Prof. Dr. Gemma Roig
M.Sc. Alperen Kantarcı
M.Sc. Gamze Akyol

Programmieren für Studierende der Naturwissenschaften

Lecture 4 – Aggregated Data Types and Functions

Contents

- L1: Basics of programming
  P1: Exercise 1 and help to installments.

- L2: Elementary data types and control structures
  P2: Exercises

- L3: Aggregated data types
  P3: Exercises

- L4: Aggregated data types and functions
  P4: Exercises

- L5:Testing,error messages and self-help
  P5: Exercises

Review of Loops

The head-driven or pre-test loop:

- first the termination condition is checked before the loop body is run through (usually indicated by the keyword WHILE (=so long-until).

The foot-driven or re-checking loop (implementable with a trick inPython):

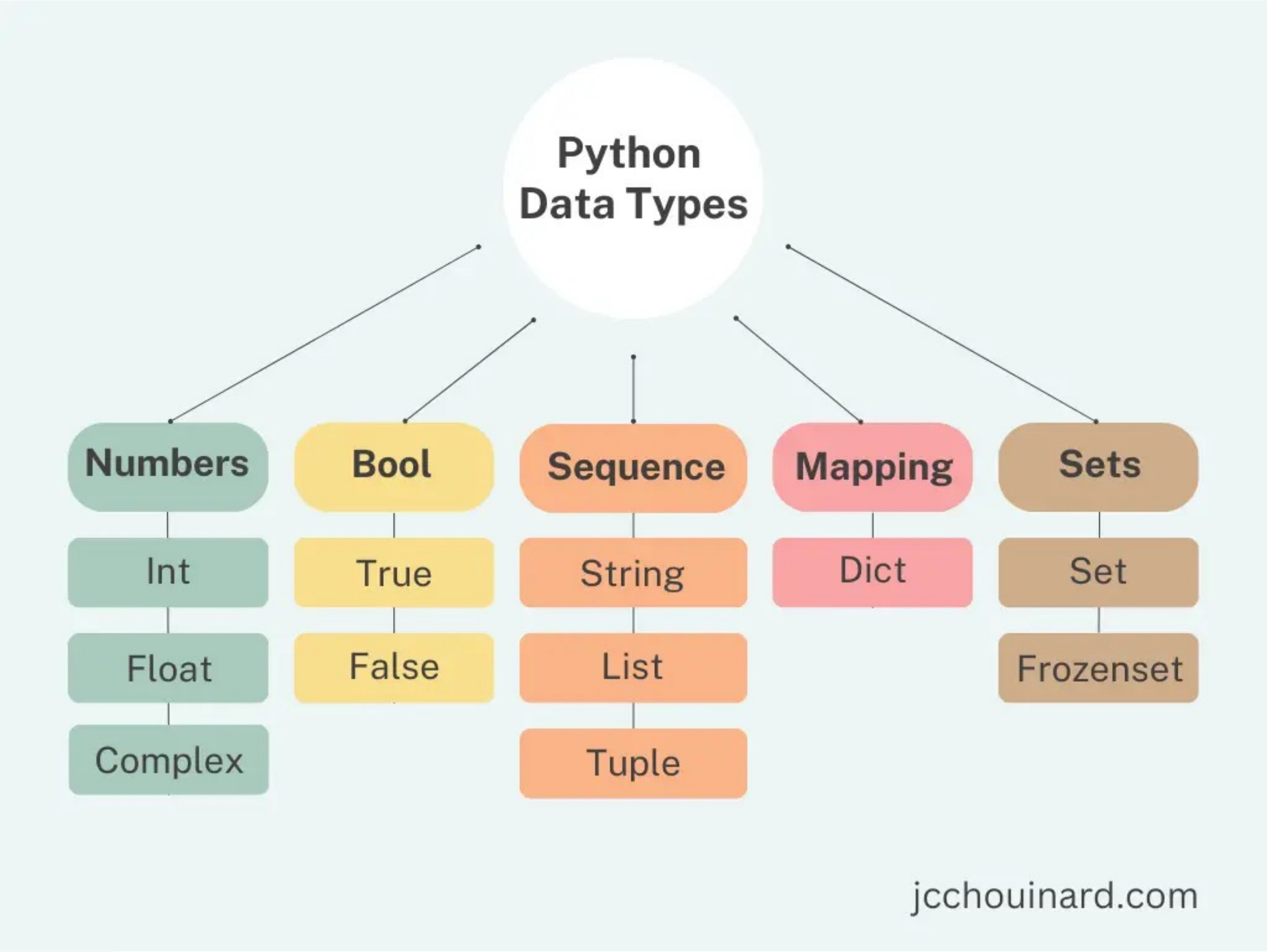
- only after the loop body has been run through, the termination condition is checked e.g. by a construct REPEAT-UNTIL (=repeat-to).

