

Inleiding Programmeren

Toon Calders

Les 3: Mutable, immutable
Dict, set, tuple

Oefening: Zeef van Eratosthenes

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

Oefening: Zeef van Eratosthenes

- Wat is input? Wat is output?
- Welke *data-structuur* gaan we gebruiken?
 - Heel belangrijk! Een goede data-structuur bespaart je veel programmeerwerk!
- Welke manipulaties op de data structuur zijn nodig?
 - Algoritme
- Nu pas: implementatie
 - Test tussentijds!

Oefening: Zeef van Eratosthenes

```
# Bereken priemgetallen mbv de Zeef van Erat
```

```
laatste=20 # grootte van de zeef
```

```
Zeef = ['Priem']*laatste
```

```
Zeef[0]='Geen priem' # 0 en 1 zijn geen priemgetallen
```

```
Zeef[1]='Geen priem'
```

```
for i in range(2,laatste): # voor elk getal
```

```
    if Zeef[i]=='Priem': # als het priem is, schrap veelvoud
```

```
        veelvoud=i*i # eerste veelvoud te beschouwen is i*i
```

```
        while veelvoud < laatste:
```

```
            Zeef[veelvoud]='Geen priem'
```

```
            veelvoud+=i
```

```
# print de output van het programma
```

```
for i in range(laatste):
```

```
    print("Het getal ",i," is ",Zeef[i])
```

Maak een lijst door de lijst
['Priem'] zoveel te
herhalen als de waarde in
laatste

Lijst van lijsten

- We kunnen allerlei zaken in lijsten bewaren
 - Mix van types
 - Opnieuw een lijst
- Manier om een matrix te bewaren:
 - Matrix = lijst van rijen
 - Rij in een matrix = lijst van getallen

Matrices

```
# Matrix als lijst van lijsten

I=[]          # wordt de identiteitsmatrix
n=5          # dimensie van de matrix

for rij in range(n):
    nieuwe_rij=[]
    for kolom in range(n):
        if rij==kolom:                # positie op de diagonaal
            nieuwe_rij.append(1)
        else:
            nieuwe_rij.append(0)
    I.append(nieuwe_rij)              # Voeg de nieuwe rij toe
```

Matrices

```
>>> print(I)
[[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0], [0, 0, 0, 0, 1]]
>>> print(I[0])
[1, 0, 0, 0, 0]
>>> print(I[3][3])
1
```

Oefening: Transponeer matrix

- Schrijf een functie die een matrix als input heeft en de getransponeerde matrix teruggeeft

Pas op: Lijsten en referenties

- Er is echter 1 groot verschil tussen lijst en “gewone” variabelen:

```
>>> a=1
>>> b=a
>>> b=2
>>> print(a)
1
```

```
>>> a=[1]
>>> b=a
>>> b[0]=2
>>> print(a)
[2]
```

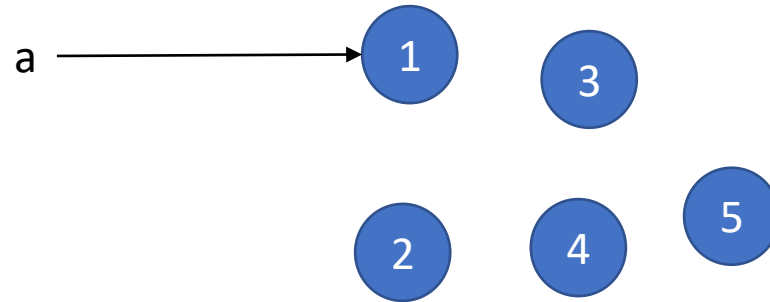
- Links: b wordt *overschreven* (b=...)
- Rechts: het *object waar b naar verwijst*, wordt *aangepast* (b[0]=...)

Python - objecten

- In Python is alles een object
 - Sommige objecten zijn *mutable*
d.i.: kunnen aangepast worden
 - Lijsten (later ook dictionaries, sets)
 - Andere objecten zijn *immutable*
d.i.: kunnen **niet** aangepast worden
 - Integers, floats, booleans, strings
- Een variabele wijst naar een object
 - `b=3` `# b wijst naar object '3'`
 - `b=a` `# b wijst naar hetzelfde object waar a naar wijst`

