Ghent University
Faculty of Sciences

Python Programming

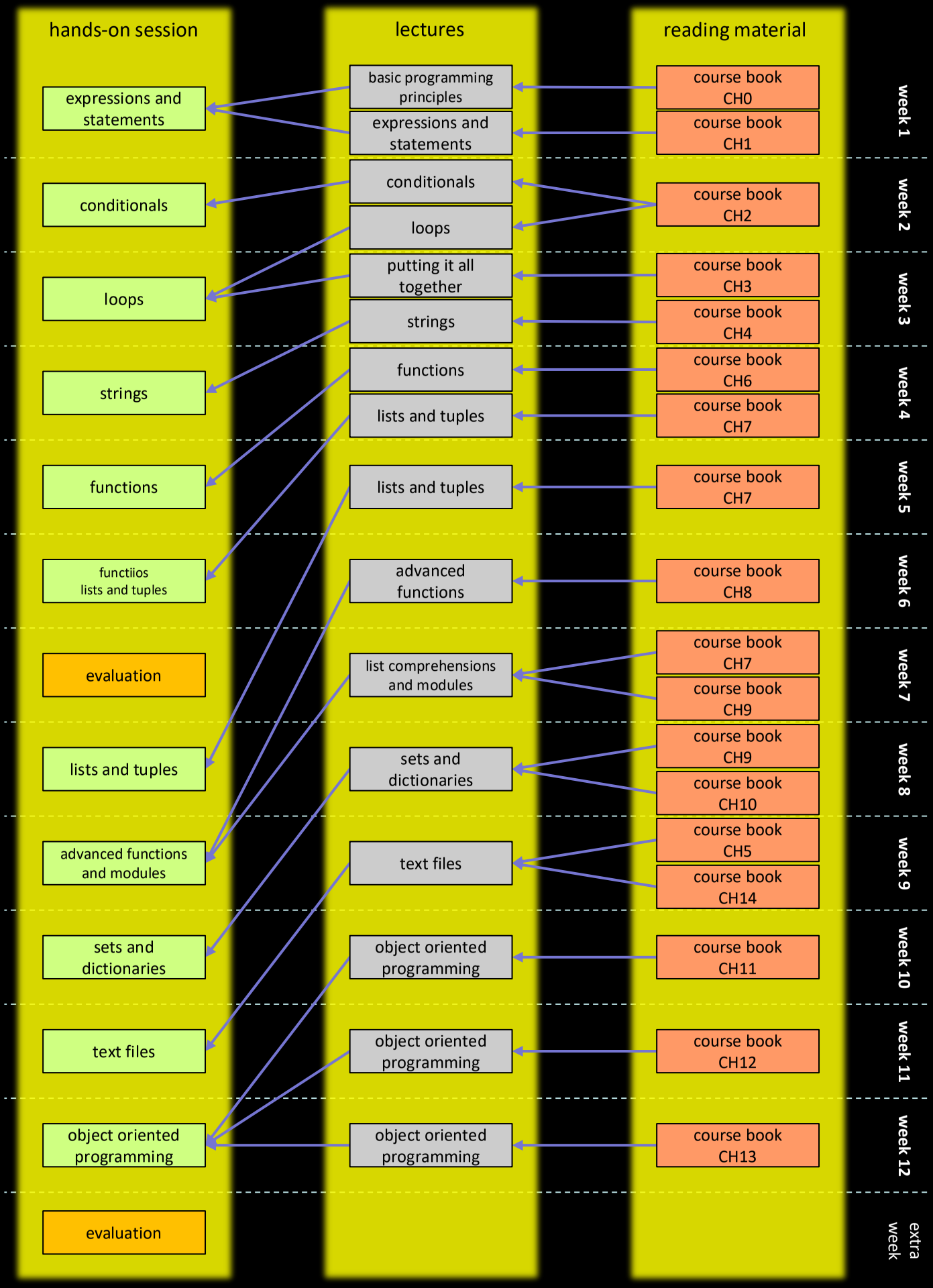
nested lists

Prof. Dr. Peter Dawyndt

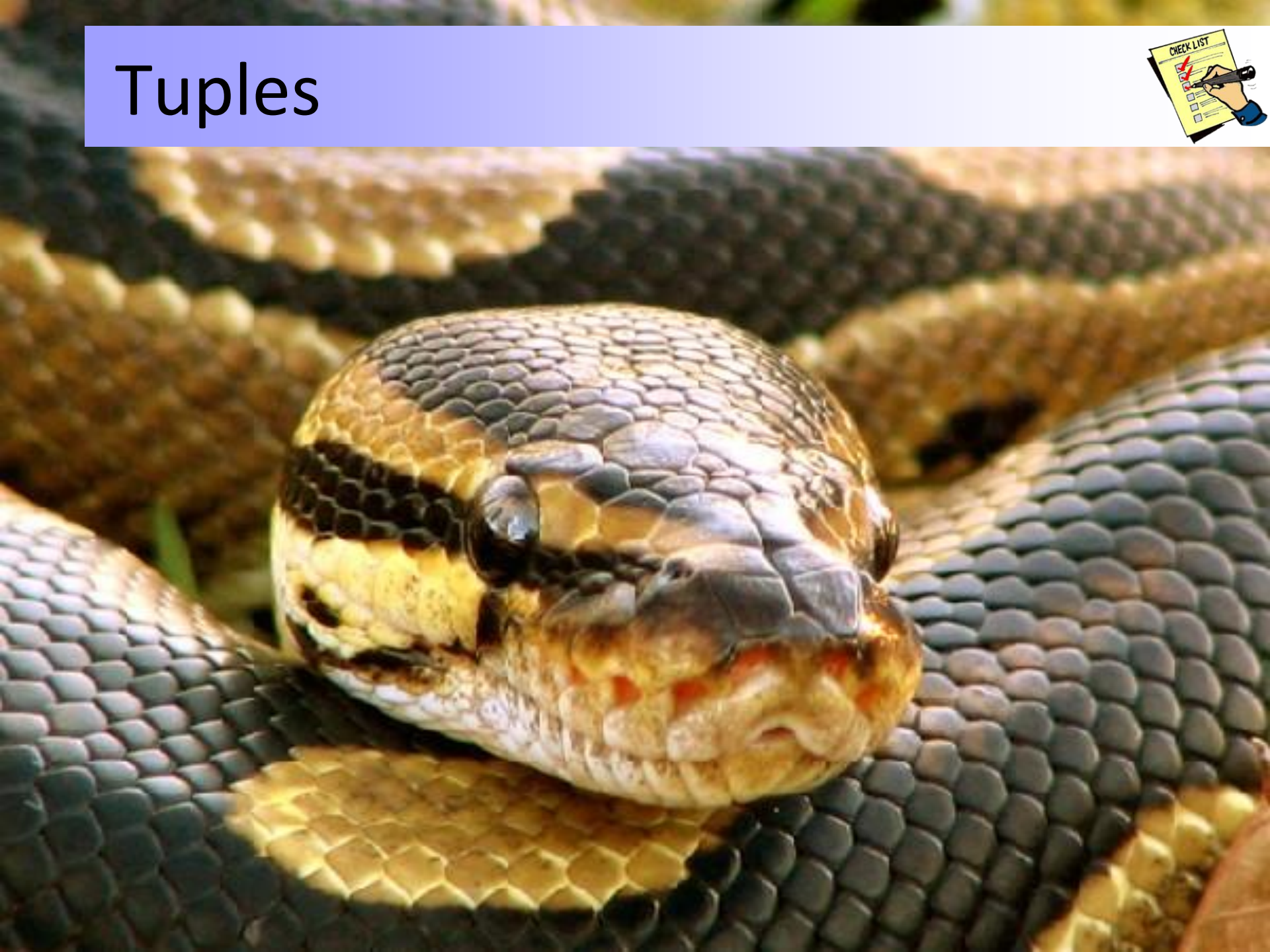
peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries
text files
object oriented programming
evaluation
basic programming principles
expressions and statements
conditionals
loops
putting it all together
strings
functions
lists and tuples
lists and tuples
advanced functions
list comprehensions and modules
sets and dictionaries
text files
object oriented programming
object oriented programming
object oriented programming
course book CH0
course book CH1
course book CH2
course book CH3
course book CH4
course book CH6
course book CH7
course book CH7
course book CH8
course book CH7
course book CH9
course book CH9
course book CH10
course book CH5
course book CH14
course book CH11
course book CH12
course book CH13

Tuples



Tuples

- **tuple**: *immutable* sequence of objects
  - *immutable* version of a list (can't be modified after creation)
  - written using parentheses instead of square brackets
    - `()` represents an empty tuple
    - tuple with a single element → extra comma: `(55,)`

```
>>> gases = ('He', 'Ne', 'Ar', 'Kr')
>>> print(gases)
('He', 'Ne', 'Ar', 'Kr')
>>> print(gases[0], gases[-1]))
He Kr
>>> print(gases[3:2])
()
>>> print(gases[2:3])
('Ar',)
```

UNIVERSITEIT GENT

Tuple unpacking

- **tuple**: *immutable* sequence of objects
  - usually not strictly necessary to put parentheses around tuples
  - multi-object assignment per element (*tuple unpacking*)
    - # names on the left must match # objects on the right

```
>>> gases = 'He', 'Ne', 'Ar'
>>> print(gases)
('He', 'Ne', 'Ar')
>>> (gas1, gas2, gas3) = gases
>>> print(gas1, gas2)
He Ne
>>> gas1, gas2, gas3 = gases
>>> gas1, gas2 = gases
ValueError: too many values to unpack
```

UNIVERSITEIT GENT

Tuple unpacking

- **tuple**: *immutable* sequence of objects
  - applications of *tuple unpacking*
    - multi-object assignment
    - switch object assignments
    - functions that seem to return multiple object

```
>>> gas1, gas2 = 'He', 'Ne'
>>> gas1, gas2 = gas2, gas1
>>> def multiple(n): return n**2, n**3
...
>>> x2, x3 = multiple(10)
>>> print(x2, x3)
100 1000
```

UNIVERSITEIT GENT

Tuple unpacking

- **tuple**: *immutable* sequence of objects
  - applications of *tuple unpacking*
    - multi-object assignment
    - switch object assignments
    - functions that seem to return multiple object
    - multi-object assignment in for loops unpack tuples on the fly

