekte als unveränderlich (*immutable*) betrachtet.
- Manche Objekte können während des Programmablaufs verändert werden.
- Veränderliche Objekte:
 - Listen (Typ `list`)
 - Instanzen von Datenklassen
- Bei veränderlichen Objekten ist die **Objektidentität** wichtig!

Identität: `is` und `is not`

- Jedes Objekt besitzt eine eigene Identität.
- Die Operatoren `is` und `is not` testen die Identität.
- `x is y` ist True, wenn `x` und `y` **dasselbe Objekt** bezeichnen, und ansonsten False (bzw. umgekehrt bei `is not`):

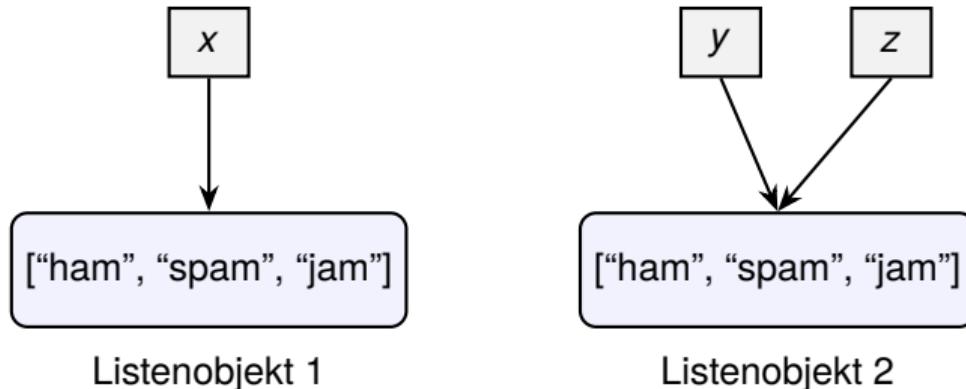
```
>>> x, y = ["ham", "spam", "jam"], ["ham", "spam", "jam"]
>>> z = y
>>> x is y, x is z, y is z
(False, False, True)
>>> x is not y, x is not z, y is not z
(True, True, False)
```

Veränderliche Daten

Identität und Gleichheit

Parameter mit Standardwert und Namen

Aliasing



- x enthält die **Adresse**, wo Listenobjekt 1 im Speicher abgelegt ist.
- y und z enthalten die **Adresse**, wo Listenobjekt 2 im Speicher abgelegt ist.
- Der `is` Operator testet die Gleichheit dieser Adressen!

Identität vs. Gleichheit



- Außer Zahlen und Strings können auch Listen und Tupel auf Gleichheit getestet werden. Der Unterschied zum Identitätstest ist wichtig:

```
>>> x = ["ham", "spam", "jam"]
>>> y = ["ham", "spam", "jam"]
>>> x == y, x is y
(True, False)
```

- Test auf *Gleichheit*: Haben x und y den gleichen Typ? Sind sie gleich lang? Sind korrespondierende Elemente gleich? (die Definition ist rekursiv)
- Test auf *Identität*: bezeichnen x und y dasselbe Objekt? (Adressvergleich)

Faustregel

Verwende in der Regel den Gleichheitstest.

Spezialfall x is None

Anmerkung zu None:

- Der Typ `NoneType` hat nur einen einzigen Wert: `None`. Daher ist es egal, ob ein Vergleich mit `None` per Gleichheit oder per Identität erfolgt.
- Vergleiche mit `None` sollten mit
`x is None` bzw. `x is not None`
und `nicht` mit
`x == None` bzw. `x != None`
erfolgen.

Veränderliche Daten

Identität und Gleichheit

Parameter mit Standardwert und Namen

Veränderlich oder unveränderlich?



Jetzt können wir auch genauer sagen, was es mit veränderlichen (*mutable*) und unveränderlichen (*immutable*) Datentypen auf sich hat:

- Instanzen von veränderlichen Datentypen können modifiziert werden.
Vorsicht bei Zuweisungen wie $x = y$:
Nachfolgende Operationen auf x beeinflussen auch y (und umgekehrt).
 - Beispiel: Listen (`list`)
- Instanzen von unveränderlichen Datentypen können nicht modifiziert werden.
Daher sind Zuweisungen wie $x = y$ völlig unkritisch:
Da das durch x bezeichnete Objekt nicht verändert werden kann, besteht keine Gefahr für y .
 - Beispiele: Zahlen (`int`, `float`, `complex`), Strings (`str`), Tupel (`tuple`)

Weitere Operationen auf Listen

Zuweisung an Listenelement



```
>>> xs, ys = ["prolific", "proselyte"], ["prolific", "proselyte"]
>>> zs = xs
>>> print(" ".join(xs))
prolific proselyte
>>> xs[1] = "procrastinator"
>>> print(" ".join(xs)); print(" ".join(zs))
prolific procrastinator
prolific procrastinator
>>> print(" ".join(ys))
prolific proselyte
```