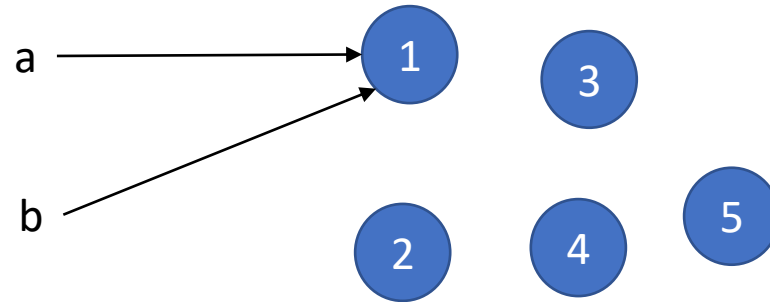
Python - objecten

```
>>> a=1
```



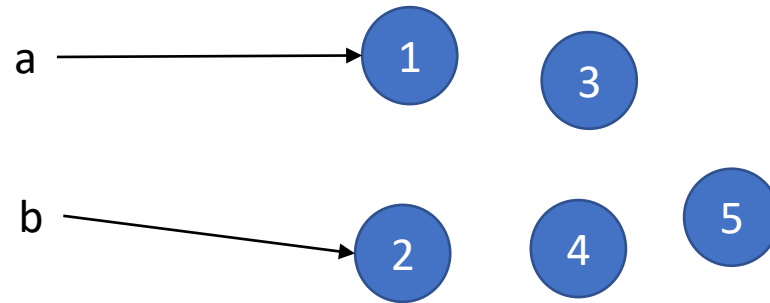
Python - objecten

```
>>> a=1  
>>> b=a
```



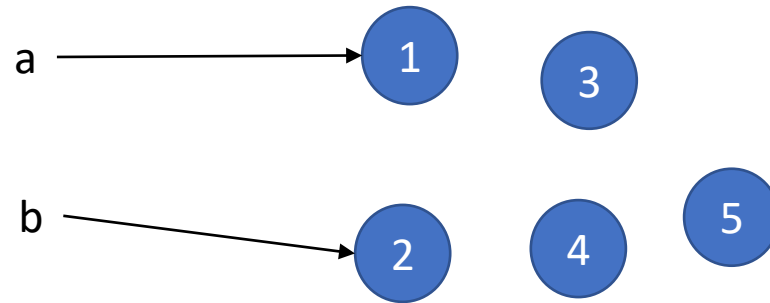
Python - objecten

```
>>> a=1  
>>> b=a  
>>> b=2  
.
```



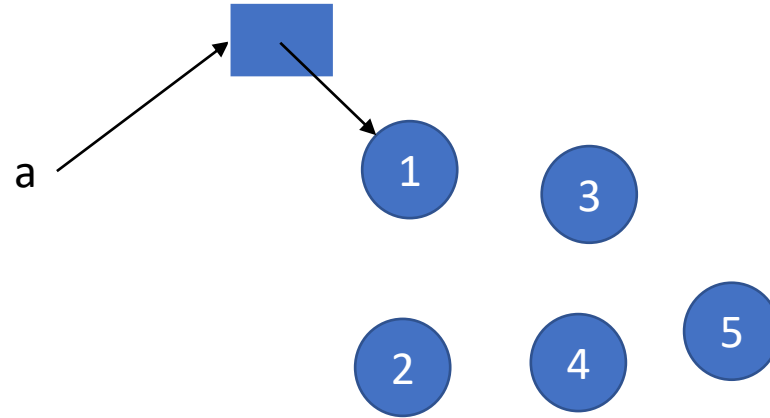
Python - objecten

```
>>> a=1  
>>> b=a  
>>> b=2  
>>> print(a)  
1
```