Thinking about problems

When you encounter with a problem:

- Decomposition into subproblems
- Formulate sub-problems linguistically (do not code it directlyfirst)
- Draw, visualize when the relationships seem complex to grasp
- Define essential objects and describe their properties, if necessary
- Do not think too detailed at the start. Start coarse and go into the details

Review of data types



jcchouinard.com

Set and Frozen Set

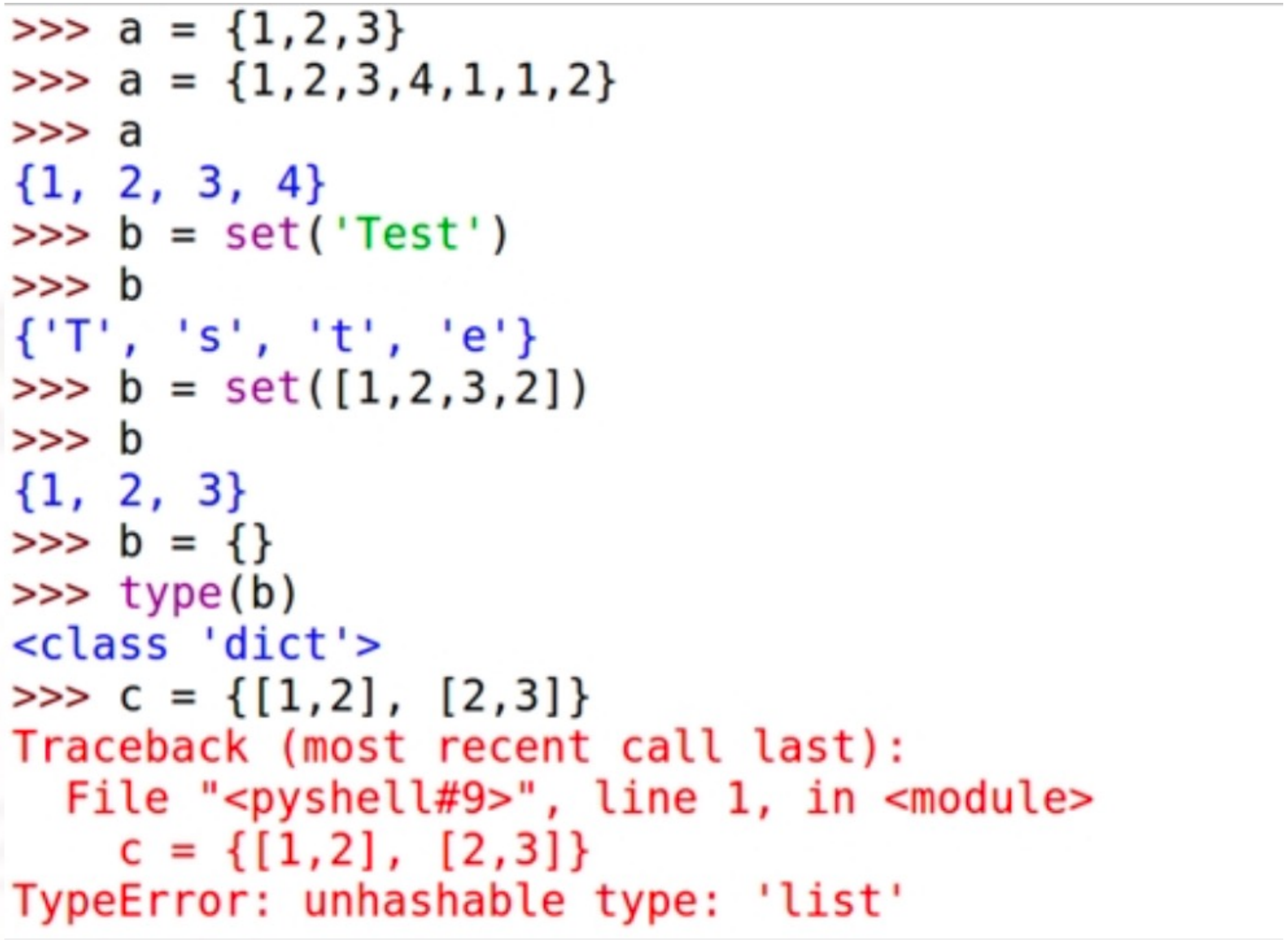
As in mathematics:

