Ghent University
Faculty of Sciences

Python Programming

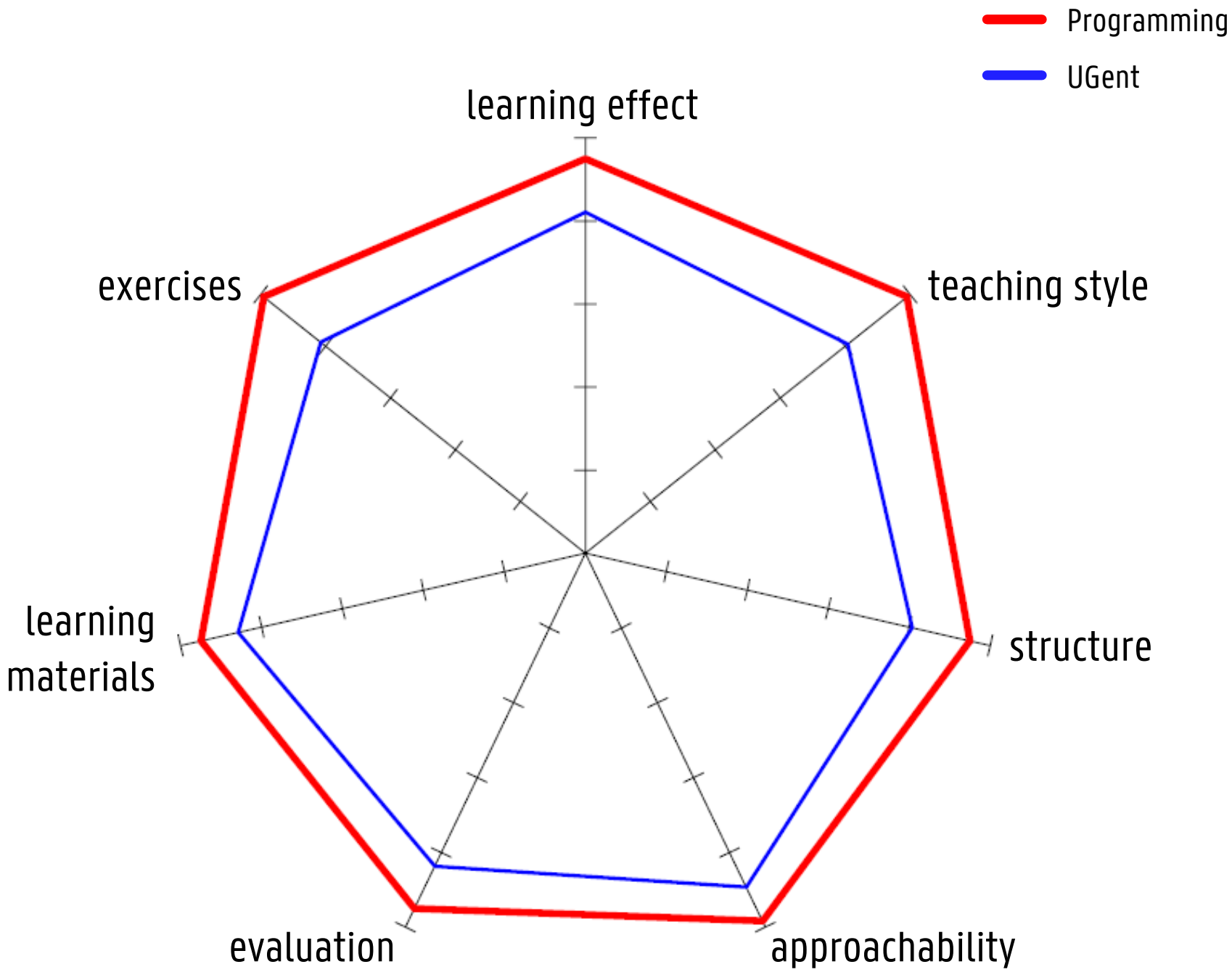
Prof. Dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


Ooooh! The last bug!


Programming
UGent
learning effect
teaching style
structure
approachability
evaluation
learning materials
exercises