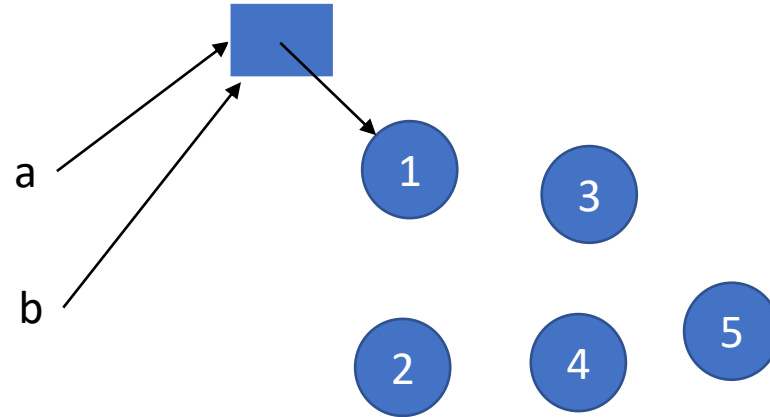
Python - objecten

```
>>> a=[1]
```



Python - objecten

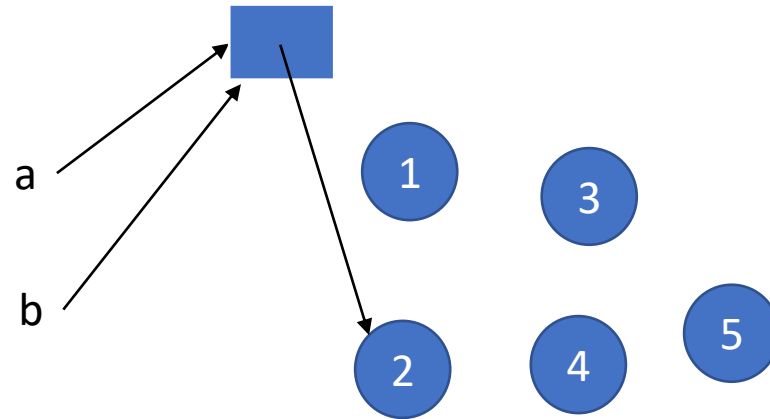
```
>>> a=[1]  
>>> b=a
```



Python - objecten

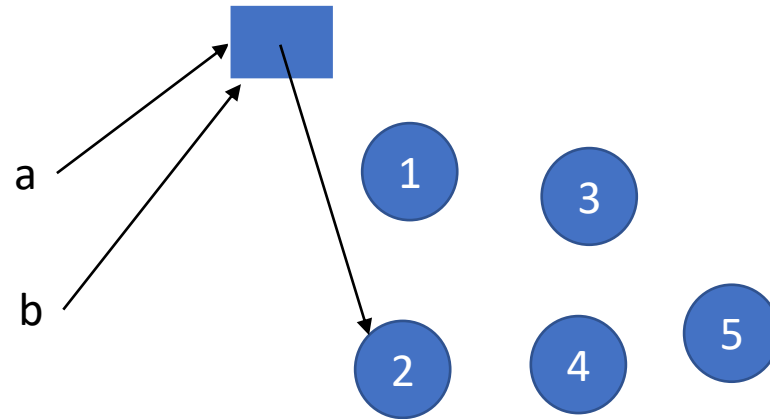
```
>>> a=[1]  
>>> b=a  
>>> b[0]=2
```

Object waar b
naar wijst,
muteert



Python - objecten

```
>>> a=[1]
>>> b=a
>>> b[0]=2
>>> print(a)
[2]
```



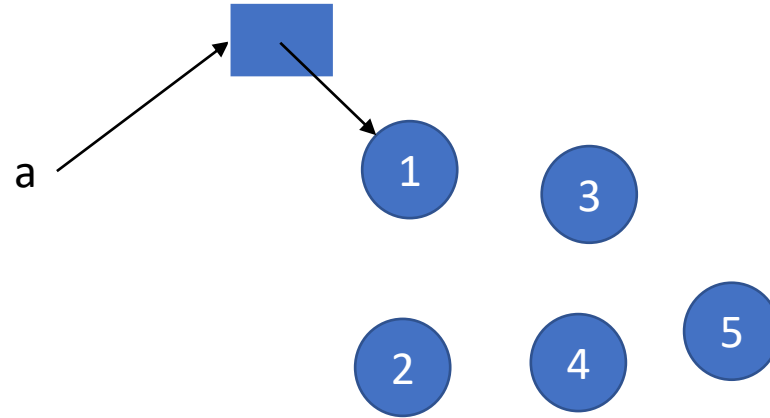
Dus: wat gebeurde er?

- Bij een assignment wordt een lijst niet gekopieerd, maar wordt er een nieuwe referentie gemaakt naar dezelfde lijst
- Als je een kopij wil van een lijst, gebruik dan `lijst[:]`

