There is a set of four cards placed on a table, each of which has a number on one side and a colored patch on the other side.

Which card(s) must be turned over in order to test the truth of the proposition that if a card shows an even number on one face, then its opposite face is red?

Ghent University
Faculty of Sciences

Python Programming

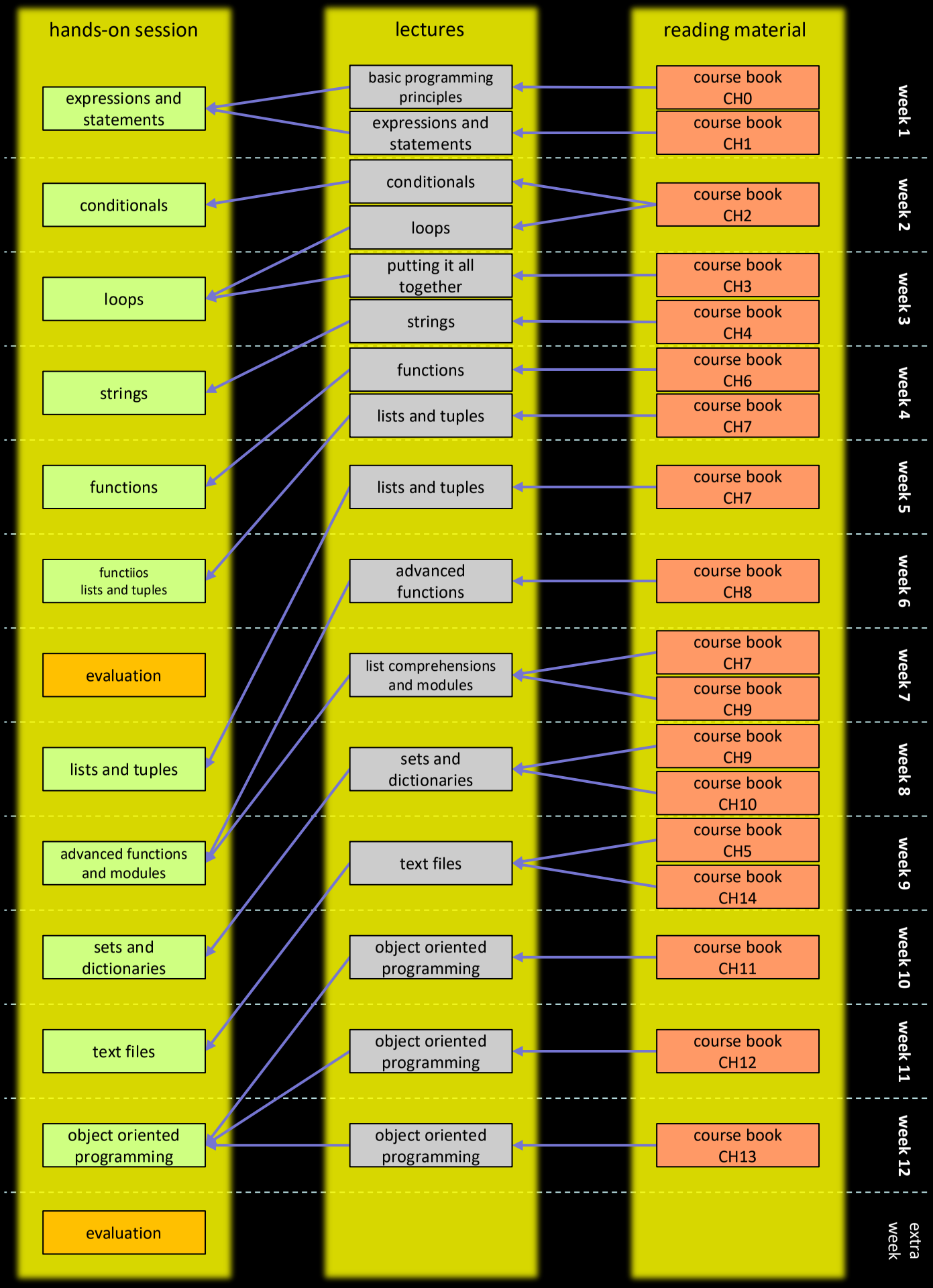
go with the flow

Prof. Dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT

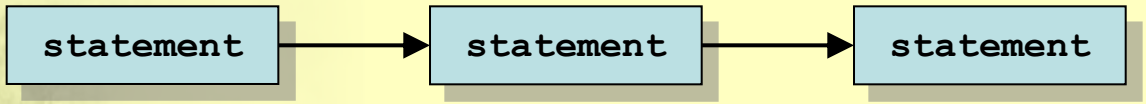

hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries
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

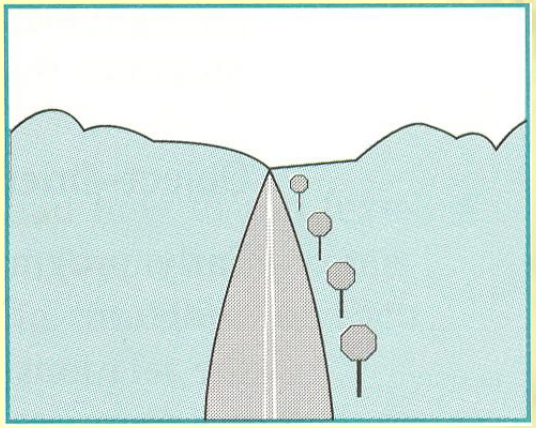
Pear and apple salad with chic

- 4 heads chicory, a mixture of red and white if possible
- 2 English eating apples
- 2 pears
- 1 handful fresh soft herbs (such as chervil, tarragon, parsley or a mixture), roughly chopped
- 50 g strong blue cheese
- 50 g crème fraîche
- 5 tablespoons extra virgin olive oil, plus a little extra for drizzling
- 4 tablespoons cider vinegar
- 6 tablespoons water

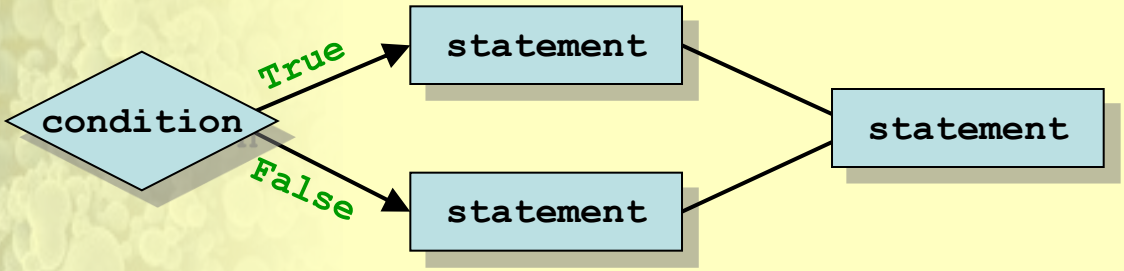
Control structures

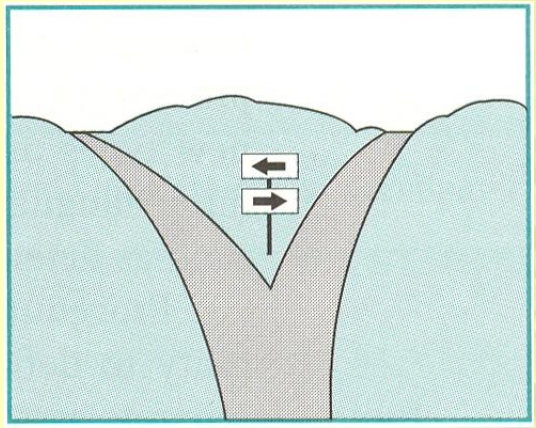
sequence





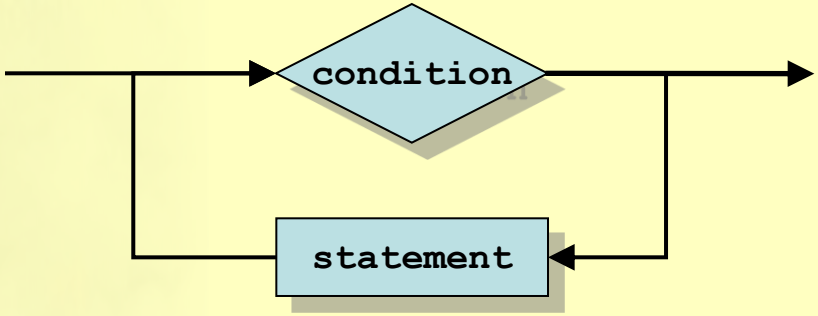
conditional statement (branch)

