Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

FS24 CAS PML - Python

Niklaus Johner

niklausbernhard.johner@bfh.ch

FS24 CAS PML - Python

10. Veränderlichkeit (mutability)

Veränderlichkeit

- In python gibt es veränderliche und unveränderliche Objekte (mutable und immutable)
- Zum Beispiel:
  - Listen sind veränderlich

```
In [24]: l = [1, 2]
    ...: l[0] = 2
    ...: print(l)
    ...:
[2, 2]
```

Veränderlichkeit

- In python gibt es veränderliche und unveränderliche Objekte (mutable und immutable)
- Zum Beispiel:
  - Listen sind veränderlich
  - Tuples sind unveränderlich

```
In [25]: t = (1, 2)
    ...: t[0] = 2
    ...:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-292c72ca9dd8> in <module>()
      1 t = (1, 2)
----> 2 t[0] = 2

TypeError: 'tuple' object does not support item assignment
```

Veränderlichkeit

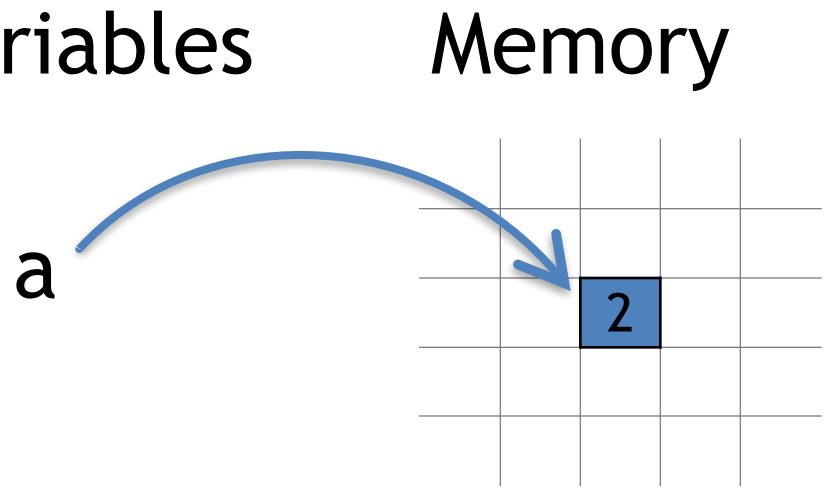
- Veränderliche Objekte können verändert werden
- Unveränderliche Objekte können nicht verändert werden

Mutable	**Immutable**
list	int, float, complex
set	tuple
dict	str

Unveränderliche Objekten

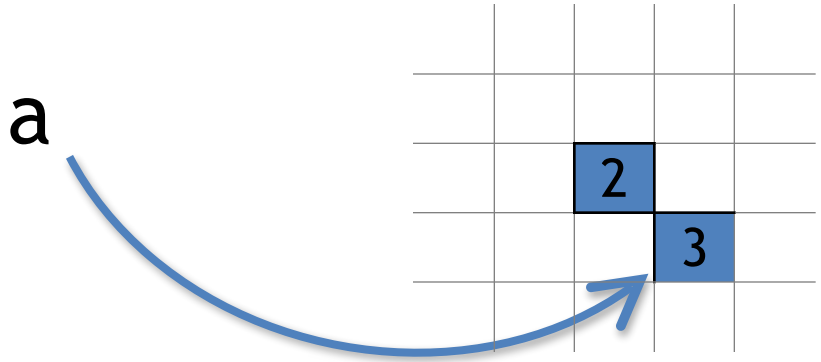
```
a = 2
print(a, id(a))
```

```
(2, 140473914148416)
```




```
a += 1
print(a, id(a))
```

