MUTASJON OG ALIAS

INF100

VÅR 2026

Crystal Chang Din

Med bidrag fra: Torstein Strømme

'dette er en streng'

egentlig bare en samling med tegn

```
['d', 'e', 't', 't', 'e', ' ', 'e', 'r', ' ', 'e', 'n', ' ', 'l', 'i', 's', 't', 'e']
```

egentlig bare en samling med ting

list versus str : likheter

x = [‘a', ‘b', ‘c']

x = ‘abc'

- Indeksering
- Beskjæring
- Løkker
- Operasjoner
  - in, not in, +, *
- Funksjoner og metoder
  - len, min, max
  - .count, .index

```
x[0]

x[1:]

for el in x:
    ...

'a' in x

x * 2

'foo' + x

['f' , 'o' , 'o'] + x

len(x)

min(x)

max(x)

x.count('a')

x.index('b')
```

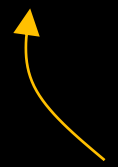
list versus str : forskjeller

x = [‘a', ‘b', ‘c']

x = ‘abc'

- Lister kan ha andre ting enn bare skrifttegn
- Lister kan muteres (endres)
- Strenger kan ikke muteres (endres)

```
x[0] = 'f'
```



Tilordning krasjer hvis x er en streng

Muterbare objekter kan endres etter at de er opprettet, mens immuterbare objekter ikke kan det.

Muterbare Objekter	**Immuterbare Objekter**
list, dict, set	**str**, int, float, bool, tuple

ENDRE FØRSTE BOKSTAV

Streng

```
s = 'abc'
s = 'x' + s[1:]

print(s) # xbc
```

Mutasjon av strenger er ikke mulig

Liste

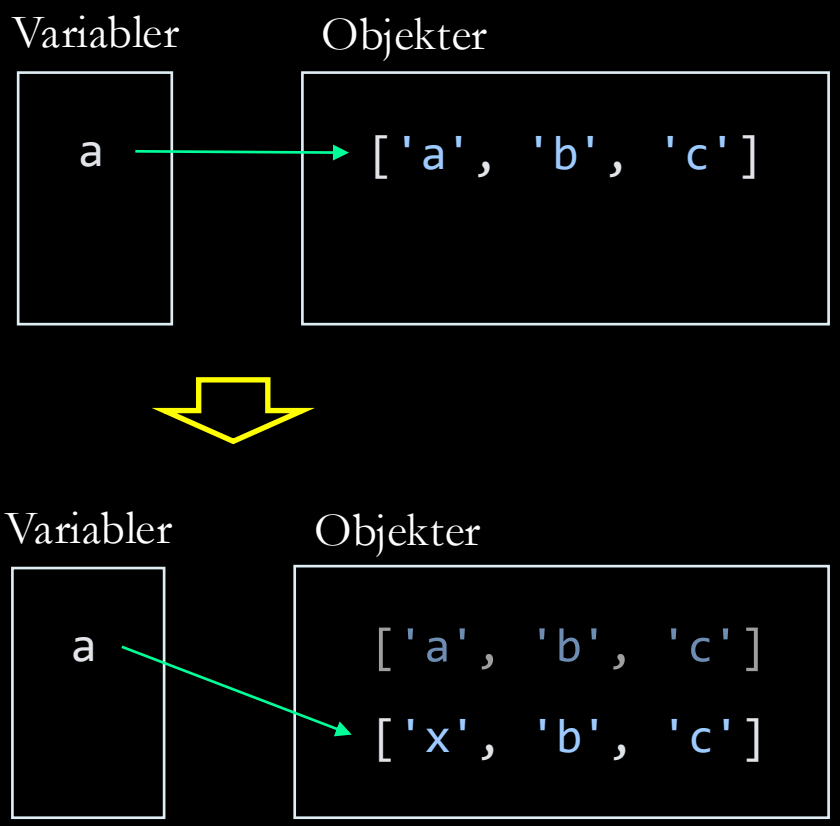
```
a = ['a', 'b', 'c']
a = ['x'] + a[1:]

print(a) # ['x', 'b', 'c']
```

Ikke-mutasjon: variabelen peker på nytt objekt

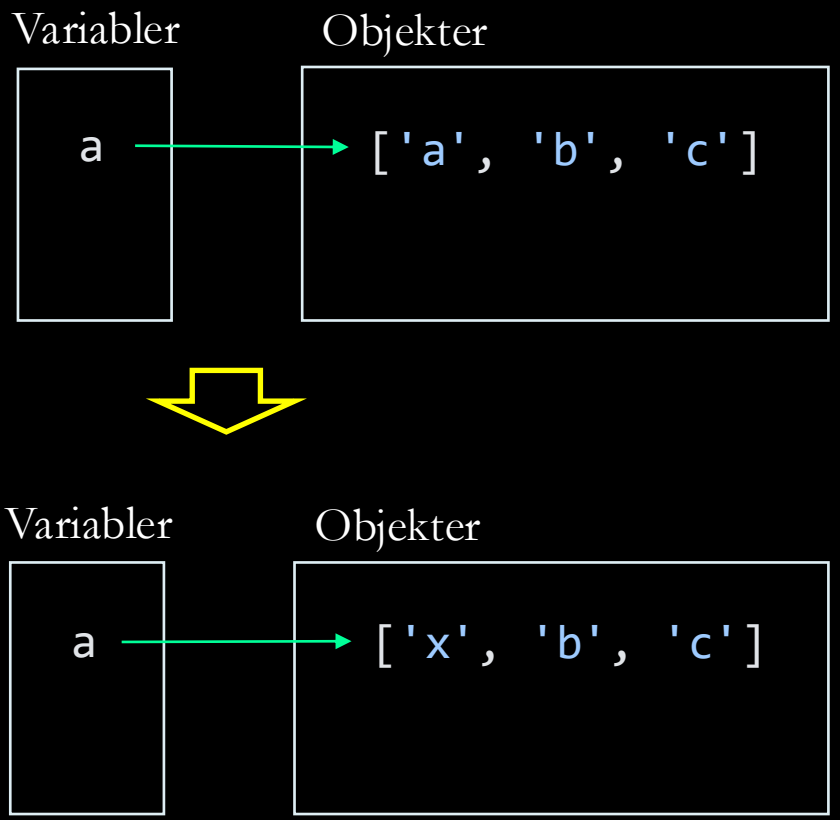
```
a = ['a', 'b', 'c']
a[0] = 'x'

print(a) # ['x', 'b', 'c']
```

Mutasjon: selve listen har blitt endret

```
a = ['a', 'b', 'c']
a = ['x'] + a[1:]

print(a) # ['x', 'b', 'c']
```




```
a = ['a', 'b', 'c']
a[0] = 'x'

print(a) # ['x', 'b', 'c']
```



MUTERE VS Å OPPRETTE NYTT

Å *mutere* et objekt innebærer å endre på objektetet i minnet.

Lister kan muteres. For eksempel, hvis vi har en liste

```
a = [42, 95, 66]
```

og ønsker å legge til en ny verdi i listen

```
a += [77, 88]
```

så er dette en endring som er slik at selve liste-objektet i minnet blir endret. Variabelen endres altså *ikke*, den peker fremdeles på akkurat samme objekt; det er bare at objektet i seg selv har blitt endret. Vi sier at listen har blitt *mutert*.

Ikke alle typer objekter kan muteres. For eksempel kan ikke `str`, `int`, `bool` eller `float` muteres. Når man gjør en operasjon for å regne ut en ny verdi av disse typene, vil det alltid opprettes et *nytt objekt* i minnet. For eksempel, hvis vi har en streng

```
s = 'abc'
```

og ønsker å legge til flere tegn i strengen

```
s += 'def'
```

så vil det opprettes et *nytt* objekt `'abcdef'` og variabelen `s` blir endret til å peke på det nye objektet.

Muterbare objekter kan endres etter at de er opprettet, mens immuterbare objekter ikke kan det.

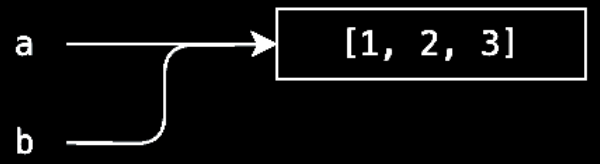
Muterbare Objekter	Immuterbare Objekter
list, dict, set	str, int, float, bool, tuple
Vi kan endre den interne innholdet i et muterbart objekt uten å endre dets unike identitet (minneadresse).	Enhver operasjon som ser ut til å endre et immuterbart objekt, oppretter faktisk et nytt objekt i minnet med en ny identitet (minneadresse).

ALIAS

To variabler er *aliaser* dersom de peker på det samme objektet.

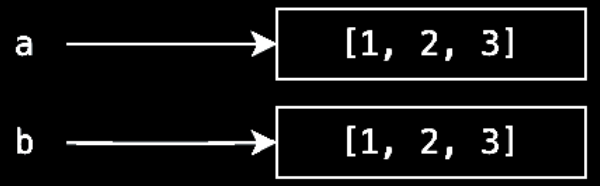
For eksempel,

```
a = [1, 2, 3]
b = a
```



vil innebære at `a` og `b` er aliaser, mens

```
a = [1, 2, 3]
b = [1, 2, 3]
```



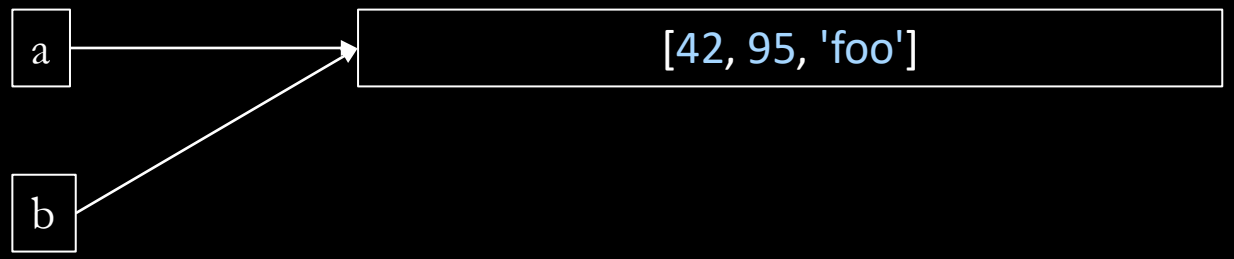
vil innebære at `a` og `b` ikke er aliaser. Selv om objektene i det sistnevnt tilfelle er *like*, er det likevel hver sine objekter med ulik plassering i minnet.

```
a = [42, 95, 'foo']
b = a
b[1] = 'bar'
```

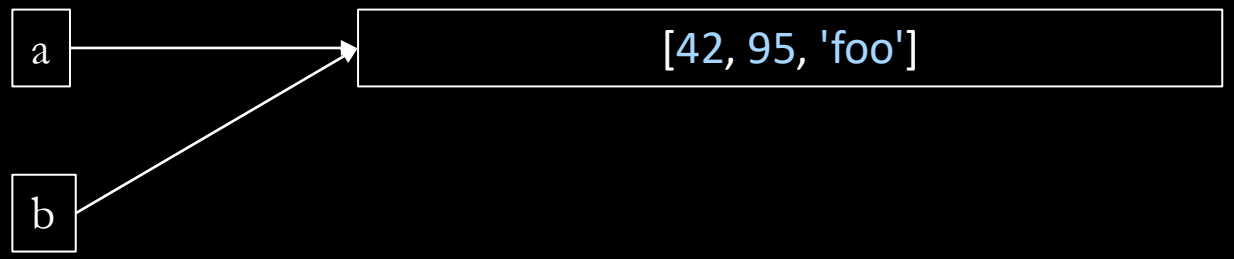
```
a = [42, 95, 'foo']
b = a
b[1] = 'bar'
```

```
a = [42, 95, 'foo']
b = a
b[1] = 'bar'
```

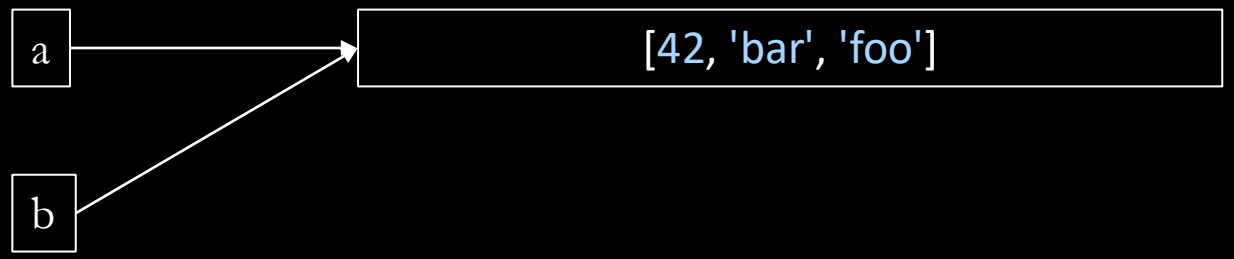
```
a = [42, 95, 'foo']
b = a
b[1] = 'bar'
```



```
a = [42, 95, 'foo']
b = a
b[1] = 'bar'
```



```
a = [42, 95, 'foo']
b = a
b[1] = 'bar'
```



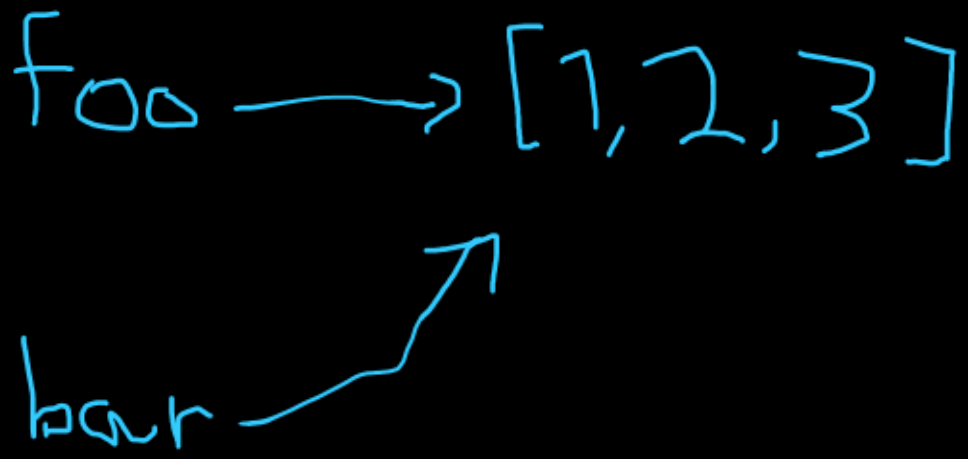
Hvis et objekt med alias blir mutert: pass på!!

