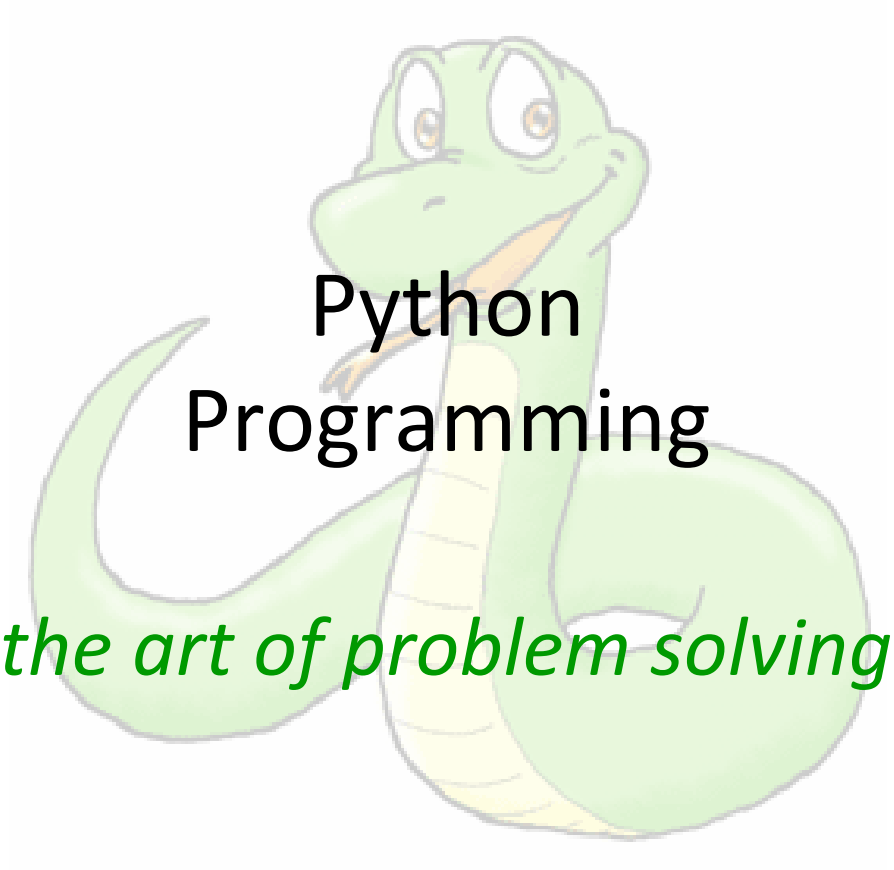
Ghent University
Faculty of Sciences



Python Programming

the art of problem solving

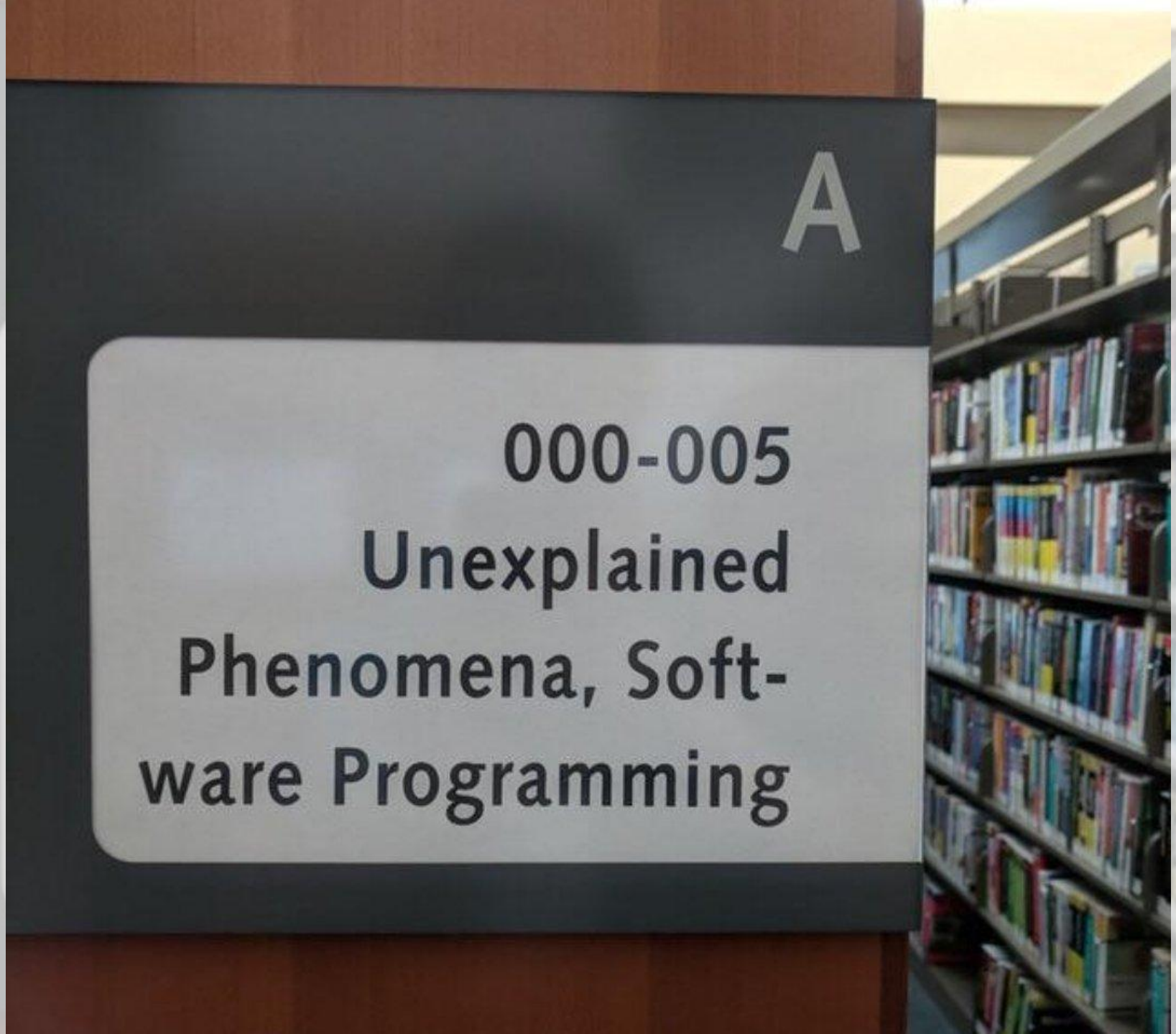
Prof. Dr. Peter Dawyndt

peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


Come closer
I have something
to tell you


A
000-005
Unexplained
Phenomena, Soft-
ware Programming

Programming experience

What programming language do you master best?