- The order is not important
- It is about ”being contained", therefore no duplicates

- Main benefits:
  - Test for membership, remove duplicates
  - Calculate classical set operations such as intersection or difference between sets
- Create quantities: By converting from an iterable data type (e.g. string or list).
- Elements must be immutable and comparable
- Attention! The empty set cannot be created by {}! We use set() for empty set.

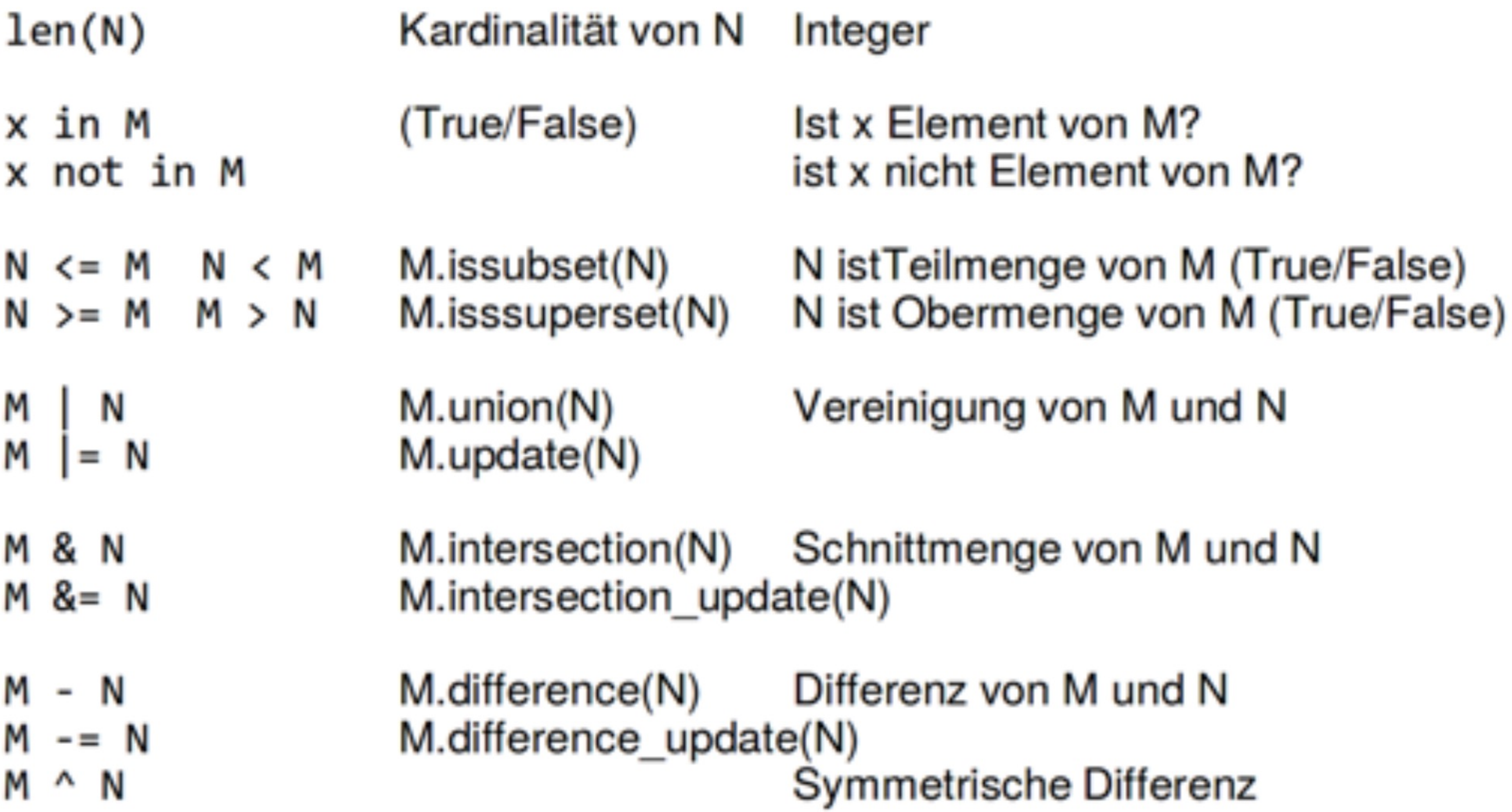
That would be a dictionary.