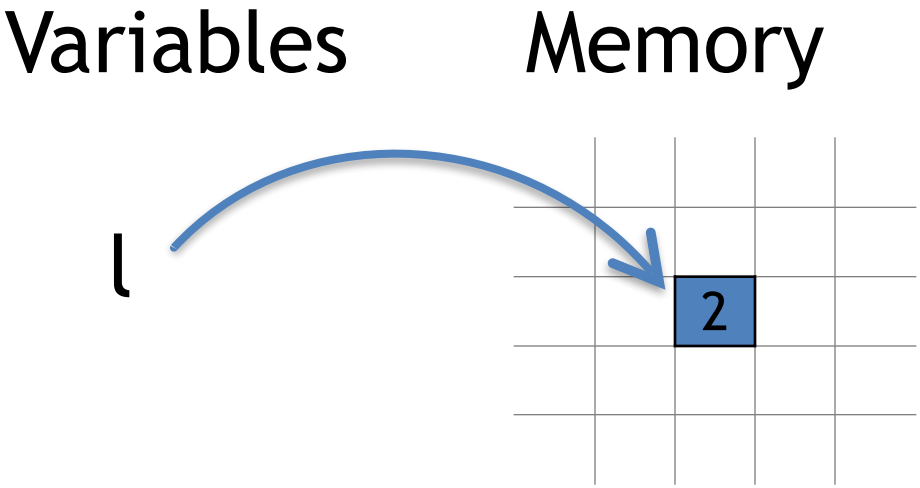
```
(3, 140473914148392)
```



Veränderliche Objekten

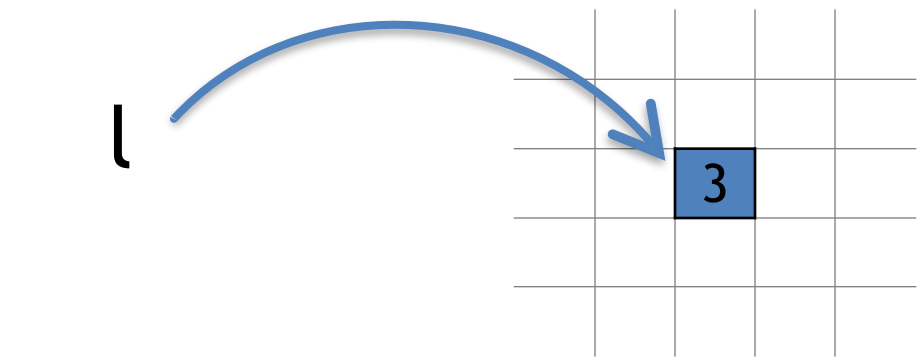
```
l = [2]
print(l, id(l))
```

```
([2], 4373917776)
```



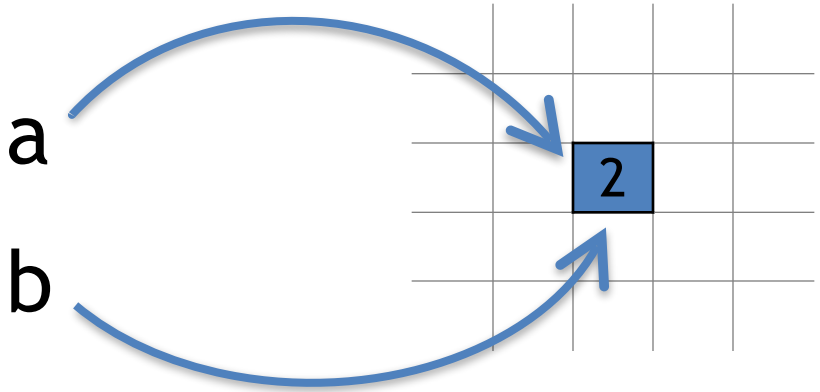

```
l[0] += 1
print(l, id(l))
```

```
([3], 4373917776)
```



Veränderlich vs unveränderlich

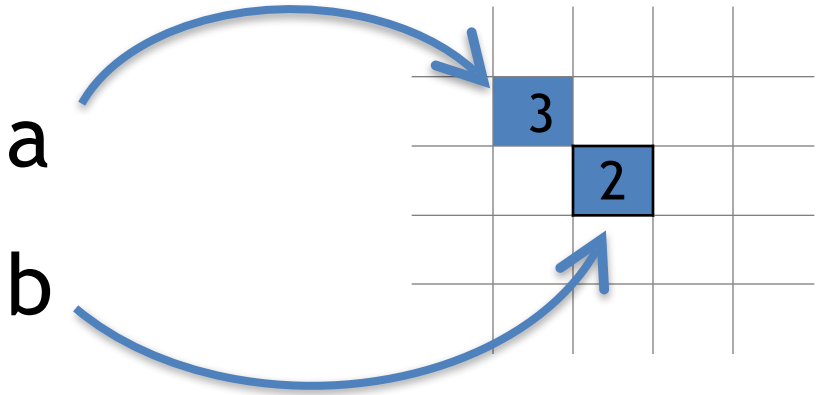
```
a = 2
b = a
print(a, b)      (2, 2)
a += 1
print(a, b)
```




```
l1 = [2]
l2 = l1
print(l1, l2)
l1[0] += 1
print(l1, l2)
```