A. Scratch
B. Java
C. Python
D. HTML/CSS/JavaScript
E. none (never programmed before)

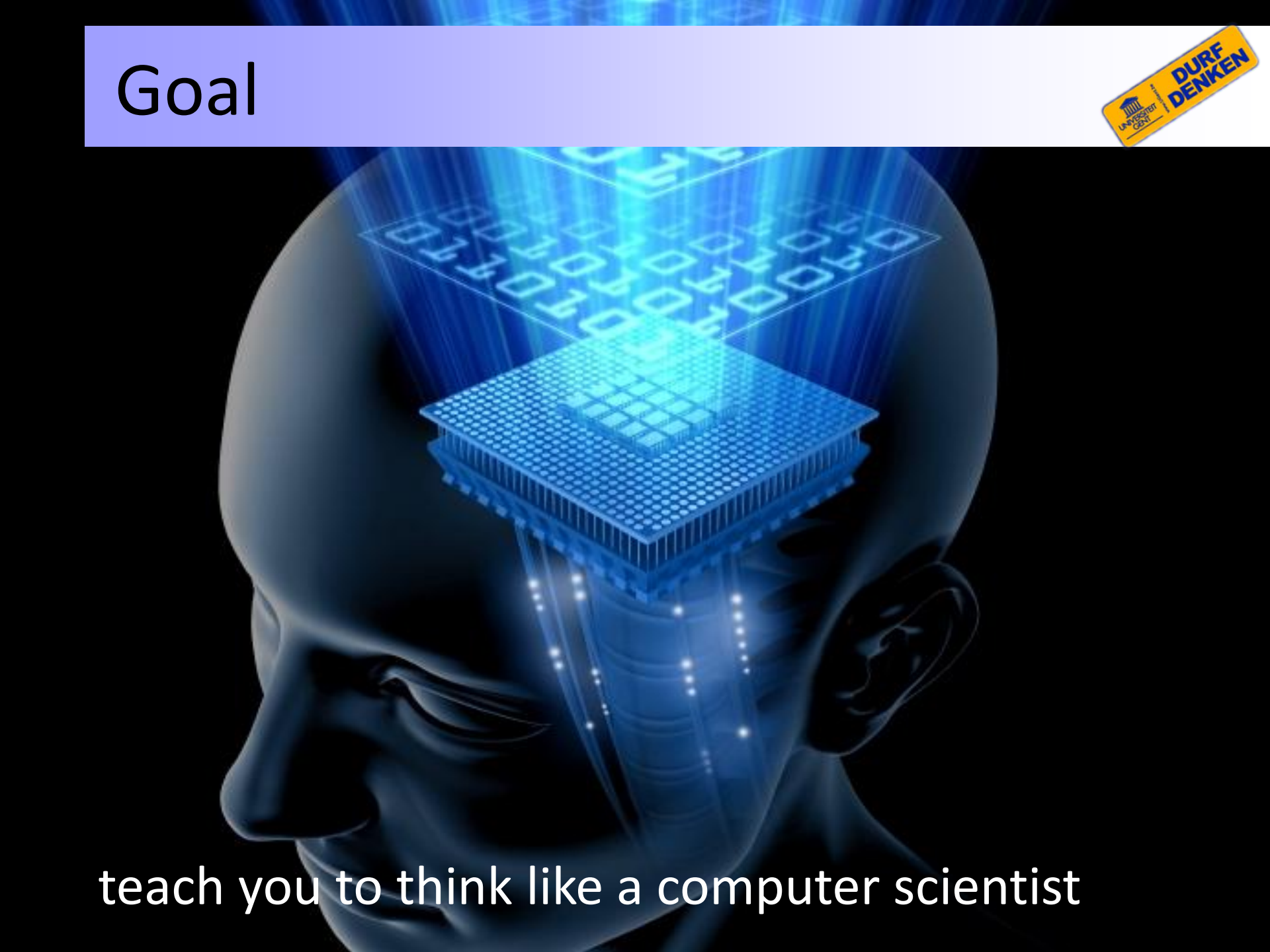
UNIVERSITEIT GENT

socrative

socrative.com

room: 407122

Goal



teach you to think like a computer scientist

Goal

this way of thinking combines best features of

- mathematics
  - use formal languages to denote ideas (computations)
- engineering
  - design things
  - assemble components into systems
  - evaluate tradeoffs among alternatives
- natural science
  - observe behaviour of complex systems
  - form hypotheses
  - test predictions

teach you to think like a computer scientist

Goal

problem solving is the single most important skill for a computer scientist

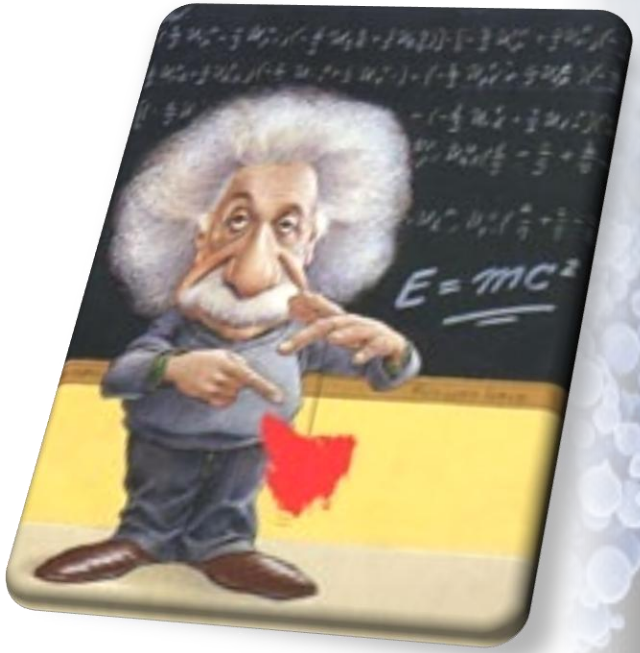
- formulate problems
- think creatively about solutions
- express a solution clearly and accurately

learning to program is all about learning to solve problems

Goal

doing > knowing

learning by doing



UNIVERSITEIT GENT

Design cycle

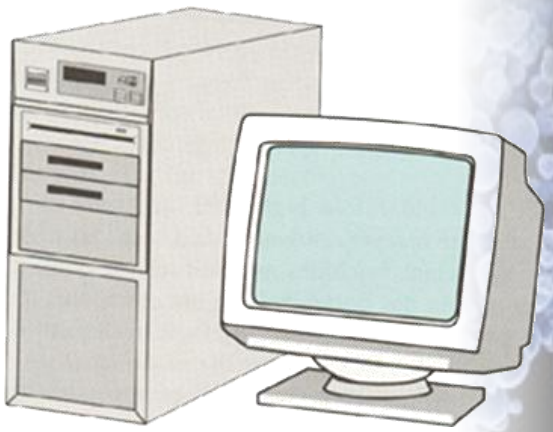
problem solving

natural language
(English, French, Arab, …)