Set and Frozen Set



```
>>> a = {1,2,3}
>>> a = {1,2,3,4,1,1,2}
>>> a
{1, 2, 3, 4}
>>> b = set('Test')
>>> b
{'T', 's', 't', 'e'}
>>> b = set([1,2,3,2])
>>> b
{1, 2, 3}
>>> b = {}
>>> type(b)
<class 'dict'>
>>> c = {[1,2], [2,3]}
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    c = {[1,2], [2,3]}
TypeError: unhashable type: 'list'
```

Operators



len(N)	Kardinalität von N	Integer
x in M	(True/False)	Ist x Element von M?
x not in M		ist x nicht Element von M?
N <= M N < M	M.issubset(N)	N istTeilmenge von M (True/False)
N >= M M > N	M.isssuperset(N)	N ist Obermenge von M (True/False)
M | N	M.union(N)	Vereinigung von M und N
M |= N	M.update(N)	
M & N	M.intersection(N)	Schnittmenge von M und N
M &= N	M.intersection_update(N)	
M - N	M.difference(N)	Differenz von M und N
M -= N	M.difference_update(N)	
M ^ N		Symmetrische Differenz

Quantities

- m.add(x)
  Add an element x to the set M. (Has no effect if x is already an element of M)

- m.clear()
  Empties the set M

- m.pop()
  Removes an element from the set M

- m.remove(x)
  Remove element x from the set M (x must be an element of M, otherwise keyerror)

Watch Python documentary and try it out!

Dictionary

- A bit different from the other aggregated data types
- Dictionaries implement partial functions. For this one uses 2-tuples (pairs) of the form (key,value), written for example as {key:value}
- Since the value can again be an n-tuple, but also a list, etc., arbitrary partial functions are implementable.
- In contrast to sequences, which are indexed by a number interval, dictionaries are indexed by keys.
- The keys must have some immutable type. So strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, tuples, frozensets

Dictionary

- A pair of braces{} creates an empty dictionary

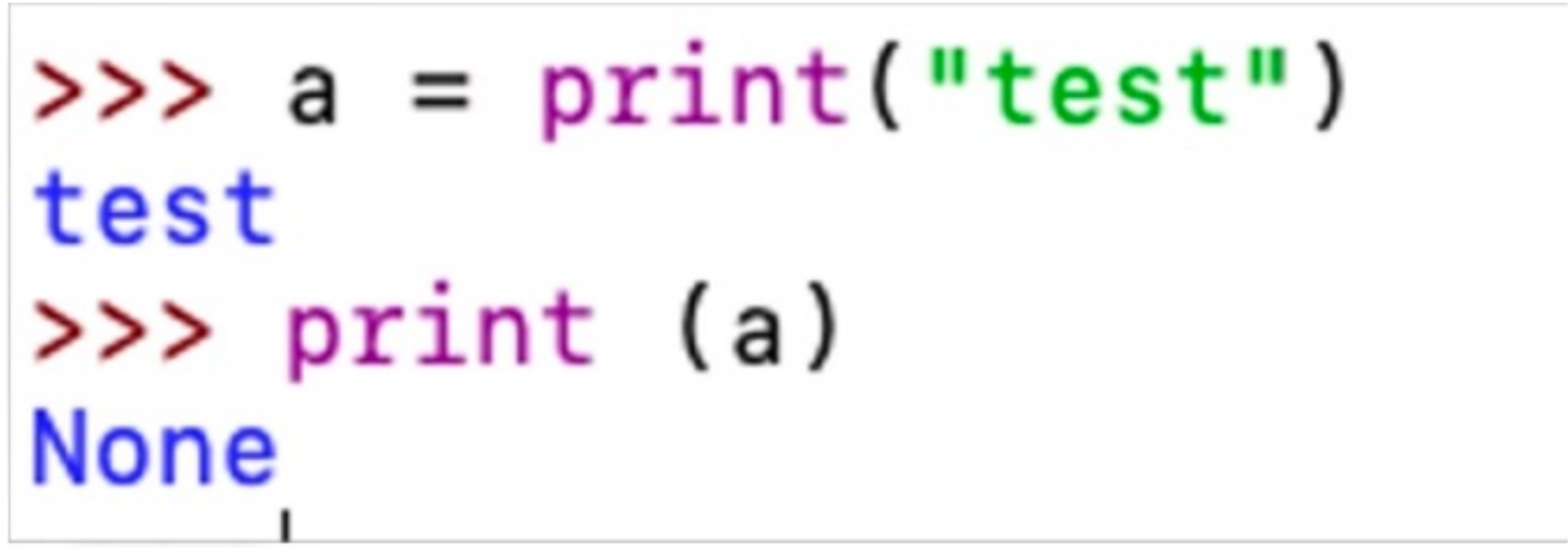
- A comma-separated sequence of (key:value) pairs inside the parentheses inserts the initial pairs into the dictionary.