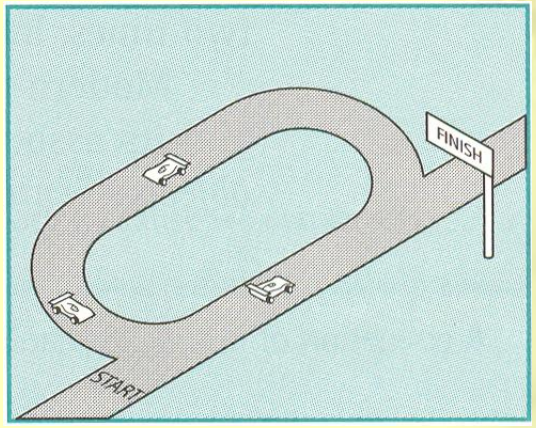




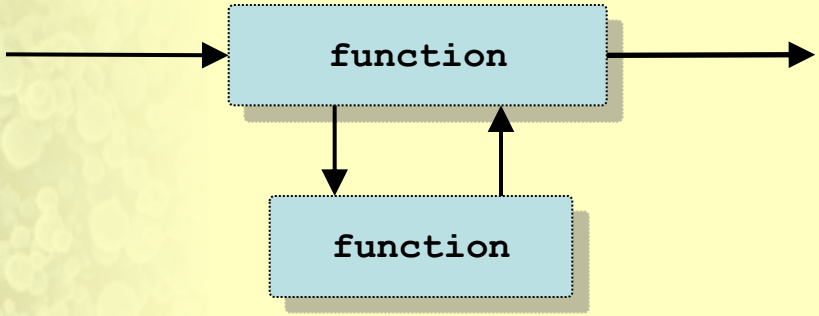
Control structures

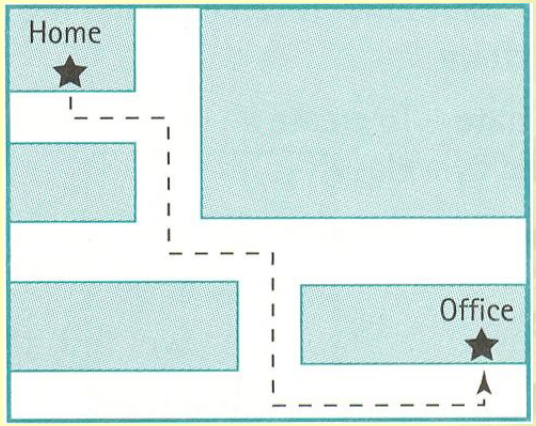
repetitive statement (loop, repetition, iteration)





function (method)





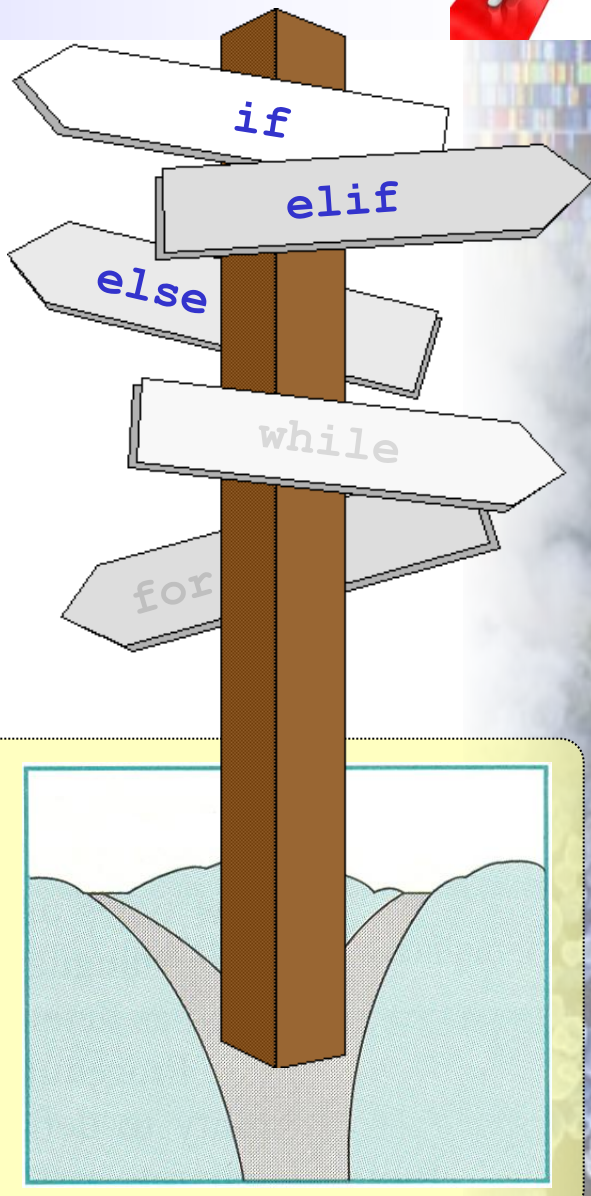
Control structures

- conditional statements
- repetitive statements
- functions

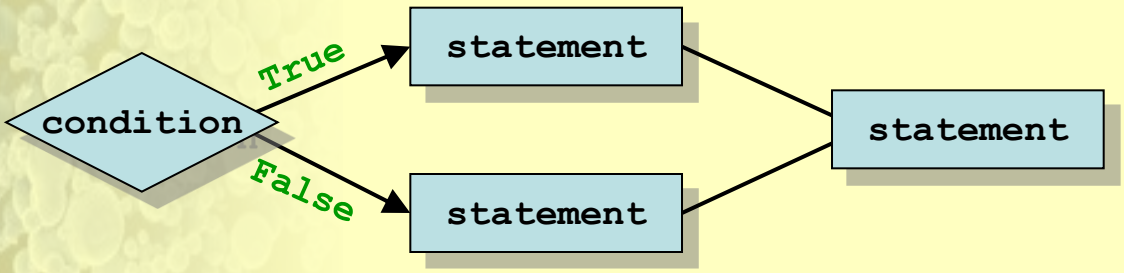
UNIVERSITEIT GENT

Control structures

- conditional statements
- repetitive statements
- functions



conditional statement (branch)



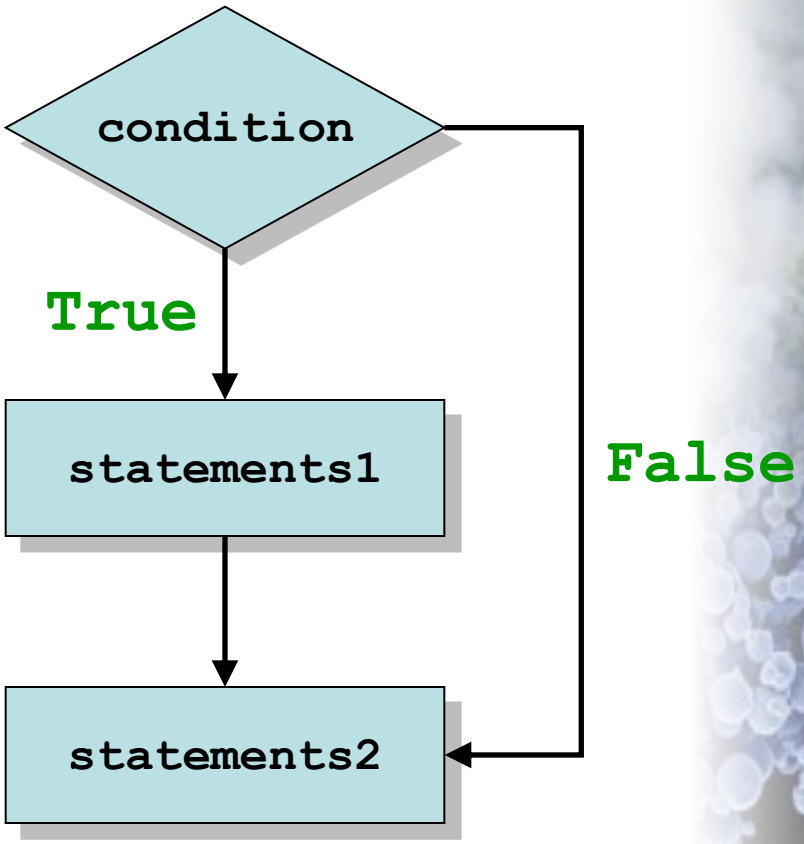
Conditional execution

syntax

```
if boolean expression:
    statements
```

example

```
if x > 0:
    print('x is positive')
```



UNIVERSITEIT GENT

Pass statement

- number of statements in body of **`if`** statement is unlimited
  - there has to be at least one
  - occasionally it is useful to have a body without statements
    - e.g. as a placeholder for code has not yet been written
  - **`pass`** statement
    - statement that does nothing

```
if boolean expression:
    statements
```

UNIVERSITEIT GENT

Pass statement

- number of statements in body of **`if`** statement is unlimited
  - there has to be at least one
  - occasionally it is useful to have a body without statements
    - e.g. as a placeholder for code has not yet been written
  - **`pass`** statement
    - statement that does nothing

UNIVERSITEIT GENT

```
if True:      # this is always true
    pass      # always executed, but it does nothing
```

Modulo operator

- modulo operator (%)
  - works both on integers and floats
  - yields remainder when first operand is divided by second

```
>>> quotient = 7 // 3
>>> print(quotient)
2
>>> remainder = 7 % 3
>>> print(remainder)
1
```

UNIVERSITEIT GENT

Modulus operator

- modulus operator (%)
  - works both on integers and floats
  - yields remainder when first operand is divided by second
- convenient operator to perform tests
  - check whether **x** is divisible by **y**
  - determine rightmost digits of an integer

```
>>> x = 18; y = 3; z = 1234
>>> x % y == 0
True
>>> z % 10              # units
4
>>> z % 100             # tens
34
```

UNIVERSITEIT GENT

Boolean values

- data type **`bool`**
  - representation of values **`True`** and **`False`**
    - *case sensitive*
  - named after British mathematician George Boole
    - founder of the Boolean algebra

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> type(true)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
```

UNIVERSITEIT GENT

Boolean expressions

- Boolean expression
  - expression that evaluates to a boolean value
  - operator **==** compares two values
    - produces a boolean value
    - example of a *comparison operator*

```
>>> 5 == 5
True
>>> 5 == 6
False
>>> x = (5 == 5)
>>> print(x)
True
```

UNIVERSITEIT GENT

Comparison operators

operator	meaning
`x == y`	is equal to
`x != y`	is not equal to
`x < y`	is less than
`x > y`	is greater than
`x <= y`	is less than or equal to
`x >= y`	is greater than or equal to

UNIVERSITEIT GENT

Logical operators

and	True	False
True	True	False
False	False	False

conjunction

or	True	False
True	True	True
False	True	False

disjunction

	not
True	False
False	True

negation

UNIVERSITEIT GENT


AND
Truth table
A
B
Output
0
0
0
0
1
0
1
1
1

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