HVORDAN OPPSTÅR ALIAS?

```
foo = [1, 2, 3]
bar = foo
```

HVORDAN OPPSTÅR ALIAS?

```
foo = [1, 2, 3]
bar = foo
```



HVORDAN OPPSTÅR ALIAS?

```
foo = [1, 2, 3]
bar = foo
```

```
foo = {
    "wha": [1, 2, 3],
    "chi": [4, 5, 6]
}
bar = foo["wha"]
```

HVORDAN OPPSTÅR ALIAS?

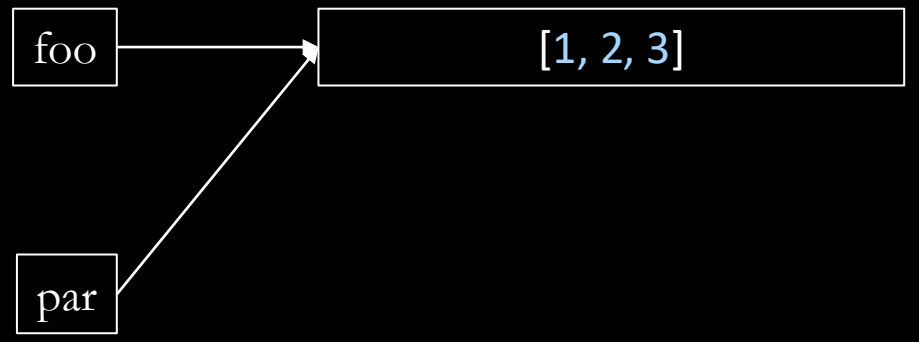
```
foo = [1, 2, 3]
bar = foo
```

```
foo = {
    "wha": [1, 2, 3],
    "chi": [4, 5, 6]
}
bar = foo["wha"]
```

```
def fun(par):
    ...

foo = [1, 2, 3]
fun(foo)
```

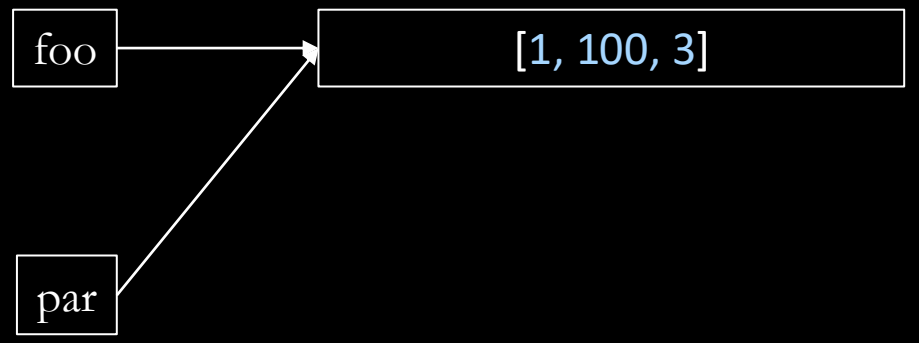
ALIAS MED FUNKSJONSPARAMETRE

```
def fun(par):
    par[1] = 100

foo = [1, 2, 3]
fun(foo)
```



ALIAS MED FUNKSJONSPARAMETRE

```
def fun(par):
    par[1] = 100

foo = [1, 2, 3]
fun(foo)
```



HVORDAN OPPSTÅR ALIAS?

```
foo = [1, 2, 3]
bar = foo
```

```
foo = {
    "wha": [1, 2, 3],
    "chi": [4, 5, 6]
}
bar = foo["wha"]
```

```
def fun(par):
    ...

foo = [1, 2, 3]
fun(foo)
```

```
foo = [[1, 2, 3], [4, 5, 6]]

for el in foo:
    ...
```

HVORDAN OPPSTÅR ALIAS?

```
foo = [1, 2, 3]
bar = foo
```

```
def fun(par):
    ...

foo = [1, 2, 3]
fun(foo)
```

```
foo = {
    "wha": [1, 2, 3],
    "chi": [4, 5, 6]
}
bar = foo["wha"]
```

```
foo = [[1, 2, 3], [4, 5, 6]]

for el in foo:
    ...
```

```
def fun(par):
    return par["chi"]

foo = {
    "wha": [1, 2, 3],
    "chi": [4, 5, 6]
}
bar = fun(foo)
```

DESTRUKTIVITET

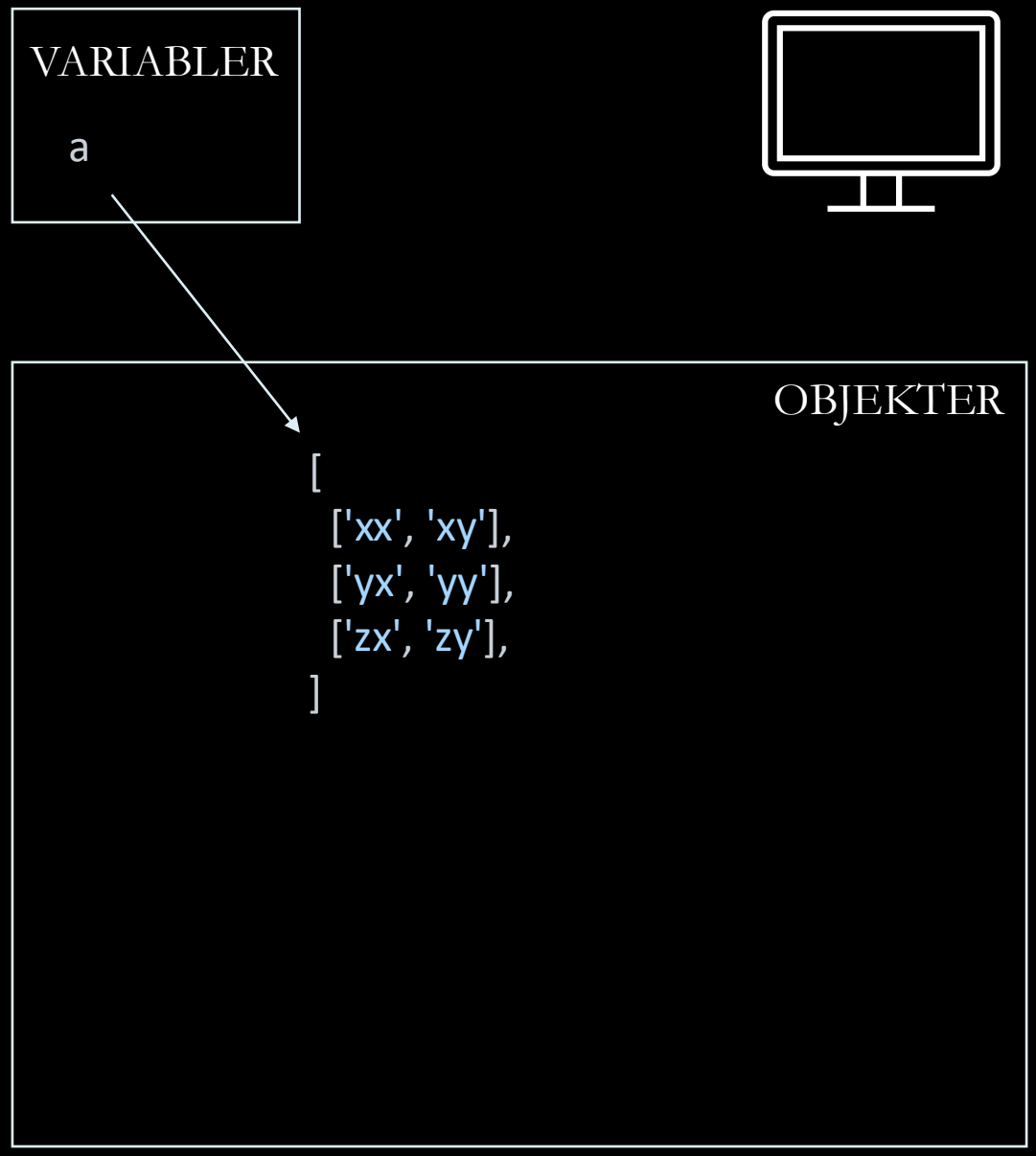
- En funksjon er *destruktiv* dersom den muterer en av sine argumenter

  Husk: Et **argument** er et objekt som pekes på av en parameter når funksjonen begynner