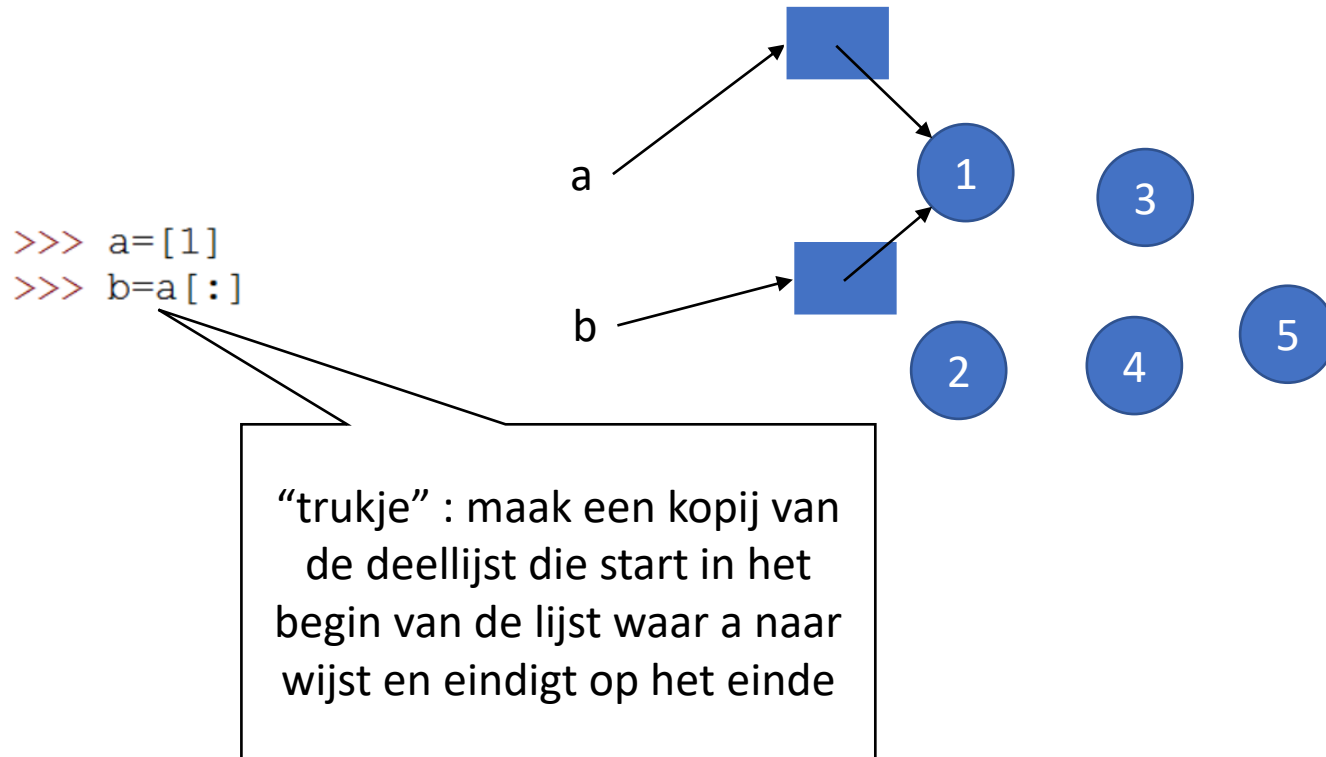
```
>>> a=[1]
>>> b=a[:]
>>> b[0]=2
>>> print(a)
[1]
```

Python - objecten

```
>>> a=[1]
```

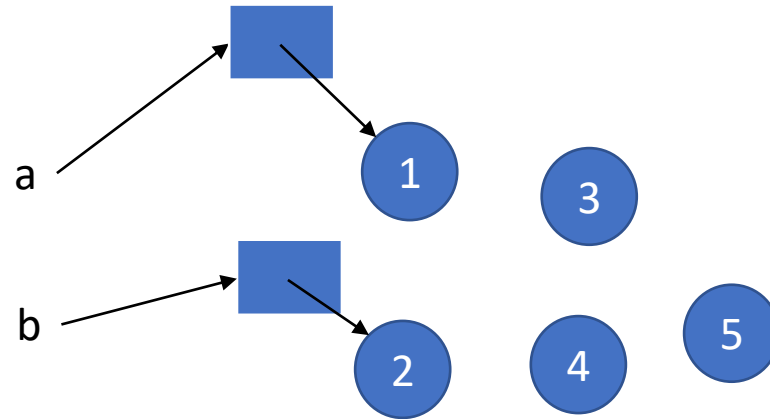


Python - objecten



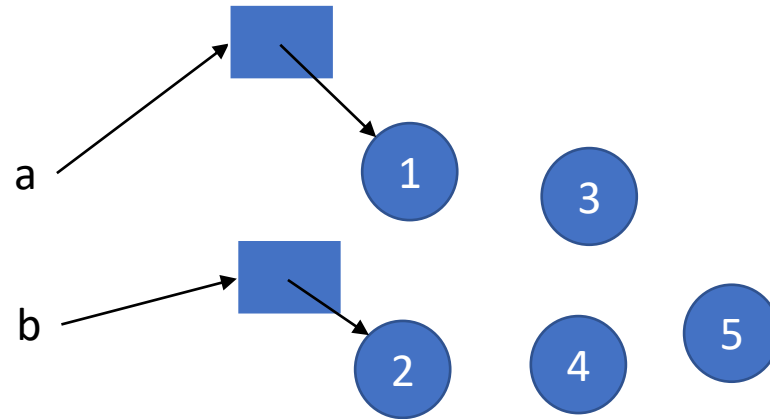
Python - objecten

```
>>> a=[1]  
>>> b=a[:]  
>>> b[0]=2
```



Python - objecten

```
>>> a=[1]
>>> b=a[:]
>>> b[0]=2
>>> print(a)
[1]
```



Vlugge vraag

- Wat is het resultaat van volgende code?

```
L=[1, 2, 3]
K=L
K.append(4)
L[2]=2
K=K[:]
K.append(3)
print(K, L)
```