```
>>> age = 12
>>> if age >= 6 and age <= 18:
...     print('You have to go to school!!')
...
You have to go to school!!
```

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

```
>>> age = 12
>>> if 6 <= age <= 18:
...     print('You have to go to school!!')
...
You have to go to school!!
```

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

```
>>> today = 'saturday'
>>> if today == 'saturday' or today == 'sunday':
...     print('It is weekend!!')
...
It is weekend!!
```

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

```
>>> today = 'saturday'
>>> if today in ['saturday', 'sunday']:
...     print('It is weekend!!')
...
It is weekend!!
```

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

```
>>> today = 'tuesday'
>>> if not(today == 'saturday' or today == 'sunday'):
...     print('Today we have lectures!!')
...
Today we have lectures!!
```

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

```
>>> today = 'tuesday'
>>> if today != 'saturday' and today != 'sunday':
...     print('Today we have lectures!!')
...
Today we have lectures!!
```

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

```
>>> today = 'tuesday'
>>> if today not in ['saturday', 'sunday']:
...     print('Today we have lectures!!')
...
Today we have lectures!!
```

UNIVERSITEIT GENT

Logical operators

operator	name	meaning
`and`	conjunction	gives `True` if both conditions are true, `False` otherwise
`or`	disjunction	gives `False` if both conditions are false, `True` otherwise
`not`	negation	converts `False` into `True`, and `True` into `False`

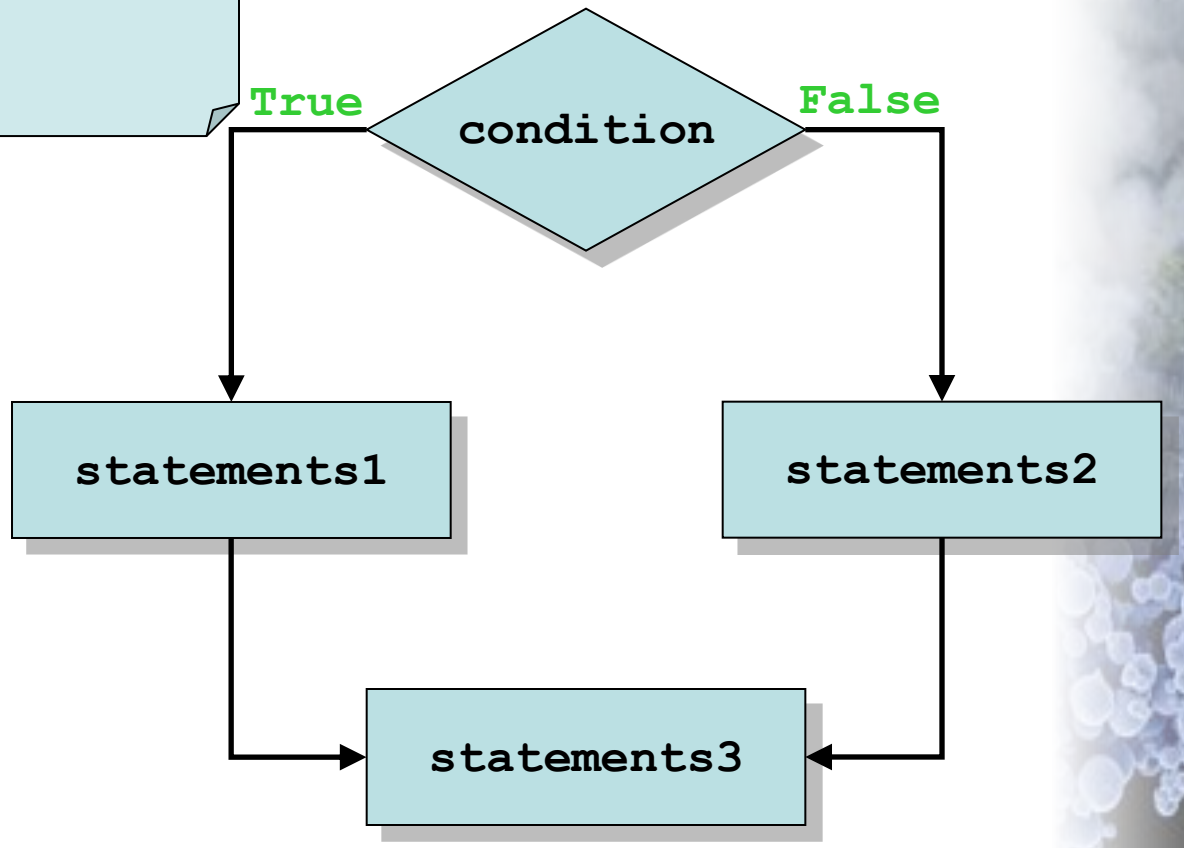
```
>>> today = 'tuesday'
>>> if not today in ['saturday', 'sunday']:
...     print('Today we have lectures!!')
...
Today we have lectures!!
```

UNIVERSITEIT GENT

Alternative execution

syntax

```
if boolean expression:
    statements1
else:
    statements2
```



UNIVERSITEIT GENT

Alternative execution

syntax

```
if boolean expression:
    statements1
else:
    statements2
```

```
>>> x = 4
>>> if x % 2 != 0:
...     print(f'{x} is odd')
... else:
...     print(f'{x} is even')
...
4 is even
```

UNIVERSITEIT GENT

Alternative execution

syntax

```
if boolean expression:
    statements1
else:
    statements2
```

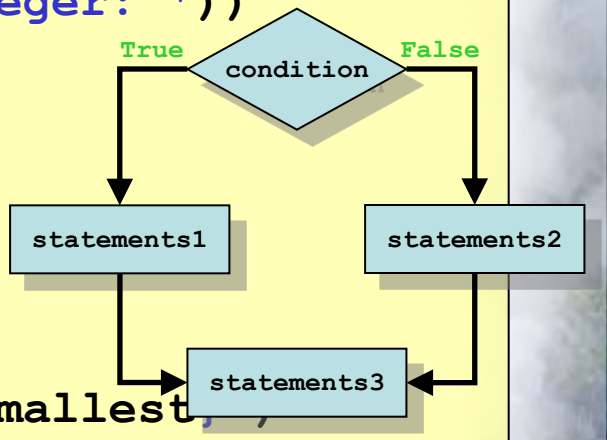
```
>>> x = 4
>>> if x % 2:
...     print(f'{x} is odd')
... else:
...     print(f'{x} is even')
...
4 is even
```

UNIVERSITEIT GENT

Alternative execution

smallest.py

```
# determine smallest of two given integer
x = float(input('Enter an integer: '))
y = float(input('Enter another integer: '))

if x < y:
    smallest = x
else:
    smallest = y

print(f'The smallest integer is {smallest}')
```

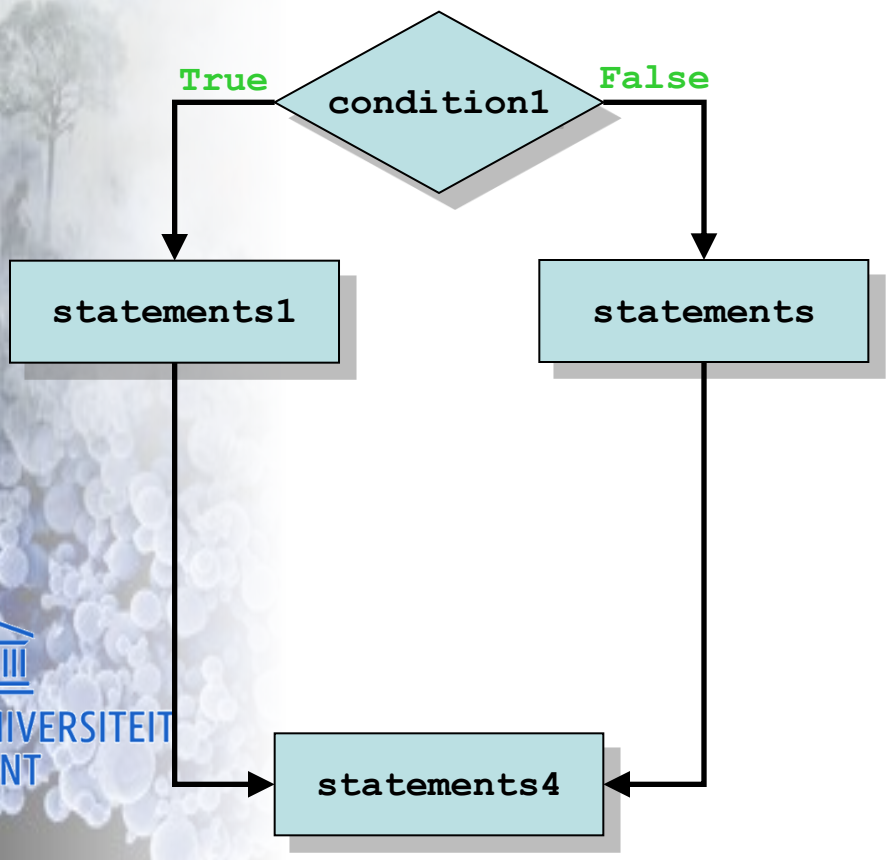


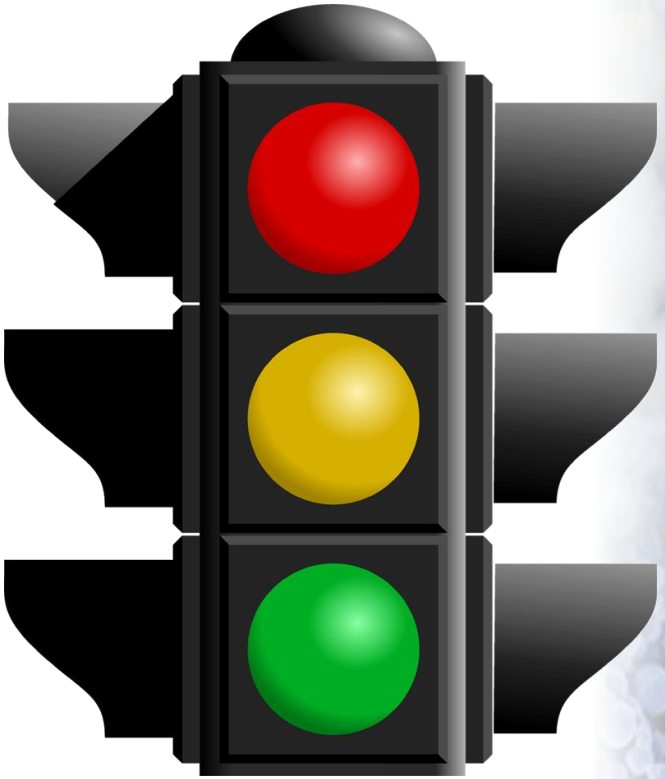