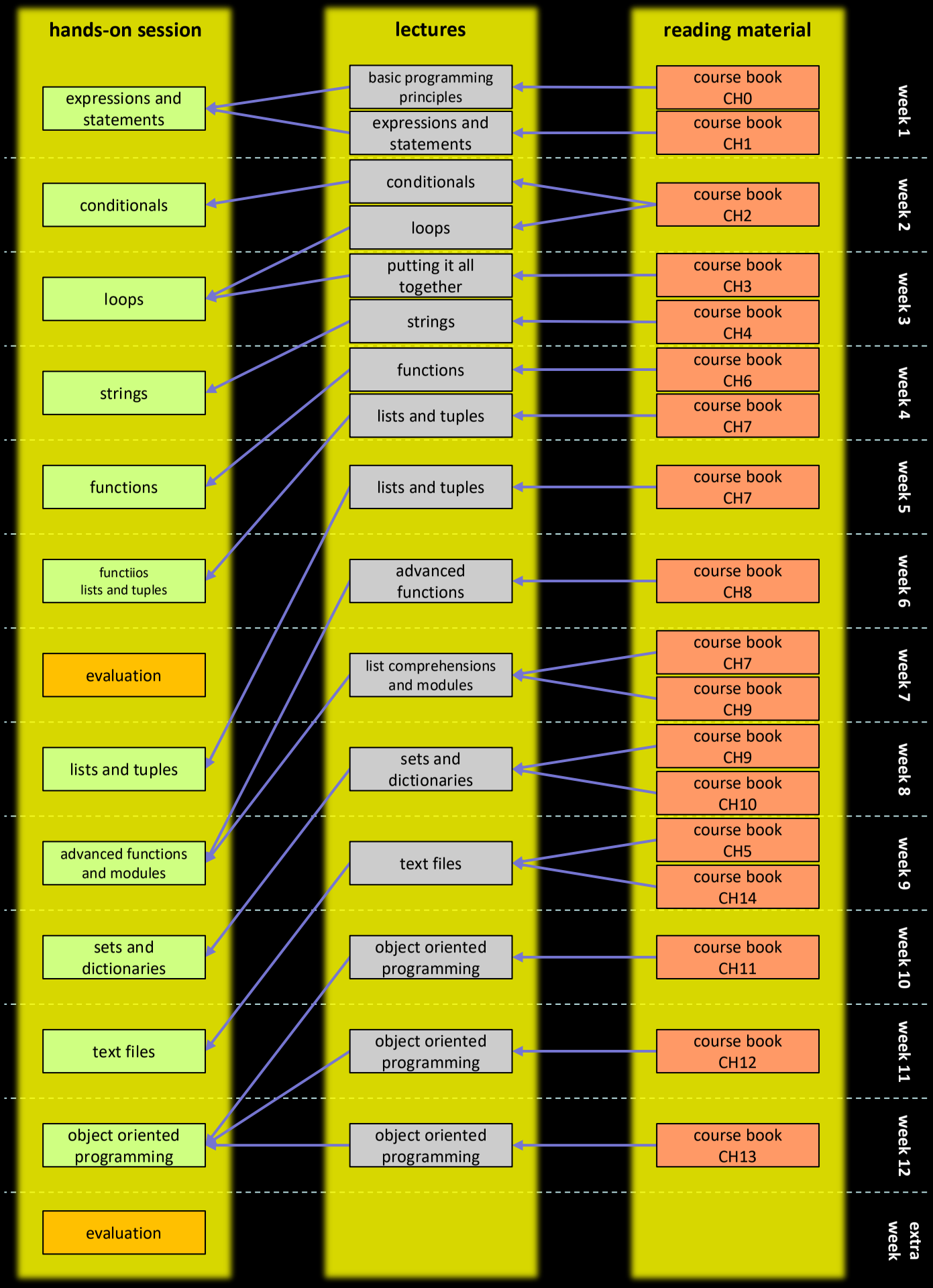
programming language

programming language
(Python, Java, C++, …)

machine language
(Pentium, Xeon, AMD, …)



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

Design cycle

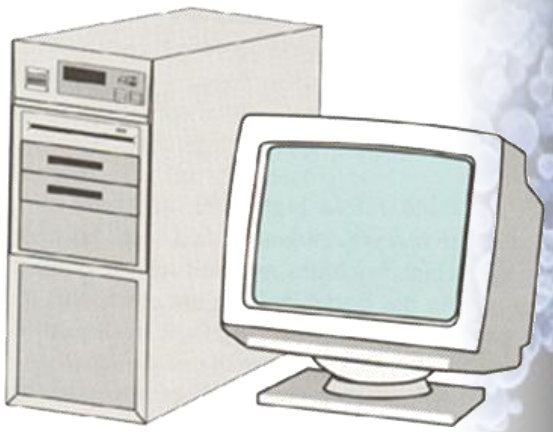
natural language
(English, French, Arab, …)

programming language
(Python, Java, C++, …)

machine language
(Pentium, Xeon, AMD, …)

problem solving

programming language



UNIVERSITEIT GENT

Design cycle

A describe and analyze the problem

- *what should the program do ?*
- *what is the input, and what is the expected output ?*

B design an *algorithm* (*pseudocode*)

- *how to achieve the result ?*

C convert algorithm into *source code*

- following syntax rules of chosen programming language

D compile source code into *machine language*

D execute the program

E trace potentional errors (*debugging*)