Veränderlich vs unveränderlich

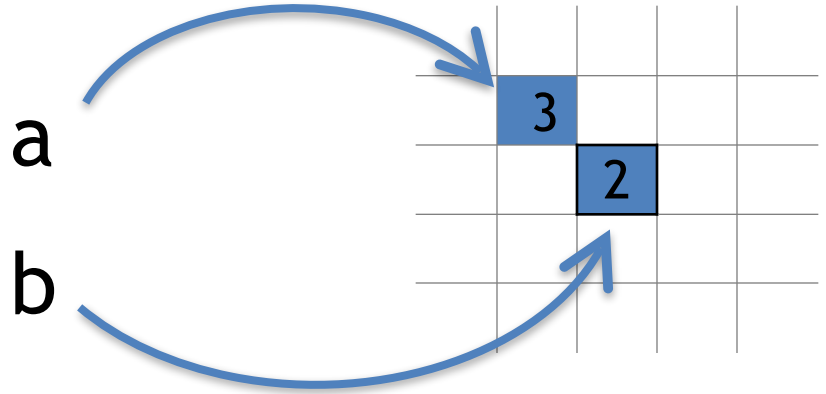
```
a = 2
b = a
print(a, b)      (2, 2)
a += 1
print(a, b)      (3, 2)
```



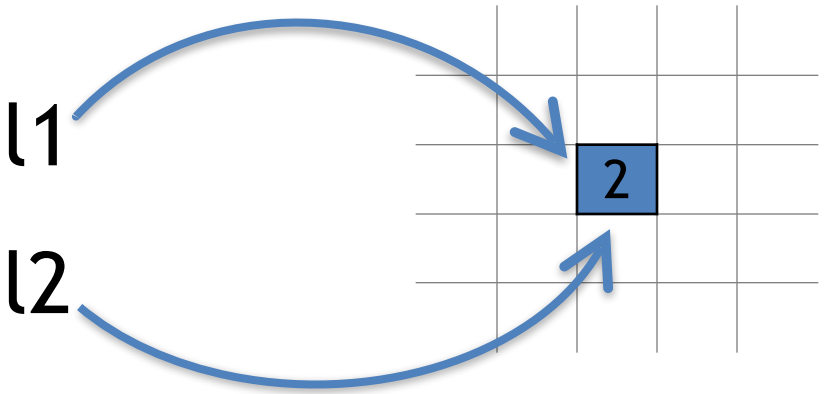

```
l1 = [2]
l2 = l1
print(l1, l2)
l1[0] += 1
print(l1, l2)
```

Veränderlich vs unveränderlich

```
a = 2
b = a
print(a, b)      (2, 2)
a += 1
print(a, b)      (3, 2)
```

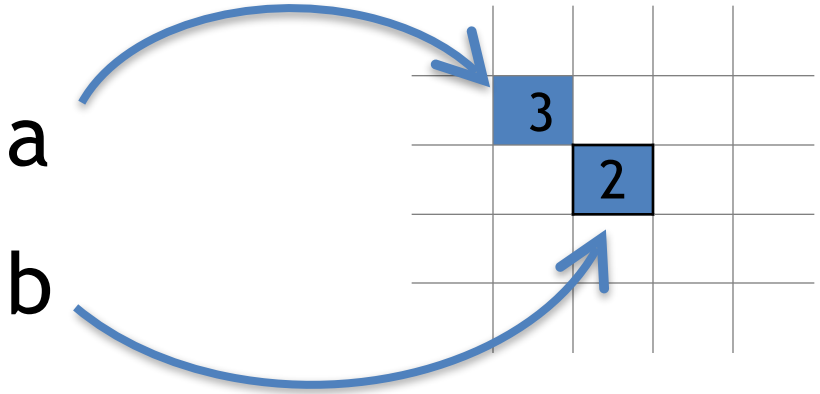



```
l1 = [2]
l2 = l1
print(l1, l2) ([2], [2])
l1[0] += 1
print(l1, l2)
```