- Main operations on a dictionary are:
  - Saving a value under a key and retrieving this value when the key is specified
  - It is also possible to delete a (key:value) pair with delitem ()
  - If a key that already exists is used when saving, the old key associated with it is forgotten.
  - It generates an error to retrieve a value with a non-existent key.

Data type None

- The None data type has only one value in Python:
  - The constant None
  - None is a keyword. It serves as a placeholder for variables that actually have no value (or not yet known)

- Functions that do not return a value implicitly have None as return value
  - When the interactive interpreter evaluates an expression, it outputs only if the return value is not None
  - Testing in the console



```
>>> a = print("test")
test
>>> print (a)
None
```

Functions

Functions

- Why are the control structures that we know so far not sufficient?

- Enable functions:
  - a better structuring of programs
  - Modularization (many components with defined interfaces)
  - Re-use of code parts externally and internally
  - Increasing the efficiency of the compilations

- Attention: In the context of this event, the differences between functions and methods are not considered in more detail!

- Students of computer science learn about these differences in the context of "object-oriented programming"

Functions - Subroutines

- A sequence of instructions is combined under one name

- Arguments(so-called parameters) can be passed to this sequence and, if necessary, a value or values can also be returned

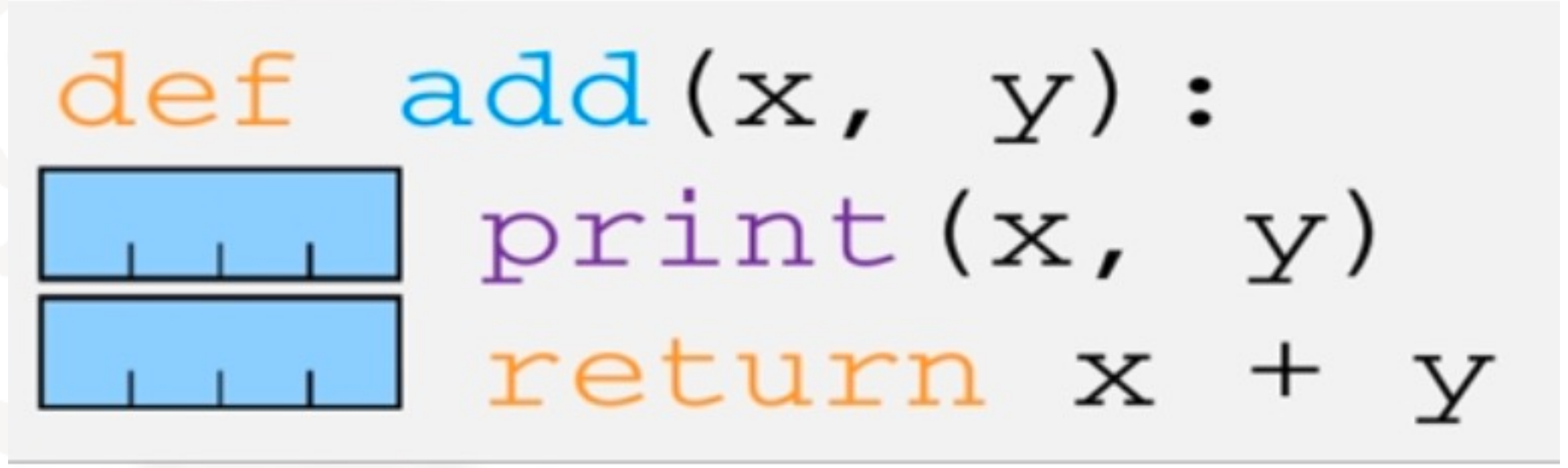
- The parameters are usually specified by order, type and number and/or by names