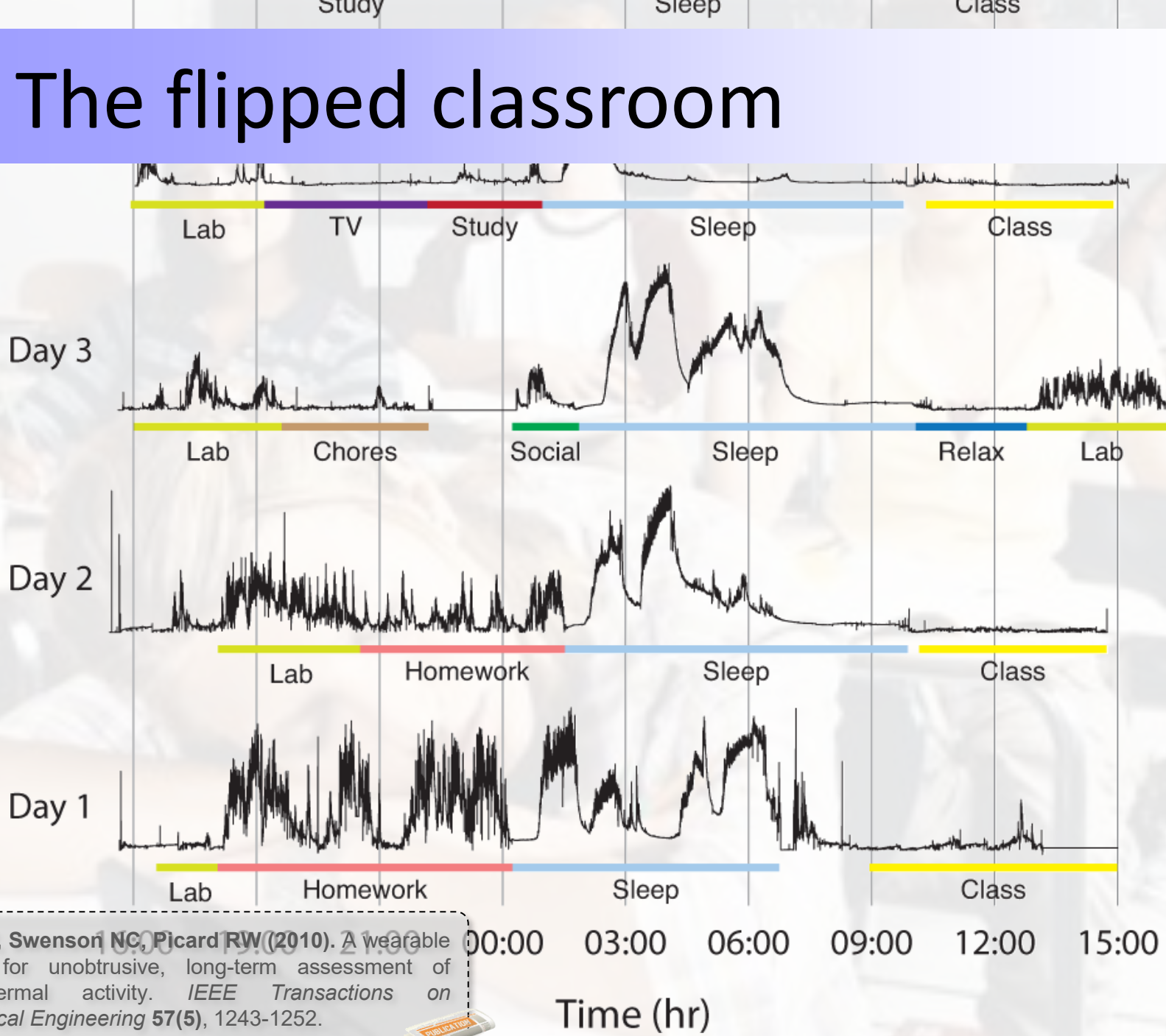
The flipped classroom

traditional classroom (ideal world)

lecture | study | lecture | study | lecture | study | study | exam

UNIVERSITEIT GENT

The flipped classroom



Poh MZ, Swenson NC, Picard RW (2010). A wearable sensor for unobtrusive, long-term assessment of electrodermal activity. *IEEE Transactions on Biomedical Engineering* **57(5)**, 1243-1252.

The flipped classroom

traditional classroom (ideal world)

lecture | study | lecture | study | lecture | study | study | exam

UNIVERSITEIT GENT

The flipped classroom

traditional classroom (real world)