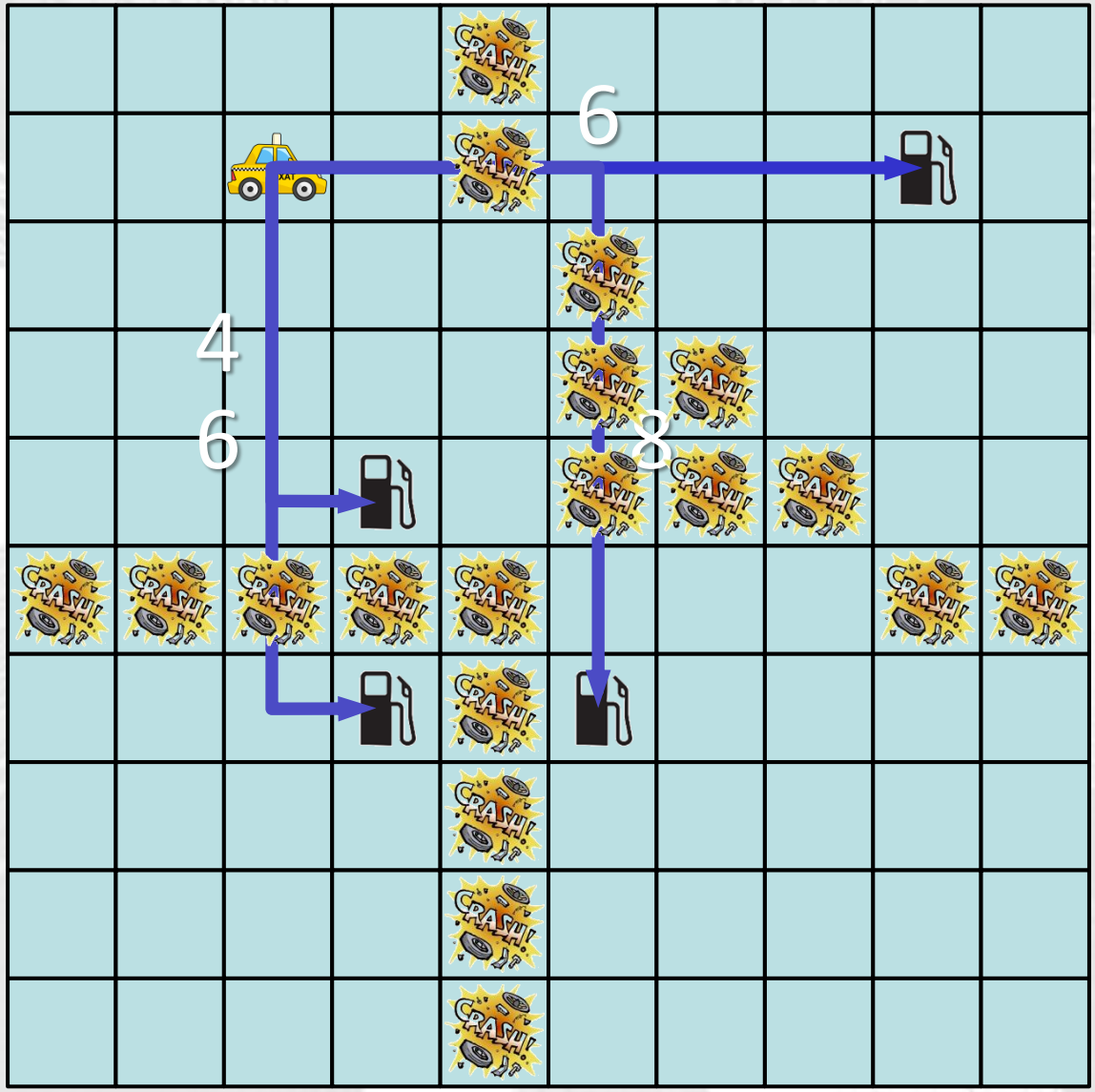
unpacking.py

```
elements = [
   ['H', 'hydrogen', 1.008],
   ['He', 'helium', 4.003],
   ['Li', 'lithium', 6.941],
   ['Be', 'beryllium', 9.012]
]

for symbol, name, mass in elements:
    print(f'{name} ({symbol}): {mass :.3f}')
```

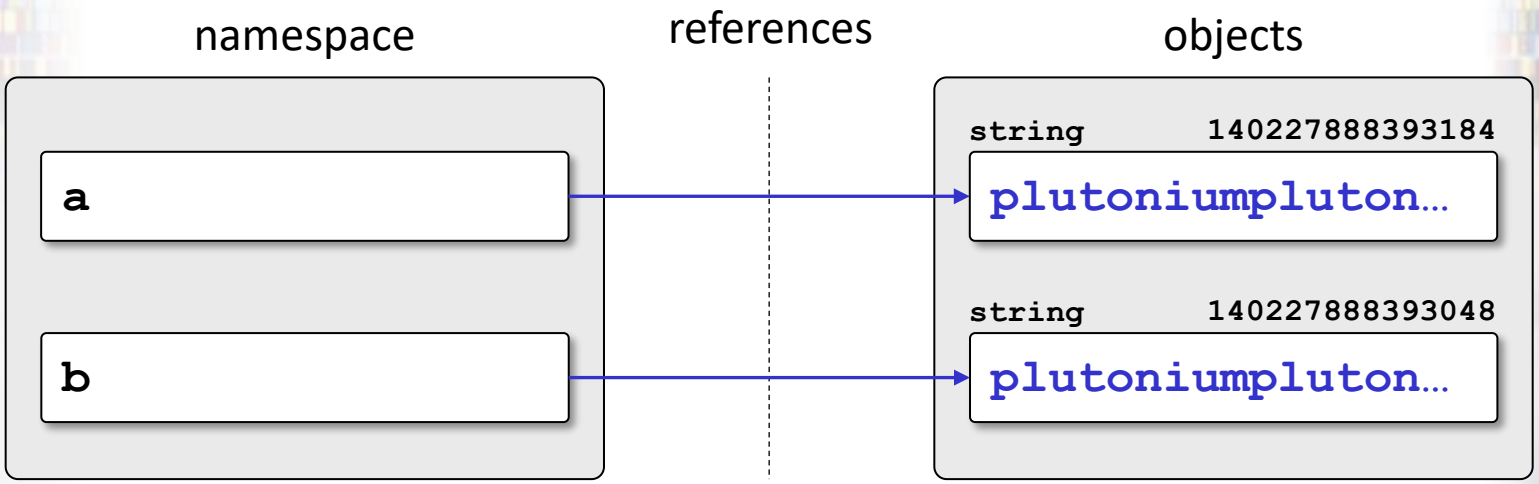
UNIVERSITEIT GENT

Manhattan



Aliases

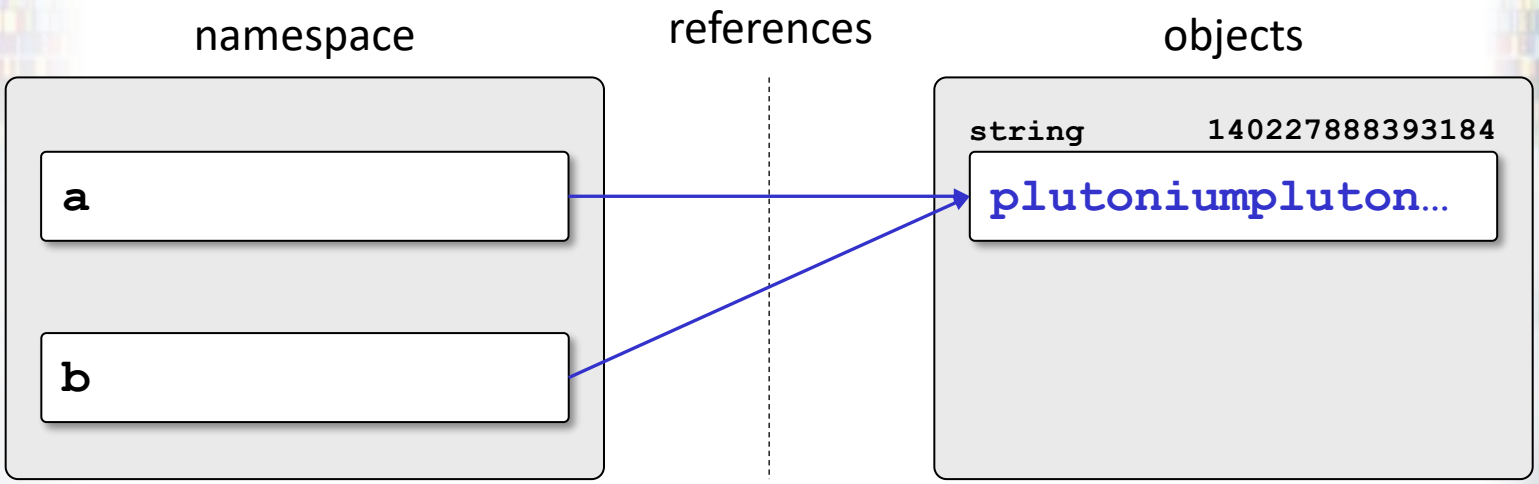
String aliases




```
>>> a = 'plutonium' * 100
>>> b = 'plutonium' * 100
>>> a == b
True
>>> a is b
False
>>> id(a), id(b)
(140227888393184, 140227888393048)
```

UNIVERSITEIT GENT

String aliases

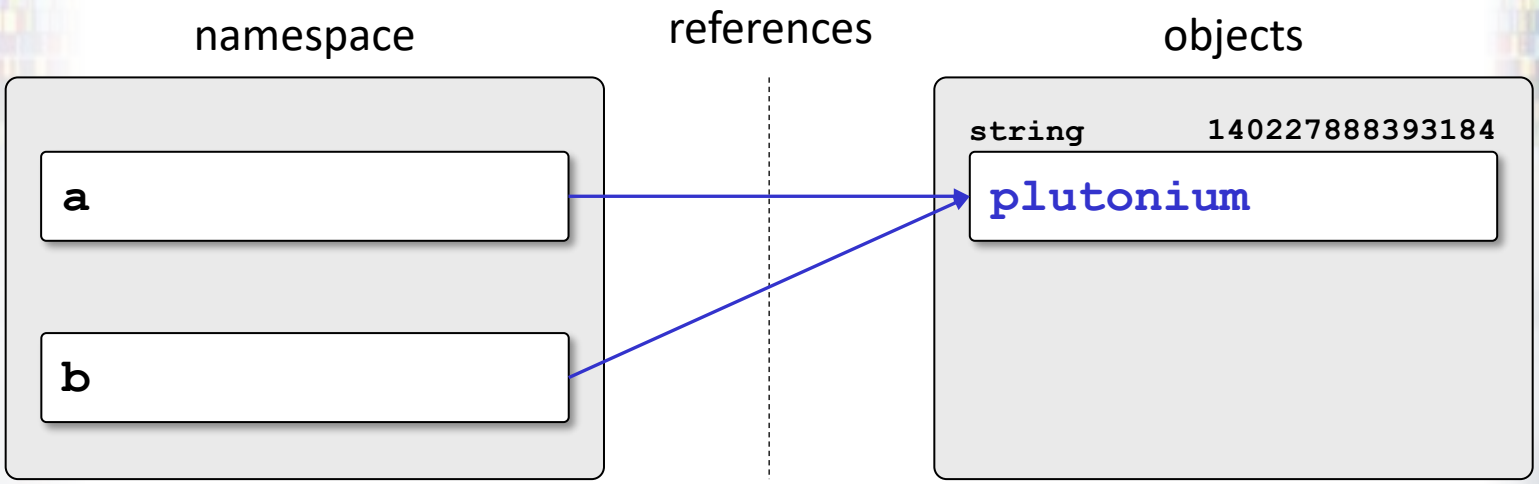



```
>>> a = 'plutonium' * 100
>>> b = a
>>> a == b
True
>>> a is b
True
>>> id(a), id(b)
(140227888393184, 140227888393184)
```

"a and b refer to the same object: they are aliases."