- **The good news is that** you already know how to call something like this!

Functions

- Functions are defined with the def statement, parameters in round brackets directly behind.

- The functional body must be indented.
- The end of the function definition is indicated by undoing the indentation.
- Return is the keyword that causes the value to be assigned to the function value and the function to terminate.
- A return is optional. But be aware!
- If no value is specified or if the return statement is omitted, the object None is returned.



```
def add(x, y):
    print(x, y)
    return x + y
```

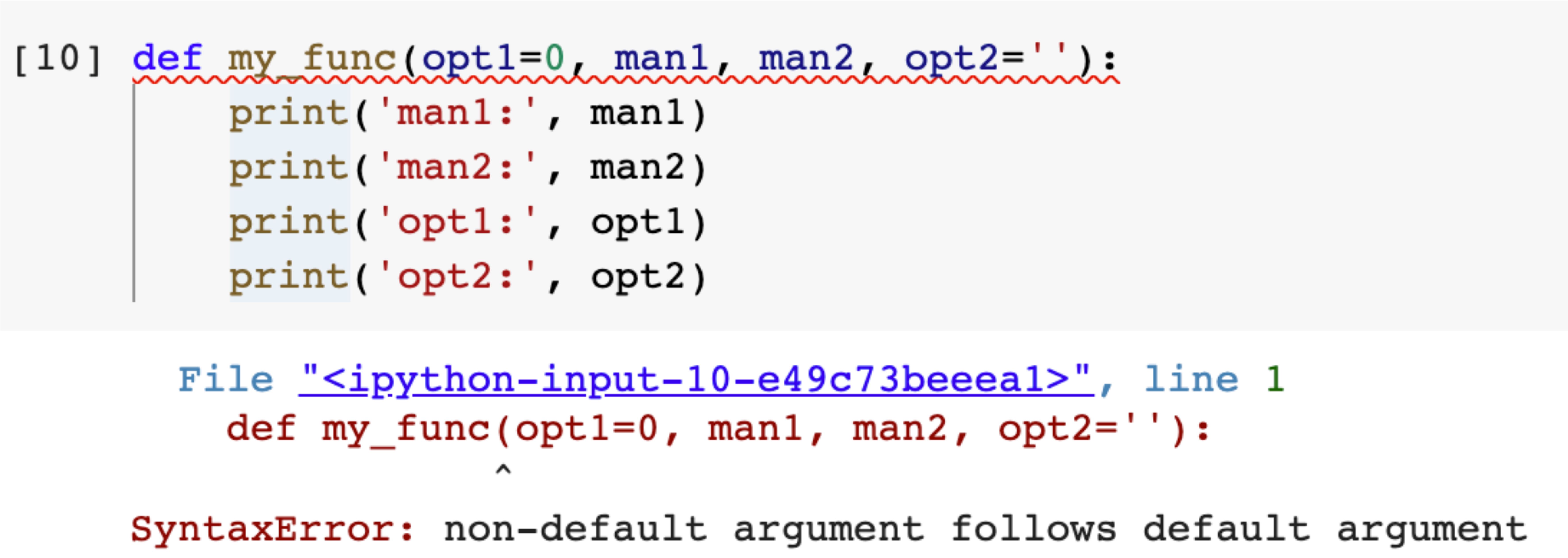
Functions - Calls

- The function definition must be made in the program text (lexically) before the call (only then the name of the function is known)

- All arguments in the function definition must be specified concretely and in the correct order when called