EKSEMPLER

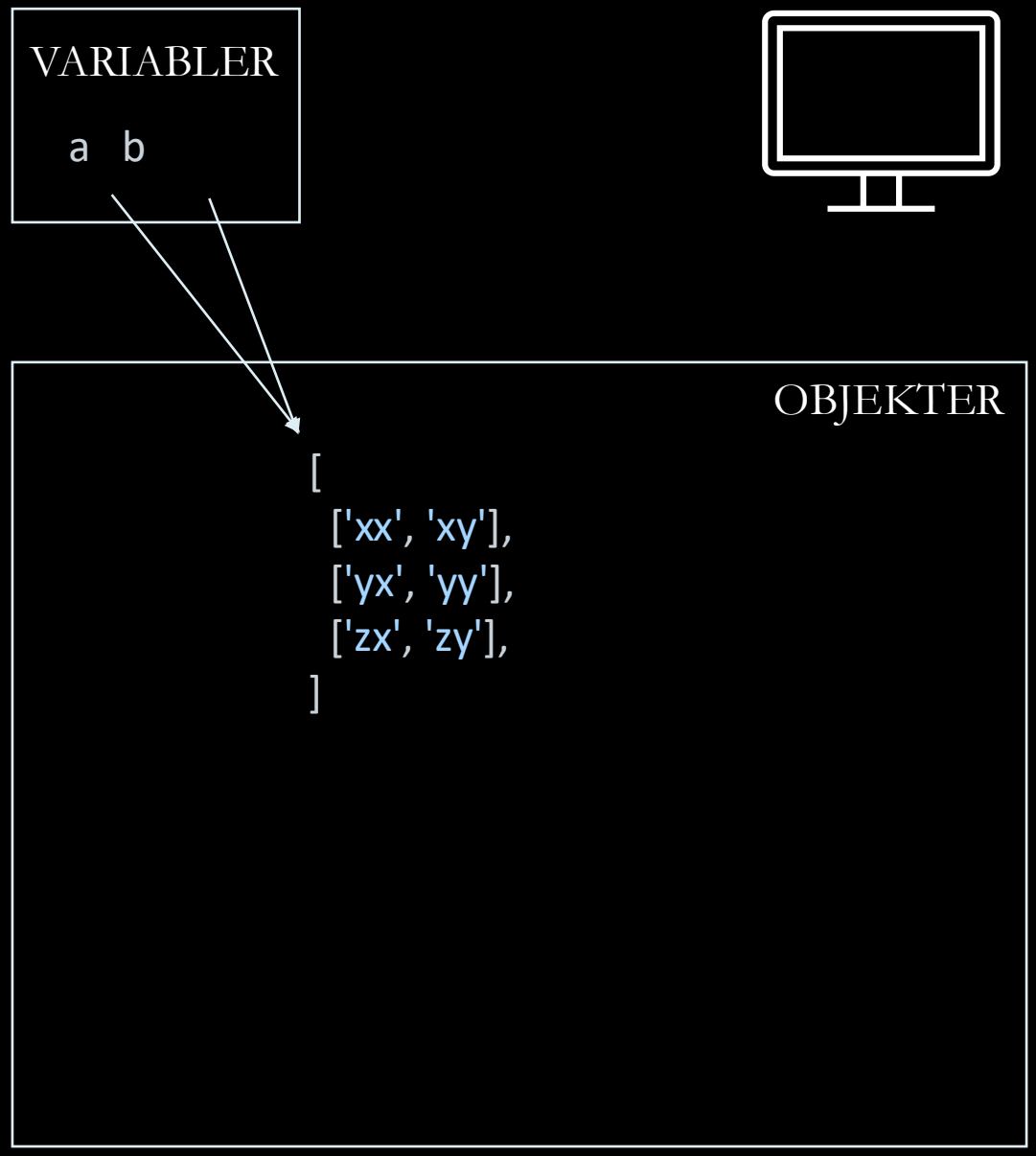
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a
b[1][0] = 'foo'
print(a[1][0])
```



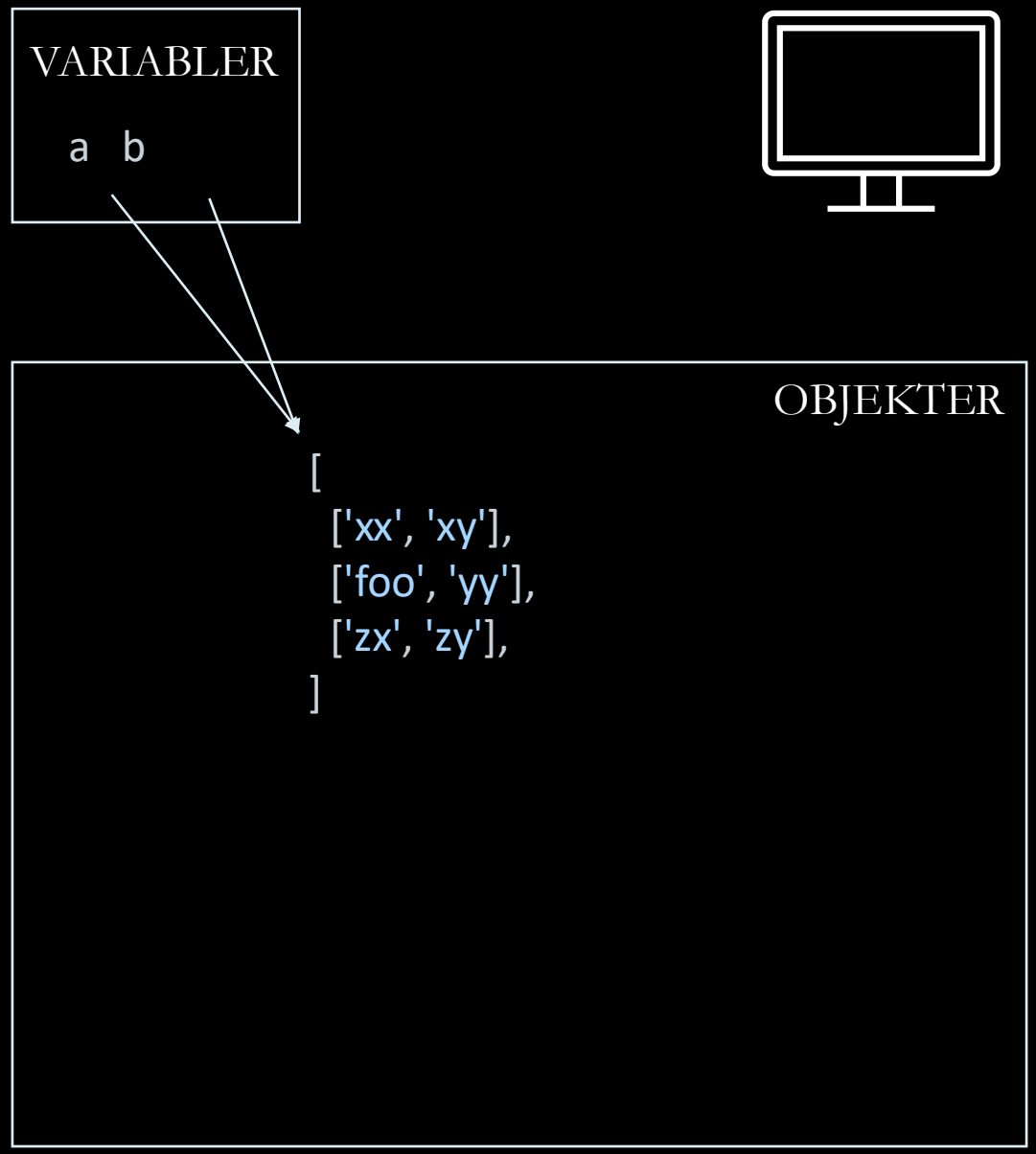
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a
b[1][0] = 'foo'
print(a[1][0])
```



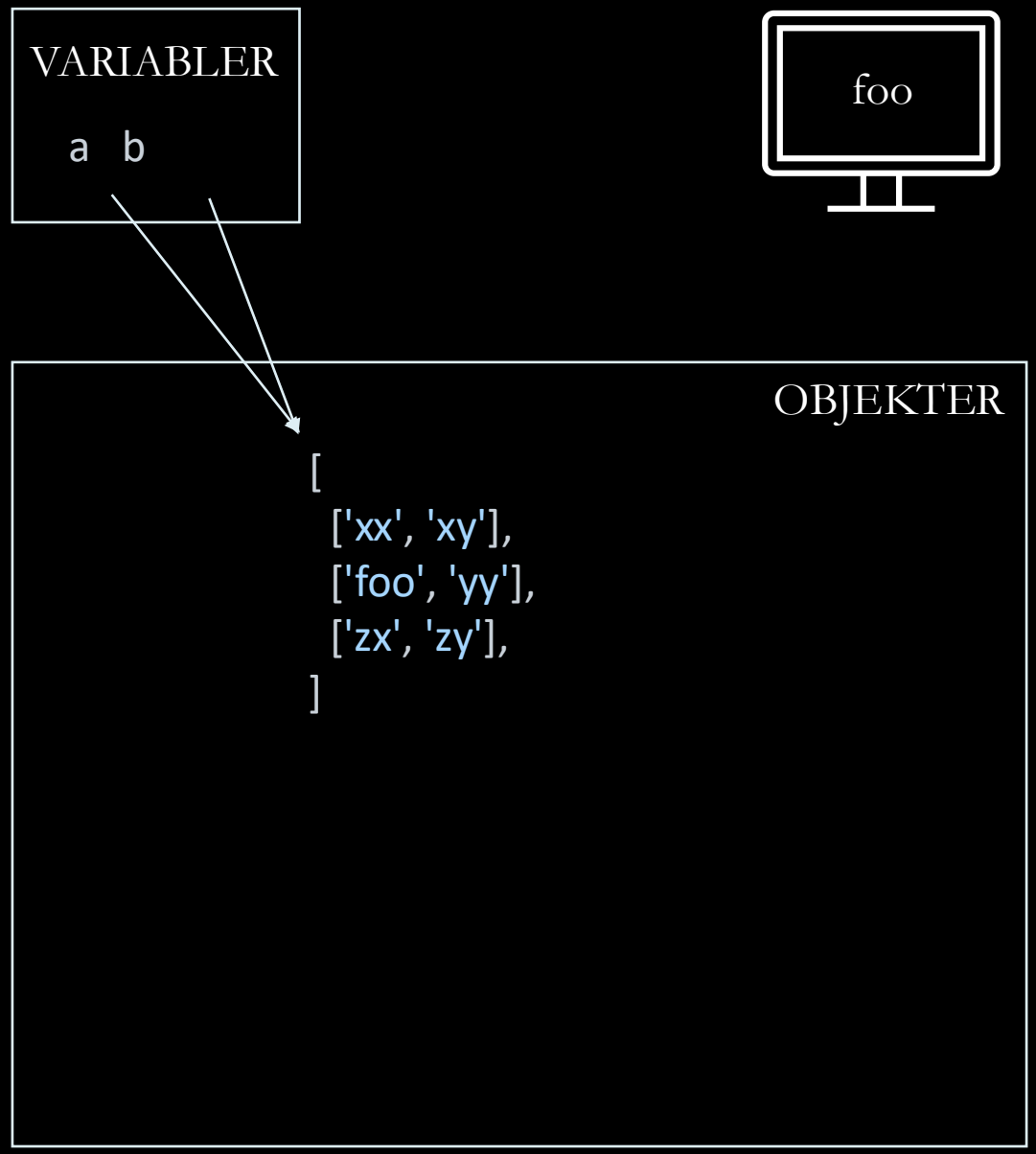
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a
b[1][0] = 'foo'
print(a[1][0])
```



2D-LISTE

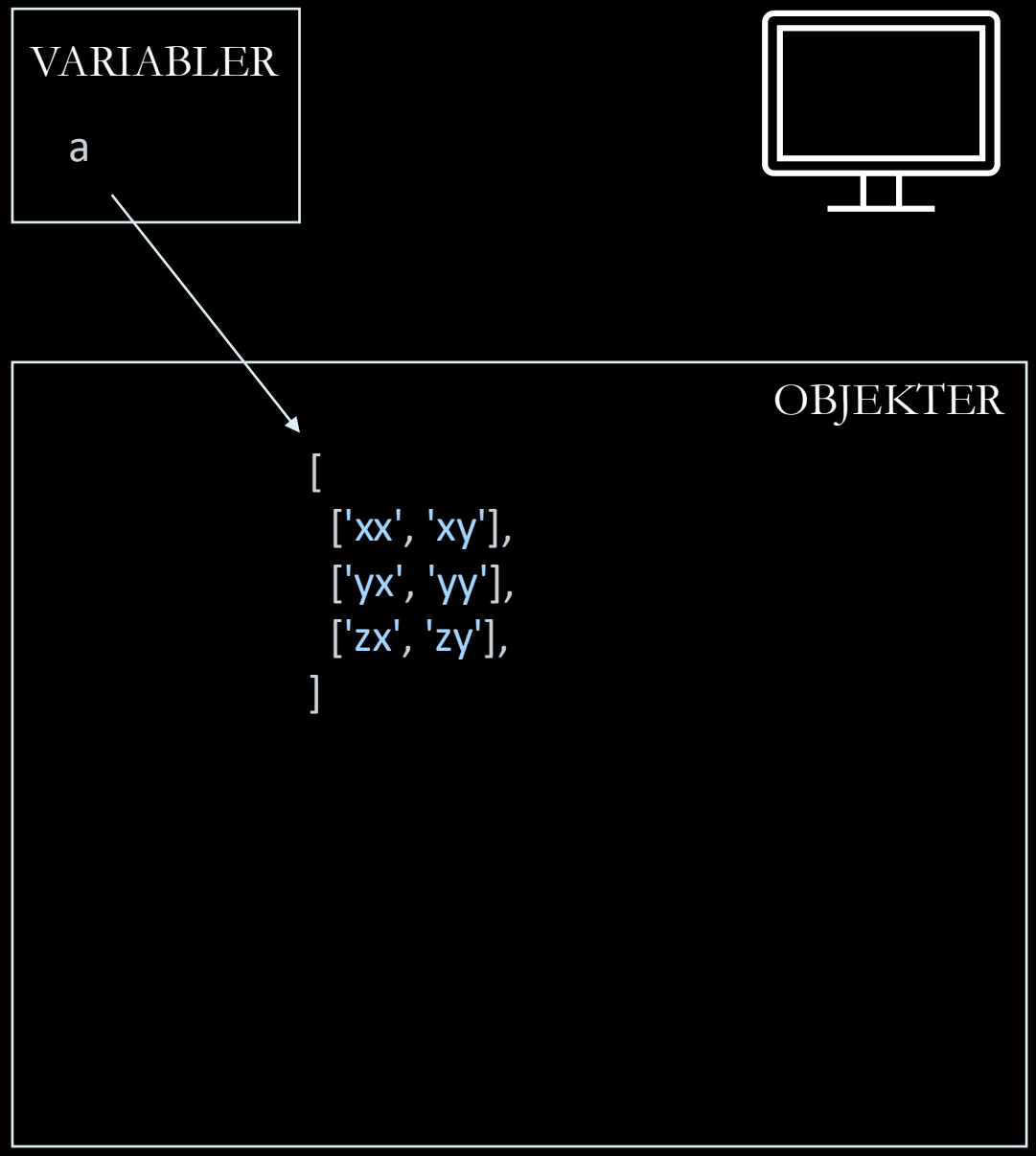
```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a
b[1][0] = 'foo'
print(a[1][0])
```



NYTT EKSEMPEL

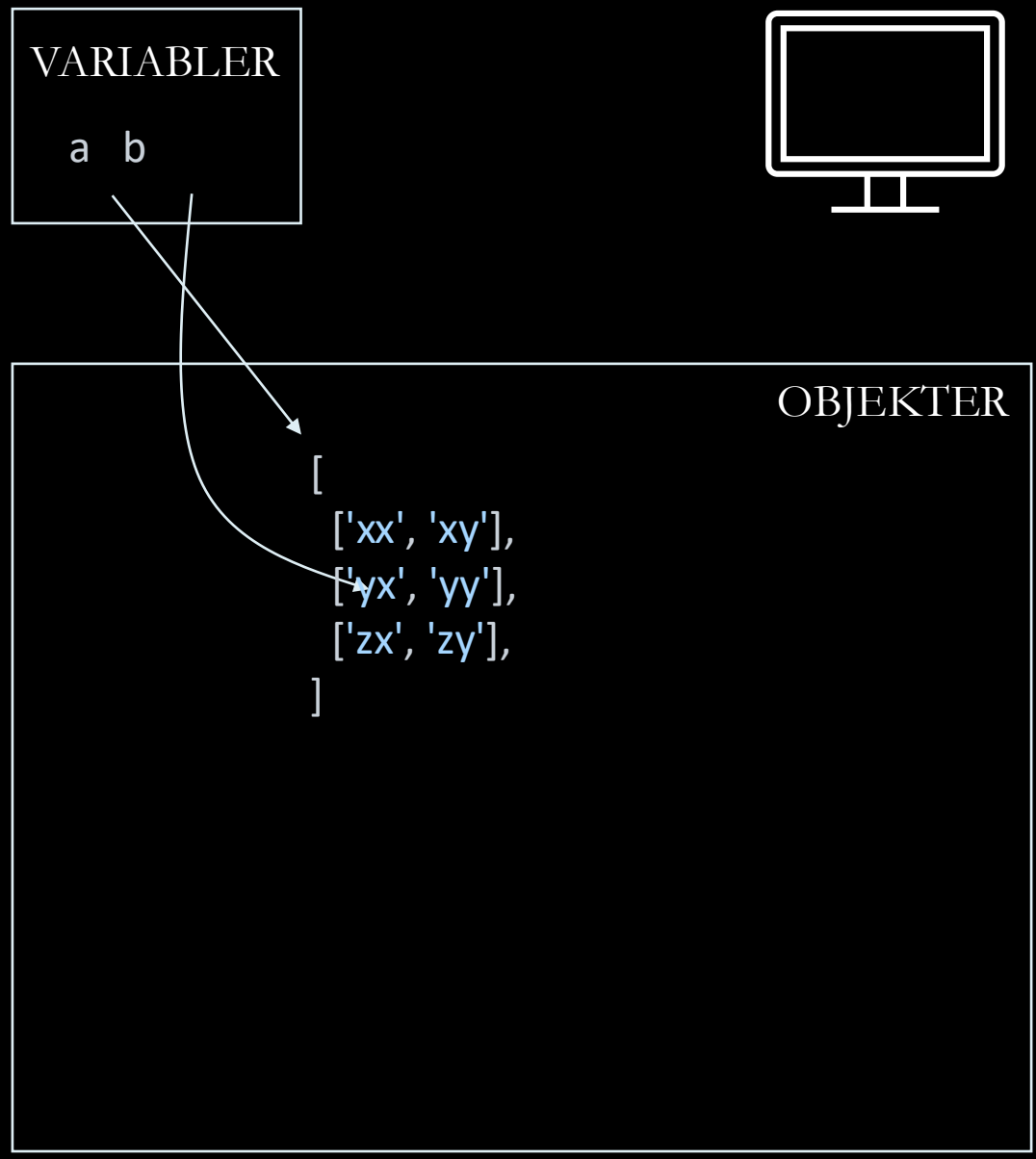
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
b[0] = 'foo'
print(a[1][0])
```