- A. [1, 2, 3, 4, 3] [1, 2, 2]
- B. [1, 2, 2, 4, 3] [1, 2, 2, 4]
- C. [1, 2, 2, 4, 3] [1, 2, 2, 4, 3]
- D. iets anders

Vlugge vraag

- Wat is het resultaat van volgende code?

```
L=[1, 2, 3]
K=L
K.append(4)
L[2]=2
K=K[: ]
K.append(3)
print(K, L)
```

- A. [1, 2, 3, 4, 3] [1, 2, 2]
- B. [1, 2, 2, 4, 3] [1, 2, 2, 4]**
- C. [1, 2, 2, 4, 3] [1, 2, 2, 4, 3]
- D. iets anders

Dit is erg verwarrend, dus let op!

```
I=[]      # wordt identiteitsmatrix
n=3      # dimensie van de matrix

rij=[0]*n # nul-rij van lengte n
for i in range(n):
    n_rij=rij
    n_rij[i]=1
    I.append(n_rij)

print(I)
```

Zie ook
interactive
code

```
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

Dit is erg verwarrend, dus let op!

```
### Correcte code
```

```
I=[]      # wordt identiteitsmatrix  
n=3      # dimensie van de matrix
```

```
rij=[0]*n  # nul-rij van lengte n  
for i in range(n):  
    n_rij=rij[:]  
    n_rij[i]=1  
    I.append(n_rij)
```

```
print(I)
```

```
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

Zie ook
interactive
code

Dit is erg verwarrend, dus let op!

```
>>> L=[[1,0,0],[0,1,0],[0,0,1]]
>>> K=L[:]
>>> del(K[0])
>>> K
[[0, 1, 0], [0, 0, 1]]
>>> L
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
>>> K[0][0]=2
>>> K
[[2, 1, 0], [0, 0, 1]]
>>> L
[[1, 0, 0], [2, 1, 0], [0, 0, 1]]
```

Oefening

- Schrijf een functie die een matrix als input heeft, en een *diepe* kopij van deze matrix teruggeeft

Copy en Deepcopy

- Bij lijsten kunnen we een kopij maken m.b.v. een “trukje”:
 - $K=L[:]$
- Dit kan ook met “copy” uit module “copy”
 - Werkt ook voor andere types
- copy: shallow kopij
- deepcopy: diepe kopij

Copy en Deepcopy

<https://docs.python.org/3.3/library/copy.html>

```
from copy import deepcopy, copy
```

```
n=3
```

```
nulrij=[0]*n
```

```
I=[]
```

```
for i in range(n):
```

```
    rij=copy(nulrij)
```

```
    rij[i]=1
```

```
    I.append(rij)
```

```
J=deepcopy(I)
```