```
$ python3 smallest.py
Enter an integer: 3.12
Enter another integer: 7.83
The smallest integer is 3.12
```

What would happen if x and y are equal ?

UNIVERSITEIT GENT

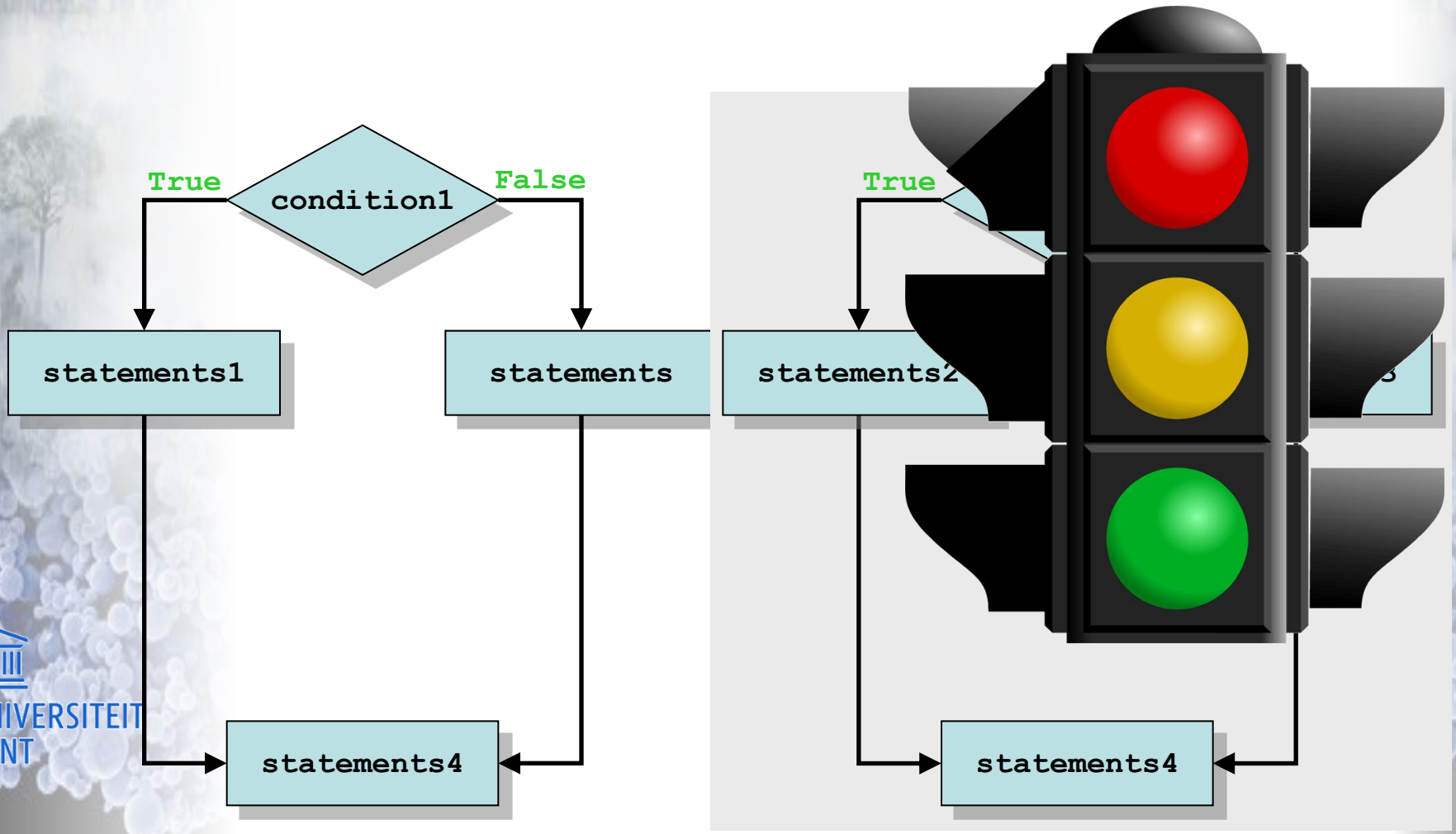
Chained conditionals





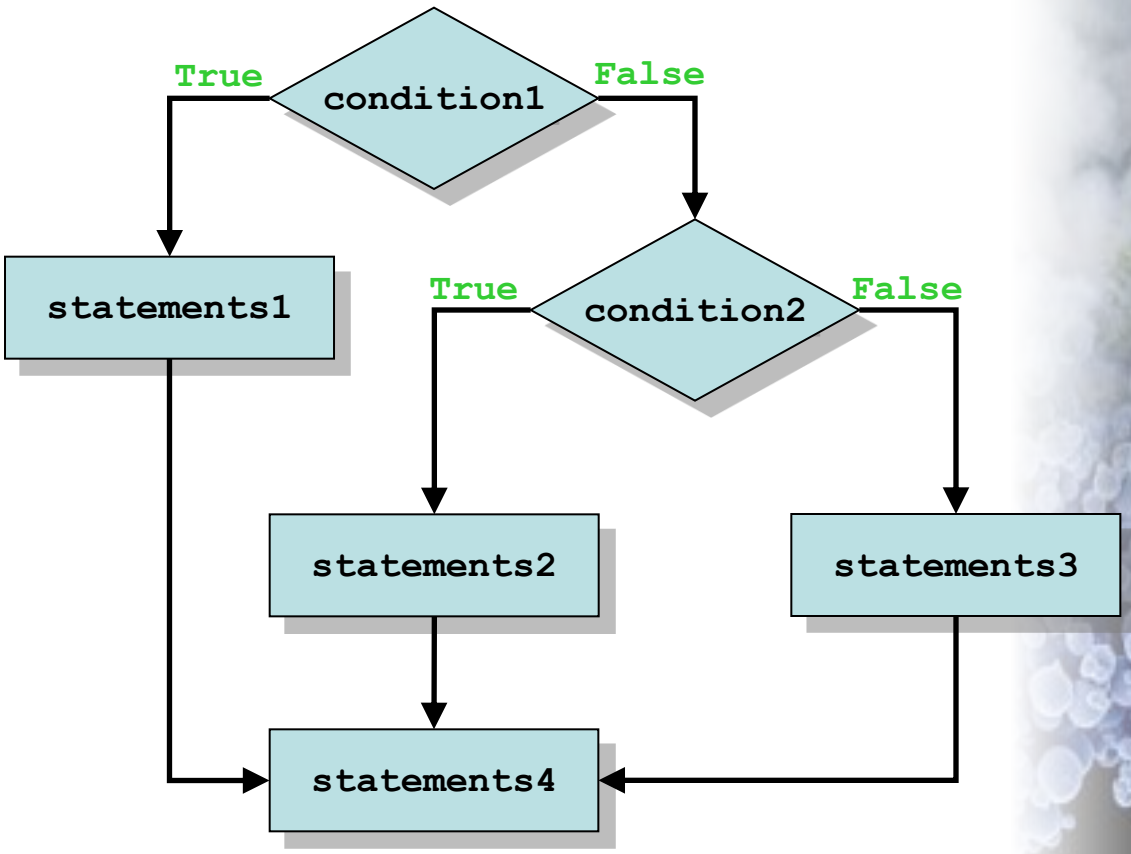
UNIVERSITEIT GENT

Chained conditionals



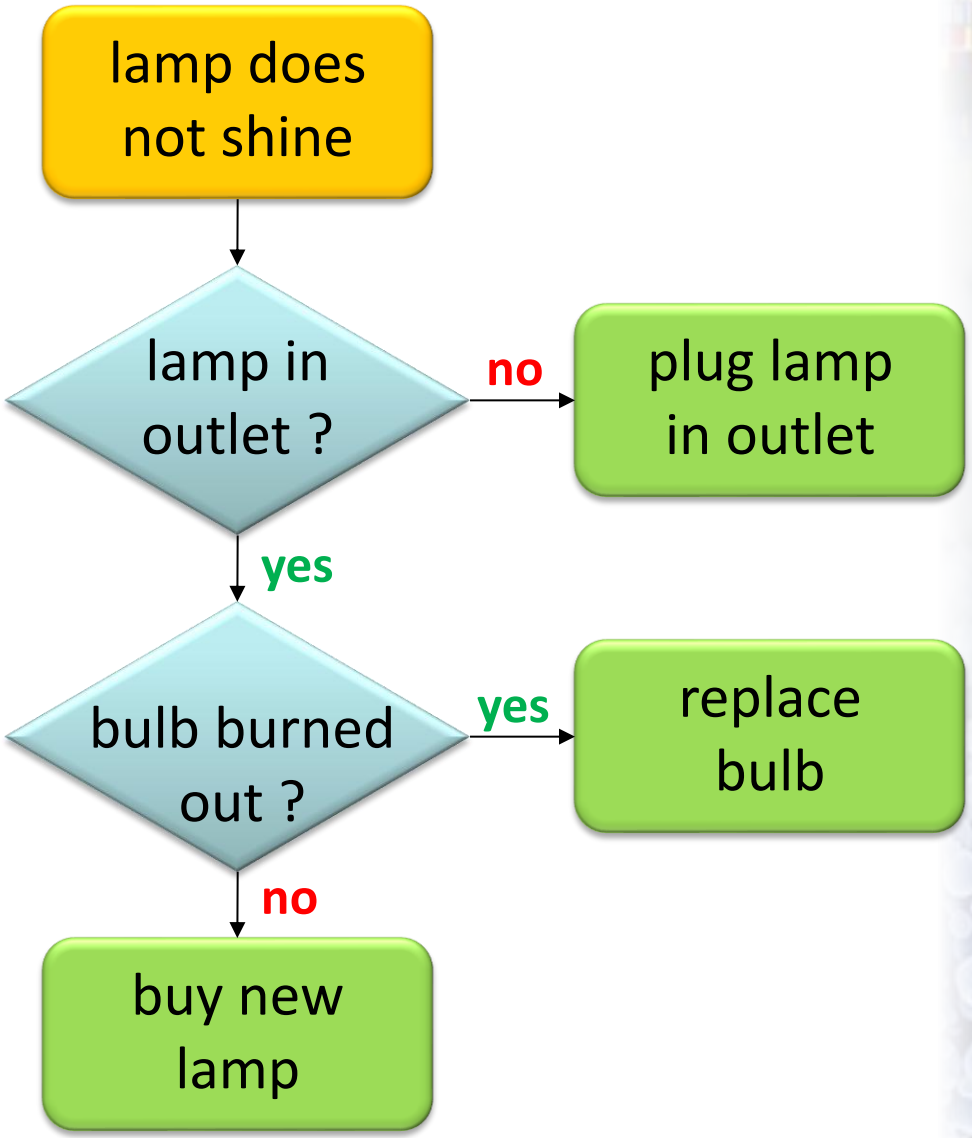
UNIVERSITEIT GENT

Chained conditionals



UNIVERSITEIT GENT

Chained conditionals

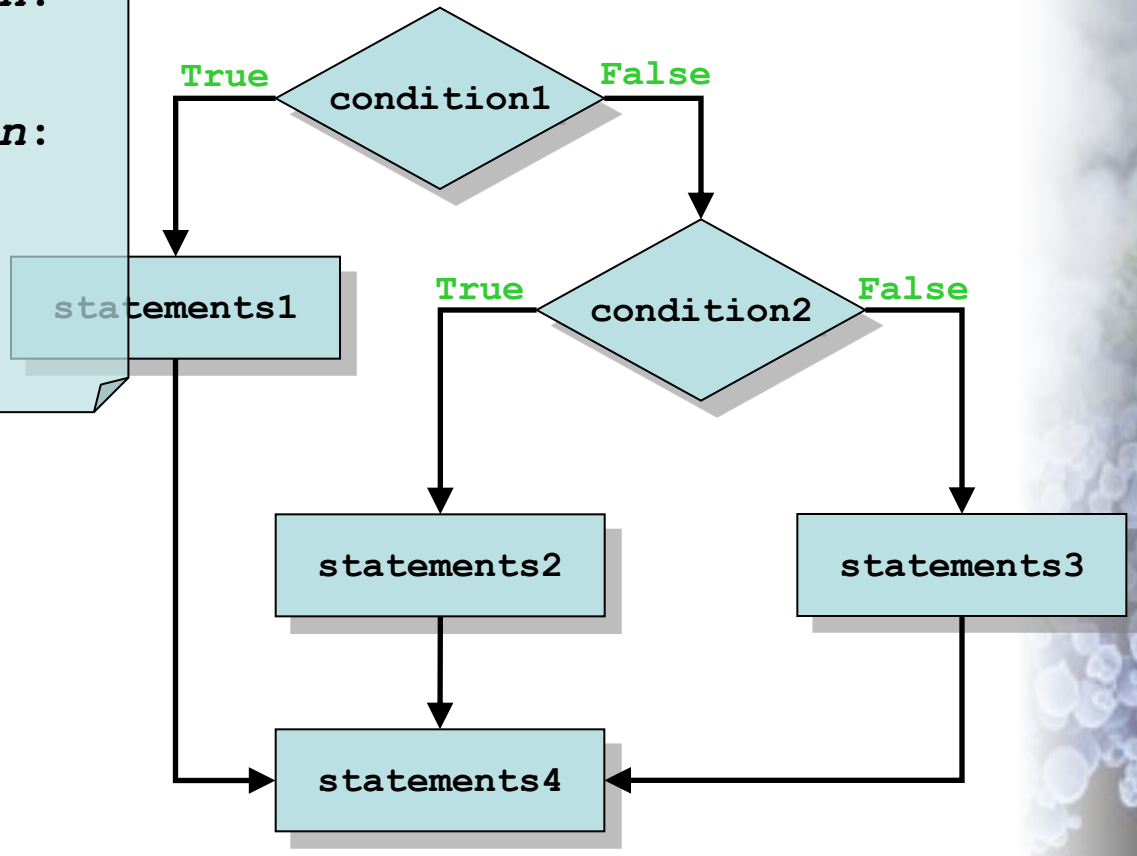


UNIVERSITEIT GENT

Chained conditionals

syntax