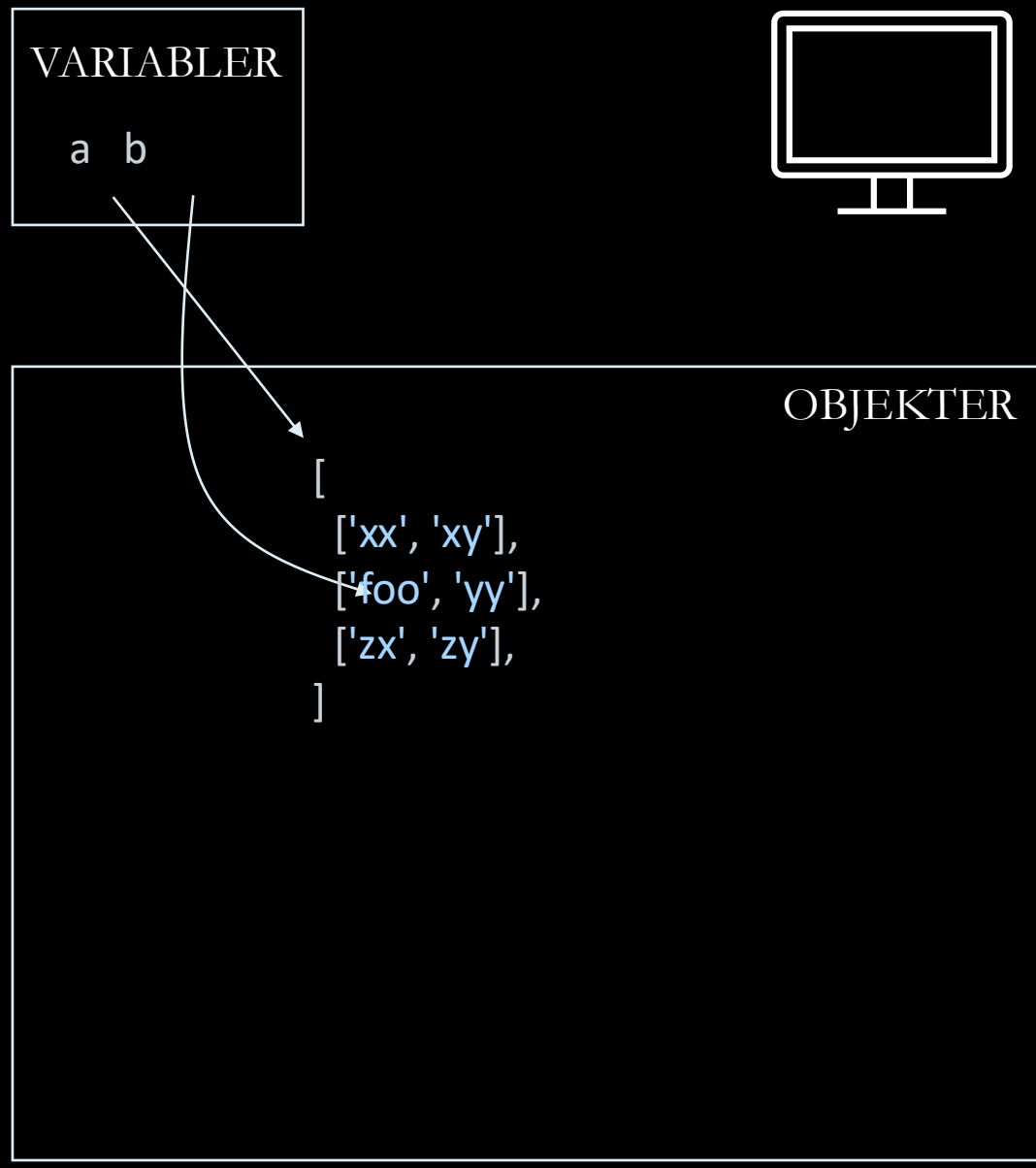
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
b[0] = 'foo'
print(a[1][0])
```



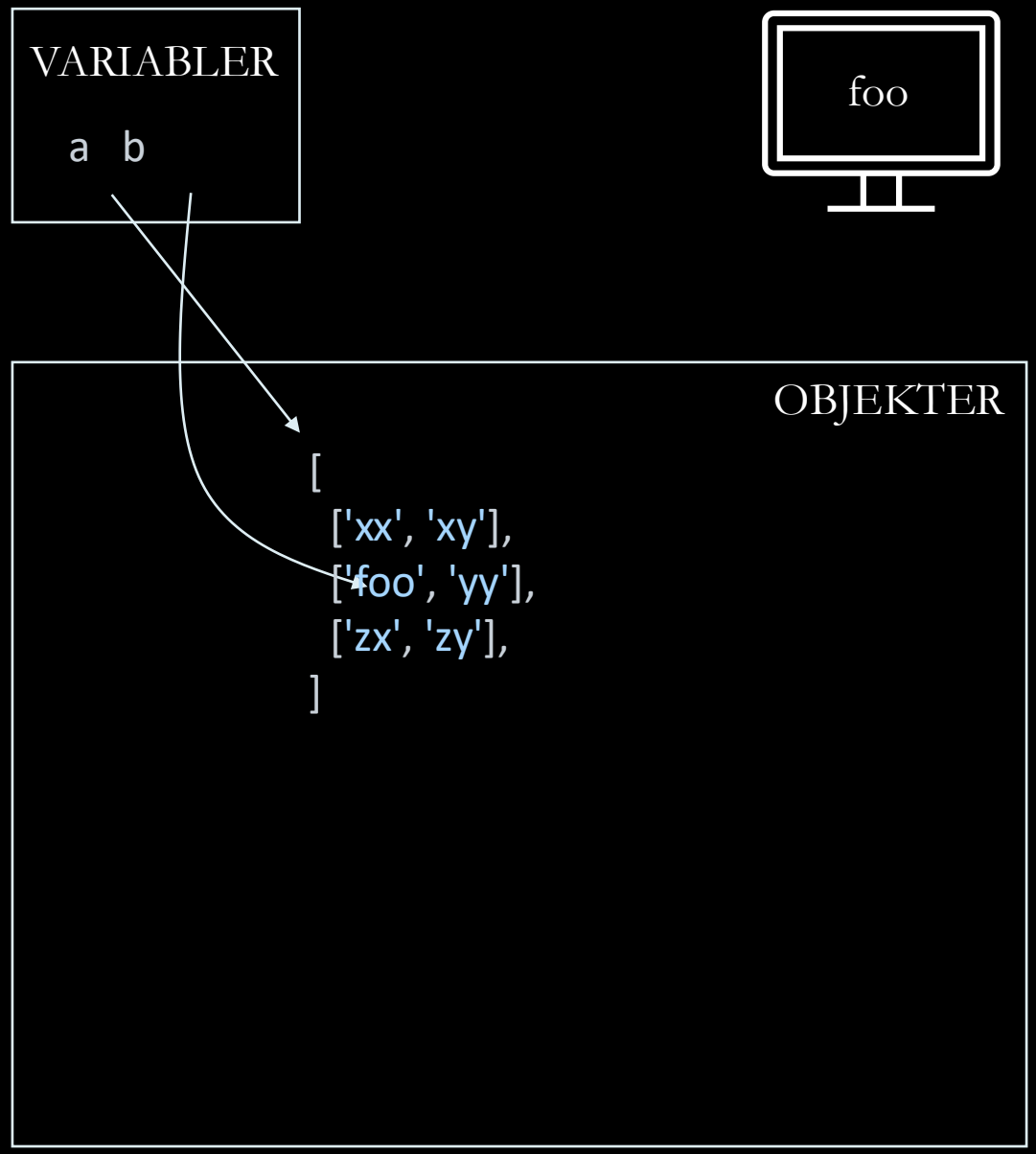
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
b[0] = 'foo'
print(a[1][0])
```