UNIVERSITEIT GENT

String aliases

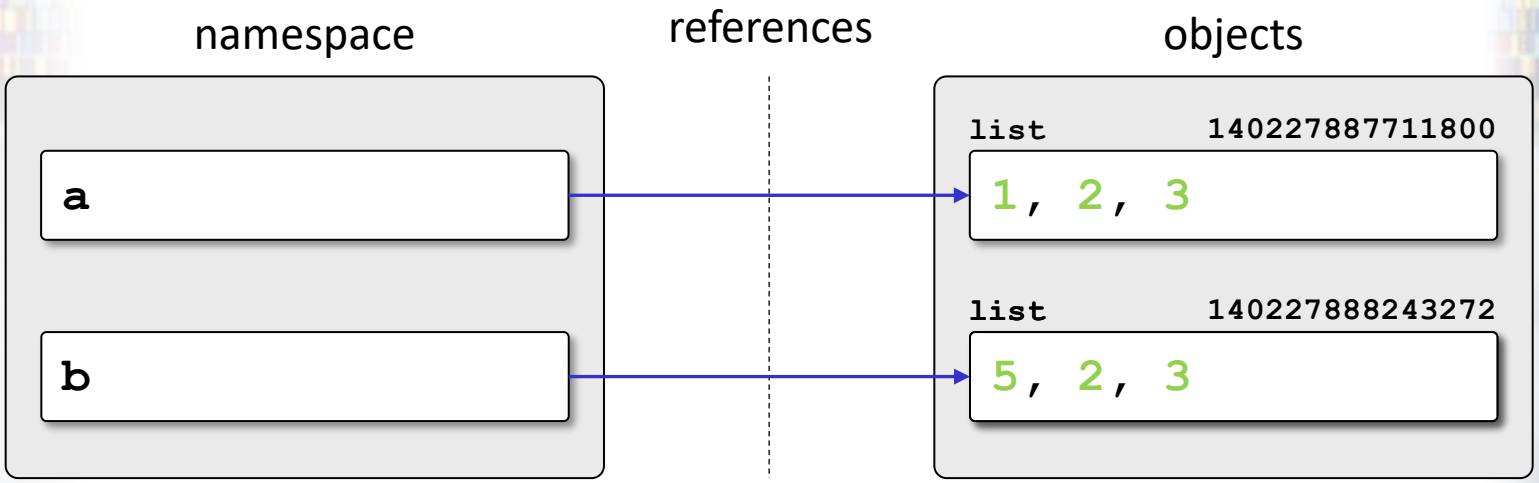



```
>>> a = 'plutonium'
>>> b = 'plutonium'
>>> a == b
True
>>> a is b
True
>>> id(a), id(b)
(140227888393184, 140227888393184)
```

"Python can apply an optimisation because strings are immutable."

UNIVERSITEIT GENT

List aliases

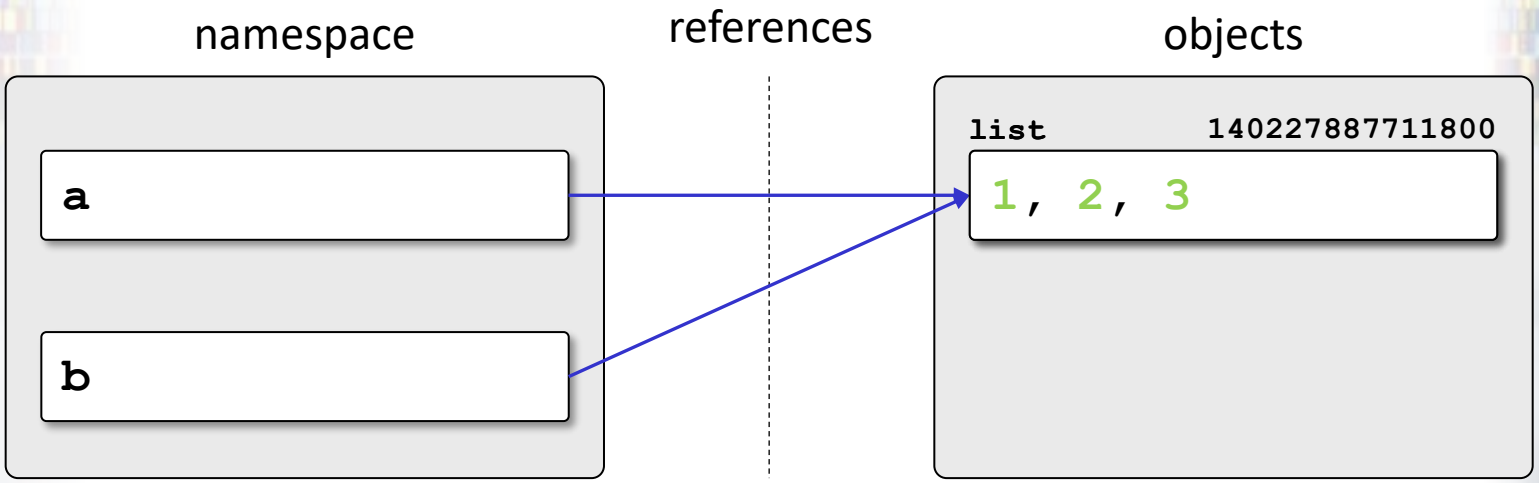



```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
>>> b[0] = 5
>>> print(a)
[1, 2, 3]
```

"Python never applies the optimisation because lists are mutable ."

UNIVERSITEIT GENT

List aliases

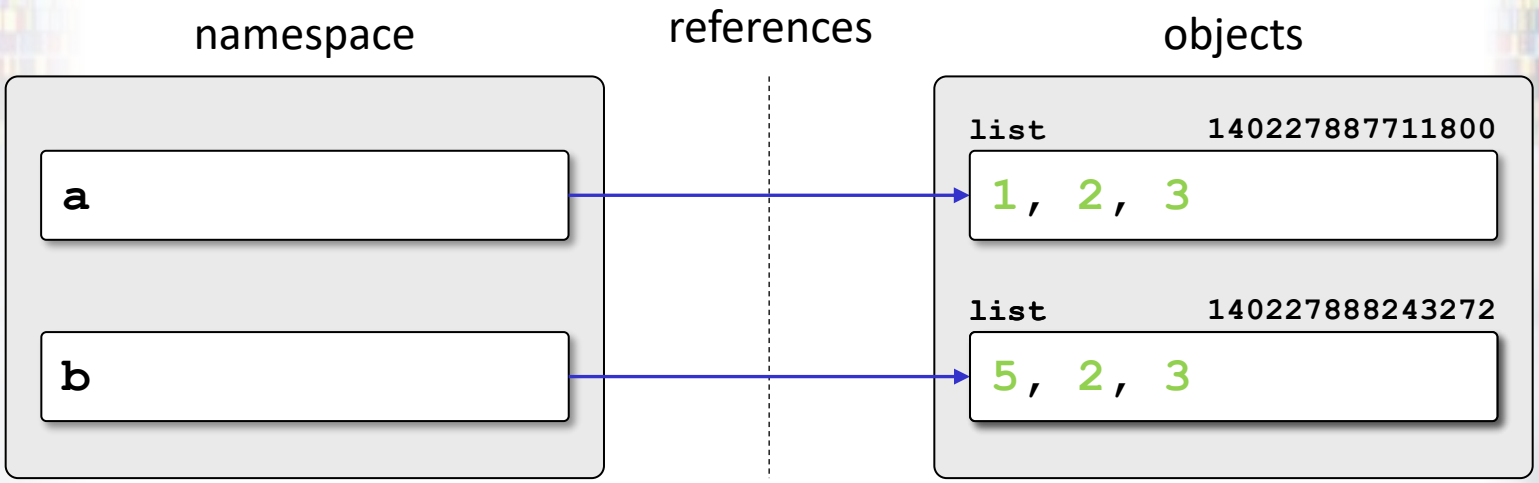



```
>>> a = [1, 2, 3]
>>> b = a
>>> a is b
True
>>> b[0] = 5
>>> print(a)
[5, 2, 3]
```

"a and b refer to the same object: they are aliases."

UNIVERSITEIT GENT

Cloning lists

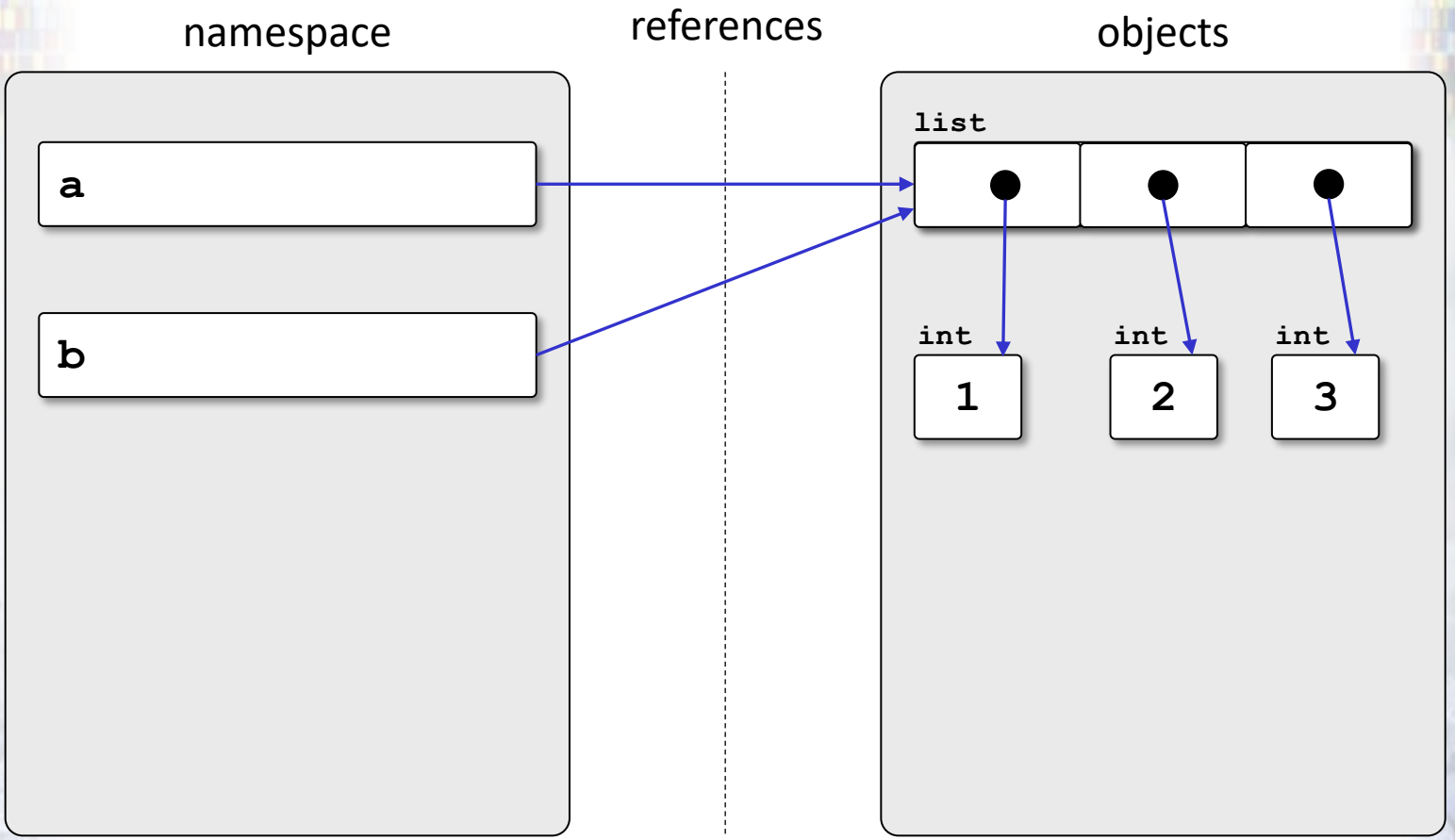



```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> a is b
False
>>> b[0] = 5
>>> print(a)
[1, 2, 3]
```

"Slicing each time creates a new list for the subsequence."