lecture

lecture

lecture

study

exam

flipped classroom

practice

study

lecture

practice

study

lecture

practice

study

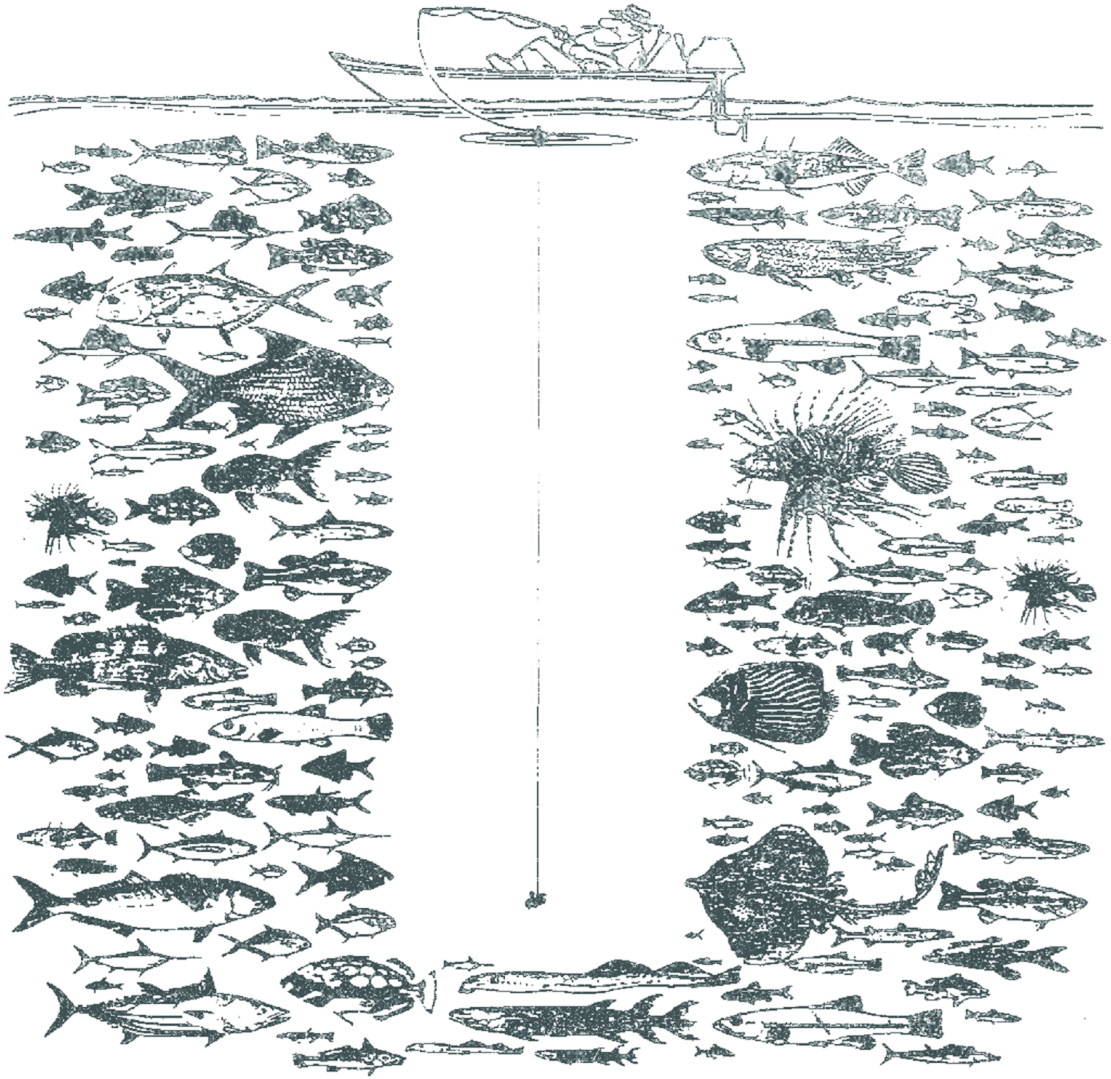
lecture

active learning

study

exam

Programming = testing software

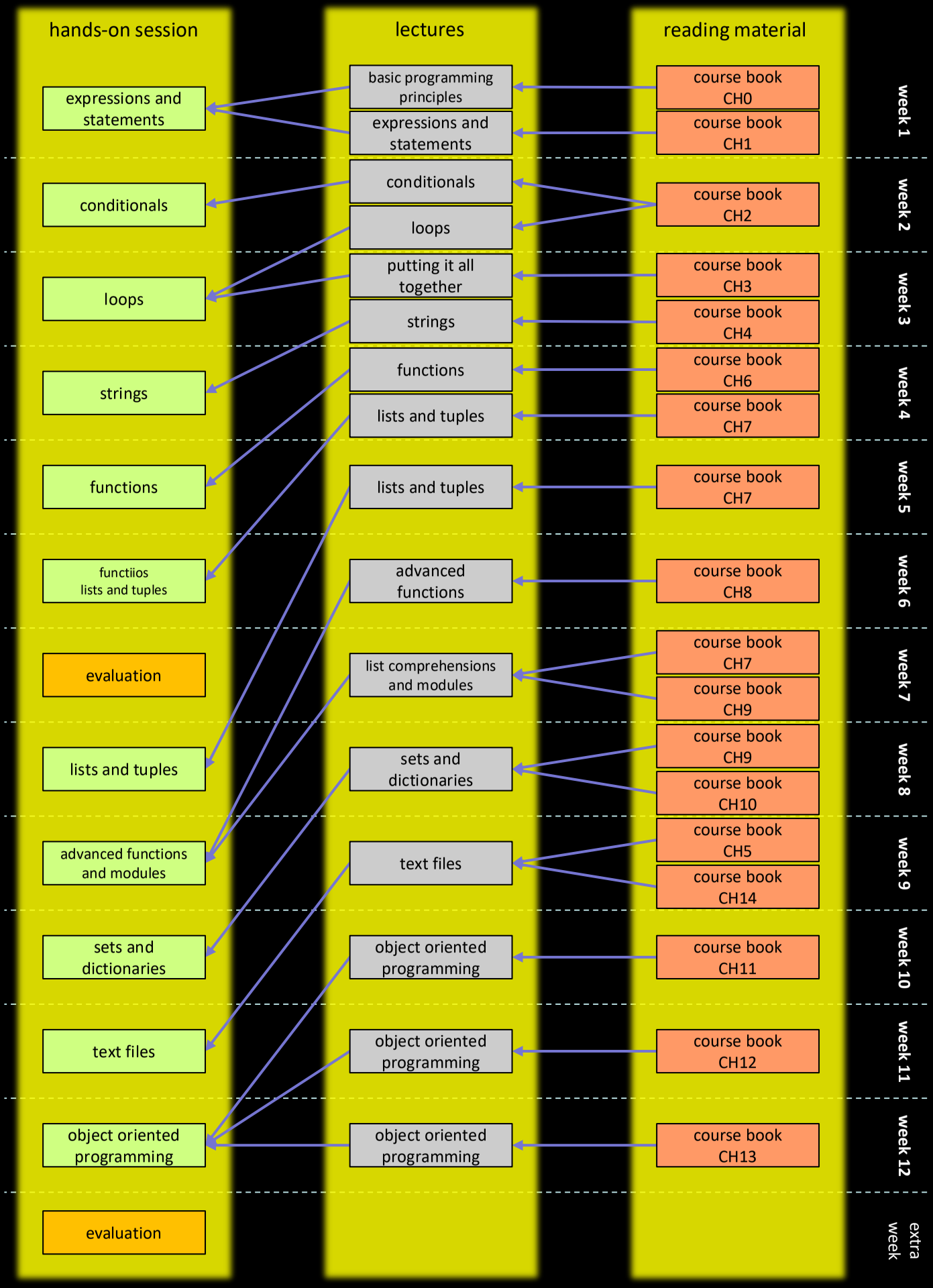


UNIVERSITEIT GENT

Evaluation

- mandatory exercises
  - evaluation series = 5 series of 6 mandatory exercises
  - push students to bring theory into practice
  - weakly deadlines on Tuesdays at 22:00
  - submit to Dodona before set deadlines (see overview)
  - gradually build up towards exam level exercises
- permanent evaluations
  - 2 evaluations in weeks 7 and 13 (during hands-on sessions)
  - per evaluation: solve 2 exercises within 2 hours
  - level: evaluation exercises are easier than exam exercises
  - tests basic programming skills
- periodic evaluation (exam)
  - exam = solve 3 exercises within 3.5 hours
  - level: exam exercises are more difficult than evaluation exercises

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

Evaluation

- mandatory exercises
  - evaluation series = 5 series of 6 mandatory exercises
  - push students to bring theory into practice
  - weakly deadlines on Tuesdays at 22:00
  - submit to Dodona before set deadlines (see overview)
  - gradually build up towards exam level exercises
- permanent evaluations
  - 2 evaluations in weeks 7 and 13 (during hands-on sessions)
  - per evaluation: solve 2 exercises within 2 hours
  - level: evaluation exercises are easier than exam exercises
  - tests basic programming skills
- periodic evaluation (exam)
  - exam = solve 3 exercises within 3.5 hours
  - level: exam exercises are more difficult than evaluation exercises

UNIVERSITEIT GENT

Practice, practice, practice

Evaluation

- evaluation score accounts for 4/20 on total score
  - triple profit !!!
  - cannot be retaken during second examination period
- other exercises
  - non-committal (use them e.g. in preparation for the exams)
  - sample exam
  - learn to program = practice, practice, practice, …

UNIVERSITEIT GENT

Permanent evaluation score

mandatory exercises

c : number of mandatory exercises submitted correctly before the weekly deadlines

a : number of mandatory exercises

evaluation exercises

s: score obtained for evaluation exercises

UNIVERSITEIT GENT

$$\text{finale score} = s \times \frac{c}{a}$$

Permanent evaluation score

mandatory exercises

c : number of mandatory exercises submitted correctly before the weekly deadlines

a : number of mandatory exercises

30/30

evaluation exercises

s: score obtained for evaluation exercises

16/20

$$\text{finale score} = s \times \frac{c}{a} = 16 \times \frac{30}{30} = 16$$

UNIVERSITEIT GENT

Permanent evaluation score

mandatory exercises

c : number of mandatory exercises submitted correctly before the weekly deadlines

a : number of mandatory exercises

18/30

evaluation exercises

s: score obtained for evaluation exercises

16/20

UNIVERSITEIT GENT

$$\text{finale score} = s \times \frac{c}{a} = 16 \times \frac{18}{30} = 9.6$$

Values and data types

Values and data types

- **object**: one of the fundamental things (like a letter or a number) that a computer program manipulates
  - strings: `'Hello, World!'`
  - integers: `17`
  - floats: `3.2`

```
>>> print(4)
4
>>> print(2.7)
2.7
```

