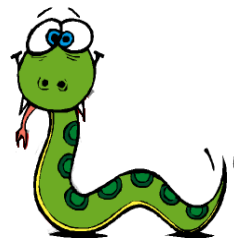


# Computer Programming

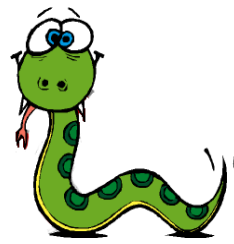


Course Details

An Introduction to Computational Tools

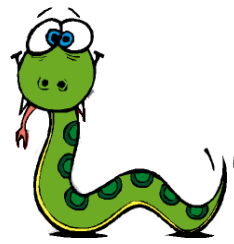
Prof. Mauro Gaspari: [mauro.gaspari@unibo.it](mailto:mauro.gaspari@unibo.it)

# Road map for today



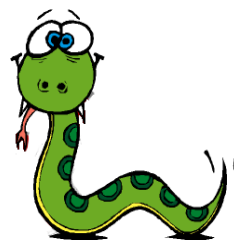
- The skills that we would like you to acquire: to think like a computer scientist.
- Course organization.
- Exams.
- Course material.
- An introduction to computational tools.
- Getting started.

# To think like a Computer Scientist.....



- To understand computational tools and terminology.
- To use these basic tools to write small scale programs.
- To understand programs written by others.
- To understand both capabilities and limitations of computational tools.
- To map (qf) problems into computational solutions.

# Python



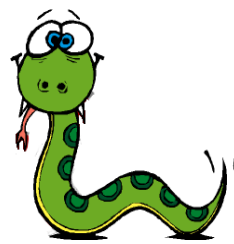
- Computational solution will be implemented in **Python** a modern programming language.
- This course is not about Python (e.g. studying Python details), but it is about using it to think and build computational solutions.
- Warning!!! reading the textbook is not enough, exercises are fundamental.



# Course Organization

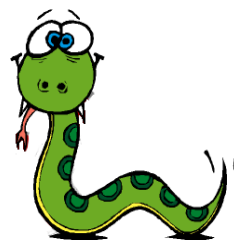
- Interactive approach - learning by doing
- **Interactive Lectures:**
  - **Exercises** solved in class (workgroup sessions)
  - **Individual support** (tutor: to be announced)
- **Home assignments** (warning home work is mandatory for class students!!!)

# Course Material



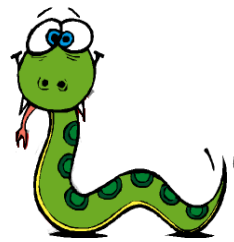
- TextBook: **How to Think Like a Computer Scientist: Learning with Python**, by A. Downey, J. f Elkner and C. Meyers. Green Tea Press (available online here: <https://media.readthedocs.org/pdf/howtothink/latest/howtothink.pdf>).
- Web site: <http://www.cs.unibo.it/~gaspari/www/teaching/>
  - Slides
  - Readings and Links
- Online software and documentation:
  - Python site: <http://www.python.org/>
  - There is an Italian version too: <http://www.python.it/>

# Exam



- The Exam of Computer Programming is composed by two parts:
  - A WRITTEN TEST
  - FOR QF STUDENTS ONLY: A DISCUSSION OF AN INDIVIDUAL (OPTIONAL) PROJECT.
- The final score is the media of the two parts, otherwise is the result of the written test.
- Passing the WRITTEN TEST is mandatory for STARTING a PROJECT and for discussing it.
- Students that passed the Crash Course Test can start their projects at the end of the course (thus before the written test).
- Projects are discussed by appointment. If the project discussion fails, namely the project is not original and/or the student is not able to change its source code, the written test must be repeated.

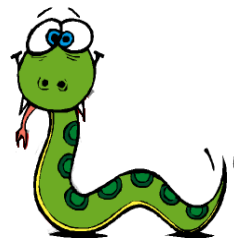
# Home Work



- **Homework are assigned to all the students at the end of each week** (Wednesday). They should be completed before the first lesson of the next week (11am Thursday).
- **Students can contact us for questions and/or problems** with the proposed exercises.
- Homework is essential for class students to follow the next lecture.