UNIVERSITEIT GENT

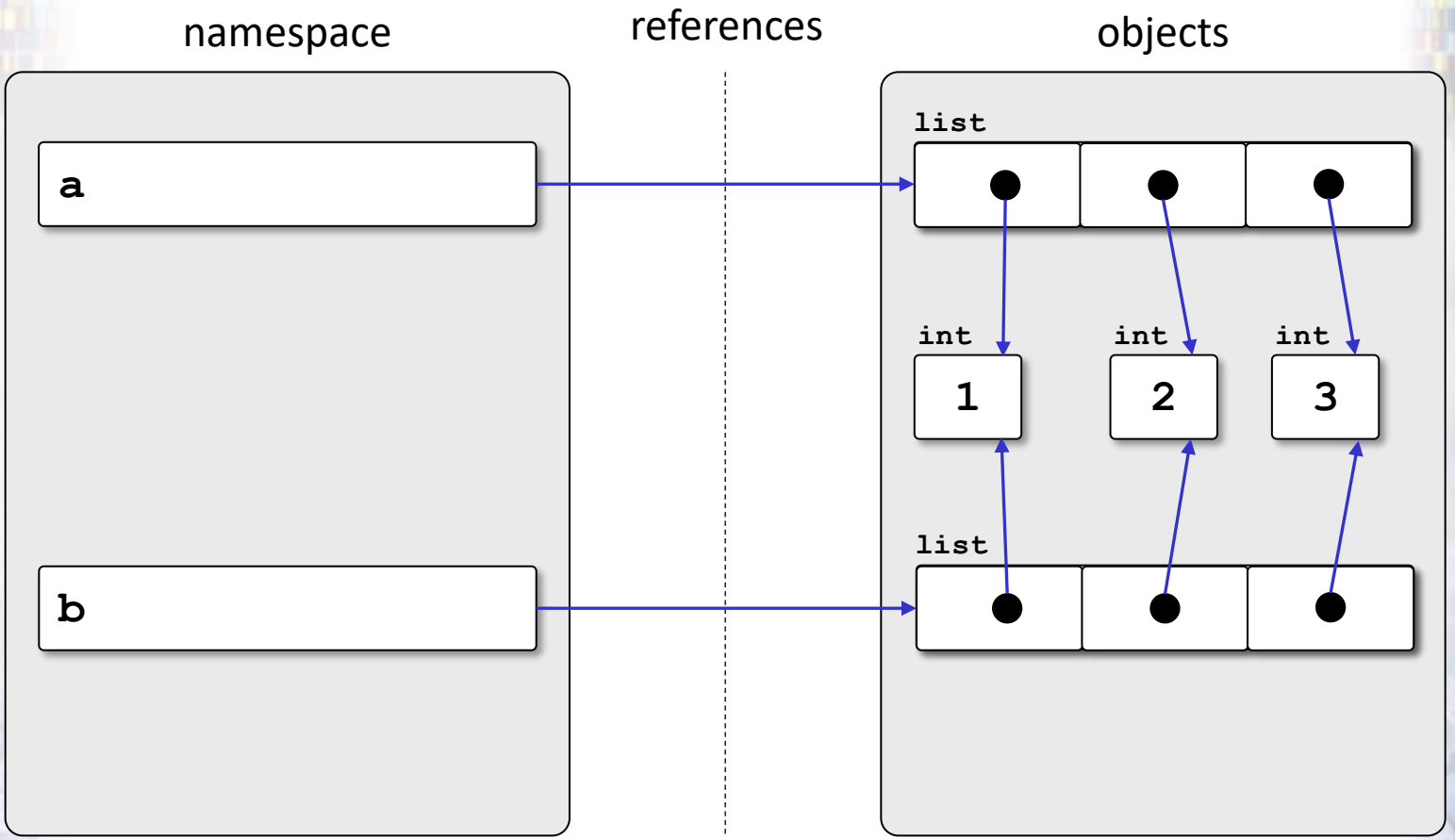
Alias vs copy vs deepcopy




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

UNIVERSITEIT GENT

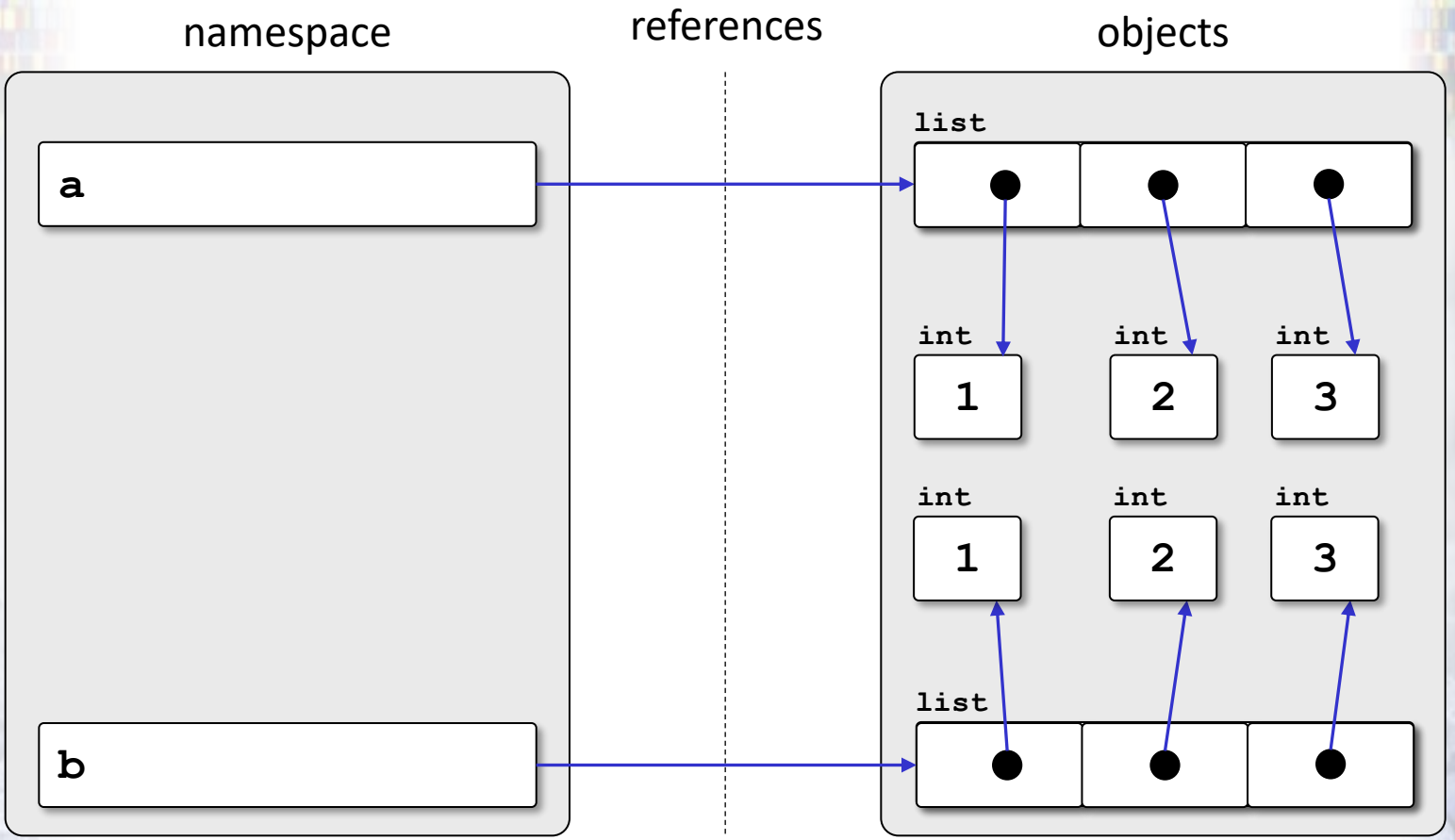
Alias vs **copy** vs deepcopy




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

UNIVERSITEIT GENT

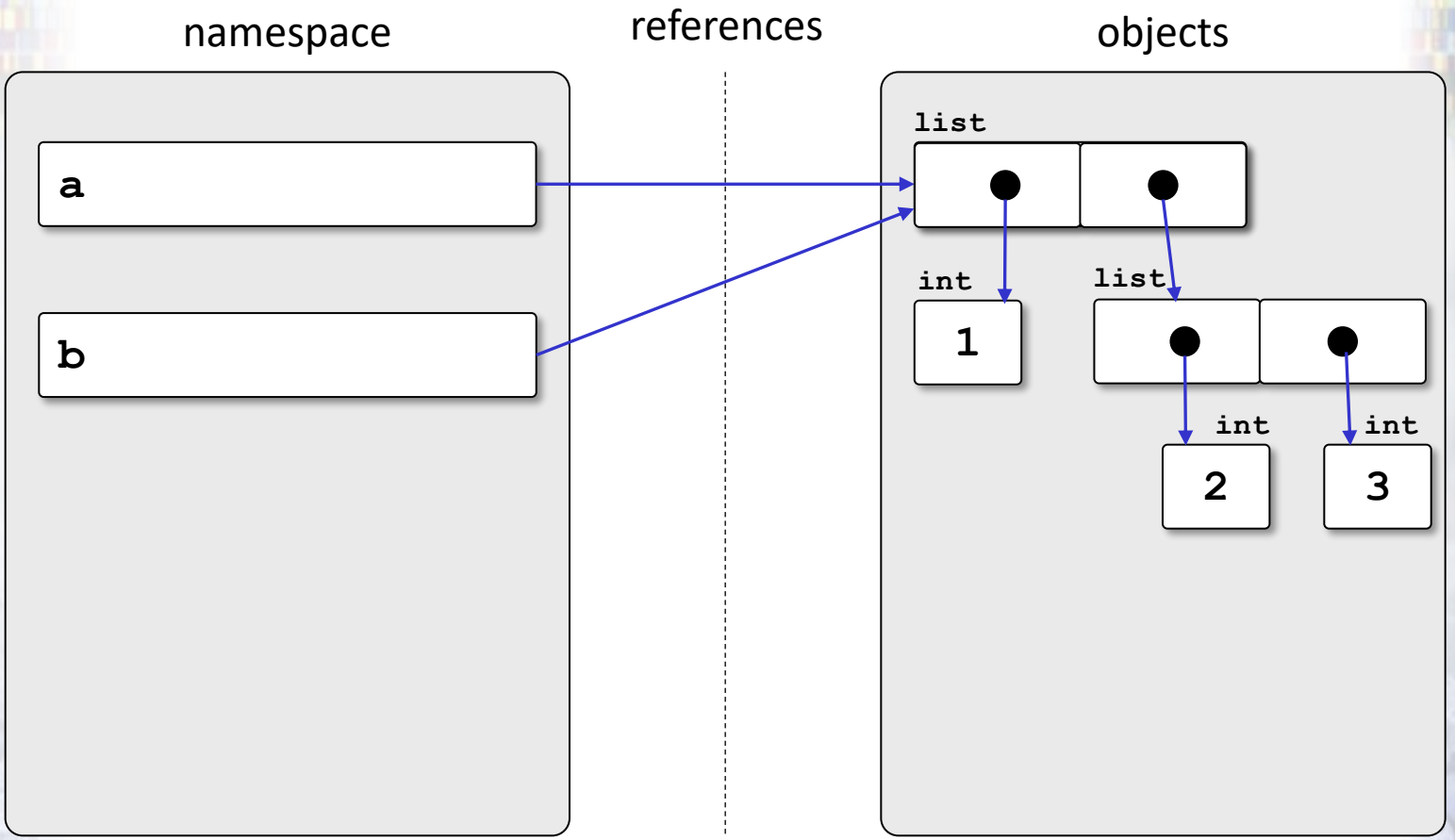
Alias vs copy vs **deepcopy**



UNIVERSITEIT GENT

```