UNIVERSITEIT GENT

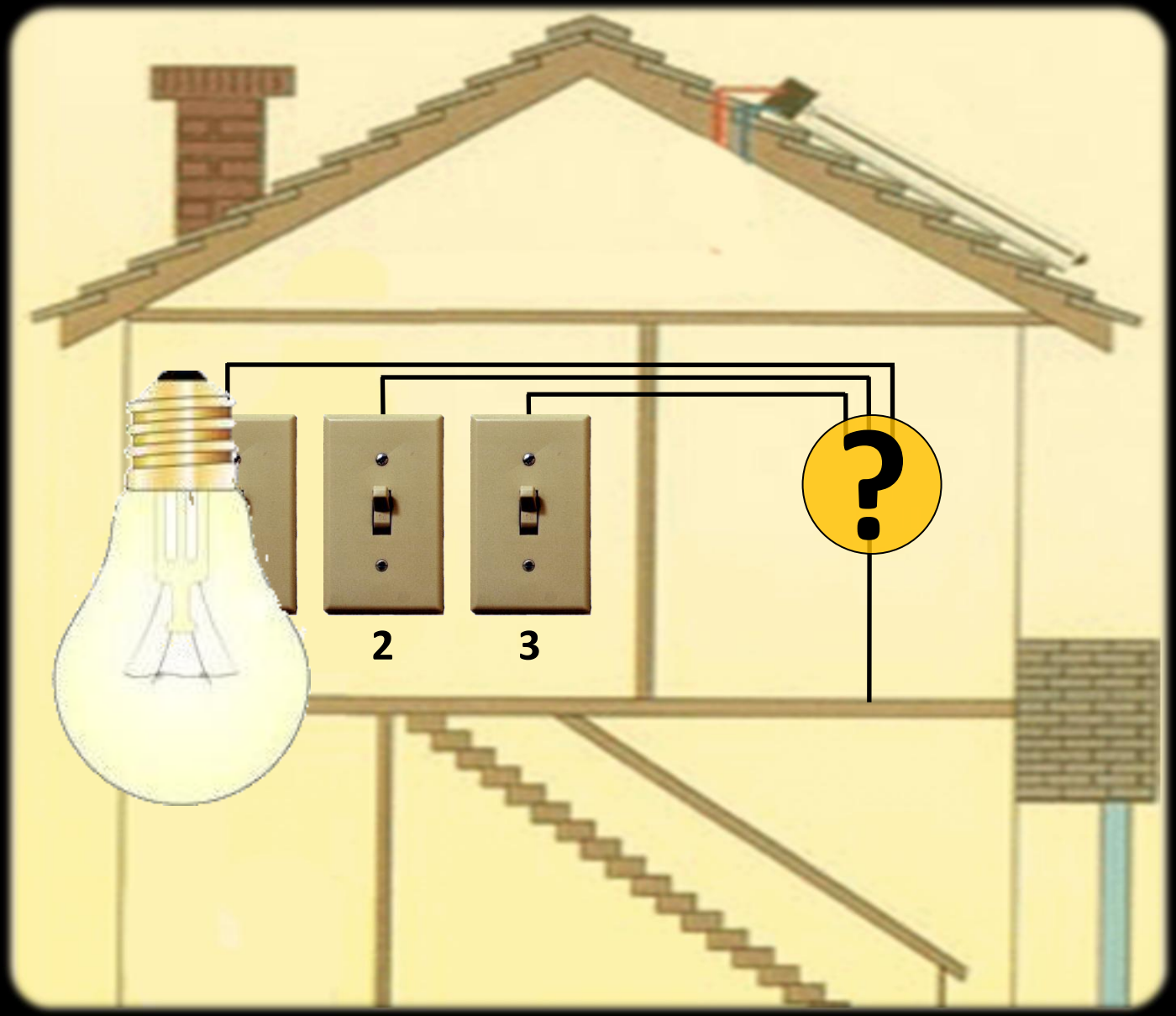
socrative

socrative.com

room: 407122

Algorithm

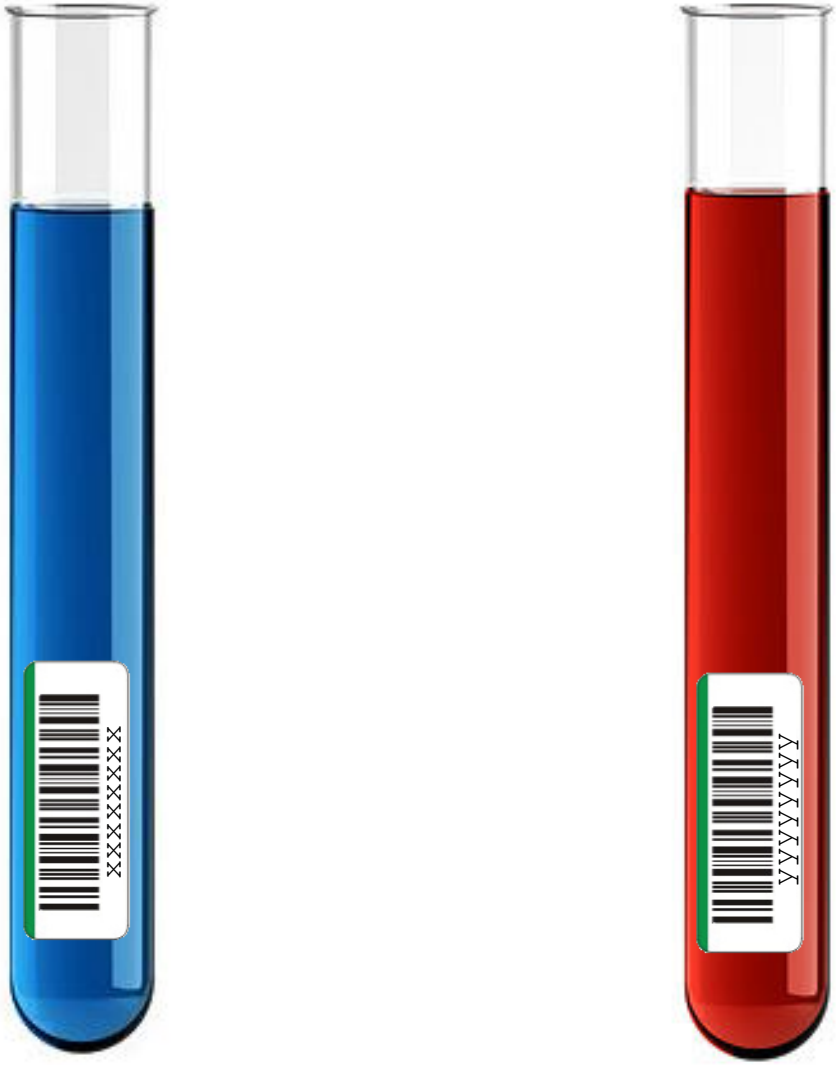
UNIVERSITEIT GENT


2
3
?

A stamp issued September 6, 1983 in the Soviet Union, commemorating Muḥammad ibn Mūsā al-Khwārizmī's (approximate) 1200th birthday. Persian mathematician, astronomer and geographer. The word algorithm refers to Algoritmi, the Latin translation of his name.