UNIVERSITEIT GENT

Values and data types

- **object**: one of the fundamental things (like a letter or number) that a computer program manipulates
  - strings: `'Hello, World!'`
  - integers: `17`
  - floats: `3.2`
- each object has a data type (*strongly typed*)

```
>>> type('Hello, World!')          # strings
<class 'str'>
>>> type(17)                       # integers
<class 'int'>
>>> type(3.2)                      # floating points
<class 'float'>
```

UNIVERSITEIT GENT

Values and data types

- **object**: one of the fundamental things (like a letter or number) that a computer program manipulates
  - strings: `'Hello, World!'`
  - integers: `17`
  - floats: `3.2`
- each object has a data type (*strongly typed*)

```
>>> type('17')
<class 'str'>
>>> type('3.2')
<class 'str'>
```

UNIVERSITEIT GENT

Values and data types

- strings are enclosed in single or double quotes
  - strings enclosed in double quotes can contain single quotes
  - strings enclosed in single quotes can contain double quotes

```
>>> type('This is a string.')
<class 'str'>
>>> type("And so is this.")
<class 'str'>
>>> print("Bruce's beard")
Bruce's beard
>>> print('The knights who say "Ni!"')
The knights who say "Ni!"
```

UNIVERSITEIT GENT

Values and data types

- it is illegal to use commas or dots between groups of three digits when typing large integers (American notation)
  - 1,000,000 is interpreted as a *tuple* containing 3 objects that need to be printed one after the other

```
>>> print(1,000,000)
1 0 0
>>> print('a', 'b', 'c')
a b c
>>> print(1.000.000)
  File "<stdin>", line 1
    print(1.000.000)
                  ^
SyntaxError: invalid syntax
```

UNIVERSITEIT GENT

Variables

- **variable**: name that points to an object
  - assignment statement
    - create new variable
    - assign object to variable
  - points to location in computer memory where object is stored
  - using name in expression fetches object from computer memory

```
>>> welcome = 'Hello, World!'
>>> print(welcome)
Hello, World!
```

computer memory (RAM)

UNIVERSITEIT GENT

Variables

- assignment operator =
  - used in assignment statement
  - should not be confused with equality test (==)
  - links an *object* (right-hand side) with a *name* (left-hand side)
    - lhs must always contain variable name
    - rhs must always contain expression that evaluates to object

```
>>> message = "What's up, Doc?"
>>> n = 17
>>> pi = 3.14159
>>> 17 = n
  File "<stdin>", line 1
SyntaxError: can't assign to literal
```

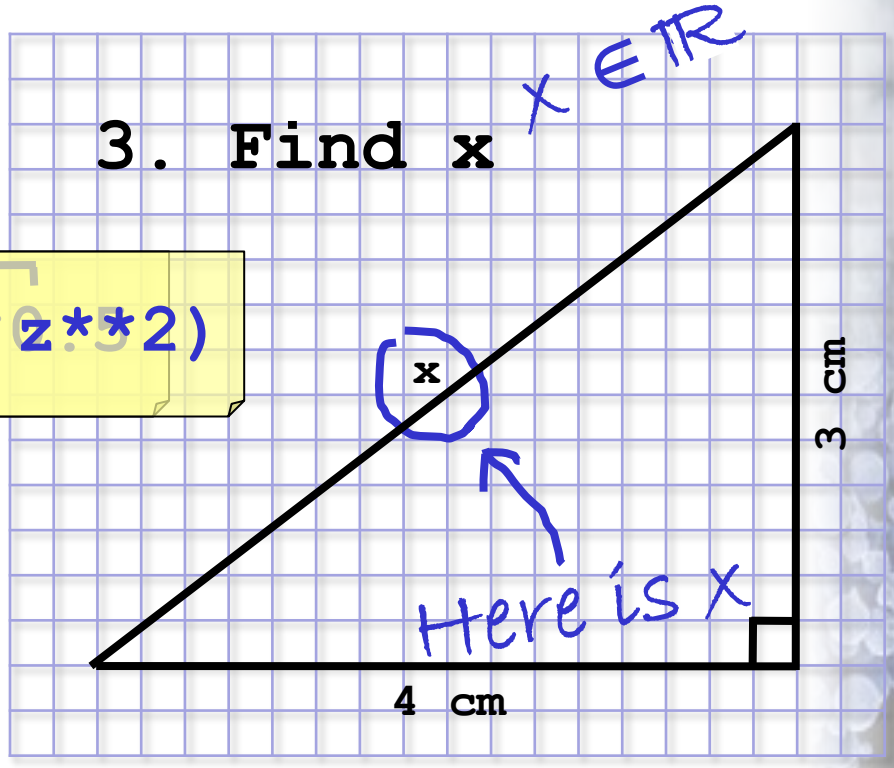
UNIVERSITEIT GENT

Variables

- **variable**: name that points to an object
  - name and object are two different things
  - data type is property of an object
    - data type can be inferred from variable name