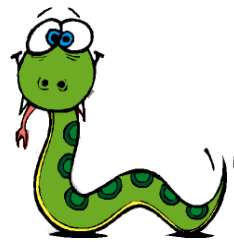


# Discussion of a project



- **Procedure:**
  - **Propose a (qf) problem and submit a specification** of what the program will do.
  - When approved **implement a computational solution.**
  - **Projects code must be consigned the day before the discussion** sending the source code by email to [mauro.gaspari@unibo.it](mailto:mauro.gaspari@unibo.it).
- **Discussion:**
  - **A demo of your project + a practical test (modify your code).**

# Vote

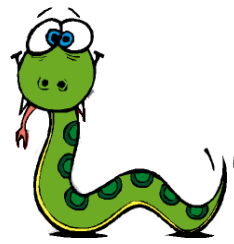


- For each student  $s$ :

$$\mathbf{Vote}(s) = (\mathbf{Written}(s) + \mathbf{Project}(s))/2$$

- Where:
  - **Written**( $s$ )  $\Rightarrow$  is the written vote
  - **Project**( $s$ )  $\Rightarrow$  is the result of the project discussion.

# Computational tools and problem solving



- Computational tools are used for problem solving.
- Humans are problem solvers: we use knowledge to solve problems.
- What is knowledge?
- We can divide knowledge into at least two categories:
  - Declarative knowledge
  - Imperative knowledge

# Example: Greatest Common Divisor



- **DECLARATIVE KNOWLEDGE:**

A, B are positive integer,  $X = \text{GCD}(A,B)$  then:

- The remainder of both  $A/X$  and  $B/X$  is equal to 0 and,
- If exists  $Y \neq X$  such that the remainder of both  $A/Y$  and  $B/Y$  is equal to 0 then  $X > Y$

- **IMPERATIVE KNOWLEDGE (EUCLID)**

IF  $A < B$ , exchange A and B.

Divide A by B and get the remainder, R. If  $R=0$ ,  $\text{GCD}(A,B) = B$ .

Replace A by B and replace B by R. Return to the previous step.