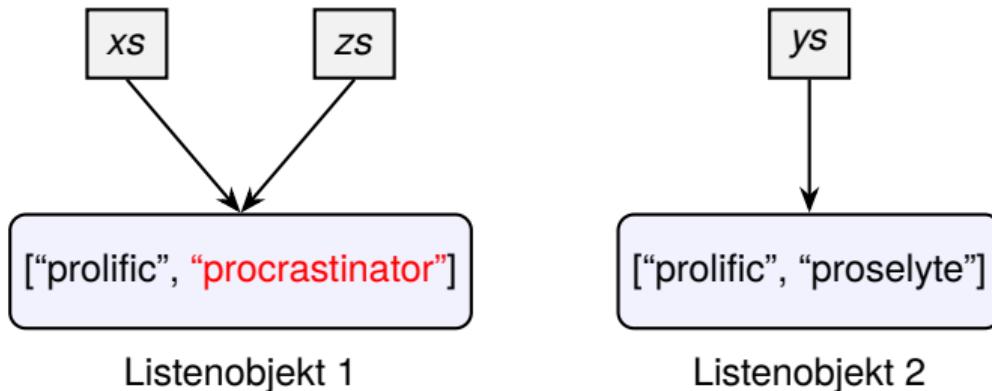
Veränderliche Daten

Identität und Gleichheit

Parameter mit Standardwert und Namen

- Zuweisung an `xs[i]` ändert Position `i` in der Liste `xs`. Dabei muss `-len(xs) <= i < len(xs)` ein legaler Index sein.
- `xs` und `zs` sind **Aliase** (sie verweisen auf das selbe Listenobjekt); daher schlägt jede Änderung an `xs` auf `zs` durch und umgekehrt.
- `ys` ist ein separates Listenobjekt und ist von Änderungen an `xs` nicht betroffen.

Zuweisung an Listenelement (Bild)



nach der Zuweisung `xs[1] = "procrastinator"`
`xs` und `zs` verändert, `ys` unverändert

Weitere Operationen auf Listen

Löschen von Listenelement



UNI
FREIBURG

```
>>> print(" ".join(xs))
prolific procrastinator
>>> del zs[0]
>>> print(" ".join(xs)); print(" ".join(zs))
procrastinator
procrastinator
>>> print(" ".join(ys))
prolific proselyte
```

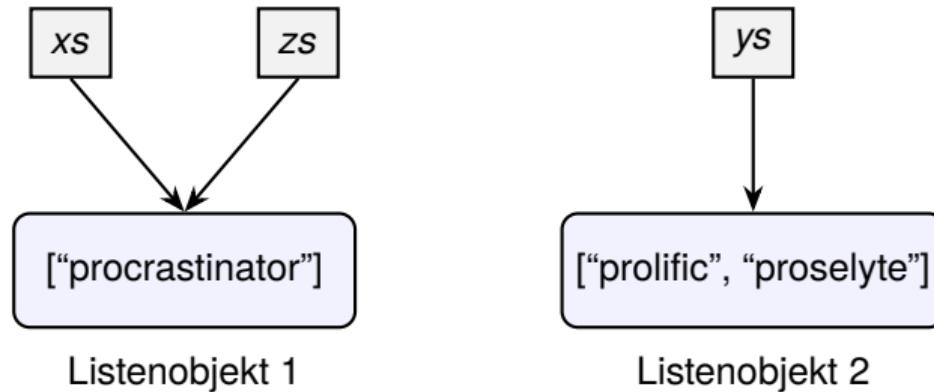
Veränderliche Daten

Identität und Gleichheit

Parameter mit Standardwert und Namen

- `del zs[i]` entfernt Position *i* aus der Liste *zs*. Dabei muss *i* ein legaler Index sein.

Löschen von Listenelement (Bild)



nach der Anweisung `del zs[0]`
`xs` und `zs` verändert, `ys` unverändert

Weitere Operationen auf Listen

Anhängen von Listenelementen



```
>>> print(" ".join(xs))
procrastinator
>>> xs.append("predator") # add a single element at the end
>>> print(" ".join(xs))
procrastinator predator
>>> xs.extend(ys) # add all elements at the end
>>> print(" ".join(xs))
procrastinator predator prolific proselyte
```

Veränderliche Daten

Identität und Gleichheit

Parameter mit Standardwert und Namen

- `append()` und `insert()` sind **Methoden** der Klasse `list` und können daher mit allen Listenobjekten verwendet werden.
- Sie **verändern** jeweils ihr Zielobjekt
- Weitere Methoden: `insert()`, `remove()`, `pop()`, `reverse()`, `sort()`, `clear()` usw.
- Siehe Dokumentation.