PICTURE WIKIMEDIA

Algorithm



Computer programming

Computer programming

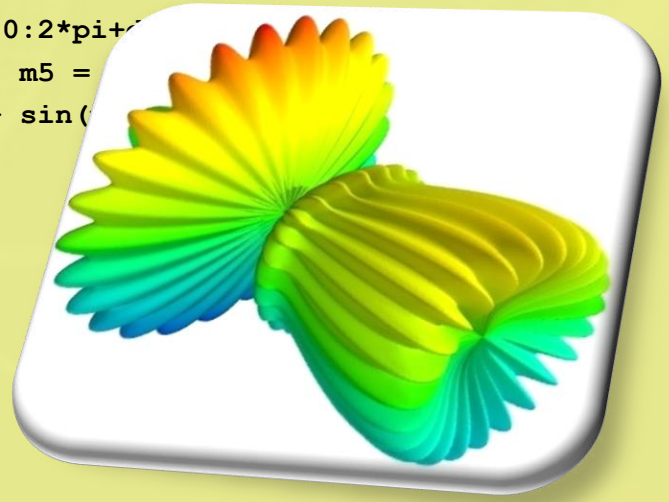
The purpose of programming is to create a set of **instructions** in a **programming language** that computers use to **automatically** perform specific operations or to exhibit desired behaviors.

Programming languages

A **programming language** can be used to create programs that control the behavior of a machine or to express algorithms precisely. Programming languages differ in **syntax** and **grammar** from natural languages, as languages that are used for interaction between people are too complex and full of ambiguities. Text written in a programming language (**source code**) should allow humans to communicate instructions that machines only can interpret in a unique way.


```
# define harmonic function
from numpy import pi, sin, cos, mgrid
dphi, dtheta = pi/250.0, pi/250.0
[phi,theta] = mgrid[0:pi+dphi*1.5:dphi,0:2*pi+
m0 = 4; m1 = 3; m2 = 2; m3 = 3; m4 = 6; m5 =
r = sin(m0*phi)**m1 + cos(m2*phi)**m3 + sin(
      cos(m6*theta)**m7
x = r*sin(phi)*cos(theta)
y = r*cos(phi)
z = r*sin(phi)*sin(theta)

# plot harmonic function
from mayavi import mlab
s = mlab.mesh(x, y, z)
mlab.show()
```



Programming languages

Programming languages

- two types of programming languages
  - high-level languages
    - Python, Java, C/C++, Perl, Ruby
  - low-level languages
    - assembly or machine languages
    - hardware-specific
      - Intel x86 (Pentium) – IBM PC compatibles
      - Motorola/IBM PowerPC – Apple Macintosh; IBM servers
      - Sun SPARC – Sun servers
- computers only execute low-level languages
  - programs written in high-level languages are translated to the machine language of a specific computer system

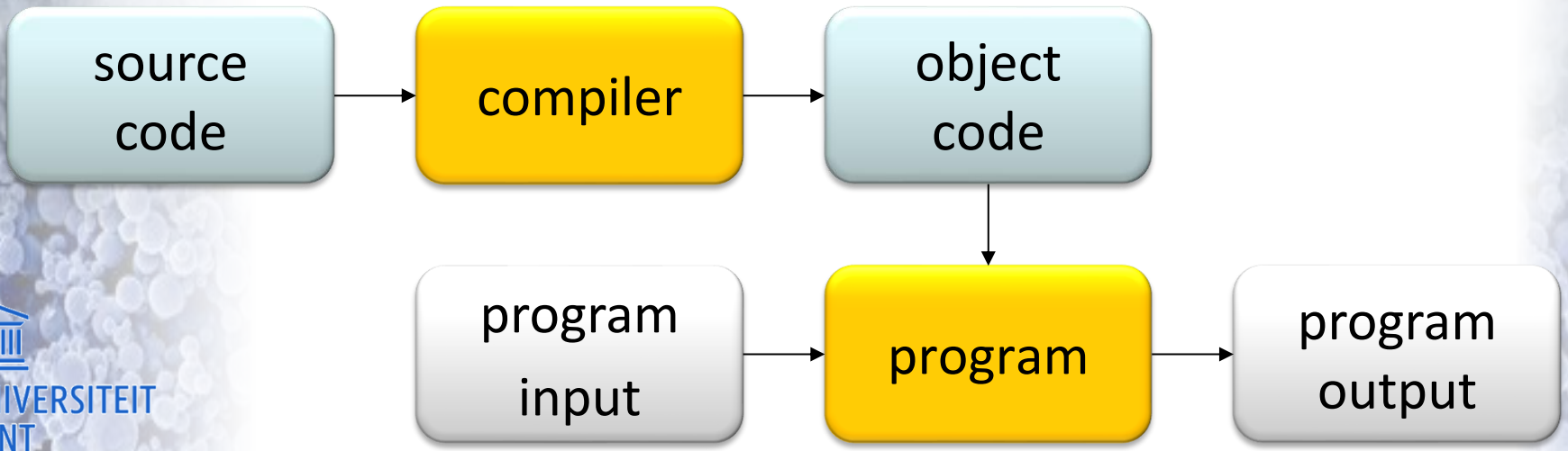
Programming languages

- advantage of low-level programming language
  - program can be tuned to specific computer system
    - maximum execution speed
    - minimum memory consumption
- advantage of high-level programming languages
  - easier to program
    - less time to write
    - shorter and easier to read
    - more likely to be correct (fewer bugs)
  - much easier to port, or modify to run on different computers

today almost all programs are written
in high-level programming languages

Translate source code

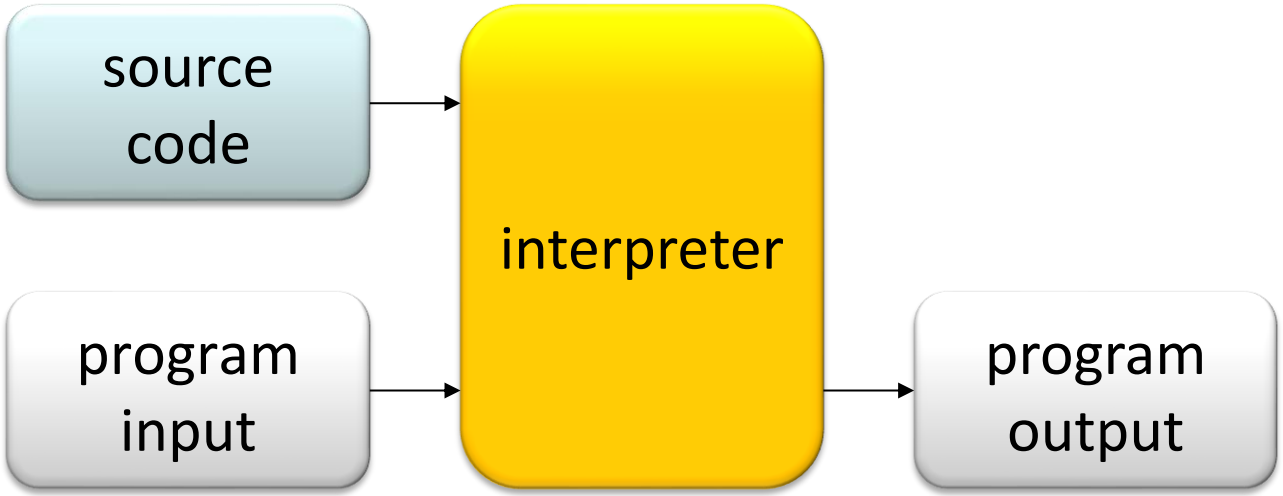
- scenario 1: compiled languages (C, C++)
  - source code translated to machine language by compiler
  - stored as an executable file (*object code*)
  - program loaded from memory and executed