- **IMPERATIVE KNOWLEDE IS THE RECIPE!**

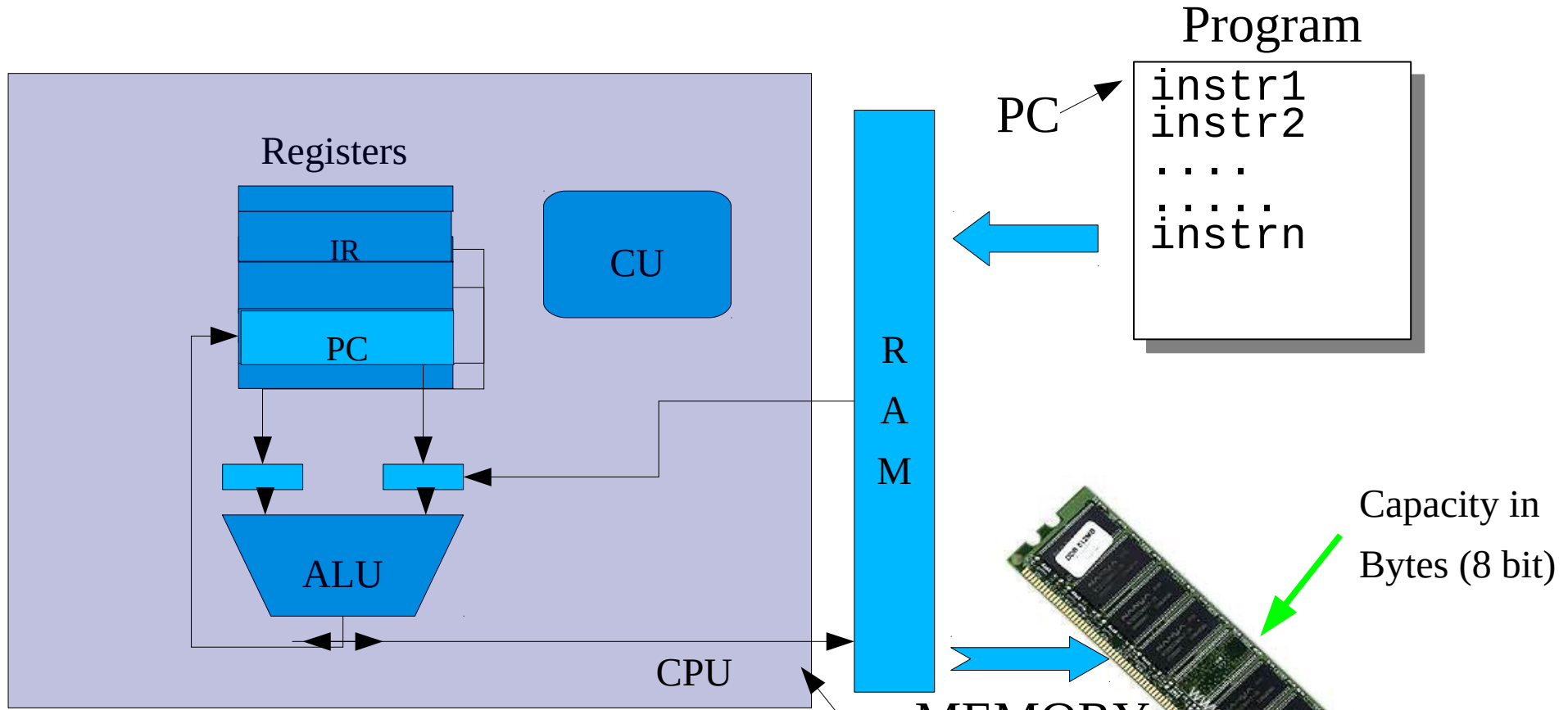
# What is a Computation?



- A computation is a recipe: a sequence of (simple) instructions (this is also called program).
- Is it possible to build a mechanical process to capture that set of computations?
- Yes, for example building a simple circuit to do this: fixed-program computer (for example a dishwasher program).
- **General purpose computer** are more complex, they are circuit able to take a “recipe” (list of simple instructions) as an input, reconfigure themselves and act as the recipe.



# How they work?



Von Neumann Machine



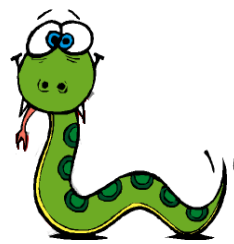
Instructions



Speed in Hertz

How many instructions in a second

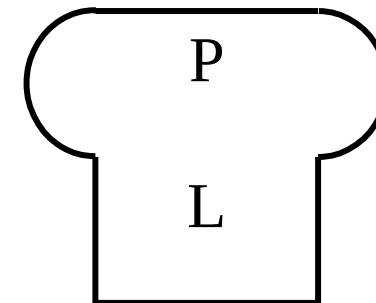
# What is a program?



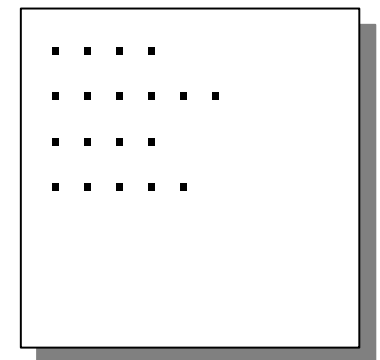
- Programs are written using programming languages.
- **Source code:** instructions of a given program.

## Program Tombstone Diagrams

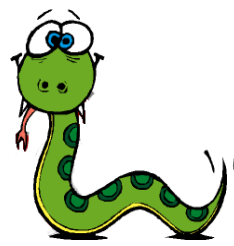
A program P written  
In a Language L



Source code of  
a program P written  
In language L is  
usually stored in  
file and loaded in  
memory



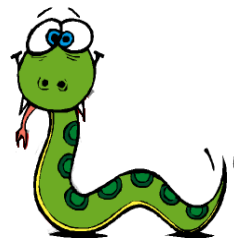
# Which Programming Languages?