```
if boolean expression:
    statements
[elif boolean expression:
    statements]
[elif boolean expression:
    statements]
[else:
    statements]
```



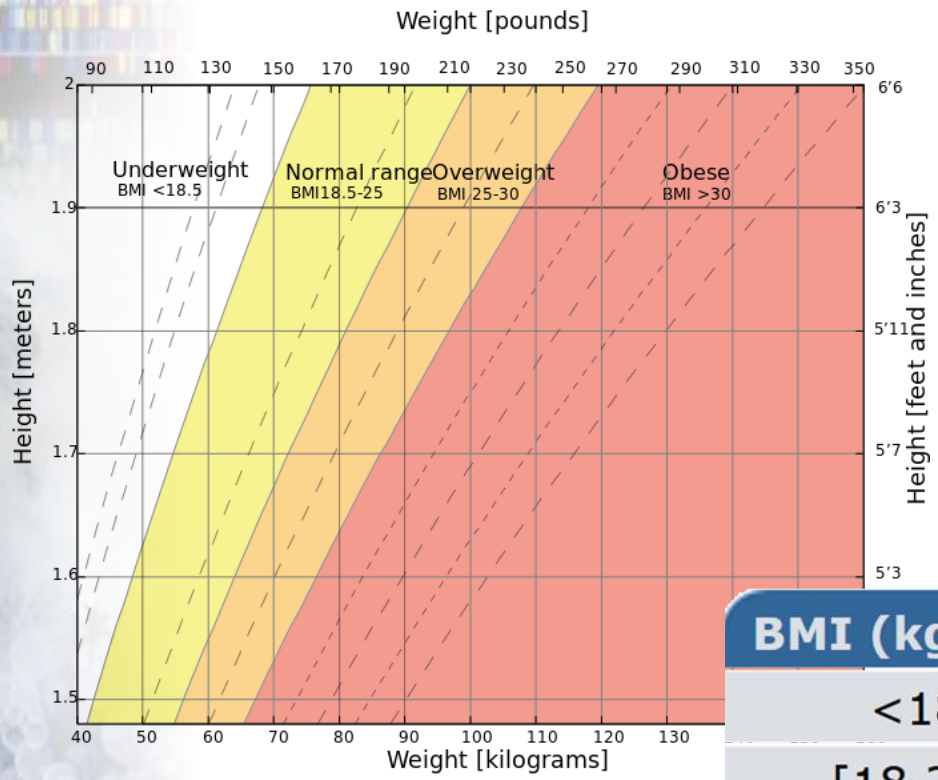
UNIVERSITEIT GENT

Chained conditionals

```
>>> x = 3; y = 4
>>> if x < y:
...     print(f'{x} is less than {y}')
... elif x > y:
...     print(f'{x} is greater than {y}')
... else:
...     print(f'{x} and {y} are equal')
...
3 is less than 4
```

UNIVERSITEIT GENT

BMI



$$\text{BMI} = \frac{\text{mass}}{\text{height}^2}$$

BMI (kg/m^2)	interpretation
<18	underweight
[18,25[	normal weight
[25,27[	overweight
[27,30[	moderate overweight
[30,40[	severe overweight
≤40	very severe overweight

UNIVERSITEIT GENT

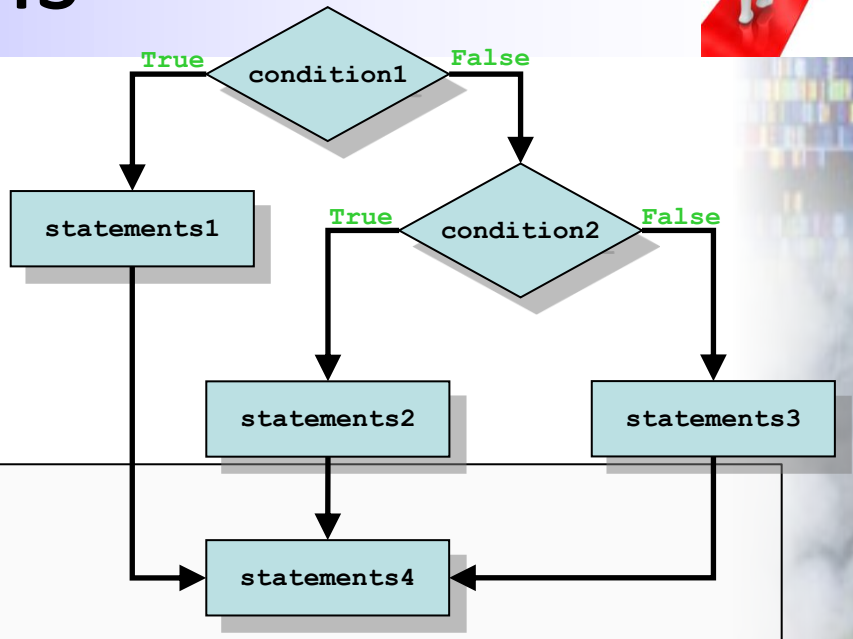
BMI

BMI.py