UNIVERSITEIT GENT

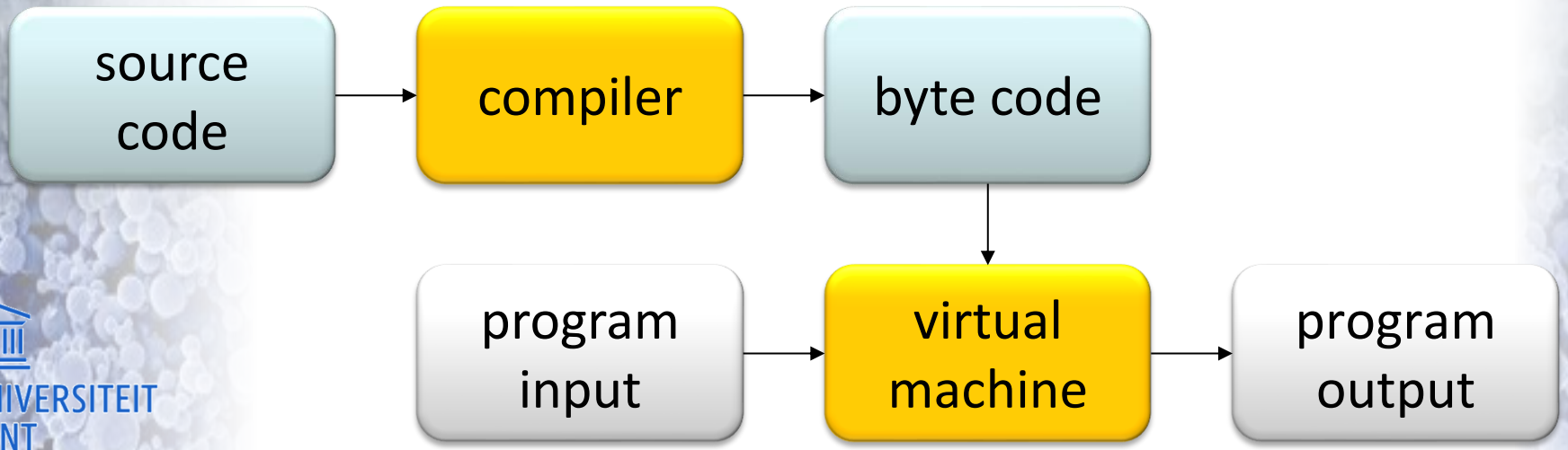
Translate source code

- scenario 2: interpreted languages (Perl, Bash)
  - source code translated to machine language by interpreter
  - interpreter immediately executes object code
  - this happens line by line



Translate source code

- scenario 3: hybrid approach (Python, Java)
  - source code translated to byte code by compiler
  - stored as an "intermediate" file
  - interpreted by separate program on execution
    - Python runtime, Java Virtual Machine (JVM)



UNIVERSITEIT GENT

Python runtime

- may be executed in two different ways
  - interactive session (*shell mode*)
    - enter a Python instruction
    - interpreter immediately executes instruction and returns result

```
$ python3
Python 3.2.3 (default, Apr 11 2012, 07:15:24)
Type "help" for more information.
>>> print(1 + 1)
2
```

UNIVERSITEIT GENT

Python runtime

- may be executed in two different ways
  - interactive session (*shell mode*)
    - enter a Python instruction
    - interpreter immediately executes instruction and returns result
  - non-interactive session (*script mode*)
    - store Python instructions into a text file
    - tell the interpreter to execute is

```
$ cat firstprogram.py
print(1 + 1)
$ python3 firstprogram.py
2
```

UNIVERSITEIT GENT

What is a program?

- a sequence of instructions that specifies how to perform a computation

- examples
  - mathematical
    - solving a system of equations
    - finding the roots of a polynomial
  - symbolic computation
    - searching and replacing text in a document
    - interpreting a program (like the Python interpreter)

What is an instruction?

- different programming languages
  → different instructions (*commands*, *statements*)
- common basic instructions
  - read input
    - get data from the keyboard, a file, or some other device
  - produce output
    - display data on the screen or send data to a file or other device
  - math
    - perform basic mathematical operations like addition and multiplication
  - conditional execution
    - check certain conditions and execute appropriate sequence of statements
  - repitition
    - perform some action repeatedly, usually with some variation

What is programming?

All programs consist of instructions like these. So one way to describe programming is the process of breaking a large, complex task up into smaller and smaller subtasks until eventually the subtasks are simple enough to be performed with one of these simple instructions.

What is debugging?

9/9

1000 " stopped - arctan ✓ 9.037 847 025
9.037 846 ?95 correct
13"uc (032) MP - MC ~~2.130476415 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11.00 test.
Relays changed

1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.

1545 

Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1700 closed down.

My code isn't working :-(

Start here...

Do you get an error when you run the code?

yes → **What type of error do you get?**

no → **Does the code use loops or if statements?**

neither

A variable that should contain a value does not
You are storing the return value of a function which changes the variable itself (e.g. sort)

A number which should be a fraction is coming out as zero in Python 2
You are dividing integers rather than floats. Convert the numbers to floats or from __future__ import division

I'm trying to print a value but getting a weird-looking string
You are printing an object (e.g. a FileObject) when you want the result of calling a method on the object

A regular expression is not matching when I expect it to
You have forgotten to use raw strings or escape backslash characters

I am reading a file but getting no input
You have already read the contents of the file earlier in the code, so the cursor is at the end.

loops

A list which should have a value for every iteration only has a single value
You have defined the list inside the loop: move it outside

A loop which uses the range function misses out the last value
The range function is exclusive at the finish: increase it by one.

I am trying to loop over a collection of strings, but am getting individual characters
You are iterating over a string by mistake

I am trying to write multiple lines to a file but only getting a single one
You have opened the file inside the loop: move it outside

if

Two numbers which should be equal are not
You are comparing a number with a string representation of a number (e.g. if 3 == "3")

A complex condition is not giving the expected result
The order of precedence in the condition is ambiguous - add some parentheses

also check...