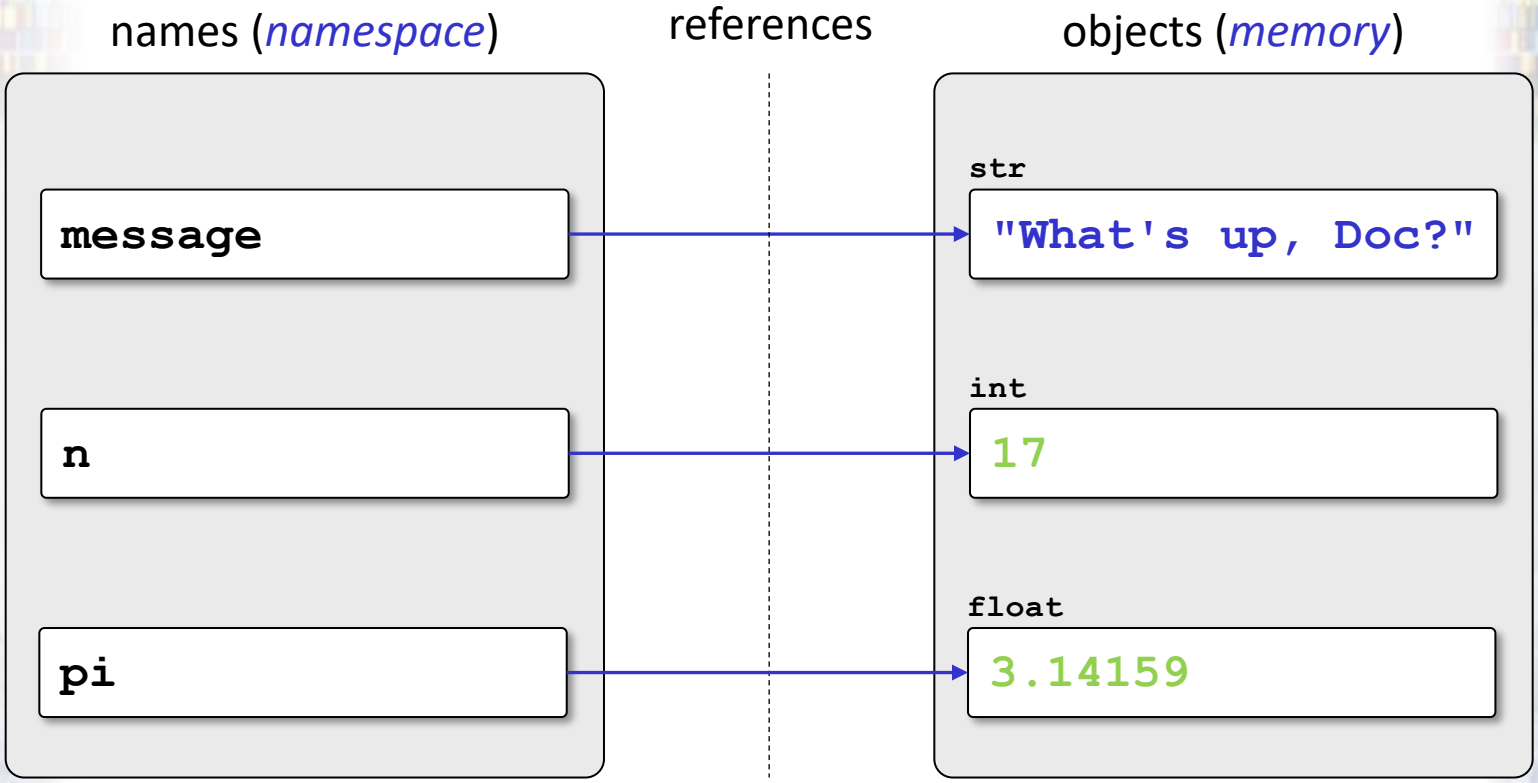
```
>>> type(message)
<class 'str'>
>>> type(n)
<class 'int'>
>>> type(pi)
<class 'float'>
```

```
x = math.sqrt(y**2 + z**2)
```



UNIVERSITEIT GENT

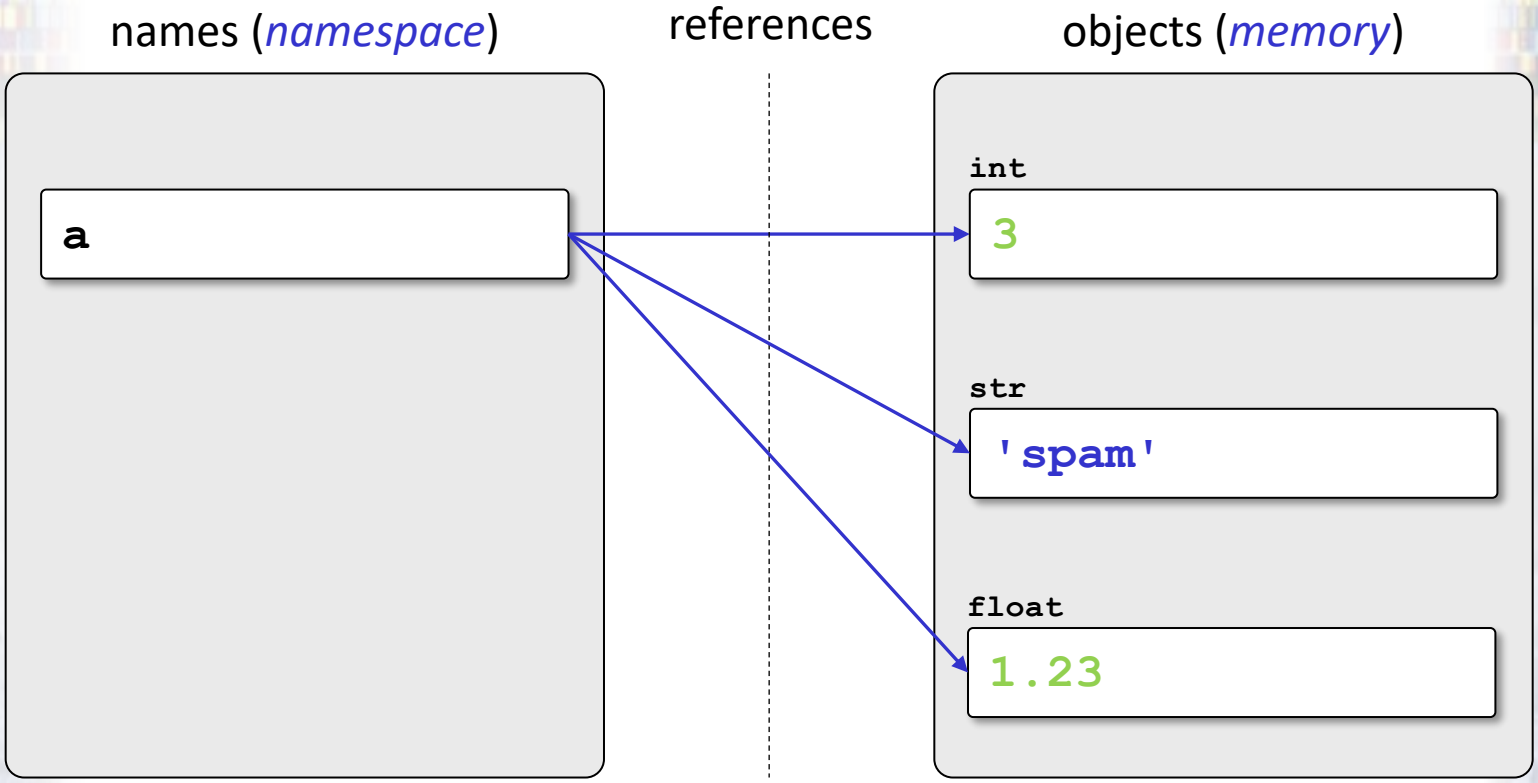
State diagram




```
>>> message = "What's up, Doc?"
>>> n = 17
>>> pi = 3.14159
```