```
# get personal data
weight = int(input('Enter your weight in kilogram: '))
length = int(input('Enter your length in centimeter: '))

# compute BMI
BMI = weight / (length / 100) ** 2    # float division (2x)

# interpret BMI
if BMI < 18:
    interpretation = 'underweight'
elif BMI < 25:
    interpretation = 'normal range'
elif BMI < 27:
    interpretation = 'overweight'
elif BMI < 30:
    interpretation = 'moderate obese'
elif BMI < 40:
    interpretation = 'severe obese'
else:
    interpretation = 'very severe obese'

# output interpretationweight
print(f'A person weighing {weight} kg an measuring {length} cm ' +
    'has {interpretation}.')
```

UNIVERSITEIT GENT

Nested conditionals




```
>>> x = 3; y = 4
>>> if x == y:
...     print(f'{x} and {y} are equal')
... else:
...     if x < y:
...         print(f'{x} is less than {y}')
...     else:
...         print(f'{x} is greater than {y}')
...
3 is less than 4
```

UNIVERSITEIT GENT

Nested conditionals

- indentation of statements makes structure apparent
  - nested conditionals become difficult to read very quickly
  - in general, it is a good idea to avoid them when you can

```
>>> x = 3; y = 4
>>> if x == y:
...     print(f'{x} and {y} are equal')
... else:
...     if x < y:
...         print(f'{x} is less than {y}')
...     else:
...         print(f'{x} is greater than {y}')
...
3 is less than 4
```

UNIVERSITEIT GENT

Nested conditionals

- indentation of statements makes structure apparent
  - nested conditionals become difficult to read very quickly
  - in general, it is a good idea to avoid them when you can
    - logical operators often provide a way to simplify nested conditional statements

```
if 0 <= x:
    if x < 10:
        print(f'{x} is a positive single digit.')
```

UNIVERSITEIT GENT

Nested conditionals

- indentation of statements makes structure apparent
  - nested conditionals become difficult to read very quickly
  - in general, it is a good idea to avoid them when you can
    - logical operators often provide a way to simplify nested conditional statements

```
if 0 <= x and x < 10:
    print(f'{x} is a positive single digit.')
```

UNIVERSITEIT GENT

Nested conditionals

- indentation of statements makes structure apparent
  - nested conditionals become difficult to read very quickly
  - in general, it is a good idea to avoid them when you can
    - logical operators often provide a way to simplify nested conditional statements

```
if 0 <= x < 10:
    print(f'{x} is a positive single digit.')
```

UNIVERSITEIT GENT

Python syntax rules

```
if condition1:                    # start of
if condition2
if condition3
statements1
elif conditi                      th third if
statements2
else:                             th third if
statements3

                                  third if
else:                             th second if
if condition5                     of fourth if
statements4

                                  fourth if
                                  second if
else:                             th first if
statements5

                                  # end of first if
```

spaghetti code

UNIVERSITEIT GENT

Python syntax rules

```
if condition1:                    # start of first if
    if condi                      of second if
        if c                      of third if
            c
        elif                      ith third if
            s
        else                      ith third if
            s
                                  third if
    else:                         ith second if
        if c                      of fourth if
            s
                                  fourth if
                                  second if
else:                             ith first if
    statemen
                                  # end of first if
```

UNIVERSITEIT GENT

Python style guide

style rule

Python Style Guide (PEP8) recommends 4 spaces per indentation level, and definitely not tabs

Python syntax rules

```
if condition1:                  # start of first if
    if condition2:              # start of second if
        if condition3:          # start of third if
            condition1
        elif condition4:        # elif with third if
            statements2
        else:                   # else with third if
            statements3
                                # end of third if
    else:                       # else with second if
        if condition5:          # start of fourth if
            statements4
                                # end of fourth if
                                # end of second if
else:                           # else with first if
    statements5
                                # end of first if
```

UNIVERSITEIT GENT

Python syntax rules

```
if condition1:                  # start of first if
    if condition2:              # start of second if
        if condition3:          # start of third if
            statements1
        elif condition4:        # elif with third if
            statements2
        else:                   # else with third if
            statements3
    else:                       # else with second if
        if condition5:          # start of fourth if
            statements4
else:                           # else with first if
    statements5
```

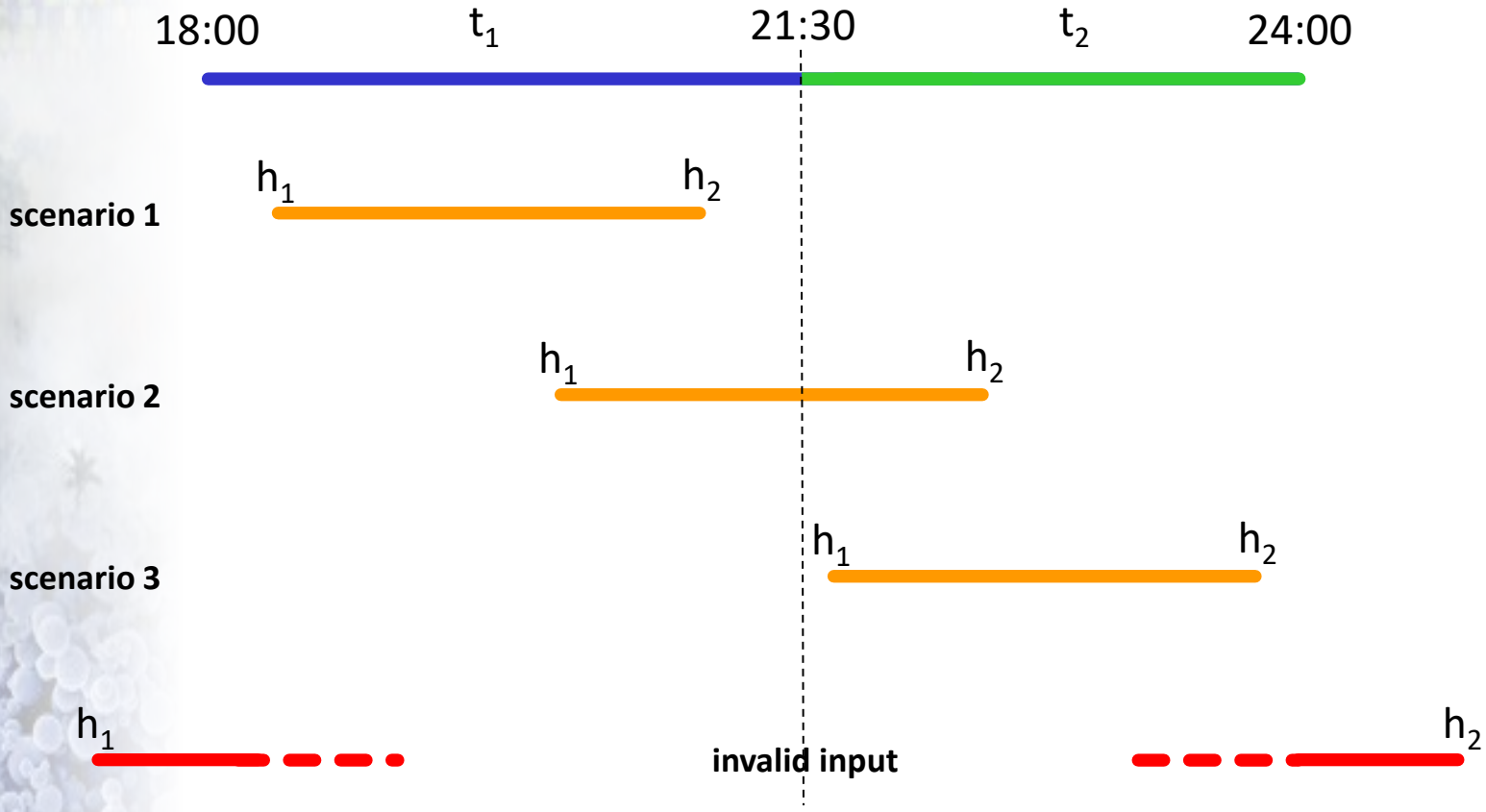
UNIVERSITEIT GENT

Babysitter

programming exercise

A babysitter charges €2 per hour between 18:00 and 21:30, and €4 per hour between 21:30 en midnight. She does not want to babysit before 18:00 nor after midnight.

Babysitter



UNIVERSITEIT GENT

Babysitter

babysit1.py

```
# read start and stop times
h1 = int(input())
m1 = int(input())
h2 = int(input())
m2 = int(input())

# convert times into hours since midnight
h1 = h1 + m1 / 60           # caution: floating point division
h2 = h2 + m2 / 60
h1800 = 18.0                # 18:00 in hours since midnight
h2130 = 21.5                # 21:30 in hours since midnight

if h1 < h1800 or h2 < h1:
    print('invalid input')
else:
    # compute babysit hours before (t1) and after (t2) 21:30
    if h2 < h2130:          # scenario 1
        t1 = h2 - h1
        t2 = 0.0
    elif u1 < h2130:        # scenario 2
        t1 = h2130 - h1
        t2 = h2 - h2130
    else:                   # scenario 3
        t2 = h2 - h1
        t1 = 0.0