Attribute Error
You are calling a method on the wrong type of object

SyntaxError
You've forgotten the quotes around a string

You have forgotten to put a colon at the end of a def/if/for line

You have different number of open and close brackets in a statement

TypeError
You're trying to use an operator on the wrong type of objects

An object which you expect to have a value is actually None

You've used non-integer numbers in a list slice

You've called a method/ function with the wrong number or type of arguments

Indentation Error
You've used a mixture of tabs and spaces

You haven't indented all lines in a block equally

NameError
You've misspelt a variable, function or method name

You've forgotten to import a module

You've forgotten to define a variable

Your code uses a variable outside the scope where it's defined

Your code calls a function before it's defined

You're trying to print a single word and have forgotten the quotes

IOError
You're trying to open a file that doesn't exist

KeyError
You're trying to look up a key that doesn't exist in a dict

http://pythonforbiologists.com

What is debugging?

three kinds of programming errors

- compiler-generated errors (syntax errors)
- errors during code execution (run-time errors)
- logical errors (semantic errors)

What is debugging?

three kinds of programming errors

- compiler-generated errors (syntax error)
  - Python cannot execute a program unless it is syntactically correct
  - otherwise it returns an error message without starting the program
  - syntax refers to program structure and rules about that structure
- example in natural language
  - grammar rule: a sentence must begin with a capital letter and end with a period
    - "this sentence contains a syntax error."
    - "So does this one"

What is debugging?

three kinds of programming errors

- compiler-generated errors (syntax error)
  - Python cannot execute a program unless it is syntactically correct
  - otherwise it returns an error message without starting the program
  - syntax refers to program structure and rules about that structure
- Python example
  - mismatching parenthesis

```
>>> print(5 / 2))
  File "<stdin>", line 1
    print(5 / 2))
                ^
SyntaxError: invalid syntax
```

What is debugging?

three kinds of programming errors

- errors during code execution (run-time error)
  - do not occur until program is run and faulty line is executed
  - called exception because something exceptional (👎) has happened
  - exection of program is usually terminated at this point
  - information about current state of program is printed
- Python example
  - division by zero

```
>>> 5 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

What is debugging?

three kinds of programming errors

- logical errors (semantic error)
  - program runs successfully
    - in the sense that it does not generate error messages
  - but it does not do what you want it to do
    - it returns the wrong result
    - it is doing what you told it to do !!

Debugging logical errors requires you to work backwards from the observed output (if any) to determine what the program is actually doing internally.

Experimental debugging

Experimental debugging

- one of the most important skills you will acquire in this class is debugging
  - some programmers believe debugging is one of the most intellectually rich, challenging, and interesting parts of programming
  - some students soon enough associate debugging with a strong feeling of frustration

Experimental debugging

- debugging is like detective work
  - you are confronted with clues
  - you have to infer the processes and events that lead to the results you see

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth"

— Sherlock Holmes
uit The Sign of Four (Arthur Conan Doyle)

Experimental debugging

- debugging is also like an experimental science
  - form hypothesis about cause of error
  - modify program, and predict new outcome
  - if results match prediction, hypothesis is correct
  - otherwise, modify hypothesis

Formal and natural languages

- natural languages
  - spoken by people (like English, French and Arab)
  - not designed by people; they evolved naturally
- formal languages
  - designed by humans for specific applications
    - mathematicians
      - mathematical notation: denotes relationships between numbers and symbols
    - chemists
      - chemical notation: represents chemical structure of molecules
    - computer science
      - programming language: designed to express computations

Formal and natural languages

- natural languages
  - spoken by people (like English, French and Arab)
  - not designed by people; they evolved naturally
- formal languages
  - tend to have strict rules about syntax
    - **3+3=6** is a syntactically correct mathematical statement
    - **3=+6$** is not
    - **H_20** is syntactically correct chemical name
    - **$_2Zz$** is not

UNIVERSITEIT GENT

Formal languages

- two types of syntax rules
  - token rules
    - tokens are basic elements of the language
      - words
      - numbers
      - chemical elements
    - in **3=+6$**, **$** is not a legal token in mathematics
    - in $_2$**Zz**, there is no chemical element with abbreviation **Zz**
  - structure rules
    - the way tokens are arranged into a statement
    - in **3=+6$**, plus is not allowed to follow equal signs
    - in $_2$**Zz**, subscripts must come after the element name, not before

UNIVERSITEIT GENT

Formal languages

- two types of syntax rules
  - token rules
  - structure rules
- process of determining the structure of a sentence in a natural or formal language is called *parsing*
  - example: "The other shoe fell."
    - "the other shoe" is the subject
    - "fell" is the verb
- after parsing a sentence you can determine the meaning or *semantics* of the sentence

Formal and natural languages

- formal and natural languages share
  - words
  - structure
  - syntax
  - semantics
- difference between formal and natural languages compares to difference between prose and poetry

UNIVERSITEIT GENT

Formal and natural languages

- formal and natural languages differ on
  - ambiguity
    - natural languages are full of ambiguity; people use contextual clues and other information
    - formal languages are designed to be unambiguous; each statement has exactly one meaning
  - redundancy
    - natural languages are redundant to make up for ambiguity and to reduce miscommunications
    - natural languages are verbose
    - formal languages are less redundant and more concise
  - literalness
    - natural languages are full of idiom and metaphor: no shoe, no falling
    - formal languages mean exactly what they say

UNIVERSITEIT GENT

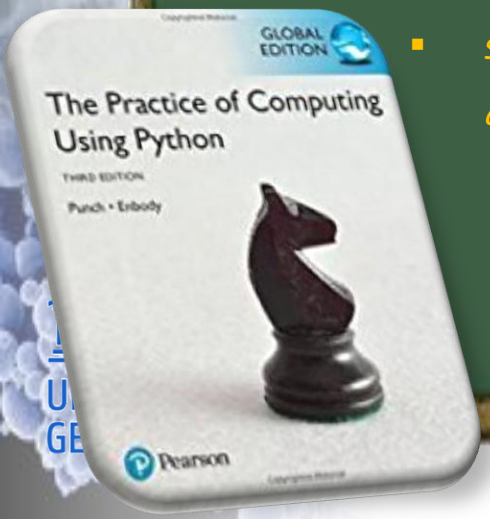
Hello, World !