>>> a = [1, 2, 3]
>>> b = deepcopy(a)
```

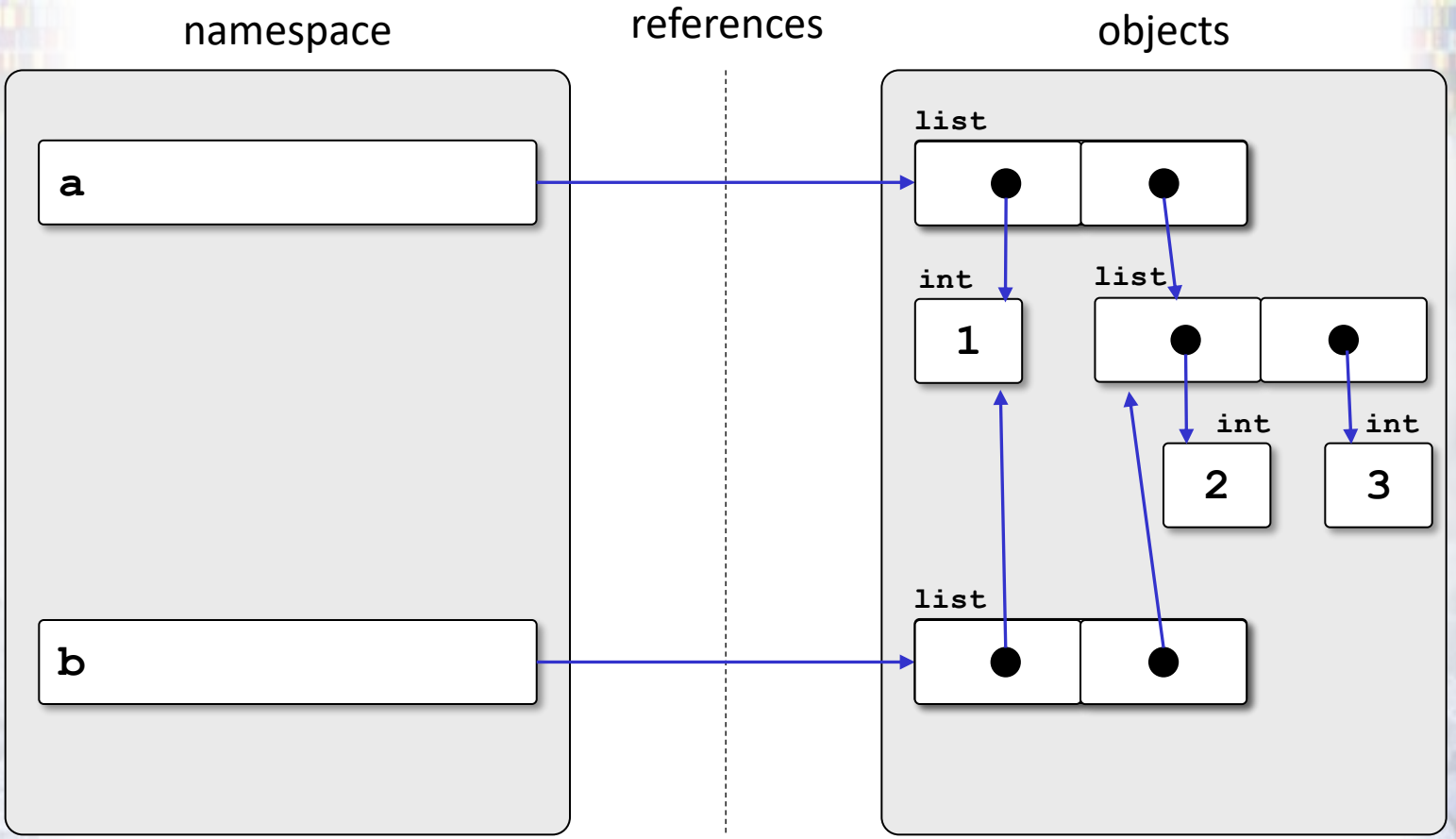
Alias vs copy vs deepcopy



UNIVERSITEIT GENT

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

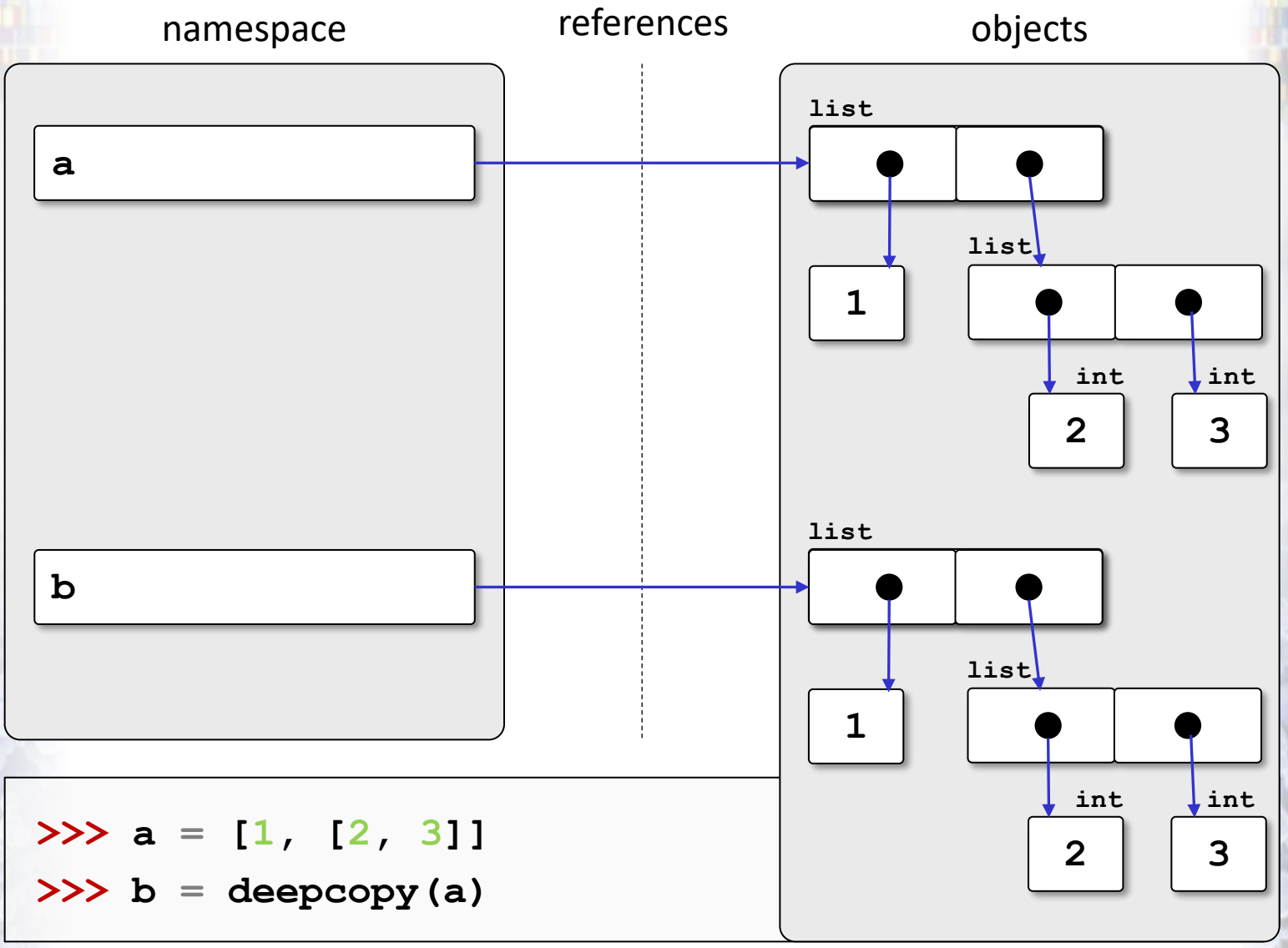
Alias vs **copy** vs deepcopy



UNIVERSITEIT GENT

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

Alias vs copy vs **deepcopy**




```
>>> a = [1, [2, 3]]
>>> b = deepcopy(a)
```