2D-LISTE

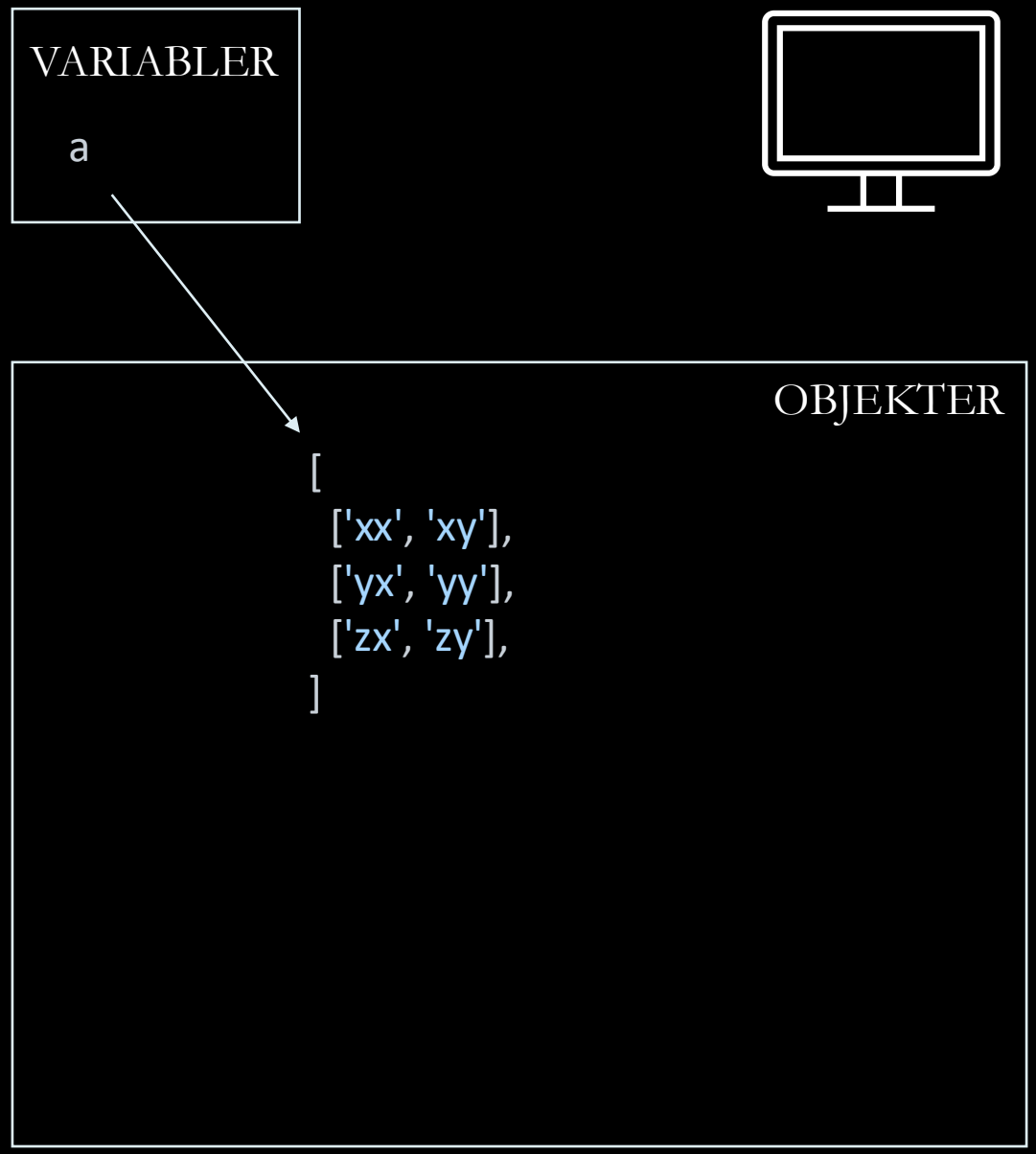
```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
b[0] = 'foo'
print(a[1][0])
```



NYTT EKSEMPEL

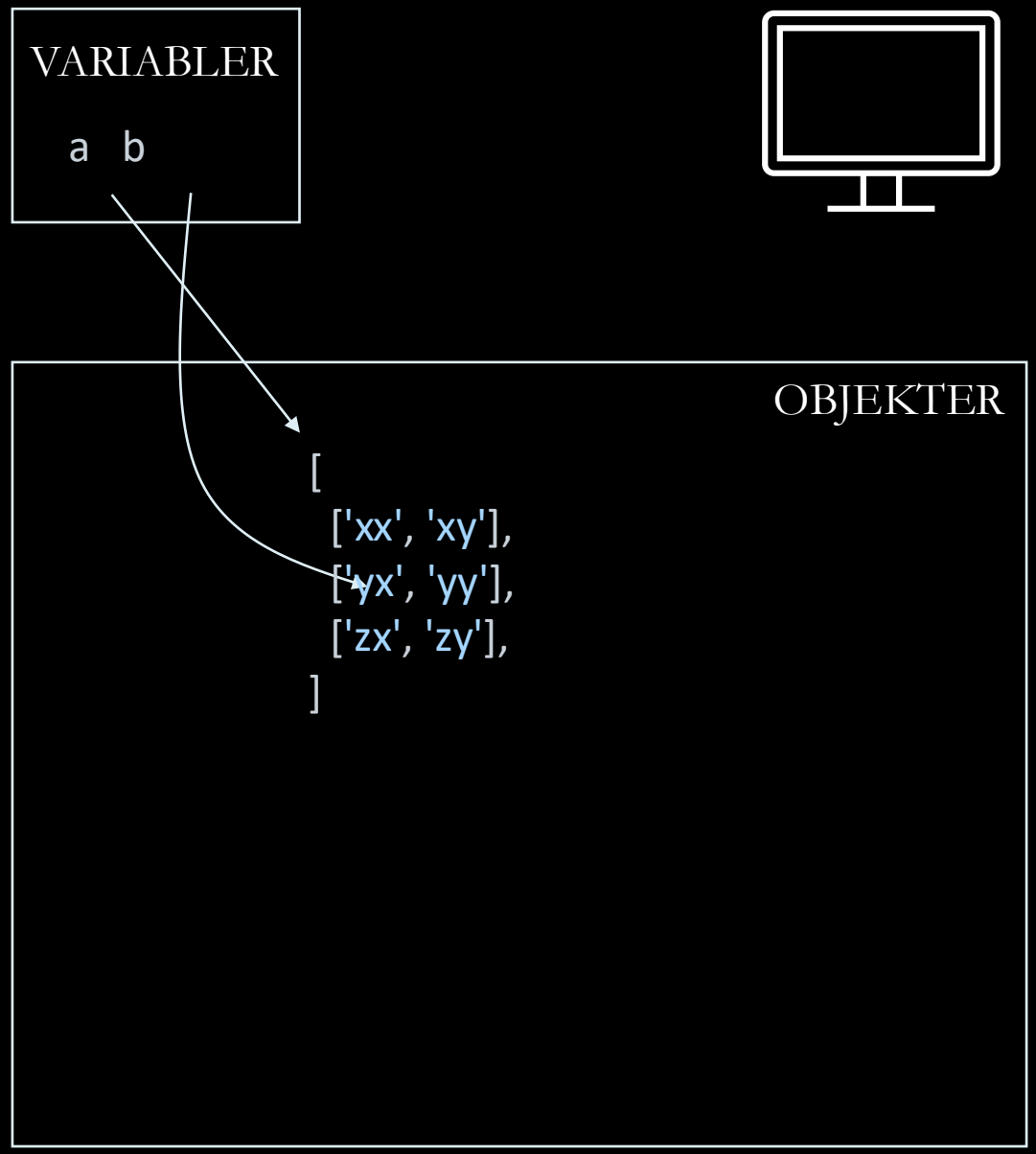
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



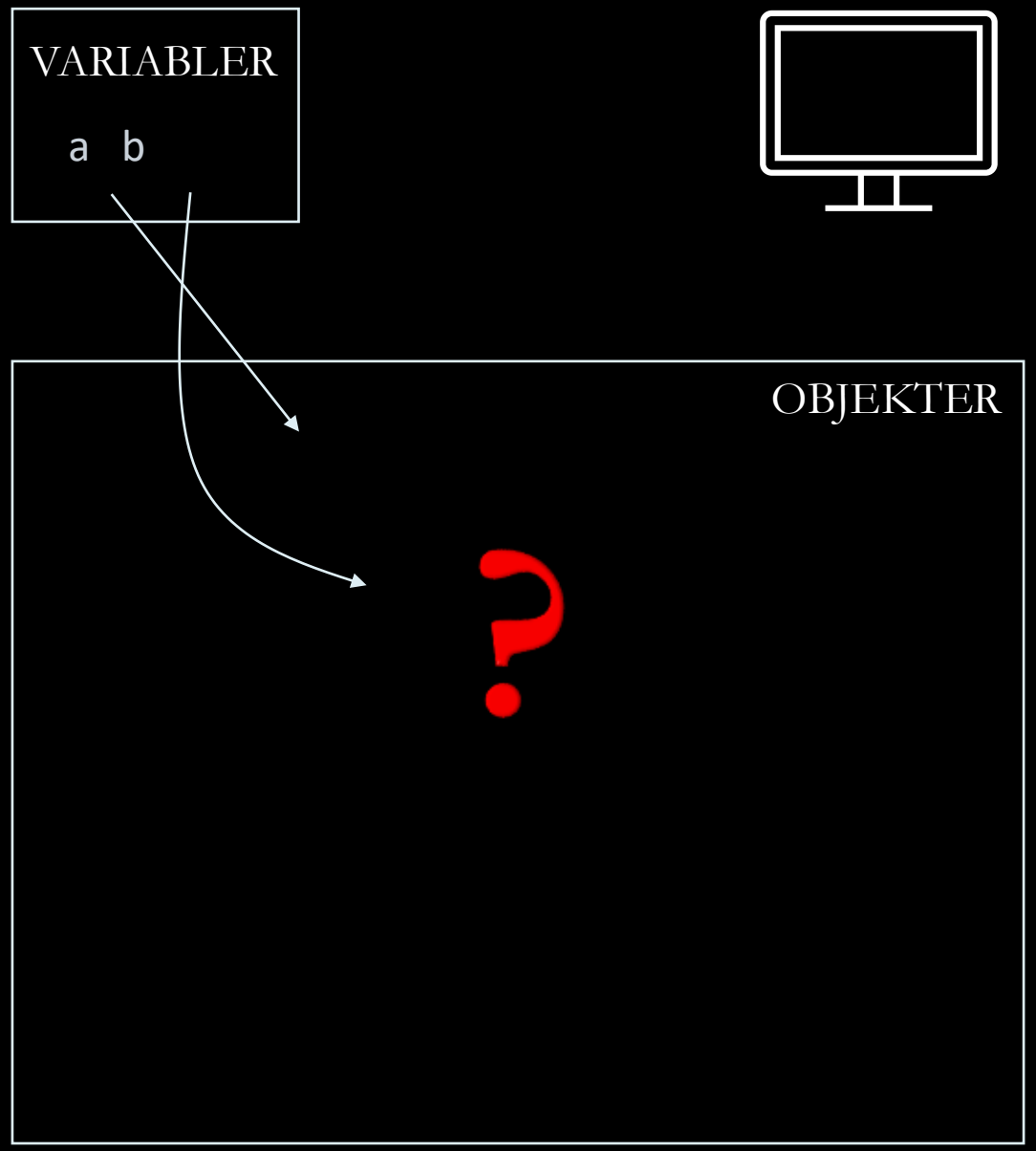
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



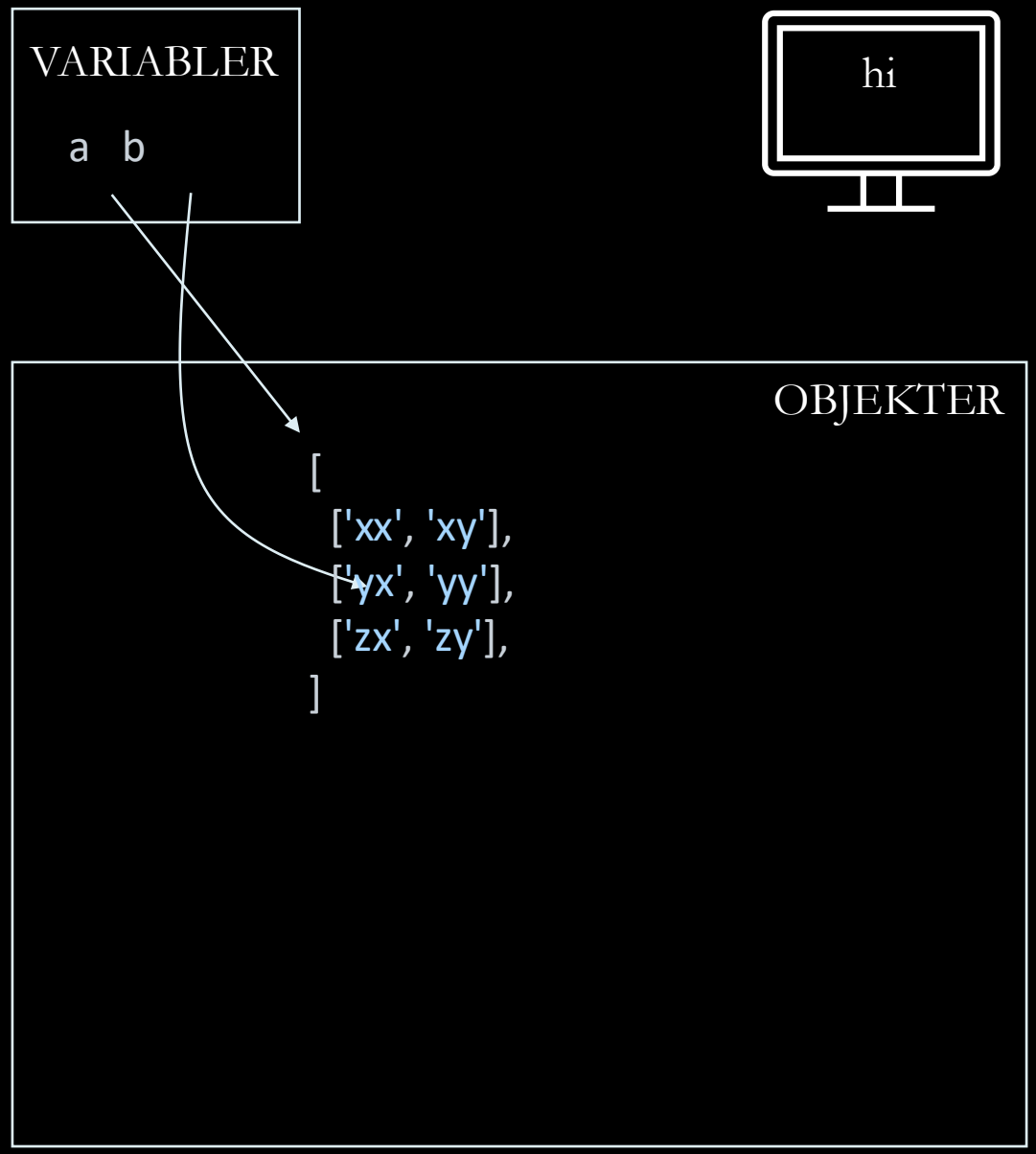
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