- **High level Languages**
  - Python, C, C++, Perl, Java, Lisp, Prolog.
- **Thus there is also a low-level**
  - Machine language
  - Assembler



# High-level vs low-level



High-level  
Languages

```
print b*h/2  
....
```

Low-level  
Languages

```
LOAD r1,b  
LOAD r2,h  
MUL r1,r2  
DIV r1,#2  
RET
```

Assembler

```
00010010010001010010  
01001110110010101101  
001..
```

Machine  
Language

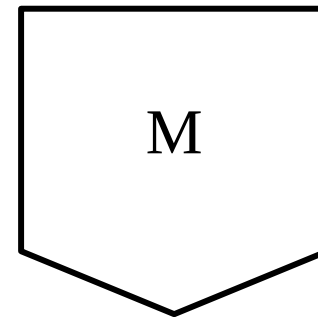


# Machine Language

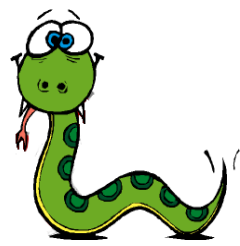
- Computer are only able to execute machine language instructions.
- Different computers may have different machine languages.

## **Computer Tombstone Diagram**

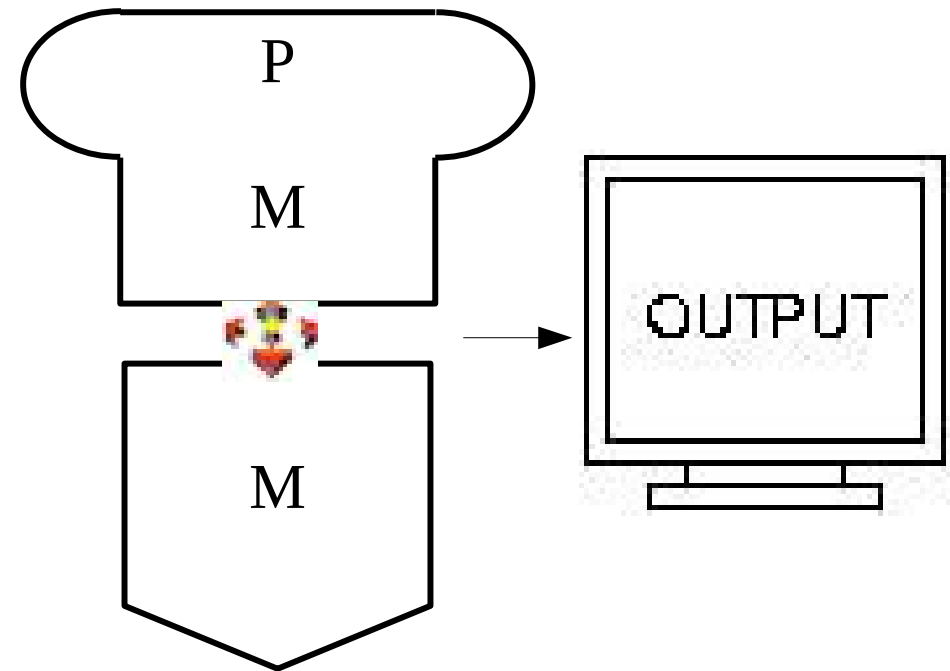
A computer having a machine language M is represented with the following tombstone diagram



# Program execution



- A computer can only execute programs written in its machine language



# Problems of machine languages



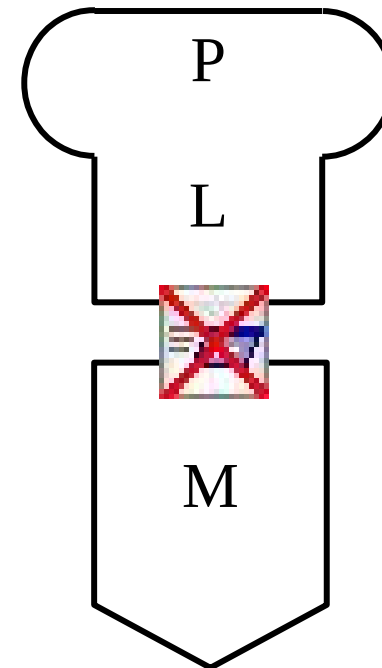
- Write programs in machine language is “almost” impossible.
- The first computers were programmed in this way.
- Now low-level languages are used only for a few specialized applications.
- Almost all programs are written in high-level programming languages.
  - Rapid development
  - Portable code