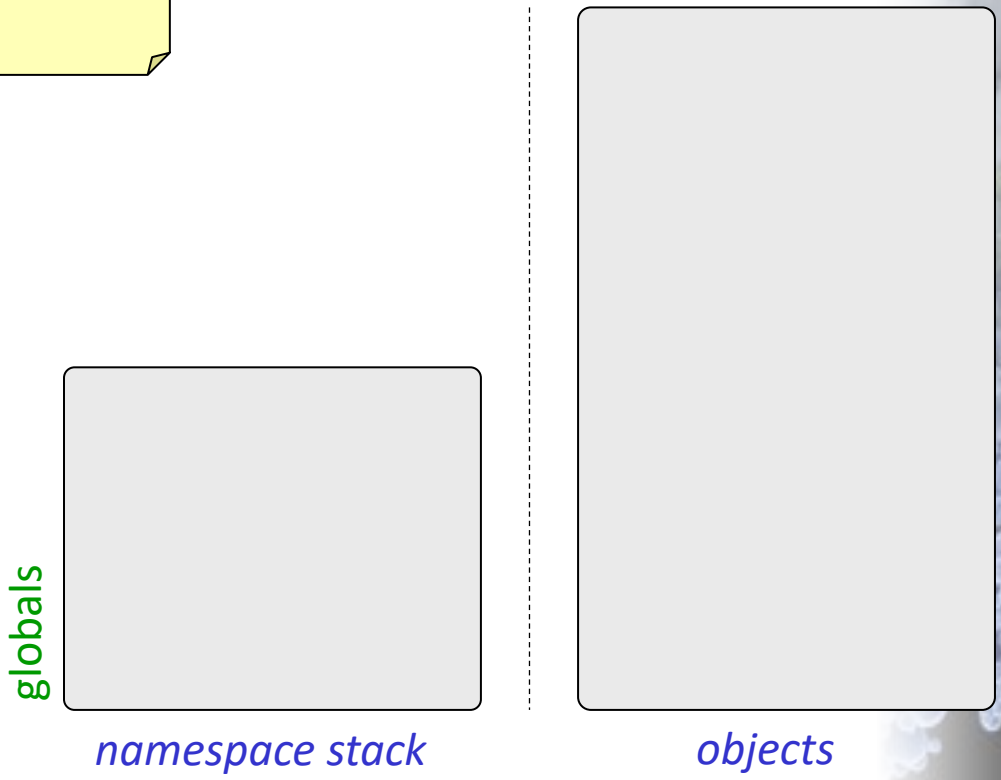
UNIVERSITEIT GENT

Lists as arguments

double.py

```
def double(seq):
    for index, value in enumerate(seq):
        seq[index] = 2 * value

things = [2, 5, 'spam', 9.5]
double(things)
print(things)
```



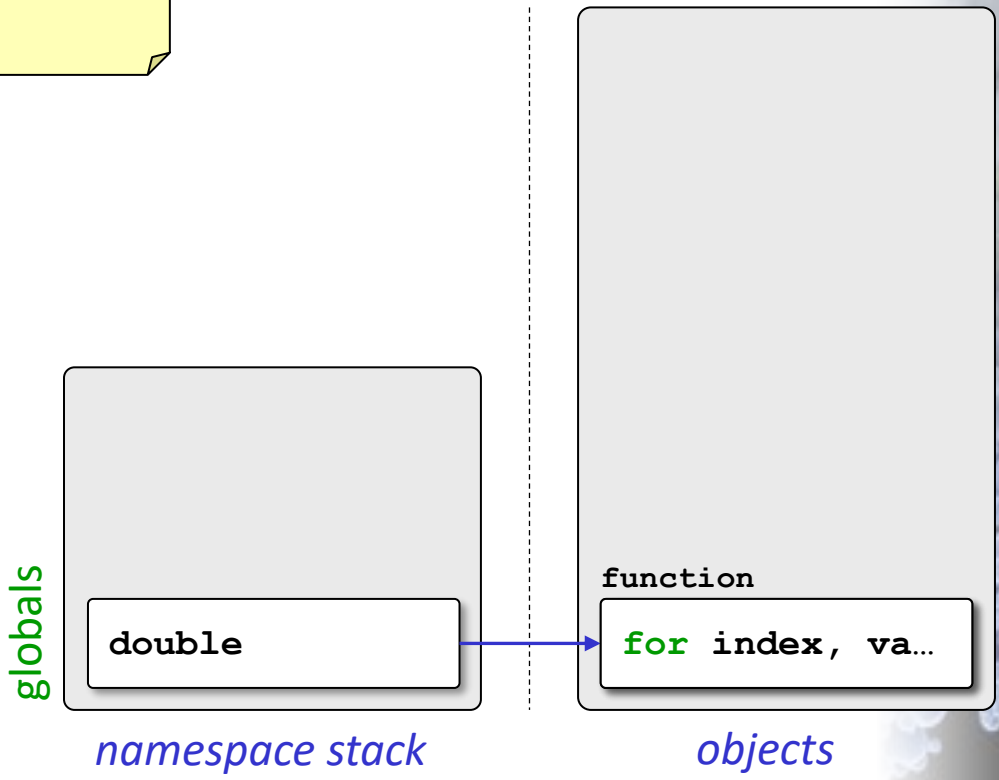
UNIVERSITEIT GENT

Lists as arguments

double.py

```
def double(seq):
    for index, value in enumerate(seq):
        seq[index] = 2 * value

things = [2, 5, 'spam', 9.5]
double(things)
print(things)
```



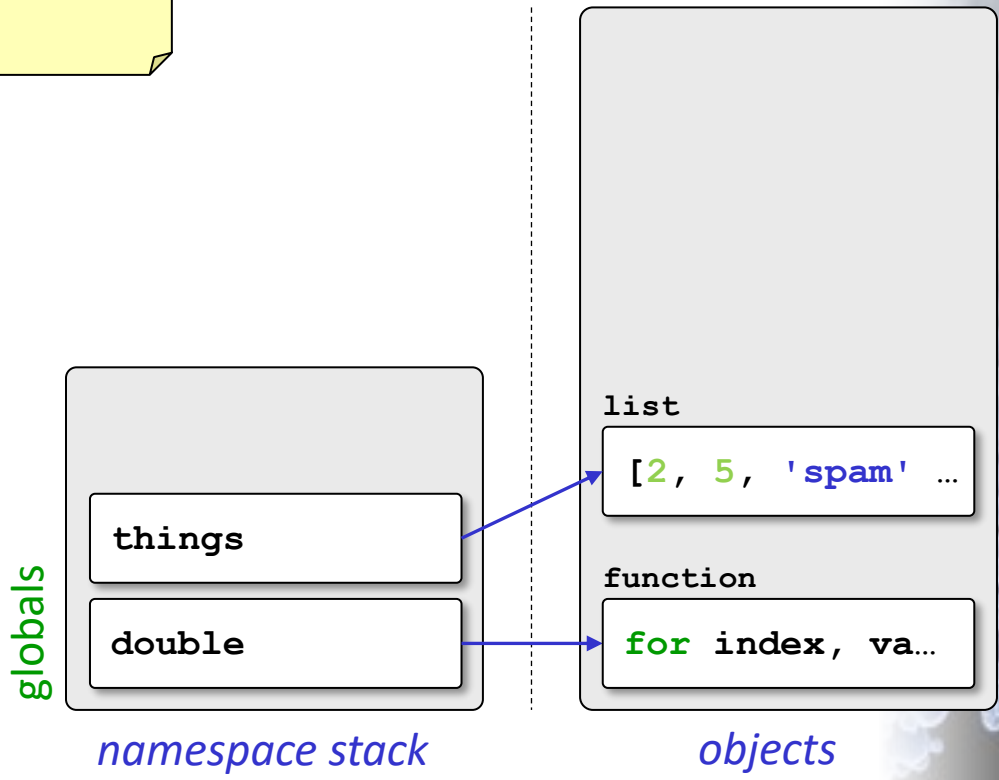
UNIVERSITEIT GENT

Lists as arguments

double.py

```
def double(seq):
    for index, value in enumerate(seq):
        seq[index] = 2 * value

things = [2, 5, 'spam', 9.5]
double(things)
print(things)
```



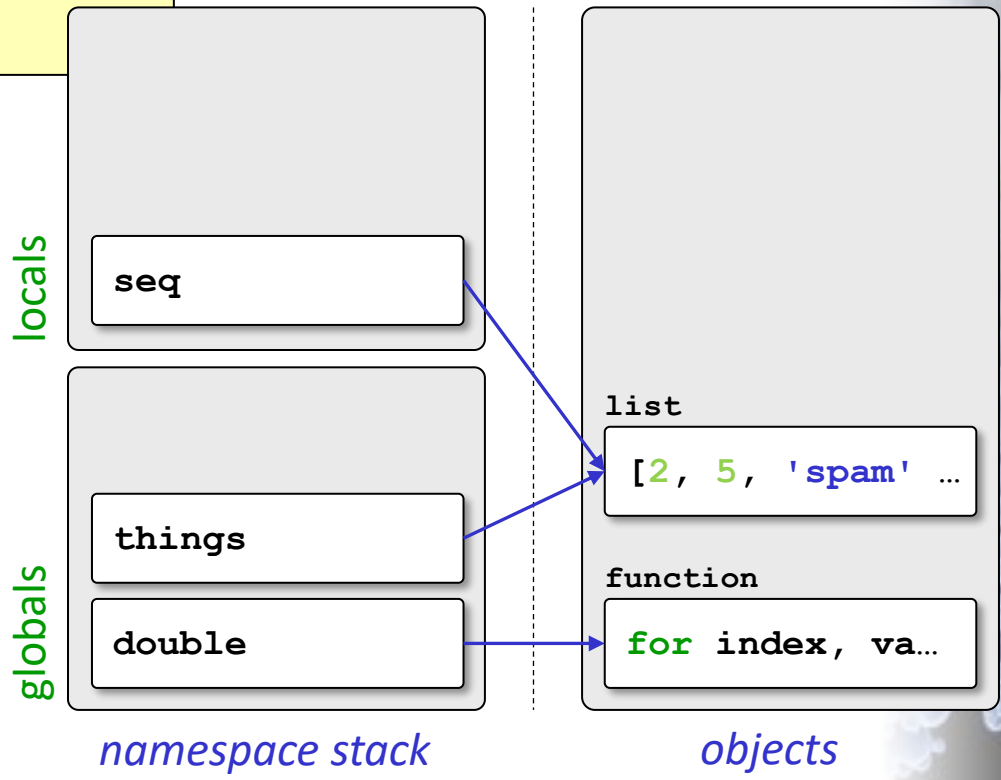
UNIVERSITEIT GENT

Lists as arguments

double.py

```
def double(seq):
    for index, value in enumerate(seq):
        seq[index] = 2 * value

things = [2, 5, 'spam', 9.5]
double(things)
print(things)
```