    # compute total amount earned
    print(2 * t1 + 4 * t2)
```

UNIVERSITEIT GENT

Babysitter

babysit2.py


```
# read start and stop times
h1 = int(input())
m1 = int(input())
h2 = int(input())
m2 = int(input())

# convert times into hours since midnight
h1 = h1 + m1 / 60            # caution: floating point division
h2 = h2 + m2 / 60
h1800 = 18.0                 # 18:00 in hours since midnight
h2130 = 21.5                 # 21:30 in hours since midnight

if h1 < h1800 or h2 < h1:
    print('invalid input')
else:
    # compute babysit hours before (t1) and after (t2) 21:30
    t1 = max(0, min(h2130, h2) - h1)
    t2 = max(0, h2 - max(u2130, h1))

    # compute total amount earned
    print(2 * t1 + 4 * t2)
```

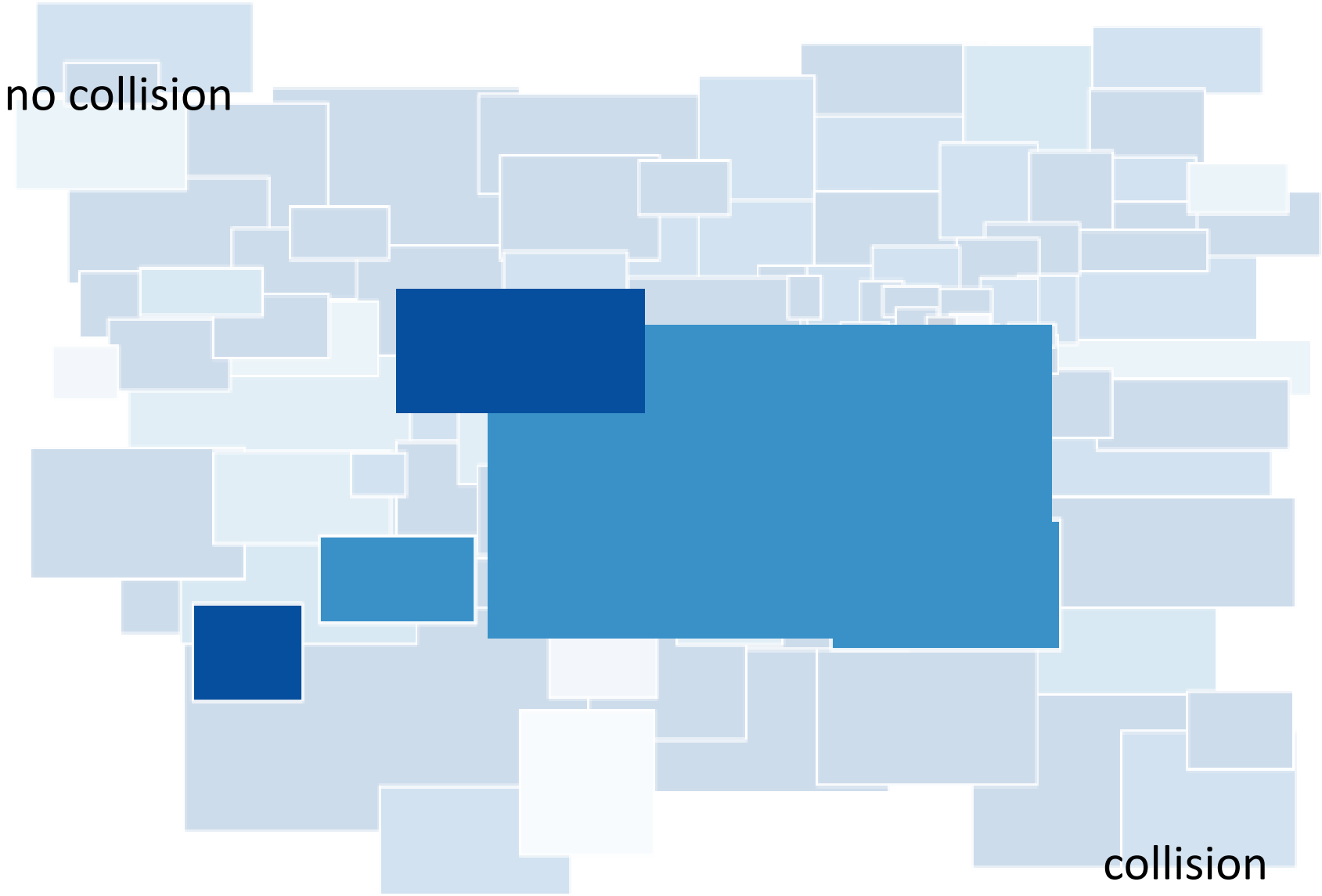
UNIVERSITEIT GENT

Collision detection

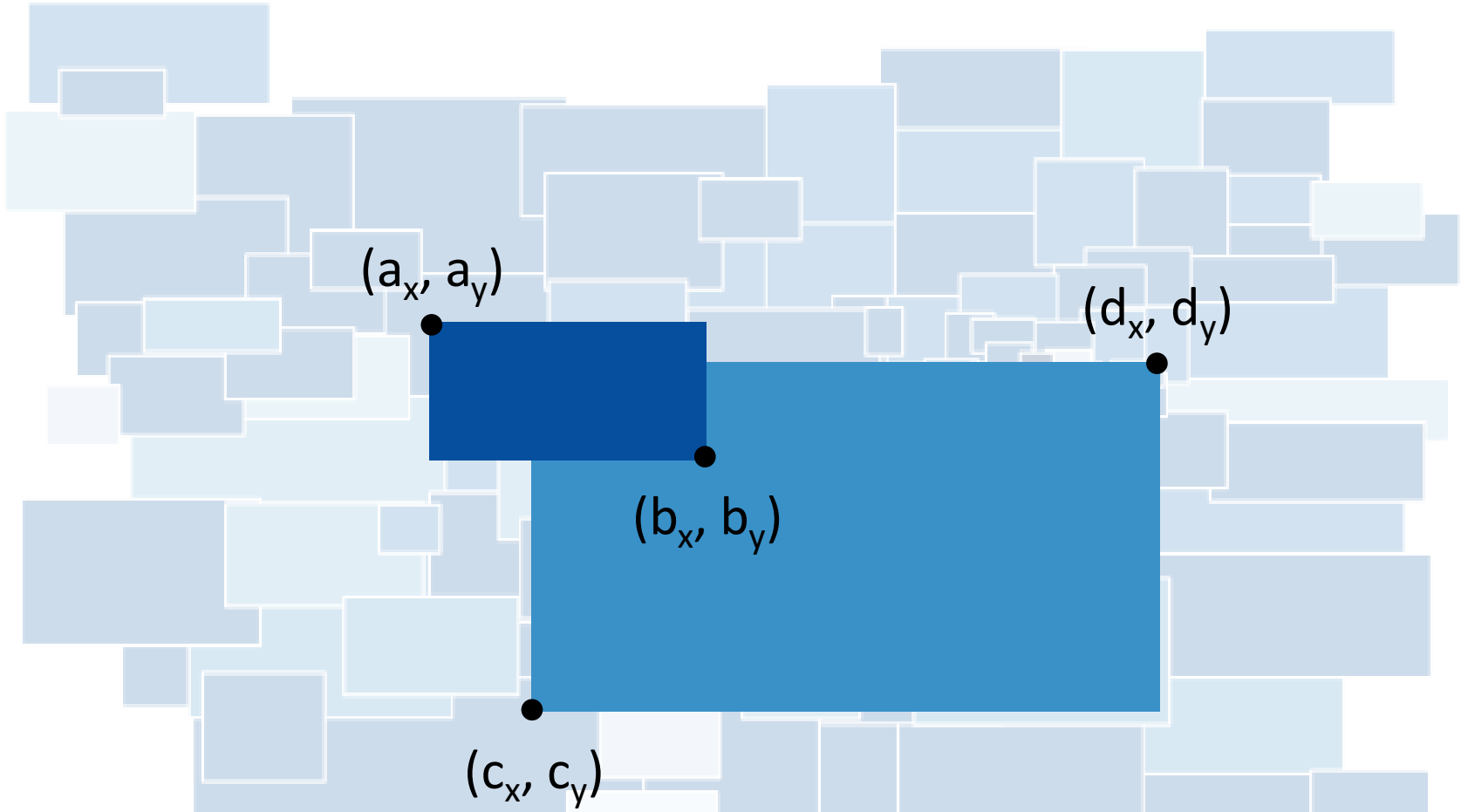
programming exercise

Determine whether or not two rectangles overlap.

Collision detection

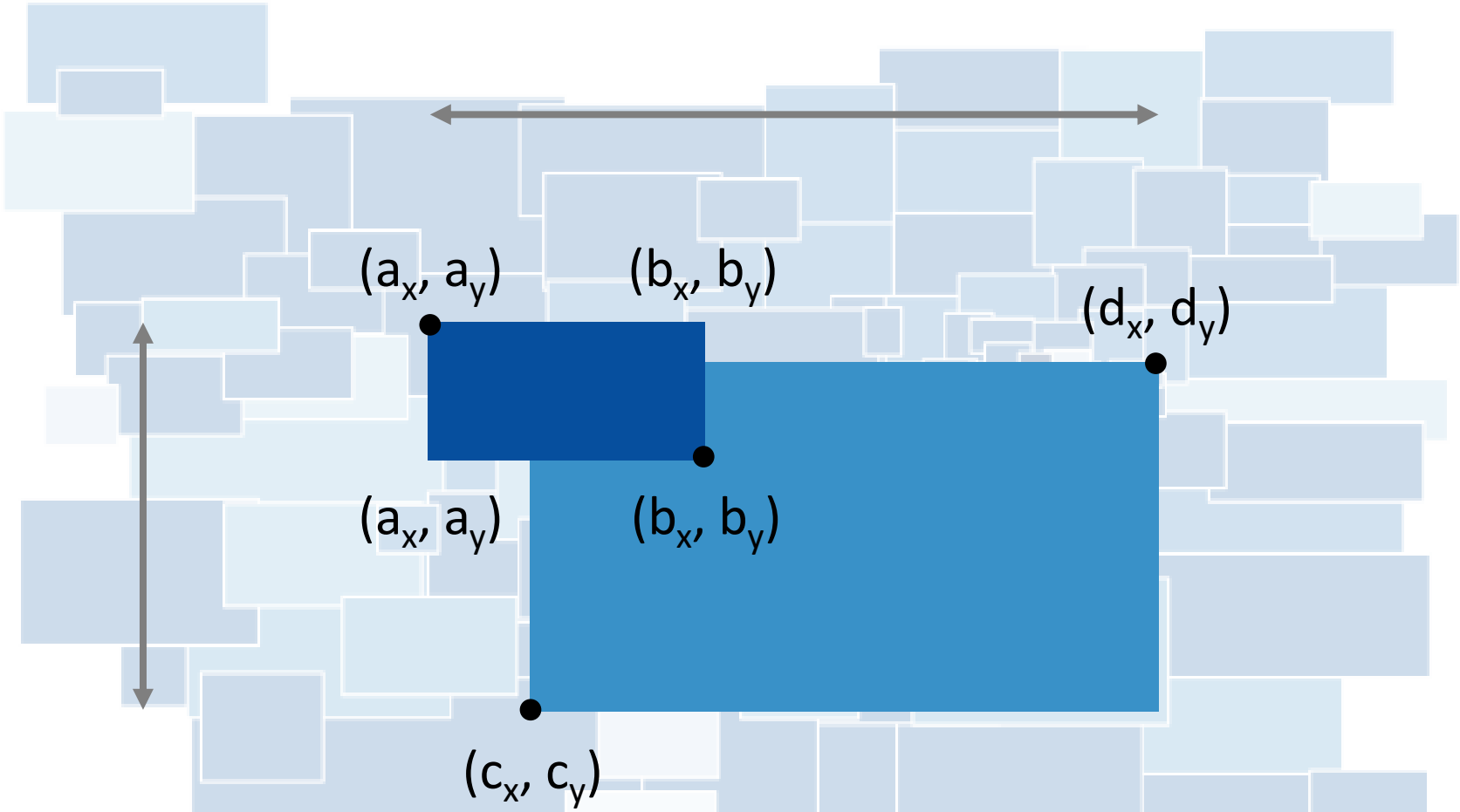


Collision detection