Erzeugung von Instanzen



- Jeder Aufruf der Klasse als Funktion erzeugt eine neue **Instanz** der Klasse.

```
class Article:  
    pass  
  
instance1 = Article()  
instance2 = Article()  
print(instance1 is instance2, instance1 == instance2)  
print(isinstance(instance1, Article) , isinstance(0, Article))
```

Ausgabe: False False True False

- Alle erzeugten Instanzen sind untereinander nicht-identisch und ungleich!
- `isinstance()` prüft ob ein Objekt Instanz einer bestimmten Klasse ist.

Veränderliche Instanzen von Datenklassen



UNI
FREIBURG

■ Erinnerung: Binärbaum

```
>>> from dataclasses import dataclass
>>> from typing import Optional
>>> @dataclass
... class INode:
...     mark : int
...     left : Optional['INode'] = None
...     right : Optional['INode'] = None
...
```

Veränderli-
che Daten

Identität und
Gleichheit

Parameter
mit Stan-
dardwert und
Namen

Veränderliche Instanzen von Datenklassen



UNI
FREIBURG

```
>>> n1, n2 = INode(42), INode(42)
>>> print(n1)
INode(mark=42, left=None, right=None)
>>> n1.mark = 0
>>> print(n1); print(n2)
INode(mark=0, left=None, right=None)
INode(mark=42, left=None, right=None)
```

Veränderli-
che Daten

Identität und
Gleichheit

Parameter
mit Stan-
dardwert und
Namen

- Zuweisung an Attribute
- n1 und n2 sind unterschiedliche Instanzen. Daher wirkt die Zuweisung n1.mark = 0 nur auf n1.

Veränderliche Instanzen von Datenklassen



UNI
FREIBURG

```
>>> n2.left = n1
>>> print(n1); print(n2)
INode(mark=0, left=None, right=None)
INode(mark=42, left=INode(mark=0, left=None, right=None), right=None)

>>> n2.right = n1
>>> print(n1); print(n2)
INode(mark=0, left=None, right=None)
INode(mark=42, left=INode(mark=0, left=None, right=None), right=INode(mark=0,
```

Veränderli-
che Daten

Identität und
Gleichheit

Parameter
mit Stan-
dardwert und
Namen

Erinnerung: Unveränderliche Daten

Tupel



UNI
FREIBURG

Veränderli-
che Daten

Identität und
Gleichheit

Parameter
mit Stan-
dardwert und
Namen

```
>>> xt = (1, 2, 3)
>>> xt[1] = -2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- Tupel sind *immutable*, sie ändern ihren Wert nicht!

Erinnerung: Unveränderliche Daten

frozen Datenklassen



UNI
FREIBURG

Veränderli-
che Daten

Identität und
Gleichheit

Parameter
mit Stan-
dardwert und
Namen

```
>>> @dataclass(frozen=True)
... class FNode:
...     mark : int
...     left : Optional['FNode'] = None
...     right : Optional['FNode'] = None
...
>>> fn1 = FNode(42)
>>> print(fn1.mark)
42
>>> fn1.mark = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<string>", line 4, in __setattr__
dataclasses.FrozenInstanceError: cannot assign to field 'mark'
```

- Der Parameter `frozen=True` macht Instanzen einer Datenklasse unveränderlich.

2 Parameter mit Standardwert und Namen



UNI
FREIBURG

Veränderli-
che Daten

Parameter
mit Stan-
dardwert und
Namen

Einschub: Parameter mit Standardwert



Die letzten Parameter einer Funktion können einen **Standardwert** haben (im Beispiel der Parameter y). Die entsprechenden Argumente dürfen beim Funktionsaufruf weggelassen werden; in dem Fall erhalten die Parameter den Standardwert.

```
>>> def f (x: int, y: int = 0) -> int:  
...     return x - y  
...  
>>> assert f (3, 5) == -2  
>>> assert f (3) == 3
```

Einschub: benannte Parameter



Argumente können auch über den Namen des Parameters übergeben werden, die Reihenfolge der benannten Argumente spielt dann keine Rolle.

```
>>> def f (x: int, y: int) -> int:  
...     return x - y  
...  
>>> assert f (3, 5) == -2  
>>> assert f (x=3, y=5) == -2  
>>> assert f (y=3, x=5) == 2
```

Veränderliche Daten

Parameter mit Standardwert und Namen

Regel für den Funktionsaufruf

- Erst die unbenannten Argumente (positional arguments).
- Dann die benannten Argumente.
- Fehlende Argumente müssen einen Standardwert haben.