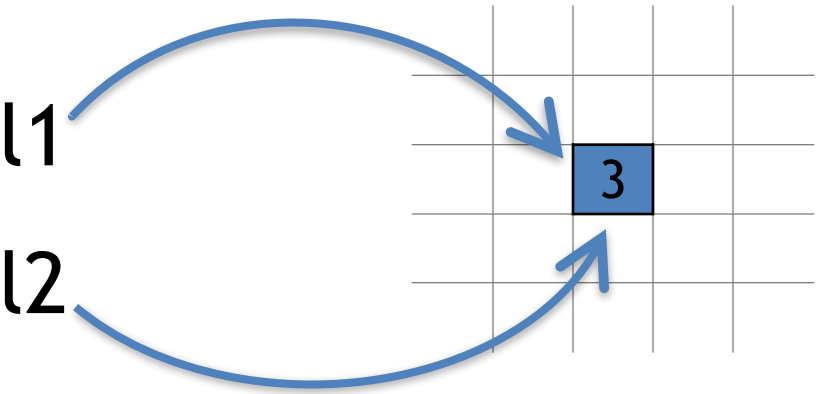


Veränderlich vs unveränderlich

```
a = 2
b = a
print(a, b)      (2, 2)
a += 1
print(a, b)      (3, 2)
```




```
l1 = [2]
l2 = l1
print(l1, l2) ([2], [2])
l1[0] += 1
print(l1, l2) ([3], [3])
```



Veränderlich vs unveränderlich

- Veränderliche Objekte werden *in place* verändert, welches alle Variablen beeinflusst die auf das Objekt zeigen
- Für unveränderliche Objekte muss ein neues Objekt kreiert werden, jedesmal wenn ein neuer Wert zu einer Variablen zugewiesen wird

```
In [33]: l1 = l2 = [1, 2]
    ...: l1 += [3, 4]
    ...: print("l1 =",l1)
    ...: print("l2 =",l2)
    ...:
l1 = [1, 2, 3, 4]
l2 = [1, 2, 3, 4]
```

```
In [34]: t1 = t2 = (1, 2)
    ...: t1 += (3, 4)
    ...: print("t1 =",t1)
    ...: print("t2 =",t2)
    ...:
t1 = (1, 2, 3, 4)
t2 = (1, 2)
```

Veränderlich vs unveränderlich

- Unveränderliche Objekte werden während Iterierung nicht modifiziert

```
In [69]: l = [1, (1,), [1], "1"]
    ...: for el in l:
    ...:     el *= 2
    ...:     print("el =", el)
    ...: print("l =", l)
    ...:
el = 2
el = (1, 1)
el = [1, 1]
el = 11
l = [1, (1,), [1, 1], '1']
```

```
el = l[0]
el *= 2
el = l[1]
el *= 2
el = l[2]
el *= 2
el = l[3]
el *= 2
```

Elemente einer Liste während Iteration modifizieren

- Um die Elemente einer Liste während einer Schleife zu modifizieren, sollte man auf das Element durch sein Index zugreifen

```
In [32]: l = [1, 2, 3, 4]
    ...: for el in l:
    ...:     el += 5
    ...: print(l)
    ...:
[1, 2, 3, 4]
```

```
In [33]: l = [1, 2, 3, 4]
    ...: for i in range(len(l)):
    ...:     l[i] += 5
    ...: print(l)
    ...:
[6, 7, 8, 9]
```

Veränderlich vs unveränderlich

- Eine Liste über welche man iteriert sollte man nicht modifizieren

```
l = [1, 2, 3, 4]
for el in l:
    p = l.pop()
    print("popped", p)
print("remaining", l)
```

```
popped 4
popped 3
remaining [1, 2]
```

```
l = [1, 2, 3, 4]
for el in l:
    p = l.pop(0)
    print("popped", p)
print("remaining", l)
```

```
popped 1
popped 2
remaining [3, 4]
```