UNIVERSITEIT GENT

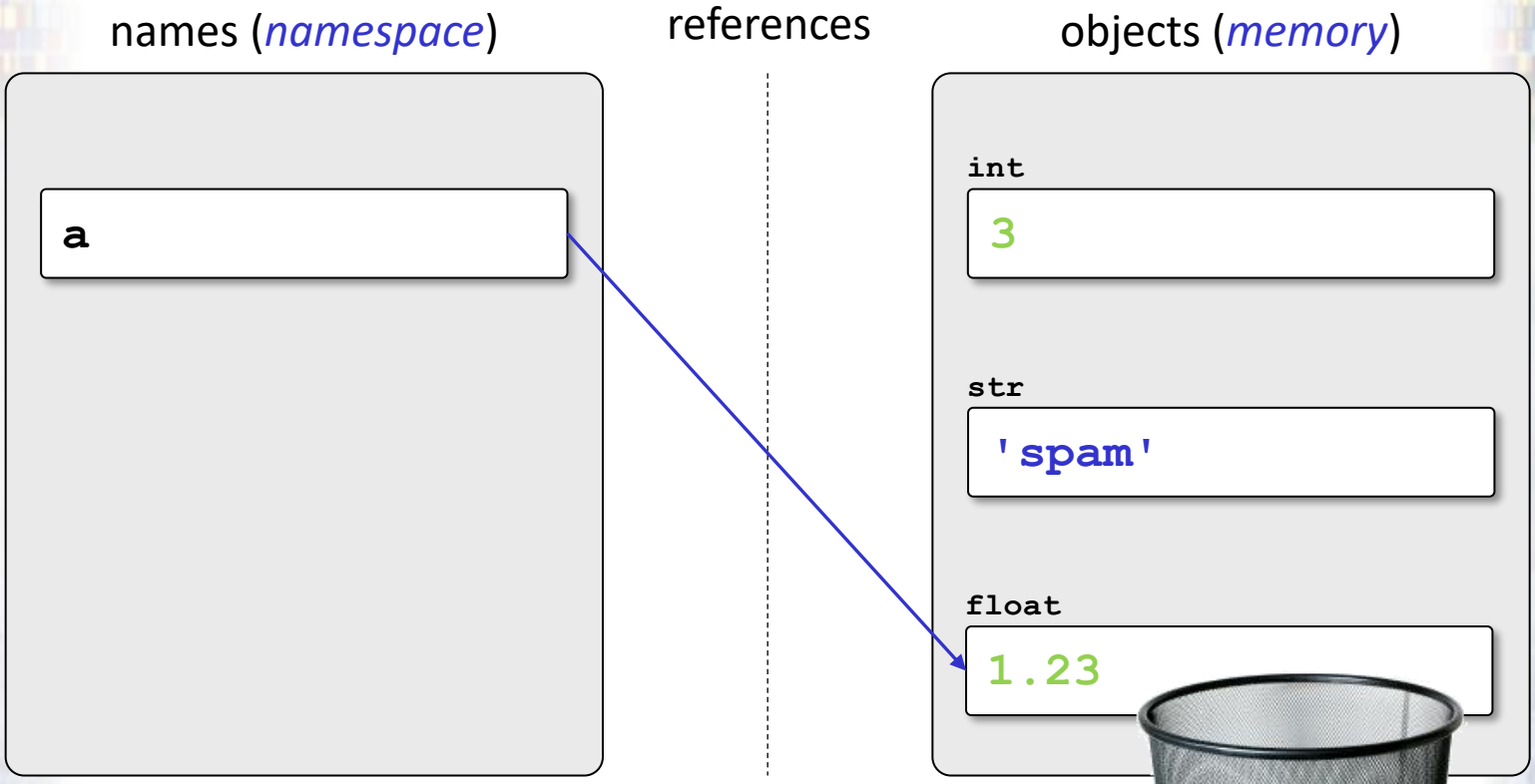
Dynamic typing




```
>>> a = 3          # a is an integer number
>>> a = 'spam'     # now it is a string
>>> a = 1.23       # and now a floating point number
```

UNIVERSITEIT GENT

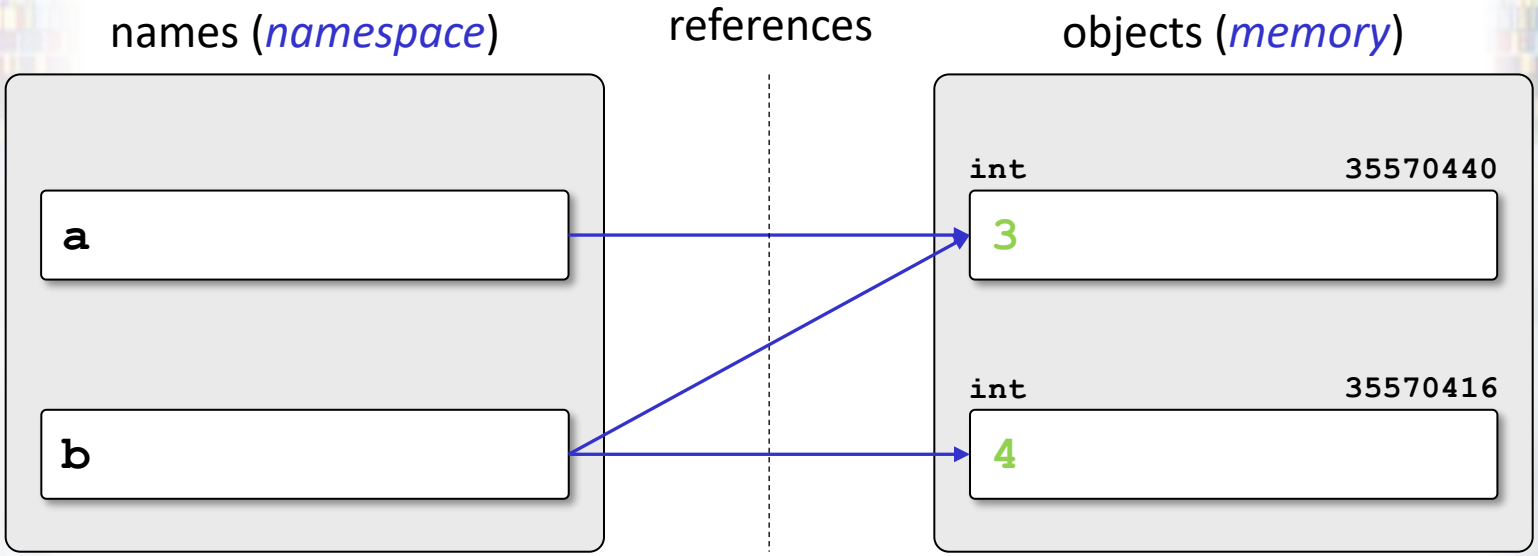
Garbage collection




```
>>> a = 3          # an integer number
>>> a = 'spam'     # now it is a string
>>> a = 1.23       # and now a float number
```