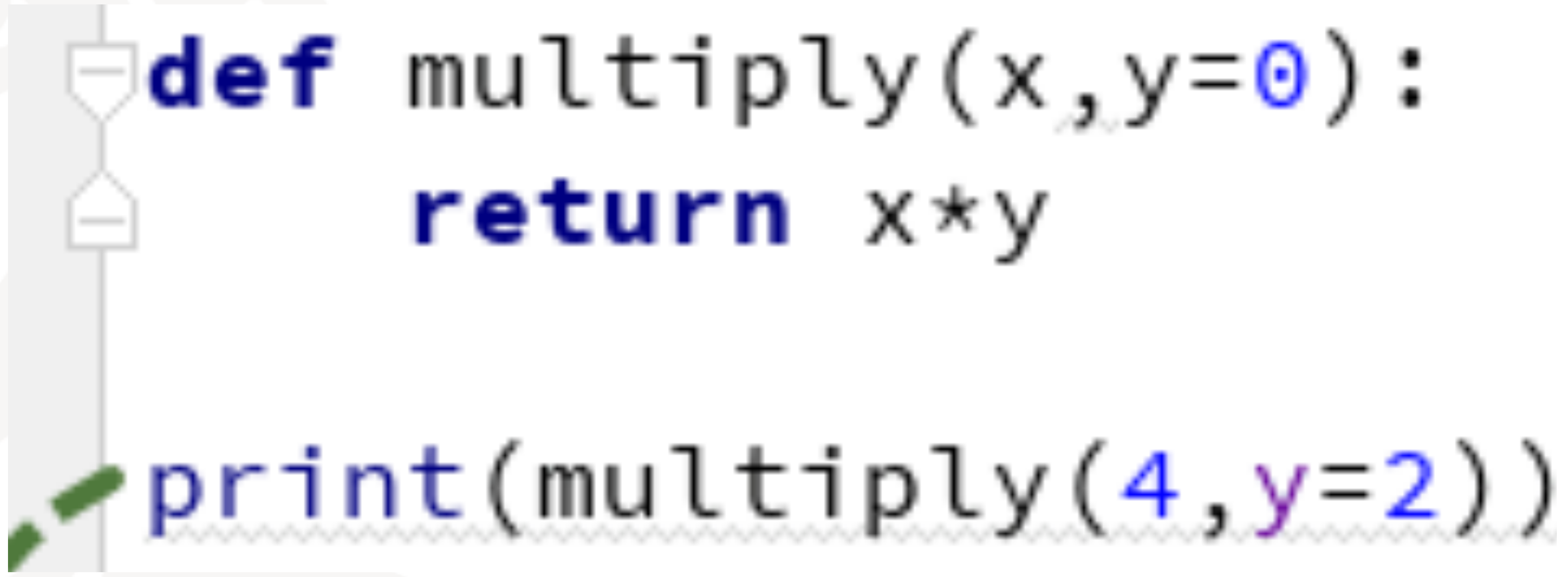
- Default-parameters

Functions - Calls



```
[10] def my_func(opt1=0, man1, man2, opt2=''):
         print('man1:', man1)
         print('man2:', man2)
         print('opt1:', opt1)
         print('opt2:', opt2)

  File "<ipython-input-10-e49c73beeea1>", line 1
    def my_func(opt1=0, man1, man2, opt2=''):
                       ^
SyntaxError: non-default argument follows default argument
```



```
def multiply(x,y=0):
    return x*y

print(multiply(4,y=2))
```

Functions - Calls

All positional arguments

```
foo(3, 4, 5)
```

✓

Positional argument(s) followed by keyword argument(s)

```
foo(3, b=4, c=5)
foo(3, 4, c=5)
```