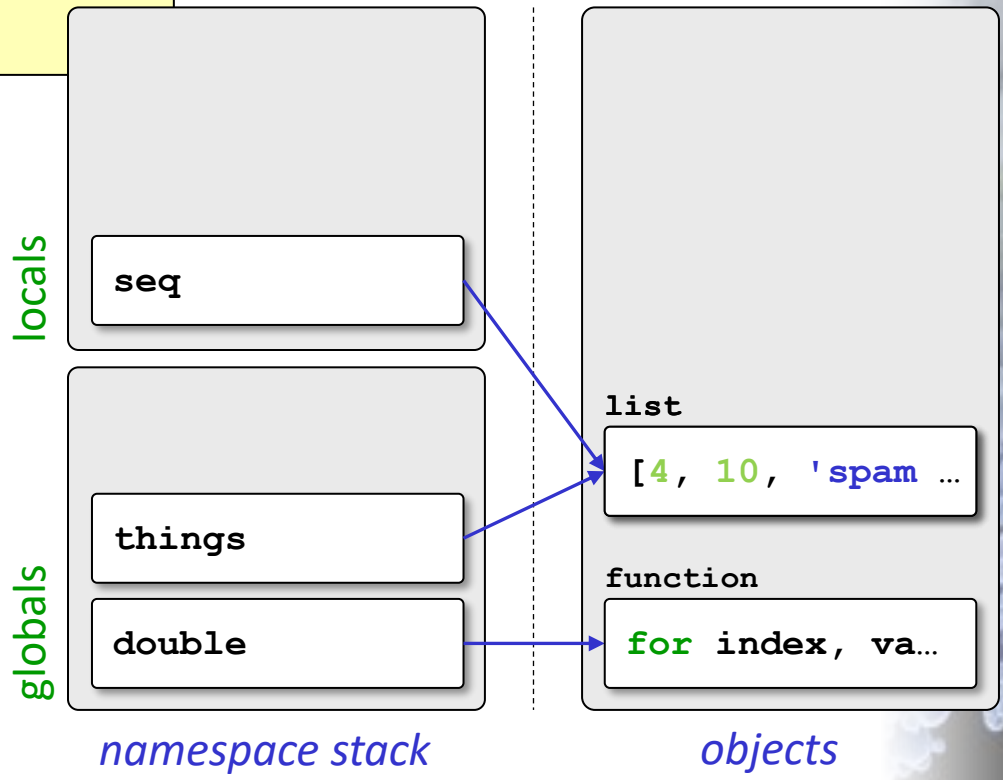
UNIVERSITEIT GENT

Lists as arguments

double.py

```
def double (seq):
    for index, value in enumerate(seq):
        seq[index] = 2 * value

things = [2, 5, 'spam', 9.5]
double(things)
print(things)
```



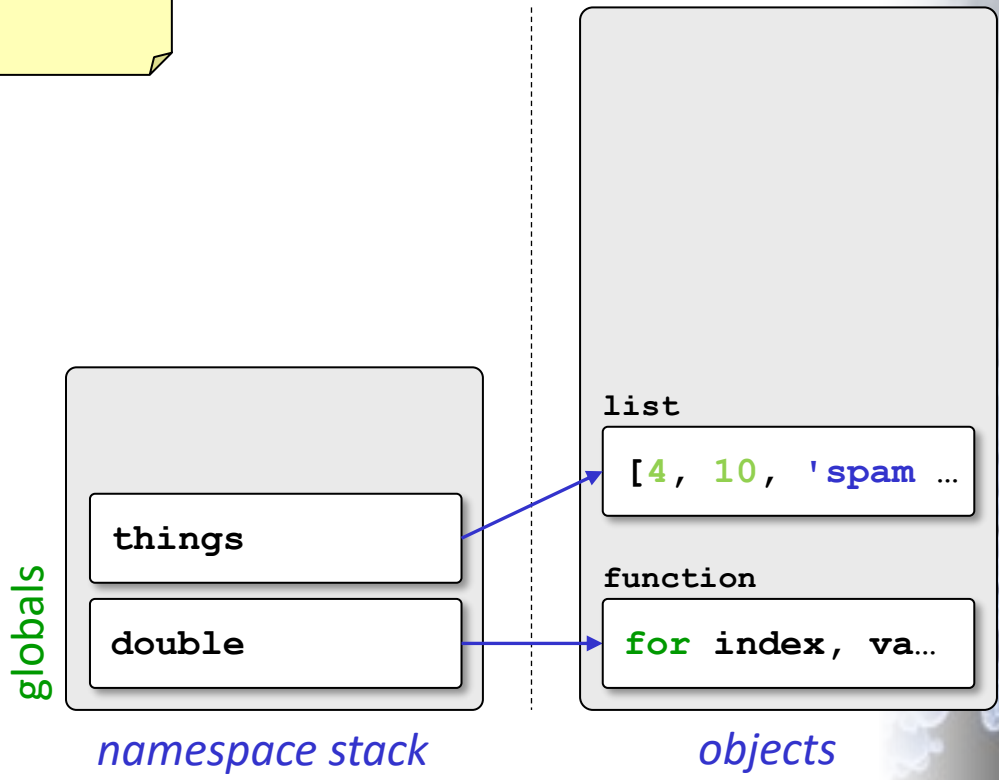
UNIVERSITEIT GENT

Lists as arguments

double.py

```
def double(seq):
    for index, value in enumerate(seq):
        seq[index] = 2 * value

things = [2, 5, 'spam', 9.5]
double(things)
print(things)
```



UNIVERSITEIT GENT

Hermit crabs

Nested lists

Nested lists

- **nested list**: list that appears as an element in another list
  - *bracket operators* evaluate from left to right

```
>>> gases = [['He', 2], ['Ne', 10], ['Ar', 18]]
>>> gas = gases[2]
>>> gas
['Ar', 18]
>>> gas[0]
'Ar'
>>> gases[2][0]
'Ar'
```

UNIVERSITEIT GENT

Matrices

- **nested list**: list that appears as an element in another list
  - *bracket operators* evaluate from left to right
  - often used to represent matrices

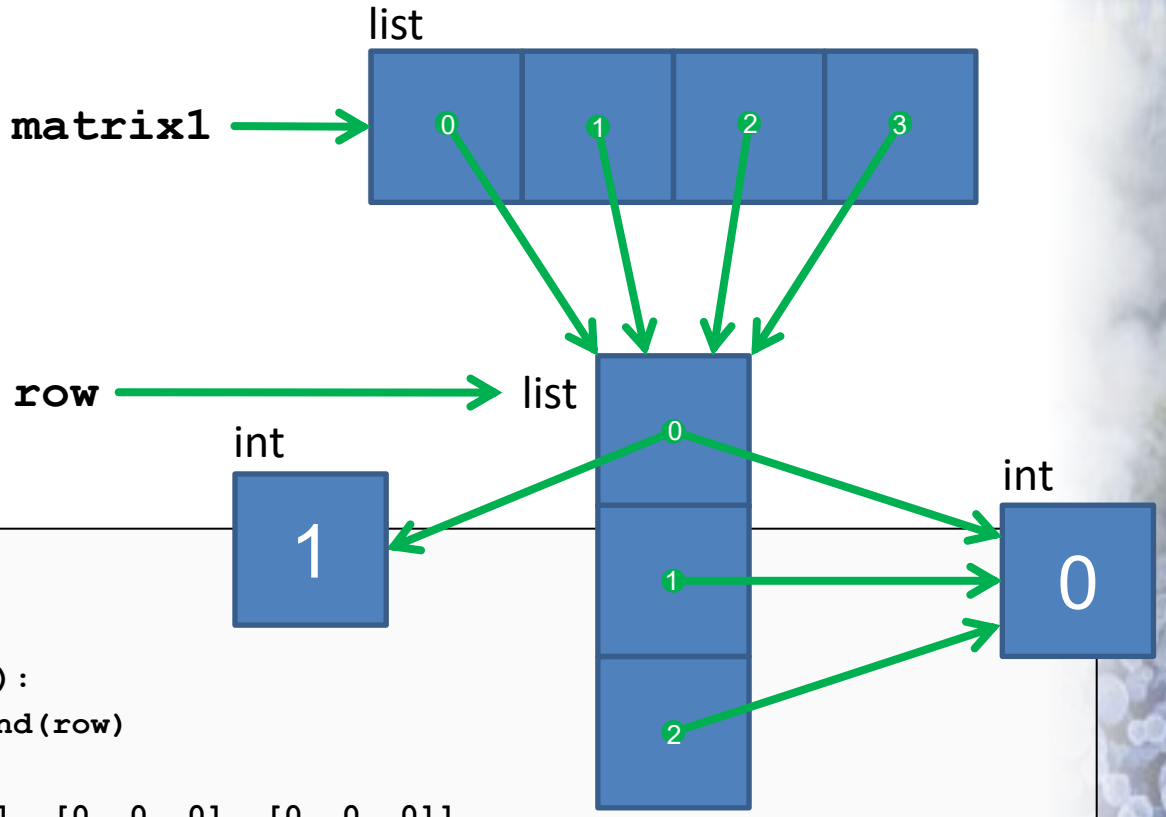
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> row = matrix[1]
>>> print(row)
[4, 5, 6]
>>> print(row[2])
6
>>> print(matrix[1][2])
6
```

UNIVERSITEIT GENT

Matrices

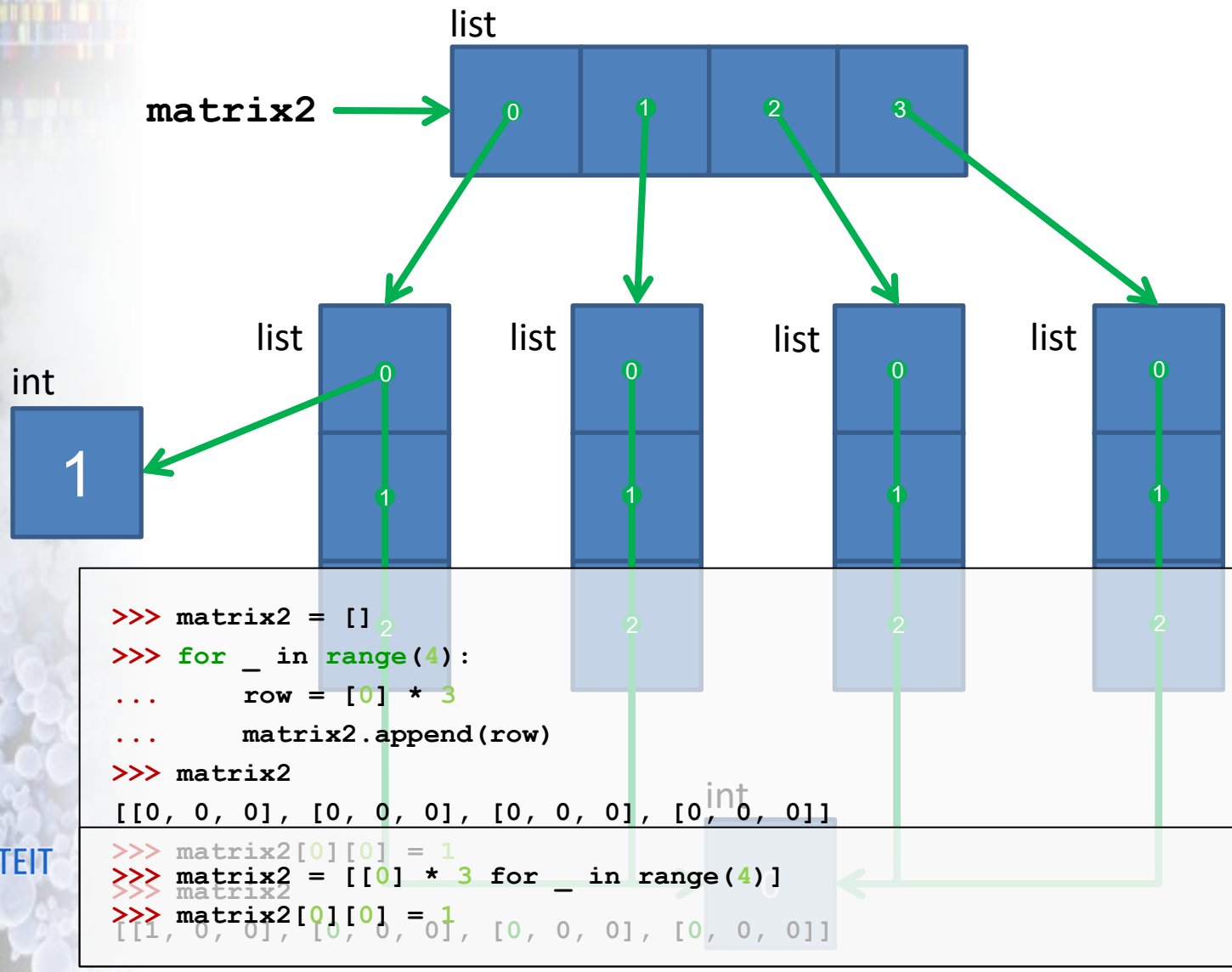
- **watch out**: lists are mutable