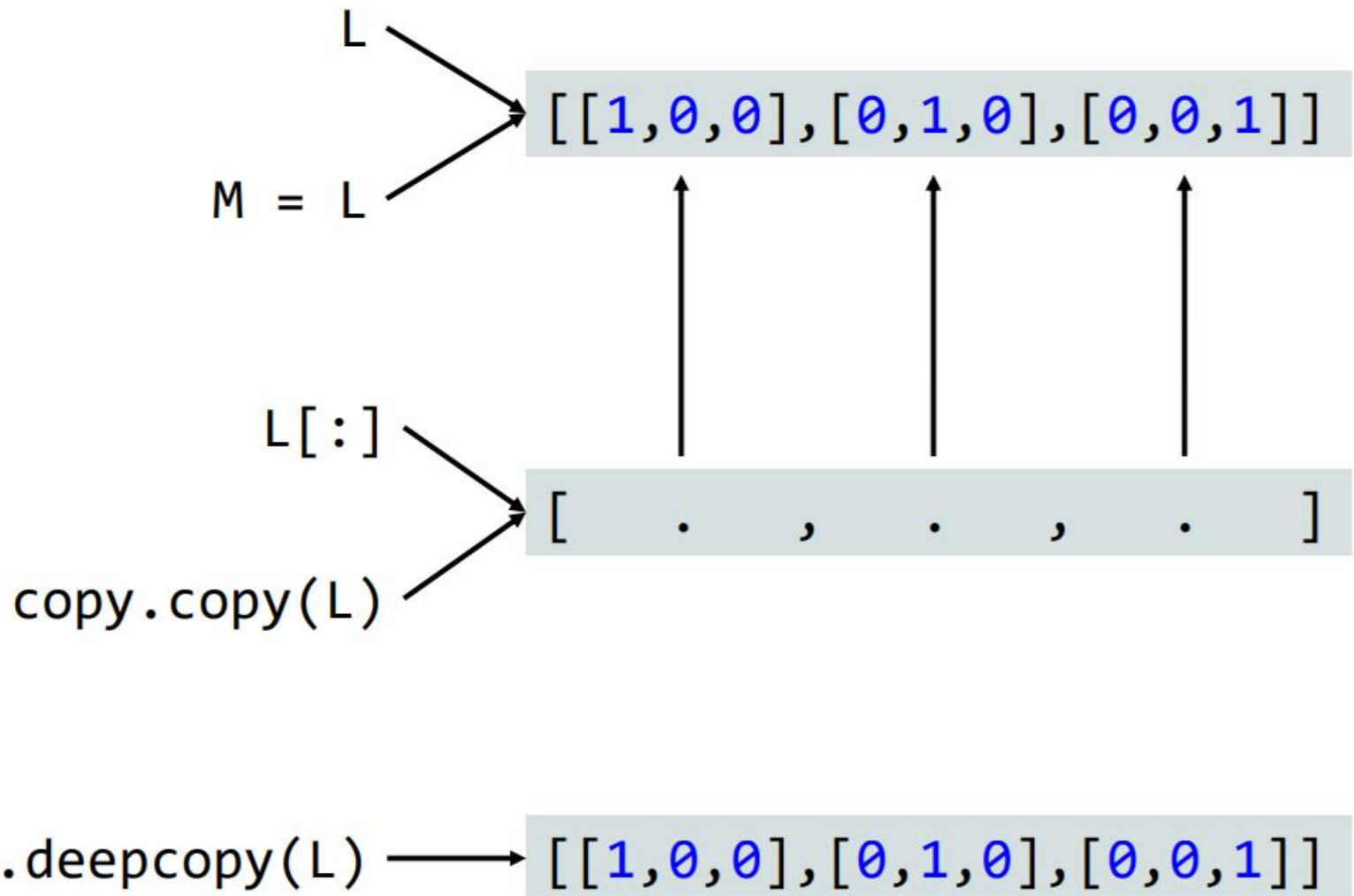
```
J[0][0]=2
```

```
print(I, J)
```

```
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

```
[[2, 0, 0], [0, 1, 0], [0, 0, 1]]
```

Verschillende vormen van copy:



Elementen verwijderen uit lijst

- Verwijder i-de element uit lijst L: `del(L[i])`
- Verwijder het eerste element met waarde "1":
`L.remove(1)`

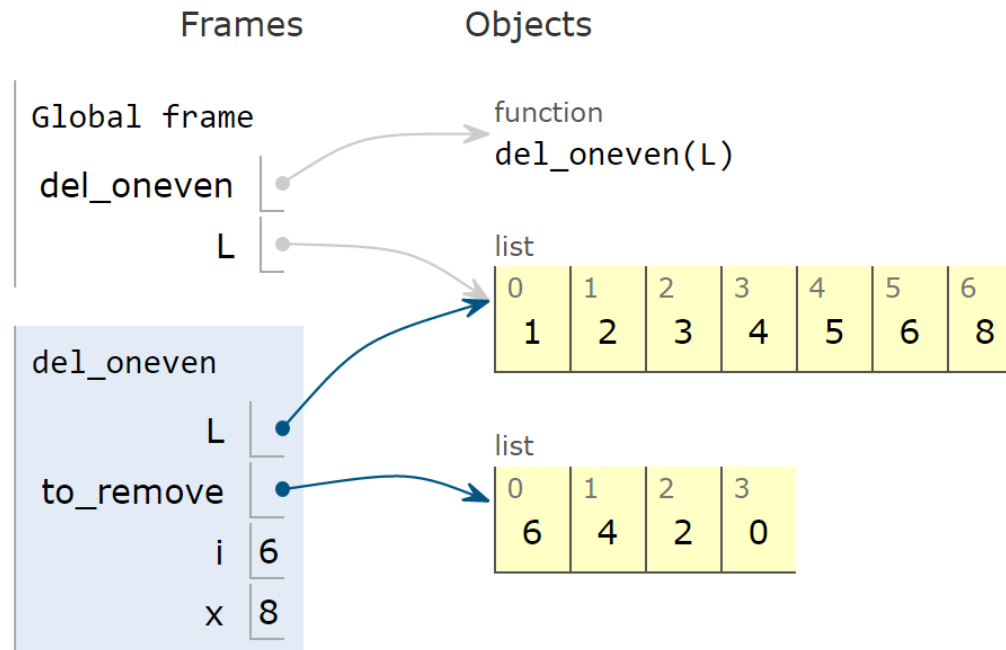
```
>>> L=list(range(4))
>>> K=L+L
>>> K
[0, 1, 2, 3, 0, 1, 2, 3]
>>> K.remove(1)
>>> K
[0, 2, 3, 0, 1, 2, 3]
>>> del(K[0])
>>> K
[2, 3, 0, 1, 2, 3]
```

Lijst Parameters