✓

All keyword arguments

```
foo(a=3, b=4, c=5)
```

✓

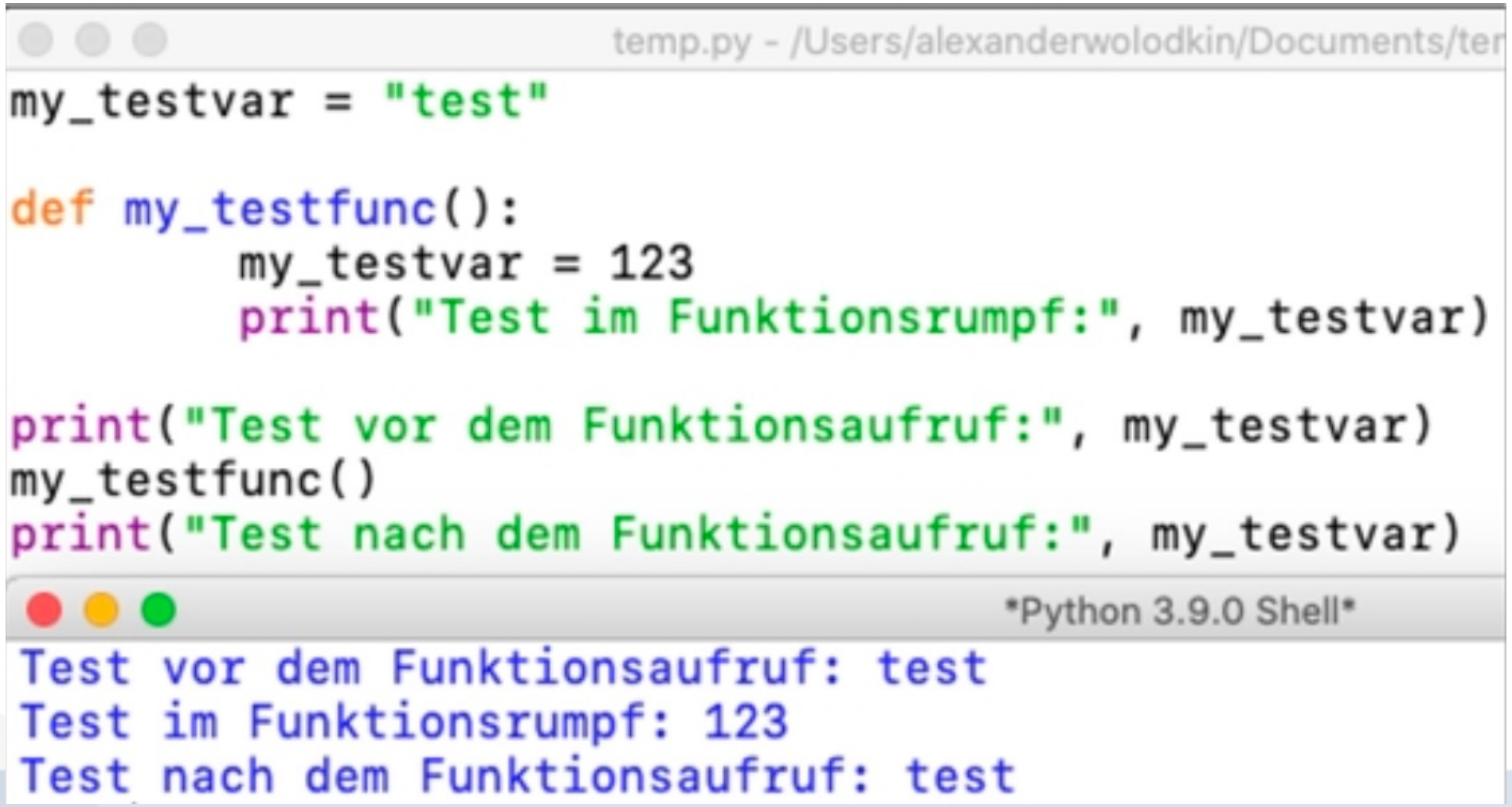
Keyword argument(s) followed by positional argument(s)

```
foo(a=3, b=4, 5)
foo(a=3, 4, c=5)
foo(3, b=4, c)
```

✗

Namespaces

- All elements that we use or reserve are in a namespace.
- Functions form their own namespaces
- Modules (see upcoming topics) can also form their own namespaces
- Accesses to elements in different namespaces are not immediately intuitively understandable. Consider the given example



```
my_testvar = "test"

def my_testfunc():
        my_testvar = 123
        print("Test im Funktionsrumpf:", my_testvar)

print("Test vor dem Funktionsaufruf:", my_testvar)
my_testfunc()
print("Test nach dem Funktionsaufruf:", my_testvar)
```

```
Test vor dem Funktionsaufruf: test
Test im Funktionsrumpf: 123
Test nach dem Funktionsaufruf: test
```