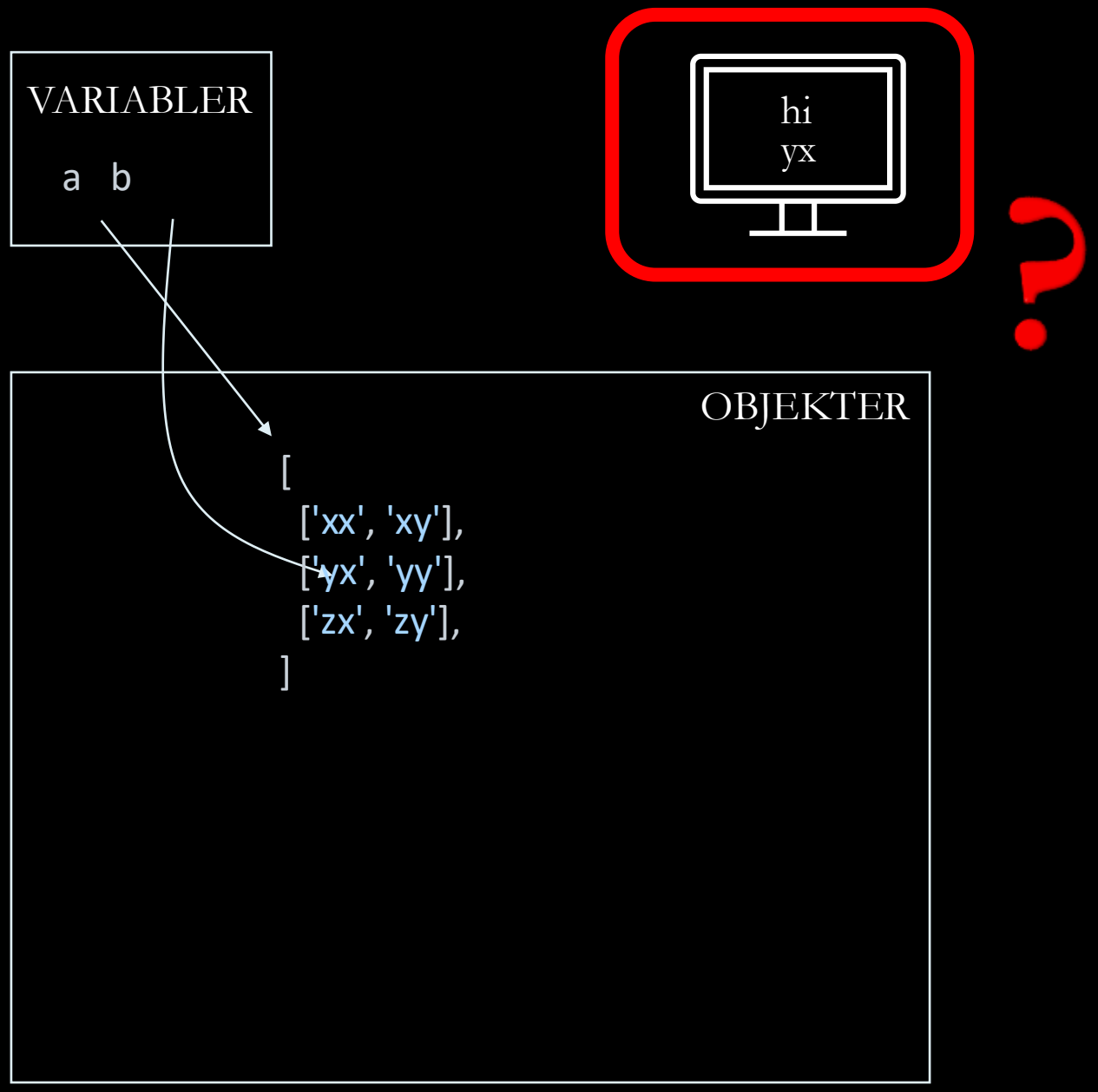
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



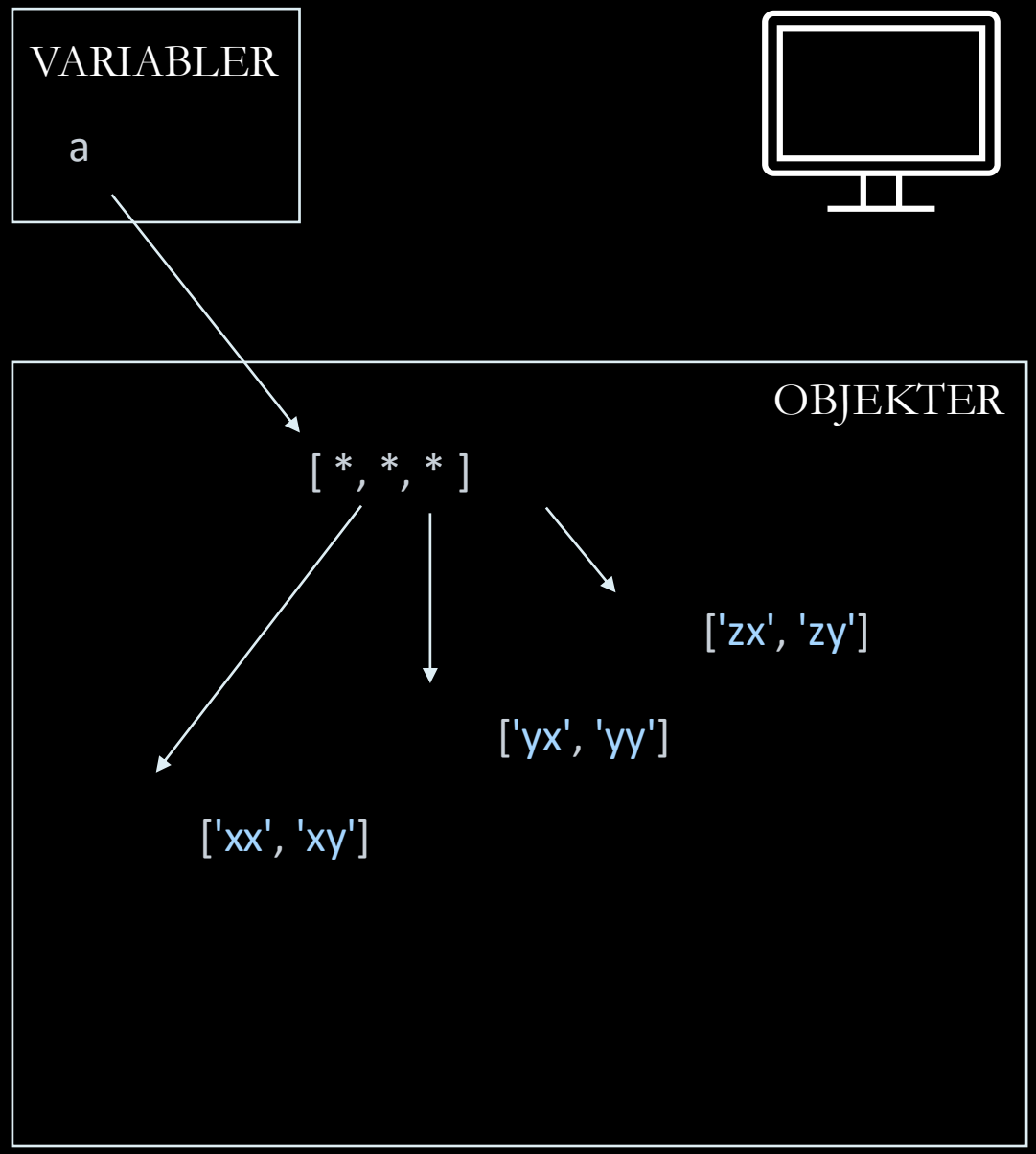
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