```
ax, bx = min(ax, bx), max(ax, bx)
```

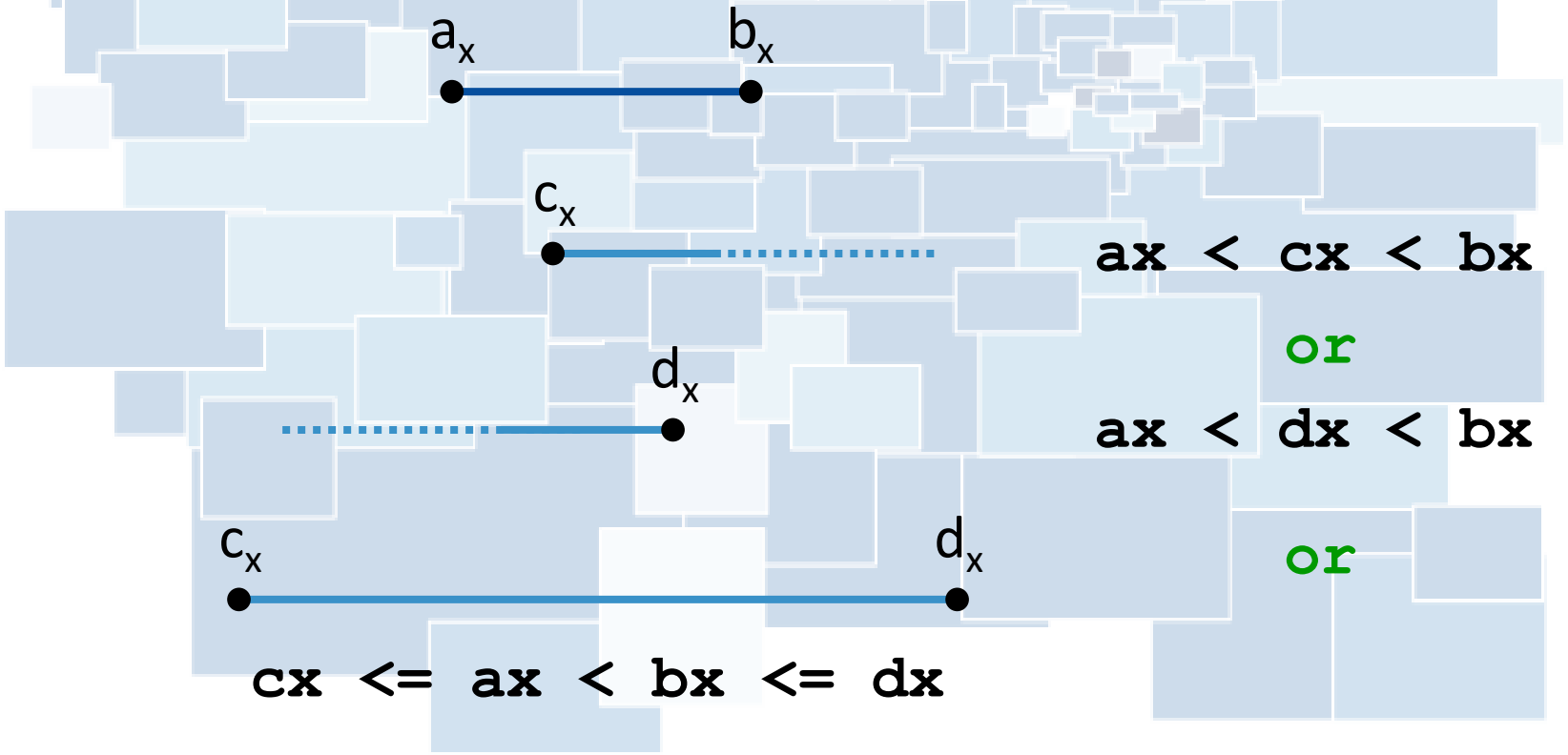
Collision detection




```
ax, bx = min(ax, bx), max(ax, bx)
```

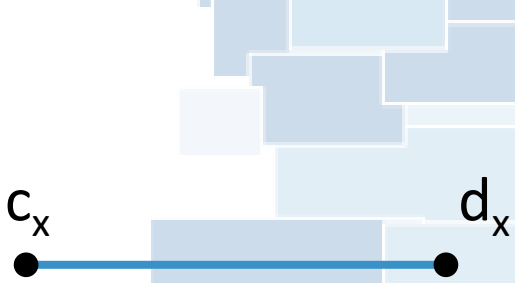
Collision detection

When does this segment overlap with segment c_x,d_x ?

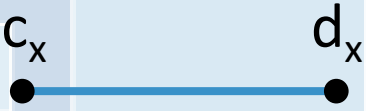


Collision detection

When **doesn't** this segment overlap with segment c_x,d_x ?




```
ax >= dx
or
cx >= bx
```




```
not(ax<dx and cx<bx)
```

Collision

botsing.py

```
# read two diametral points of two rectangles
ax, ay = int(input()), int(input())
bx, by = int(input()), int(input())
cx, cy = int(input()), int(input())
dx, dy = int(input()), int(input())

# assure diametral points are bottom left and top right
ax, bx = min(ax, bx), max(ax, bx)
ay, by = min(ay, by), max(ay, by)
cx, dx = min(cx, dx), max(cx, dx)
cy, dy = min(cy, dy), max(cy, dy)

# check if rectangles overlap horizontally and vertically
horizontal = ax < dx and cx < bx
vertical = ay < dy and cy < by

# check if both rectangles overlap
if horizontal and vertical:
    print('collision')
else:
    print('no collision')
```

UNIVERSITEIT GENT

Collision

botsing.py

```
# read two diametral points of two rectangles
ax, ay = int(input()), int(input())
bx, by = int(input()), int(input())
cx, cy = int(input()), int(input())
dx, dy = int(input()), int(input())

# assure diametral points are bottom left and top right
ax, bx = min(ax, bx), max(ax, bx)
ay, by = min(ay, by), max(ay, by)
cx, dx = min(cx, dx), max(cx, dx)
cy, dy = min(cy, dy), max(cy, dy)

# check if rectangles overlap horizontally and vertically
horizontal = ax < dx and cx < bx
vertical = ay < dy and cy < by

# check if both rectangles overlap
print('collision' if horizontal and vertical else 'no collision')
```

UNIVERSITEIT GENT

Homework (next lecture)

- course book: read chapters 2 and 3
  - chapter 2 (control loops)
  - chapter 3 (putting it all together)
- read problem description of demo exercises
  - Collatz conjecture
  - Birthday paradox

UNIVI
GENT

Homework (hands-on sessions)

- solve mandatory exercises of series 02 (deadline Tuesday, October 7, 2025, 22:00)
  - ISBN (demo, video)
  - Reynolds number
  - Rock-paper-scissors
  - Seasons
  - Trilateration
  - Darts

UNIVI GENT

Questions or remarks ?

UNIVERSITEIT GENT

The sky is the limit …

"What I hear I forget.
What I see I remember.
What I do I understand."

— Confucius

UNIVERSITEIT GENT