```
def del_oneven(L):  
    to_remove=[]  
    for i in range(len(L)):  
        x=L[i]  
        if x%2==1:  
            to_remove=[i]+to_remove  
    for i in to_remove:  
        del(L[i])
```

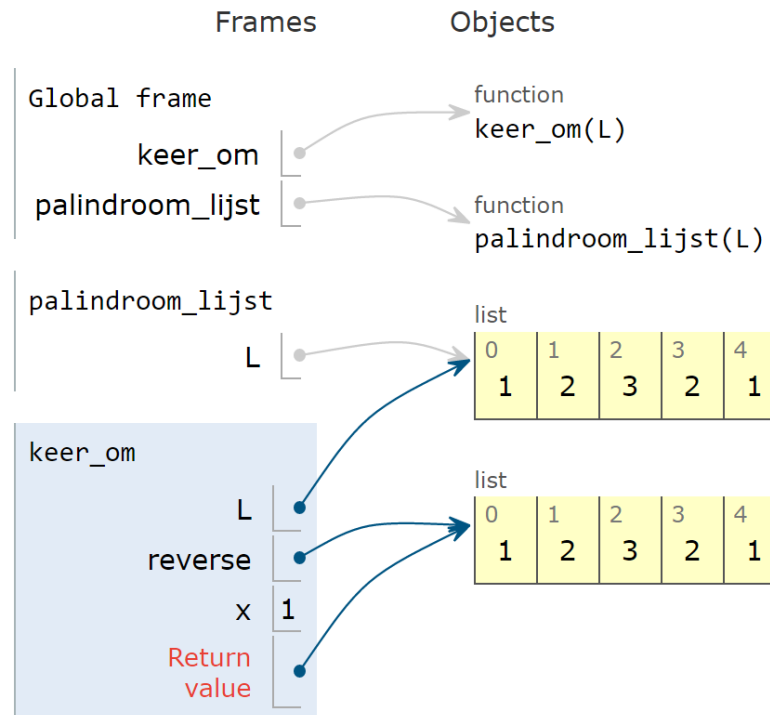
```
L=[1,2,3,4,5,6,7,8]  
del_oneven(L)  
print(L)
```

```
[2, 4, 6, 8]
```



Lijst parameters

```
def keer_om(L):  
    reverse=[]  
    for x in L:  
        reverse=[x]+reverse  
  
    return reverse  
  
def palindroom_lijst(L):  
    if keer_om(L)==L:  
        return True  
    return False  
  
print(palindroom_lijst([1,2,3,2,1]))
```



Vlugge vraag: wat loopt er hier mis?

```
def priem(x):
    for i in range(2, x):
        if x%i==0:
            return False
    return True
```

```
def verwijder_niet_priem(L):
    for x in L:
        print(x, "wordt getest")
        if not priem(x):
            L.remove(x)
            print(x, "is niet priem")
```

```
priemen=list(range(10))
verwijder_niet_priem(priemen)
print(priemen)
```

0 wordt getest
1 wordt getest
2 wordt getest
3 wordt getest
4 wordt getest
4 is niet priem
6 wordt getest
6 is niet priem
8 wordt getest
8 is niet priem
[0, 1, 2, 3, 5, 7, 9]

Handwritten red annotations: A vertical line of question marks (?) is next to the first four lines. A bracket connects the '4' in '4 wordt getest' to the '4' in '4 is niet priem'. Another bracket connects the '6' in '6 wordt getest' to the '6' in '6 is niet priem'. A '9?' is written next to the final list. The number '9' in the list is circled in red, with a red arrow pointing to it from below.

```
### Juist 1:
def verwijder_niet_priem(L):
    for x in L[:]:
        print(x, "wordt getest")
        if not priem(x):
            L.remove(x)
        print(x, "is niet priem")

### Juist 2:
def verwijder_niet_priem(L):
    to_remove=[]
    for x in L:
        print(x, "wordt getest")
        if not priem(x):
            to_remove.append(x)
        print(x, "is niet priem")
    for x in to_remove:
        L.remove(x)
```

Juist 1:

```
def verwijder_niet_priem(L):  
    for x in L[:]:  
        print(x, "wordt getest")  
        if not priem(x):  
            L.remove(x)  
            print(x, "is niet priem")
```

Handwritten annotations:

- A red circle around `L[:]` with an arrow pointing to the text "kopij van L".
- A red circle around `L.remove(x)` with a red "X" next to it.

Juist 2:

```
def verwijder_niet_priem(L):  
    to_remove=[]  
    for x in L:  
        print(x, "wordt getest")  
        if not priem(x):  
            to_remove.append(x)  
            print(x, "is niet priem")  
    for x in to_remove:  
        L.remove(x)
```

Handwritten annotations:

- A red circle around `to_remove=[]`.
- A red circle around `to_remove.append(x)`.
- A red circle around `to_remove` in the second loop.
- A red circle around `L.remove(x)` with a red "X" next to it.

Vlugge vraag: wat is het resultaat?