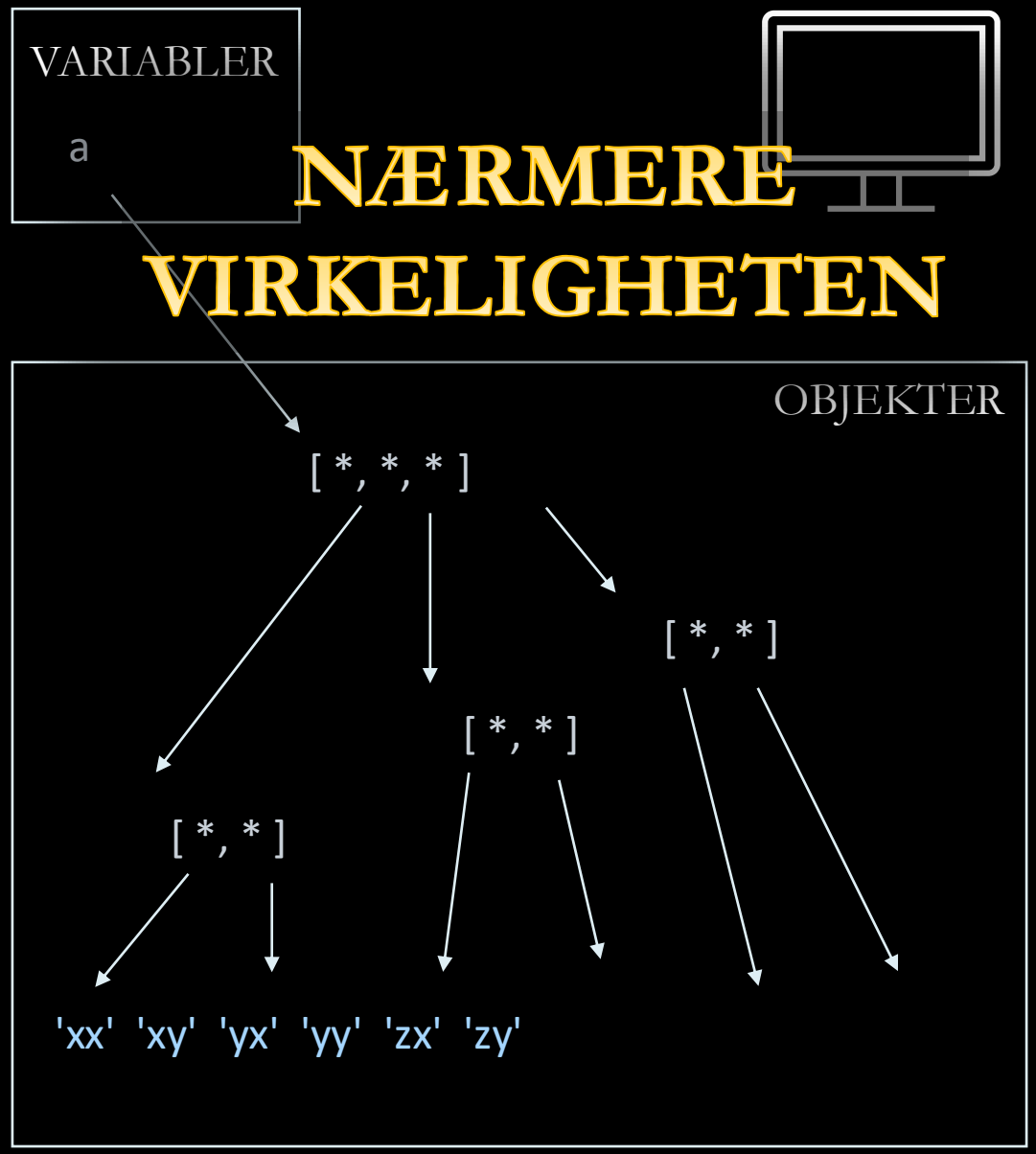
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]
```

ENDA NÆRMERE VIRKELIGHETEN

VARIABLER

Variabelnavn	Adresse
a	10240

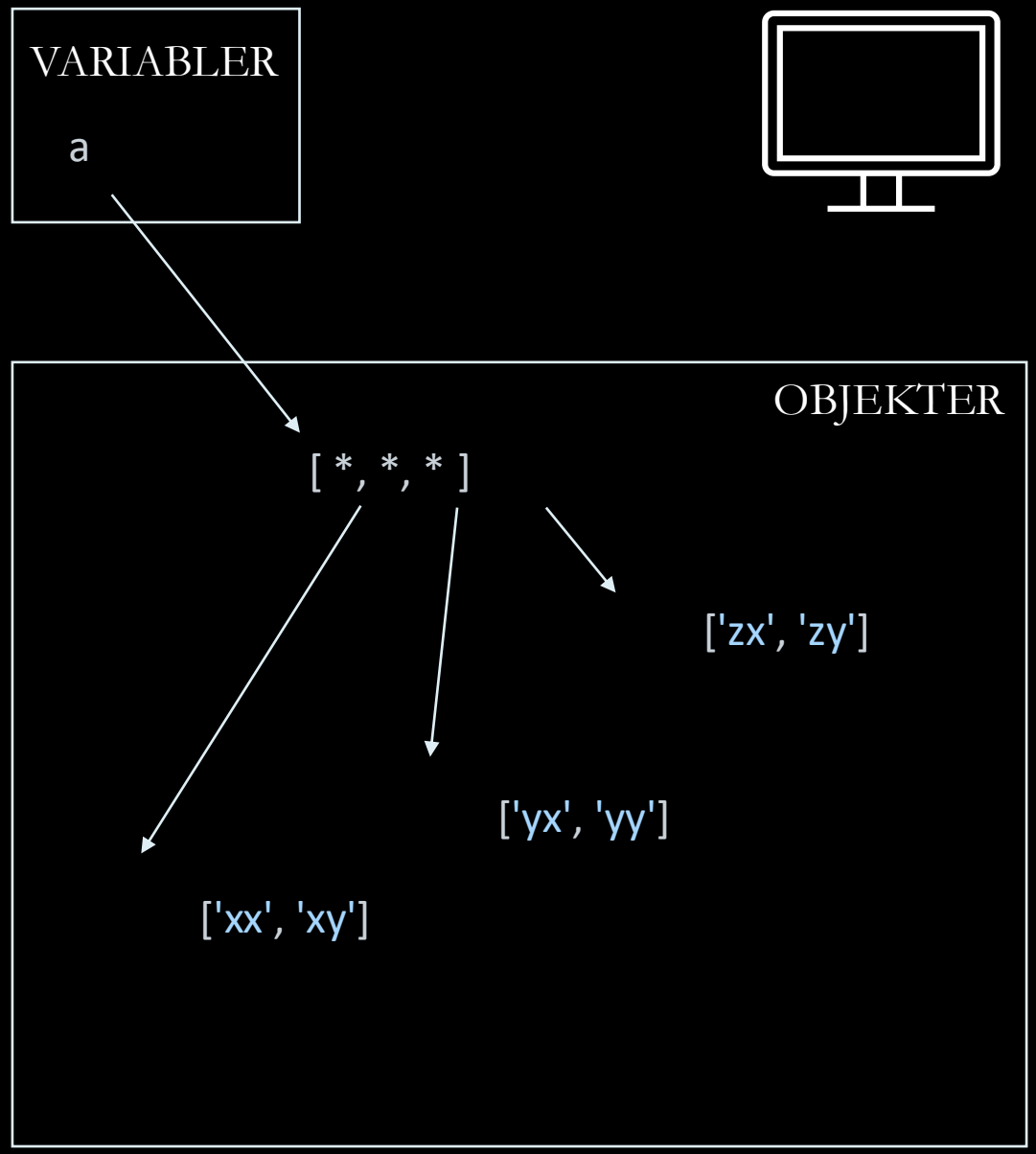
OBJEKTER

Adresse	Klasse	Verdi
1024	str	'xx'
2048	str	'xy'
3072	str	'yx'
4096	str	'yy'
5120	str	'zx'
6144	str	'zy'
7168	list	[1024, 2048]
8192	list	[3072, 4096]
9216	list	[5120, 6144]
10240	list	[7168, 8192, 9216]

NYTT FORSØK

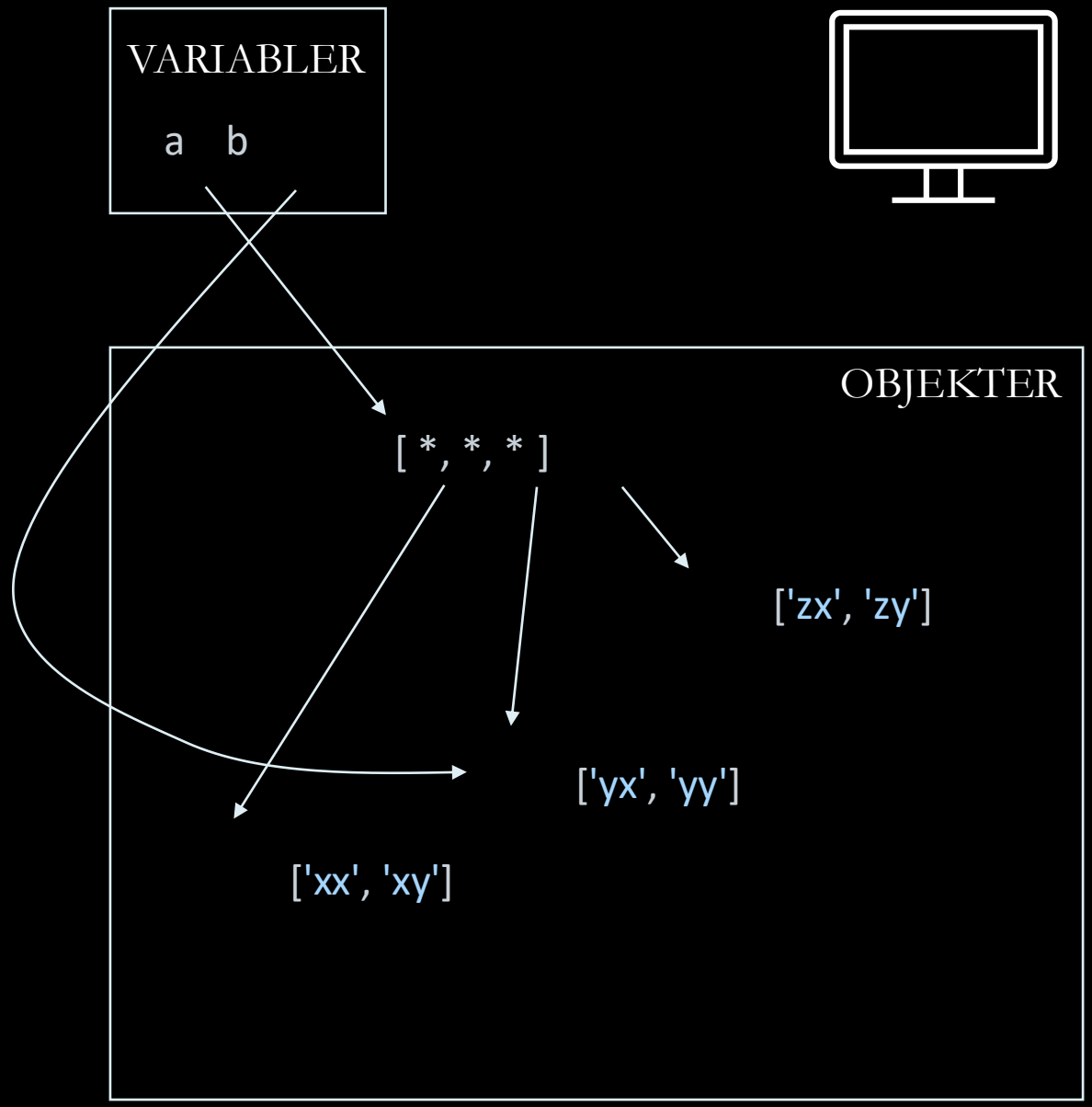
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



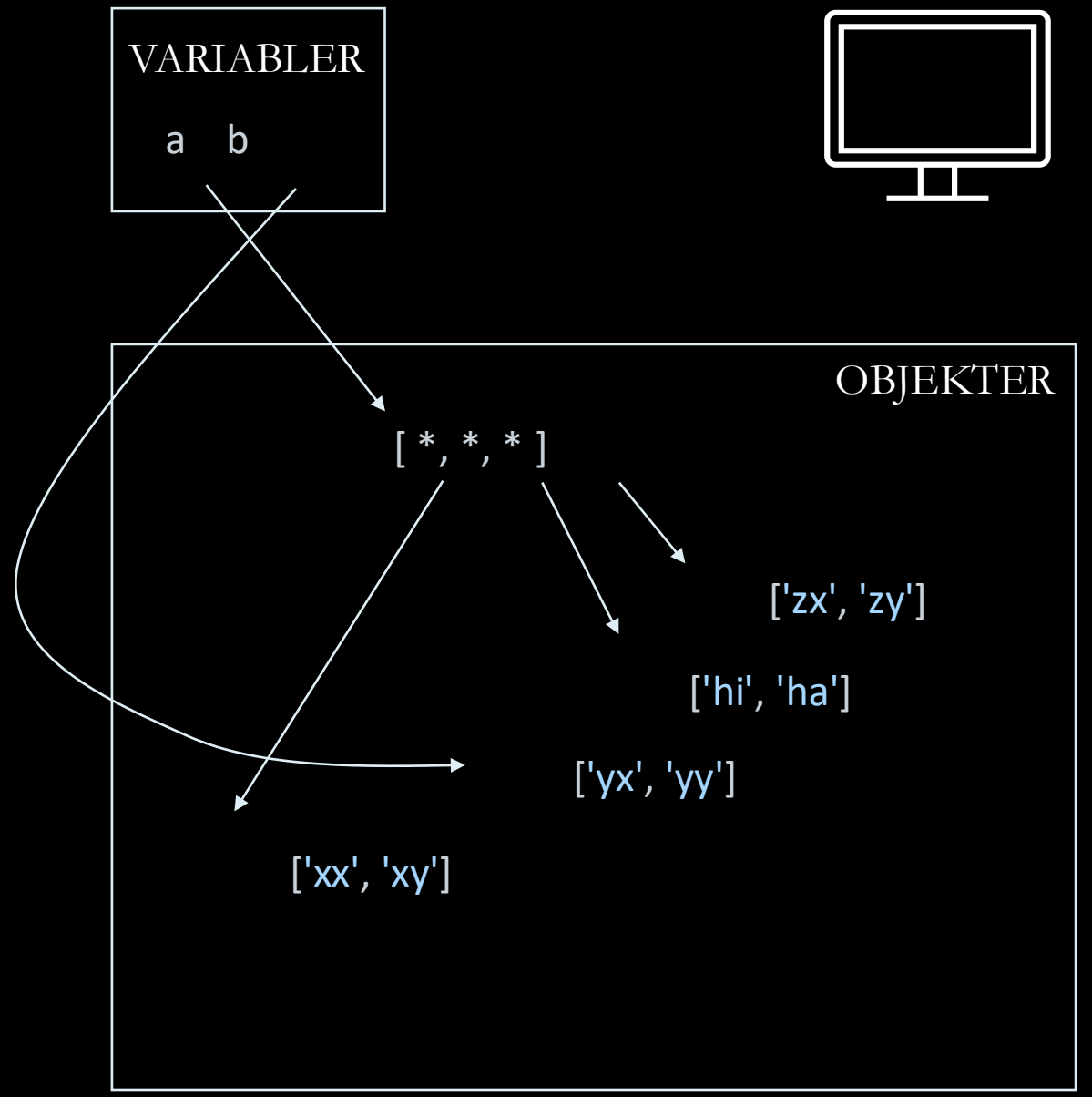
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



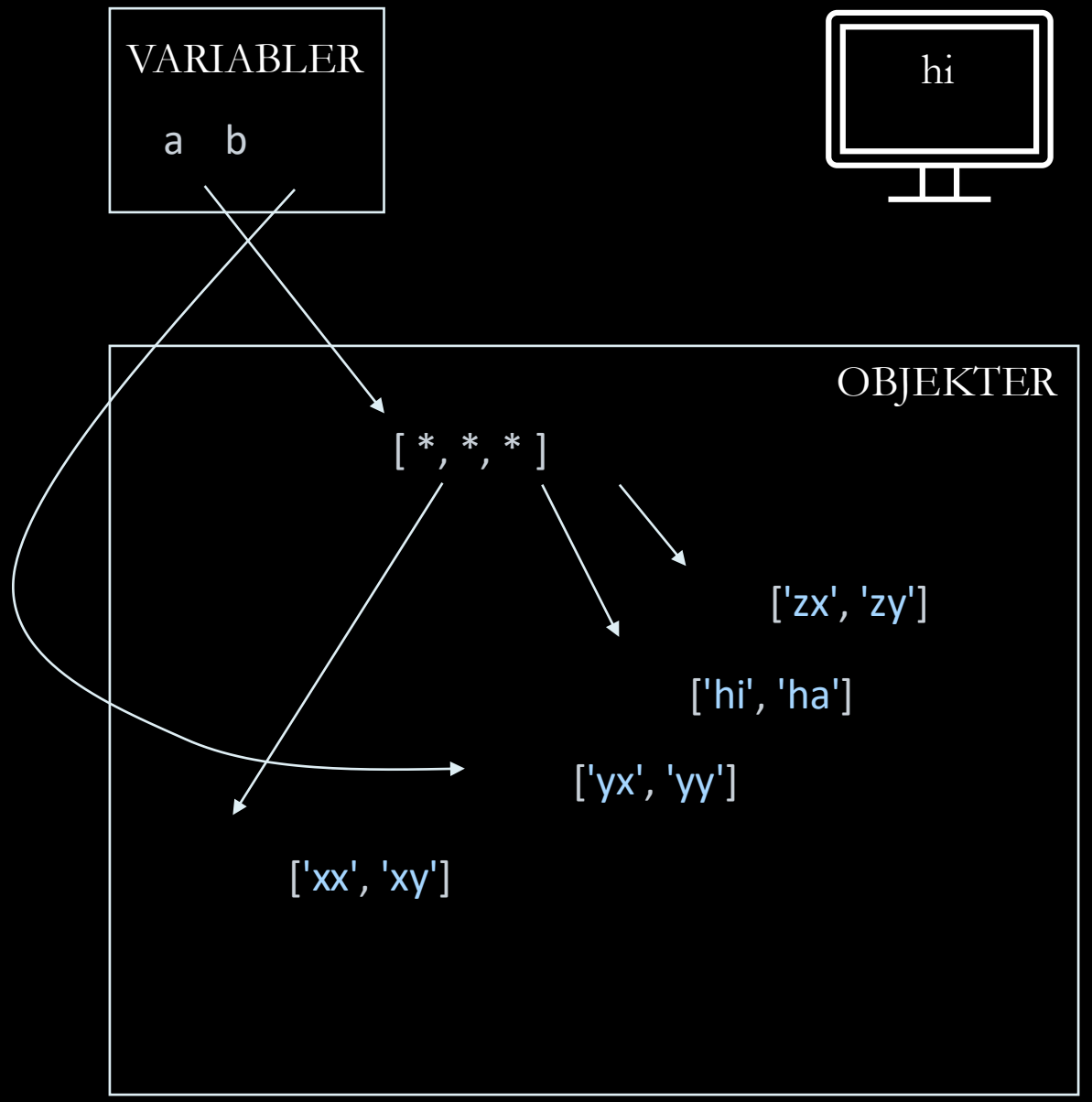
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



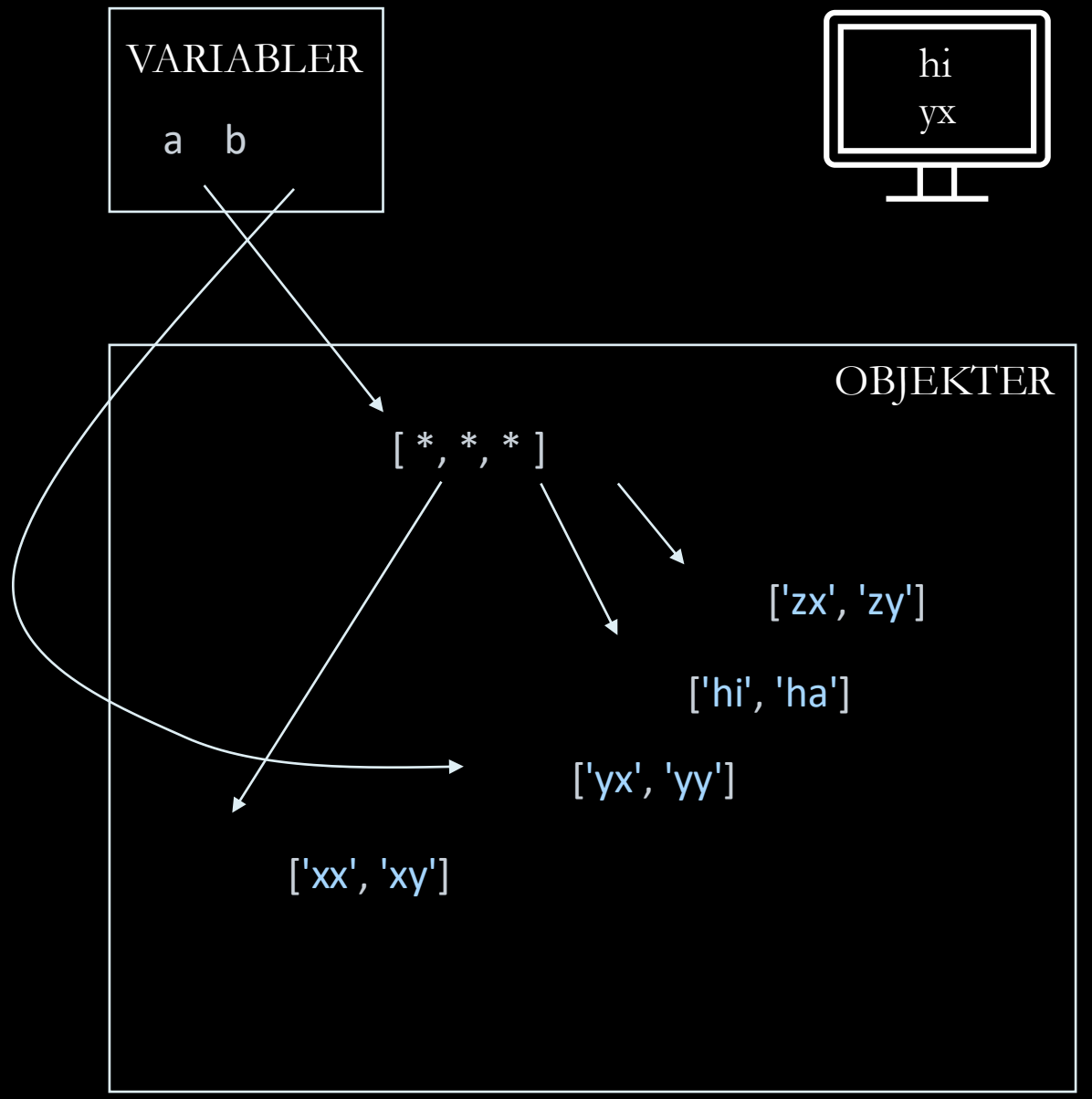
2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



2D-LISTE