UNIVERSITEIT GENT

Multiple references




```
>>> a = 3
>>> b = 4
>>> print(id(a), id(b))
35570440 35570416
>>> b = a
>>> print(id(a), id(b))
35570440 35570440
```

UNIVERSITEIT GENT

Python style guide

style rule

use long (informative) names that document the semantics of variables

Variable names

- variable names can be arbitrarily long
  - should not start with a digit
  - can contain letters, digits and underscores (_)
  - case sensitive: difference between uppercase and lowercase
    - **`spam`** and **`SPAM`** are different names

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> googol$ = 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
SyntaxError: invalid syntax
```

UNIVERSITEIT GENT

Python style guide

style rule

avoid using names that only
contain uppercase letters

Variable names

- variable names can be arbitrarily long
  - should not start with a digit
  - can contain letters, digits and underscores (_)
  - case sensitive: difference between uppercase and lowercase
    - **spam** and **SPAM** are different names
  - Python keywords cannot be used as variable names
    - they define the language's rules and structure

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> googol$ = 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
SyntaxError: invalid syntax
>>> class = "Programming"
SyntaxError: invalid syntax
```

UNIVERSITEIT GENT

Python keywords

False	None	True	and	as	assert
break	class	continue	def	del	elif
else	except	finally	for	from	global
if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try
while	with	yield			

UNIVERSITEIT GENT

Operators and operands

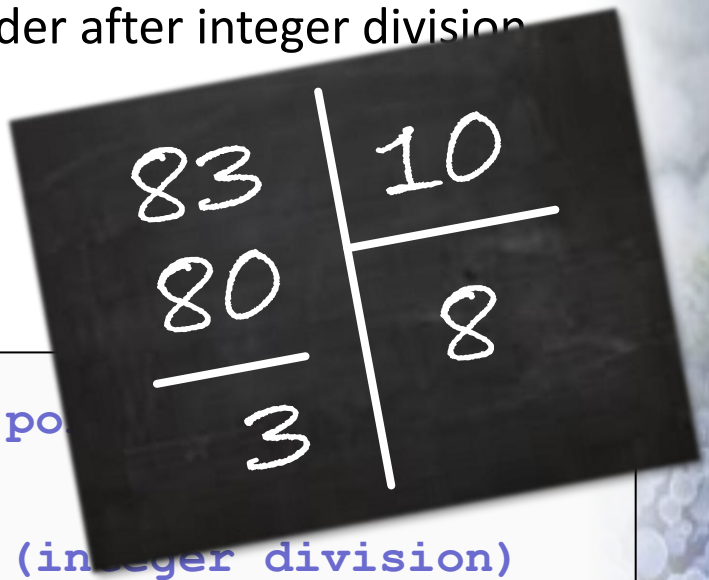
- **operator**: special symbol that represents a computation
  - examples
    - addition: **+**
    - multiplication: *****
  - **operand**: value used by operator
  - when a variable name appears in place of an operand, it is replaced with its value before the operation is performed

```
20 + 32
hours - 1
hours * 60 + minutes
minutes // 60
5 ** 2
(5 + 9) * (15 - 7)
```

UNIVERSITEIT GENT

Division

- Python differentiates **floating point** from **integer division**
  - operator / computes floating point division (always results in a float)
  - operator // computes quotient (integer division)
  - modulo operator % computes remainder after integer division




```
>>> 83 / 10          # floating po
8.3
>>> 83 // 10         # quotient (in eger division)
8
>>> 83 % 10          # remainder (modulo)
3
```

UNIVERSITEIT GENT

Division

- Python differentiates **floating point** from **integer division**
  - operator **/** computes floating point division (always result in a float)
  - operator **//** computes quotient (integer division)
  - modulo operator **%** computes remainder after integer division

```
>>> 83.0 / 10.0     # floating point division
8.3
>>> 83.0 // 10      # quotient (integer division)
8.0
>>> 83 // 10.0      # quotient (integer division)
8.0
>>> 83.0 % 10       # remainder (modulo)
3.0
>>> 83 % 10.0       # remainder (modulo)
3.0
```

UNIVERSITEIT GENT

Type conversion

- each Python type comes with a built-in function that attempts to convert objects of another type into that type
  - **`int(`*`argument`*`)`** converts any object into an integer number
  - runtime exception if argument cannot be converted

```
>>> float(83) // 10
8.0
>>> int('32')
32
>>> int('Hello')
ValueError: invalid literal for int() with base 10
```

UNIVERSITEIT GENT

Type conversion

- each Python type comes with a built-in function that attempts to convert objects of another type into that type
  - **`int(`*`argument`*`)`** converts any object into an integer number
  - runtime exception if argument cannot be converted
  - fractional part is truncated if floats are converted to integers

```
>>> int(-2.3)
-2
>>> int(3.99999)
3
>>> int('42')
42
>>> int(1.0)
1
```

UNIVERSITEIT GENT

Type conversion

- each Python type comes with a built-in function that attempts to convert objects of another type into that type
  - **`float(`*`argument`*`)`** converts any object into a float number
  - difference between integer **`1`** and floating point **`1.0`**
    - distinct data types
    - distinct internal representation in computer memory (see additional slides)

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
>>> float(1)
1.0
```

UNIVERSITEIT GENT

Type conversion

- each Python type comes with a built-in function that attempts to convert objects of another type into that type
  - **`str(`*`argument`*`)`** converts any object into a string

```
>>> str(32)
'32'
>>> str(3.14149)
'3.14149'
>>> str(True)
'True'
>>> str(true)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

UNIVERSITEIT GENT

Order of operations

- order of evaluation depends on rules of precedence
  - operators that are higher in table have higher precedence
  - operators on same row are evaluated from left to right
  - see appendix E in course book for complete precedence table

operator	description
`()`	brackets (grouping)
`**`	power
`+x`, `-x`	positive, negative (unairy)
`*`, `/`, `%`	multiplication, division, remainder
`+`, `-`	addition, subtraction

UNIVERSITEIT GENT

Operations on strings

- in general you cannot compute with strings
  - even if the strings look like numbers
  - Python does not perform implicit type conversions
    - except between numerical types: int, float, complex, …
    - mixed numerical operations: int → float → complex

```
>>> message = "What's up, Doc?"
>>> message - 1
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> 'Hello' / 123
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>> message * 'Hello'
TypeError: can't multiply sequence by non-int of type 'str'
>>> '15' + 2
TypeError: cannot concatenate 'str' and 'int' objects
```