```
def verdubbel(L):
    """ Verwacht een lijst getallen en verdubbelt
    elk getal """
    for i in range(len(L)):
        L[i]=2*L[i]
```

```
def twee_maal(L):
    """ Doel: maal L gelijk aan L+L. """
    L=L+L
```

```
L=[1,2,3]
verdubbel(L)
twee_maal(L)
print(L)
```

- (A)[1,2,3]
- (B)[2,4,6]
- (C)[1,2,3,1,2,3]
- (D)[2,4,6,2,4,6]



Lijsten als parameters

```
def verdubbel(L):  
    """ Verwacht een lijst getallen en verdubbelt  
    elk getal """  
    for i in range(len(L)):  
        L[i]=2*L[i]
```



```
def twee_maal(L):  
    """ Doel: maal L gelijk aan L+L. """  
    for x in L[:]:  
        L.append(x)
```

```
L=[1,2,3]  
verdubbel(L)  
twee_maal(L)  
print(L)
```

Vlugge vraag

- Wat doet de volgende functie?

```
def bubble(L):  
    swap=True  
    while swap:  
        swap=False  
        for i in range(len(L)-1):  
            if L[i]>L[i+1]:  
                L[i], L[i + 1] = L[i + 1], L[i] # wissel L[i] en L[i+1]  
                swap=True
```

Opmerking: lijst sorteren

- Sorteert een lijst L:
 - `L.sort()`
- Geef gesorteerde versie van L, maar laat L zelf ongemoeid:
 - `sorted_L = sorted(L)`
- We kunnen een key en reverse geven:
 - `key = functie`
 - `reverse = True` of `= False` ; by default `reverse == False`

Voorbeeld : lijst sorteren

```
L=["Michael","Jim","Dwight","Pam","Ryan","Stanley","Kevin"  
  ,"Meredith","Angela","Oscar","Phyllis","Toby","Kelly"]
```

```
L.sort(reverse=True)
```

```
print(sorted(L))
```

```
print(L)
```

```
M=[[1,2,3],[2,0,1],[3,5,7]]
```

```
M.sort()
```

```
print(M)
```

```
['Angela', 'Dwight', 'Jim', 'Kelly', 'Kevin', 'Meredith', 'Michael', 'Oscar', 'Pam', ...  
['Toby', 'Stanley', 'Ryan', 'Phyllis', 'Pam', 'Oscar', 'Michael', 'Meredith', 'Kevin', ...  
[[1, 2, 3], [2, 0, 1], [3, 5, 7]]
```

Voorbeeld : lijst sorteren

```
M=[[1,2,3],[2,0,1],[3,5,7]]
```

```
def rowsum(r):
```

```
    som=0
```

```
    for x in r:
```

```
        som += x
```

```
    return som
```

```
print(sorted(M,key=rowsum))
```

```
[[2, 0, 1], [1, 2, 3], [3, 5, 7]]
```

Opmerking: strings

- strings zijn in python immutable
 - ~~s.append("abc")~~
- We moeten expliciet een nieuwe string maken:
 - `s = s+"abc"`
- Strings zijn *itereerbaar* ; we kunnen dus een for-loop gebruiken om over de karakters te lopen

Voorbeeld: strings

```
s="abcdadbcde"
```

```
a = 0
```

```
for letter in s:
```

```
    if letter == "a":
```

```
        a += 1
```

```
print("De string bevat",a,"maal de letter a")
```


String functies

- Er bestaan heel wat handige stringfuncties:
 - `s.trim()` : verwijder witruimte
 - `s.split()` : splits de string
 - `s.upper()` : maak upper case
 - ...
- Merk op: strings zijn immutable; al deze functies geven een nieuwe string als resultaat maar laten de string waarop ze worden aangeroepen ongewijzigd

Oefening

- Gegeven:
 - Lijst met woorden woordenBoek
 - Patroon P dat bestaat uit letters en wildcards (“?”)
- Geef alle woorden uit woordenBoek die je kan maken door de wildcards te vervangen door letters
- “aap” voldoet aan “?ap” maar niet aan “b??” of “a?ap”

```
from words import woordenBoek  
print(match("p??h??", woordenBoek))
```