```
a = [
    ['xx', 'xy'],
    ['yx', 'yy'],
    ['zx', 'zy'],
]

b = a[1]
a[1] = ['hi', 'ha']
print(a[1][0])
print(b[0])
```



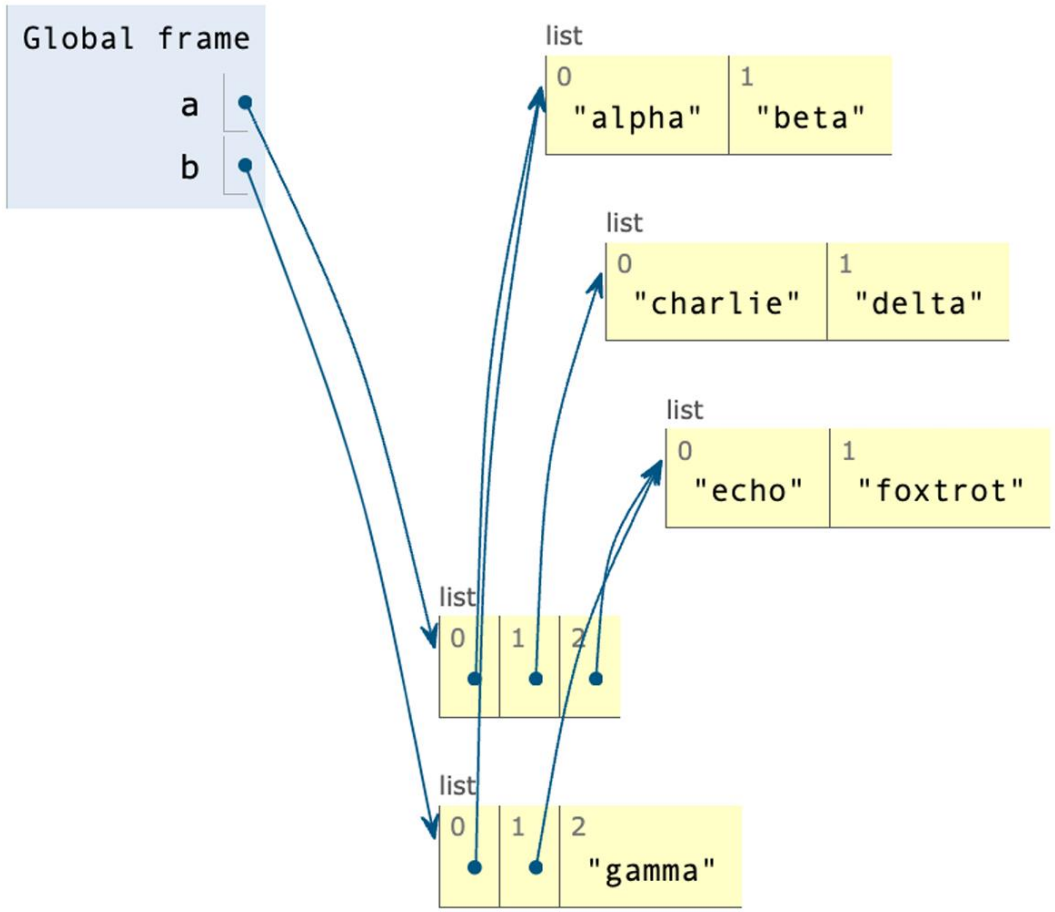
PYTHON TUTOR

- Gratis, reklamefinansiert visualiseringsverktøy

https://pythontutor.com/

EKSAMENSOPPGAVE HØST 2023

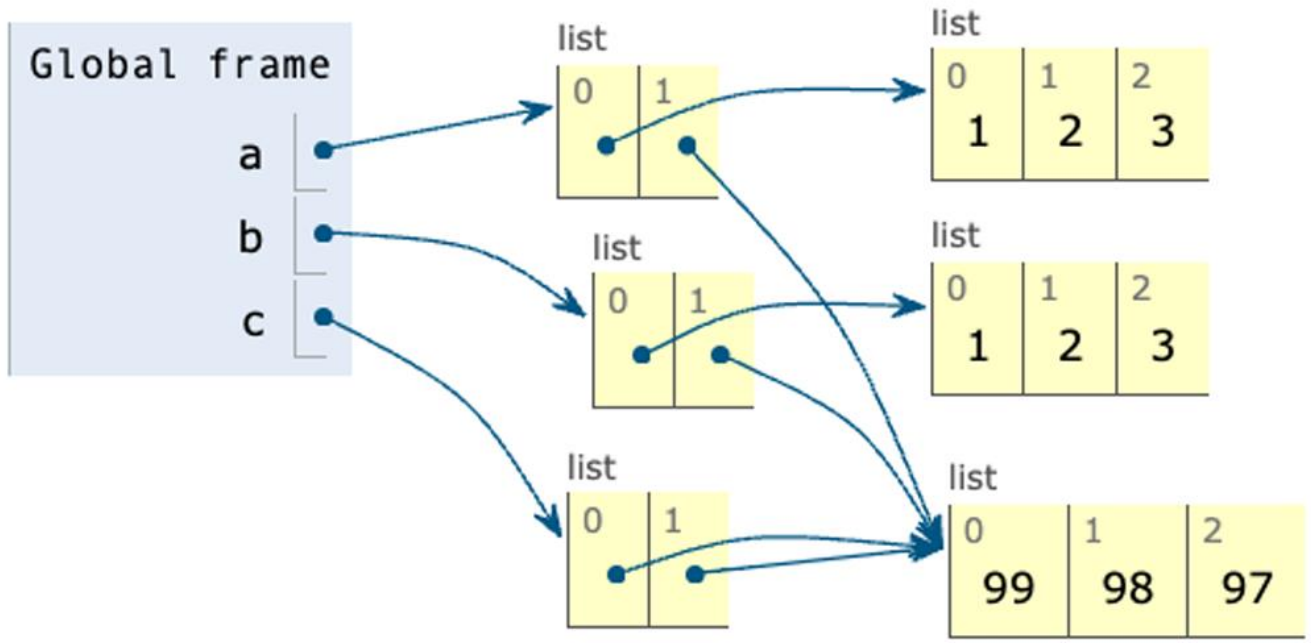
1(l)



Gitt at minnet har tilstanden vist over, hva blir skrevet ut etter setningen **print(a[1][1] + b[1][1])**?
(hvis programmet krasjer, skriv kun 'Error')

EKSAMENSOPPGAVE HØST 2023

3(a)



Skriv en kodesnutt slik at variabler og minnets tilstand for variablene a, b og c blir som vist over.

2D-LISTER

- Opprette en 2D-liste med 100 rader og 10 kolonner med kun 0'ere
- Sett elementet på posisjon [1][2] til 9999

- Bruk en nøstet løkke for å telle sammen summen av tallene i 2D-listen

2D-LISTER

- Opprette en 2D-liste med 100 rader og 10 kolonner med kun 0'ere

Buggy!

```
one_row = [0] * 10
table = [one_row] * 100
```

Riktig!

```
table = []
for _ in range(100):
    one_row = [0] * 10
    table.append(one_row)
```

2D-LISTER

- Opprette en 2D-liste med 100 rader og 10 kolonner med kun 0'ere

Buggy!

```
table = [[0] * 10] * 100
```

Riktig!

```
table = [[0] * 10 for _ in range(100)]
```