```
$ cat HelloWorld.py
print('Hello, World!')
$ python3 HelloWorld.py
Hello, World !
```

UNIVERSITEIT GENT

Homework (hands-on sessions)

- course book: read chapters 0 and 1
  - in preparation of the next hands-on session
  - Punch W & Enbody R (2017, global edition)
    The practice of computing using Python. Addison-Wesley
    ISBN-13: 978-1-29216-662-9
    - PDF-version available on Ufora
    - second and third edition of the book can still be used as it also covers Python 3



Homework (hands-on sessions)

- video tutorials
  - first Python program in PyCharm
  - interactive Python sessions in PyCharm
- check practical information
  - Ufora
  - Dodona

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

Teaching methods

- independent work
  - detailed planning in Dodona course
  - read course book: essential preparation for lectures
  - additional preparatory tasks: essential preparation for hands-on sessions
  - mandatory exercises: essential for learning to code
  - try to solve as many exercises as possible before the hands-on sessions
  - ask questions via Dodona, Ufora, or during lectures and hands-on sessions
  - good preparation is the best way to learn to code

UNIVERSITEIT GENT

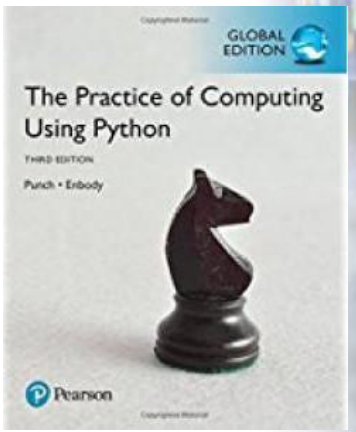
Teaching methods

- lectures
  - programming techniques explained through examples
  - essential preparation: read course book, read in-class assignments
  - opportunity to think along and ask questions !!!
  - arrive in time and turn off your cell phone

- hands-on sessions
  - work on mandatory programming exercises in PC-room
  - use course book and Python documentation as reference material
  - use your own laptop to solve exercises (BYOD)
  - opportunity to ask questions to peers and teaching assistants
  - collaboration between students allowed (max 3 students per group)

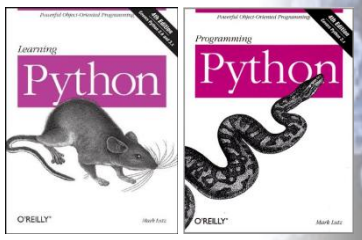
UNIVERSITEIT GENT

Study material



- course book
  - Punch W & Enbody R (global edition, 2017). The practice of computing using Python. Addison-Wesley, 978-1-29216-662-9.

- reference books
  - Mark Lutz (2009). Learning Python: Powerful Object-Oriented Programming (4th edition). O'Reilly Media, 978-0596158064.
  - Mark Lutz (2011). Programming Python (4th edition). O'Reilly Media, 978-0596158101.



UNIVERSITEIT GENT

Study material

- powerpoint slides
  - shared on Ufora
  - please report any errors or ambiguities
  - updated versions of slides may replace older ones
- background material
  - additional reading material
  - shared on Ufora
- reference material
  - help function in interactive session (built-in documentation)
  - The Python Tutorial
  - The Python Standard Library

UNIVERSITEIT GENT

Contact hours

- lectures
  - join one of these two sessions
    - initial assignment in TimeEdit based on your study program
    - *feel free* to join another sessions (no need to ask permission to switch sessions)
    - available for all lectures: livestream + recording (see Ufora Calendar)
  - Dutch track
    - Monday 11:30 – 12:45 (week 1-4)
      - Campus Sterre, S9 building, auditorium A2 (second floor)
    - Thursday 08:30 – 09:45 (week 1-12)
      - Campus Sterre, S9 building, auditorium A0 (ground floor)
  - English track
    - Monday 14:30 – 17:15 (week 1-12)
      - online (MS Teams)

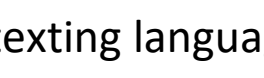
UNIVERSITEIT GENT

Contact hours

- hands-on sessions
  - join one of these two sessions
    - initial assignment in TimeEdit based on your study program
    - *feel free* to join another sessions (no need to ask permission to switch sessions)
  - Friday 10:00 – 12:15 (week 1-12)
    - Campus Boekentoren, room 0.9 (ground floor + floor -1)
    - study programs: biochemistry & biotechnology, biology, geography, pharmaceutical engineering
  - Friday 14:30 – 17:15 (week 1-12)
    - Campus Boekentoren, room 0.9 (ground floor + floor -1)
    - study programs: chemistry, geology, physics & astronomy, statistical data analysis

UNIVERSITEIT GENT

Study coaching

- all "ad valvas" announcements via Ufora
- ask questions and make comments
  - during lectures and hands-on sessions
  - via Ufora discussions
  - immediately after lectures
  - during a personal appointment (arrange via email)
    - do not send emails from an anonymous email account
    - always use your Ugent email account
    - always mention your name, study program and related course
    - avoid typos, spelling mistakes and texting language ()
    - use clear language and a correct form of address

UNIVERSITEIT GENT

Study coaching: lectures

- lecturer
  - Prof. Dr. Peter Dawyndt
    - Department of Mathematics, Computer Science and Statistics
    - office: Campus Sterre, Krijgslaan 281, S9 building
    - phone: +32 9 264 47 79
    - email: peter.dawyndt@ugent.be
  - research topics
    - computational biology (bioinformatics)
    - computer science education

UNIVERSITEIT GENT

Study coaching: hands-on sessions

Maarten Stevens
maarten.stevens@ugent.be

Thomas Van Mullem
thomas.vanmullem@ugent.be

Rien Maertens
rien.maertens@ugent.be

Mustapha Regragui
mustapha.regragui@ugent.be

Vincent Batens
vincent.batens@ugent.be

Emma Vandewalle
emmavdwa.vandewalle@ugent.be

Tibo De Peuter
tibo.depeuter@ugent.be

Wouter De Bolle
wouter.debolle@ugent.be

UNIVERSITEIT GENT

Evaluations

- mandatory programming exercises
  - put theory into practice
  - weekly deadlines
    - series 01 – series 05: Tuesday 22:00
    - series 06 – series 10: Tuesday 22:00
- evaluations
  - solve two exercises in two hours
  - follows same procedure as exam
  - lower level of difficulty compared to exam: tests basic skills
- other programming exercises
  - non-mandatory
  - learning to code = practice, practice, practice, …

UNIVERSITEIT GENT

Evaluations

- non-periodic evaluations count for 4/20
  - triple profit !!!
  - cannot be taken again for resit exam
- evaluation score takes into account mandatory exercises solved before weekly deadlines
- assessment
  - solutions for mandatory exercises before weekly deadlines
  - solutions for two new exercises during evaluation
  - feedback
    - sample solutions shared after weekly deadlines
    - score and personal feedback shared shortly after evaluation

UNIVERSITEIT GENT

Questions or remarks ?

UNIVERSITEIT GENT

The sky is the limit …

"Whatever you thought, think again"

— National Geographic Magazine