UNIVERSITEIT GENT

Operations on strings

- in general you cannot compute with strings
  - two exceptions:
    - + operator joins strings together (*concatenation*)
    - * operator repeats string a number of times (*repetition*)
      - one operand must be a string, the other an integer
    - analogy with numerical addition and multiplication
      - `4 * 3 == 4 + 4 + 4`
      - `'Spam' * 3 == 'Spam' + 'Spam' + 'Spam'`

```
>>> fruit = 'banana'
>>> baked_good = ' nut bread'
>>> print(fruit + baked_good)
banana nut bread
>>> 'Spam' * 3
'SpamSpamSpam'
>>> 3 * 'Spam'
'SpamSpamSpam'
```

UNIVERSITEIT GENT

Operations on strings

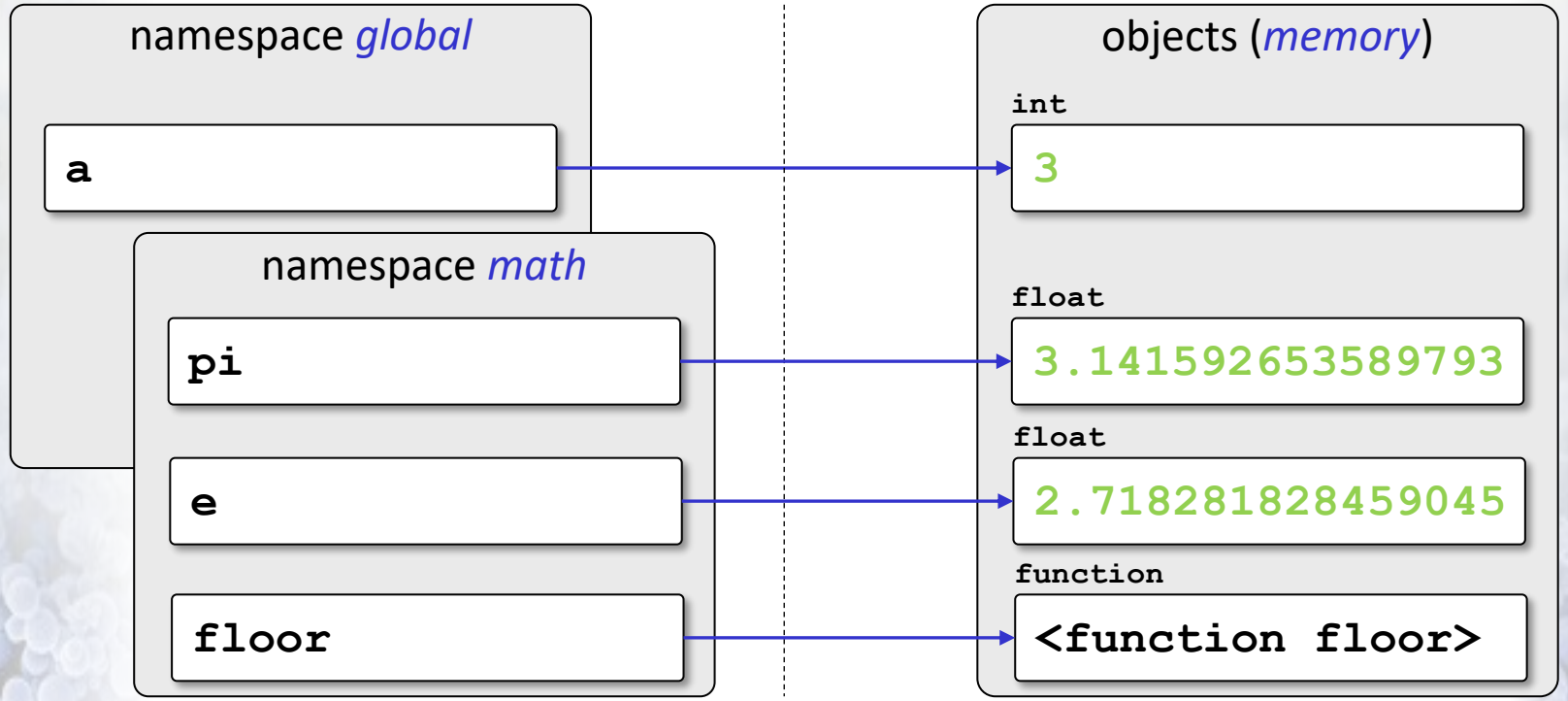
- additional string operations
  - use built-in **`help`**-function
  - check online Python documentation

```
>>> help(str)
Help on class str in module __builtin__:
...
>>> message = "What's up, Doc?"
>>> help(message.upper)
Help on built-in function upper:

upper(...)
    S.upper() -> string

    Return a copy of the string S converted to uppercase.
>>> print(message.upper())
WHAT'S UP, DOC?
```

UNIVERSITEIT GENT

Python extensions

- standard language can be extended using modules
  - example: **`math`** module




```
>>> a = 3
>>> import math
```

UNIVERSITEIT GENT

Python extensions

- standard language can be extended using modules
  - example: **`math`** module

```
>>> import math
>>> help(math)
Help on built-in module math:
...
>>> help(math.floor)
Help on built-in function floor in module math:

floor(...)
    floor(x)

    Return the floor of x as a float.
    This is the largest integral value <= x.
>>> math.floor(3.7)
3.0
```

Python extensions

- standard language can be extended using modules
  - example: **`math`** module

```
>>> math.floor(3.7)          # round down
3
>>> int(3.7)                 # truncate
3
>>> round(3.7)               # round
4
>>> math.floor(-3.7)         # round down
-4
>>> int(-3.7)                # truncate
-3
>>> round(-3.7)              # round
-4
>>> math.ceil(-3.7)          # round up
-3
```

UNIVERSITEIT GENT

Dodona

hint

use the Q&A module of Dodona to ask questions about programming tasks

Homework (hands-on sessions)

- solve mandatory exercises series 01
  (deadline Tuesday, September 30, 2025, 22:00)
  - ISBN (demo, video)
  - Sum of two integers
  - Best laid plans
  - The pudding guy
  - Human Development Index
  - The stopped clock

Homework (next lecture)

- course book: read chapter 2 (control structures)
- read through the Python Style Guide
- learn yourself to use the online help
  - use built-in help-function to study the math module
  - Python standard library: Python extensions
- study extra slides about internal representation of objects in memory
- demo exercises next lecture (series 02)
  - Body-mass index
  - Babysitter
  - Collision detection

UNIVI GENT

Questions or remarks ?

UNIVERSITEIT GENT

The sky is the limit …

"Open your arms to change,
but don't let go of your values."

— Dalai Lama

UNIVERSITEIT GENT