```
>>> row = [0] * 3
>>> matrix1 = []
>>> for _ in range(4):
...     matrix1.append(row)
>>> matrix1
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> matrix1[0][0] = 1
>>> matrix1
[[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]]
```

UNIVERSITEIT GENT

Matrices




```
>>> matrix2 = []
>>> for _ in range(4):
...     row = [0] * 3
...     matrix2.append(row)
>>> matrix2
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

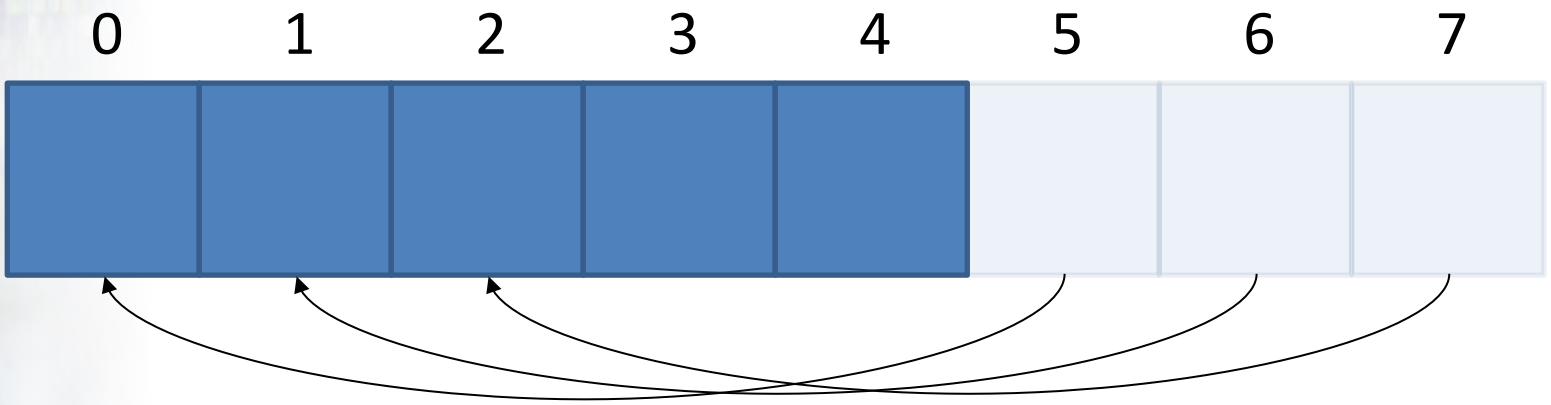
```
>>> matrix2[0][0] = 1
>>> matrix2 = [[0] * 3 for _ in range(4)]
>>> matrix2
>>> matrix2[0][0] = 1
[[1, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

int

UNIVERSITEIT GENT

Circular lists

Circular lists




```
>>> names = ['spam', 'eggs', 'bacon', 'shrubbery', 'ni']
>>> names[6]
Traceback (most recent call last):
IndexError: list index out of range
>>> names[6 % len(names)]
'eggs'
>>> names[-18 % len(names)]
'bacon'
```

UNIVERSITEIT GENT

Strings and lists

- built-in function **`list()`**
  - takes a sequence as its argument
  - creates a list from its elements and returns it
- built-in function **`str()`**
  - converts an object (*argument*) into a string (*return value*)

```
>>> list('nitrogen')
['n', 'i', 't', 'r', 'o', 'g', 'e', 'n']
>>> str(5)
'5'
>>> str(None)
'None'
>>> str(list('nitrogen'))
"['n', 'i', 't', 'r', 'o', 'g', 'e', 'n']"
```

UNIVERSITEIT GENT

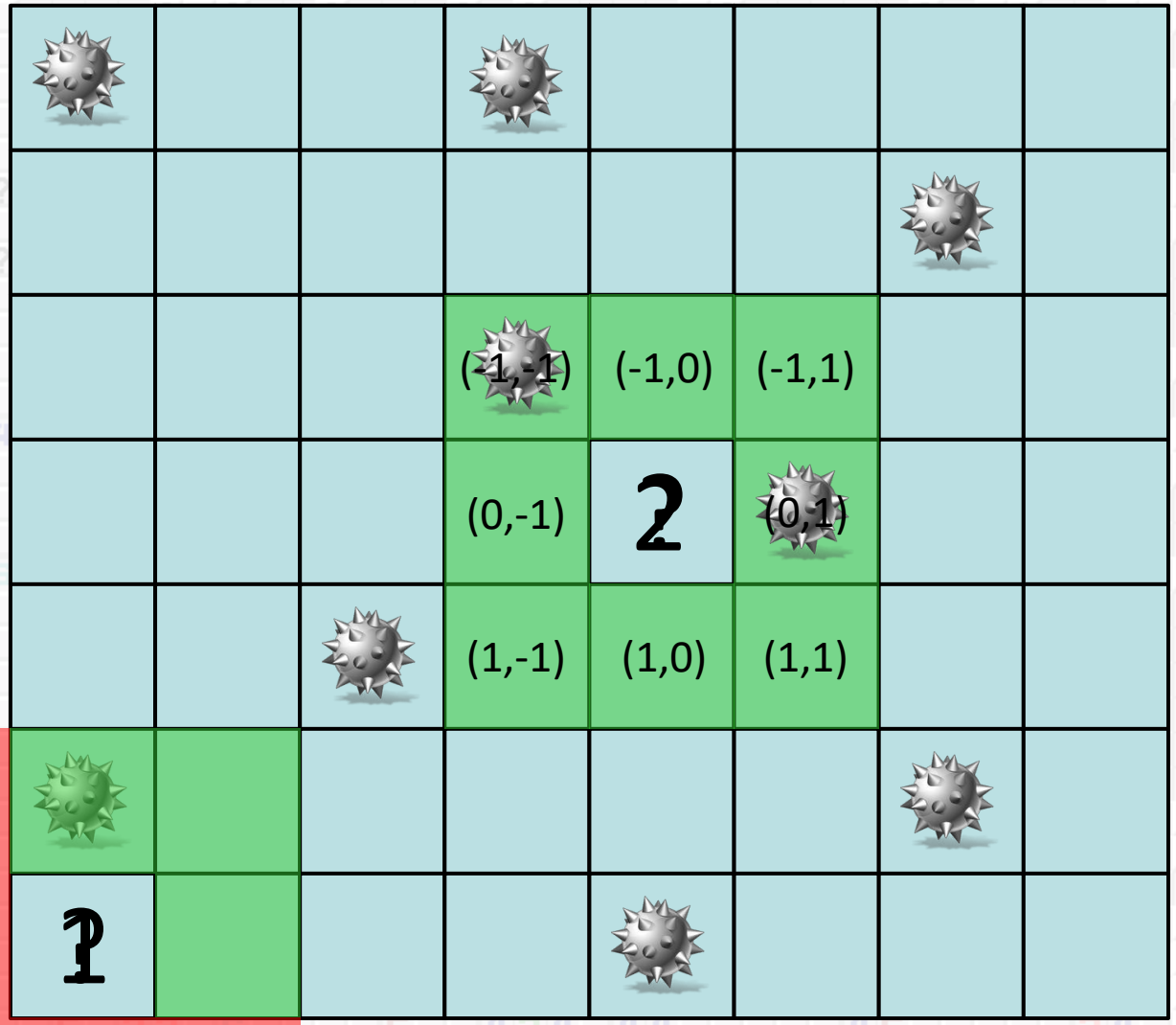
Strings and lists

- string method **`split()`**
  - string → list of strings (delimiter-based splitting)
  - by default, sequence of whitespace delimits fields
- string method **`join()`**
  - list of strings → string (joined together with delimiter)

```
>>> song = 'The rain in Spain...'
>>> song.split()
['The', 'rain', 'in', 'Spain...']
>>> song.split('ai')
['The r', 'n in Sp', 'n...']
>>> ' '.join(['The', 'rain', 'in', 'Spain...'])
'The rain in Spain...'
>>> 'ai'.join(['The r', 'n in Sp', 'n...'])
'The rain in Spain...'
```

UNIVERSITEIT GENT

Minesweeper



Homework (next lecture)

- course book
  - read chapter 8 (modules)
- read problem description of classroom exercises
  - Pub game
  - Location filter
  - Lunar phases

UNIVI
GENT

Homework (hands-on sessions)

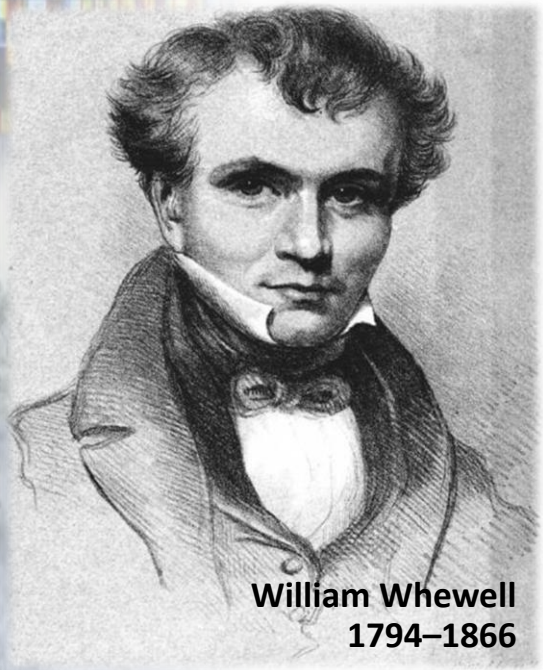
- Python help
  - read through list methods: help(list)
- hands-on session next week
  - Monday: finish mandatory exercises (series 05)

UNIVI
GENT

Remarks or questions?

UNIVERSITEIT GENT

The sky is the limit …



William Whewell
1794–1866

UNIVERSITEIT GENT

"Every failure is a step towards success."

— William Whewell