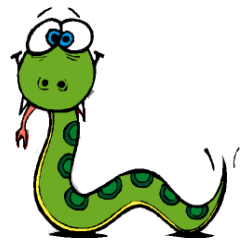
# However

- Apparently high level languages cannot be directly executed on computers.

A program P written in L cannot be executed on a machine M.



# Programming Language Processors



- Specific tools to support the execution of high-level programming languages:
  - Interpreters.
  - Compilers.

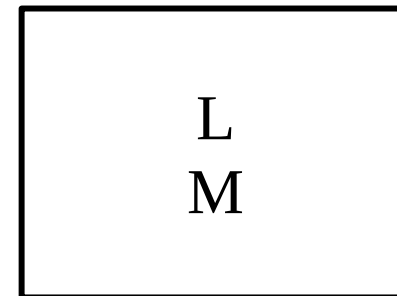


# Interpreter

- An interpreter for a language  $L$  is a program that takes as input a program written in  $L$  and executes it instruction by instruction.
- An interpreter can be written in machine language.

## Interpreter Tombstone Diagram

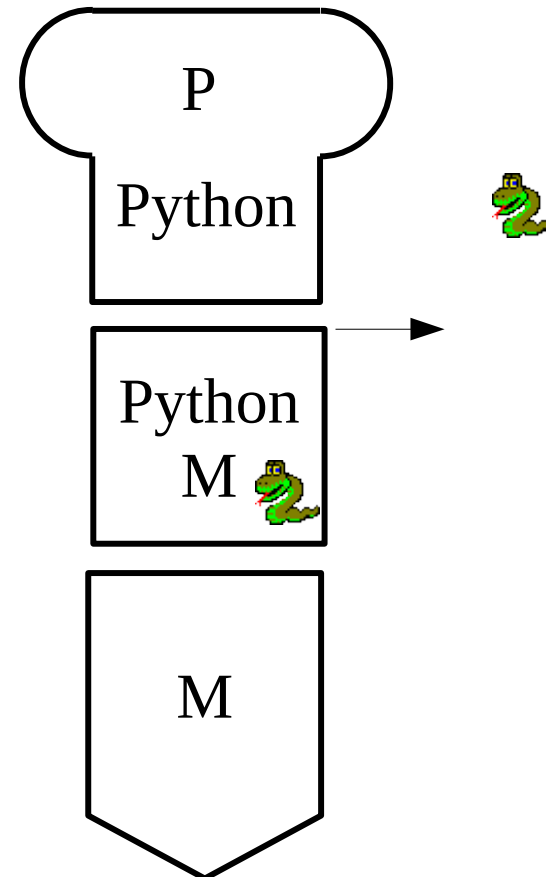
An interpreter for a language  $L$  written in  $M$ .



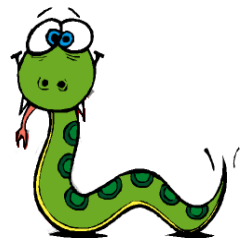
# Executing Python programs



- The python interpreter allows a user to execute a program P written in Python in computer M.
- **This means that Python an interpreted language.**



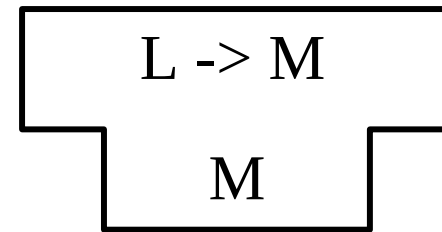




# Compilers

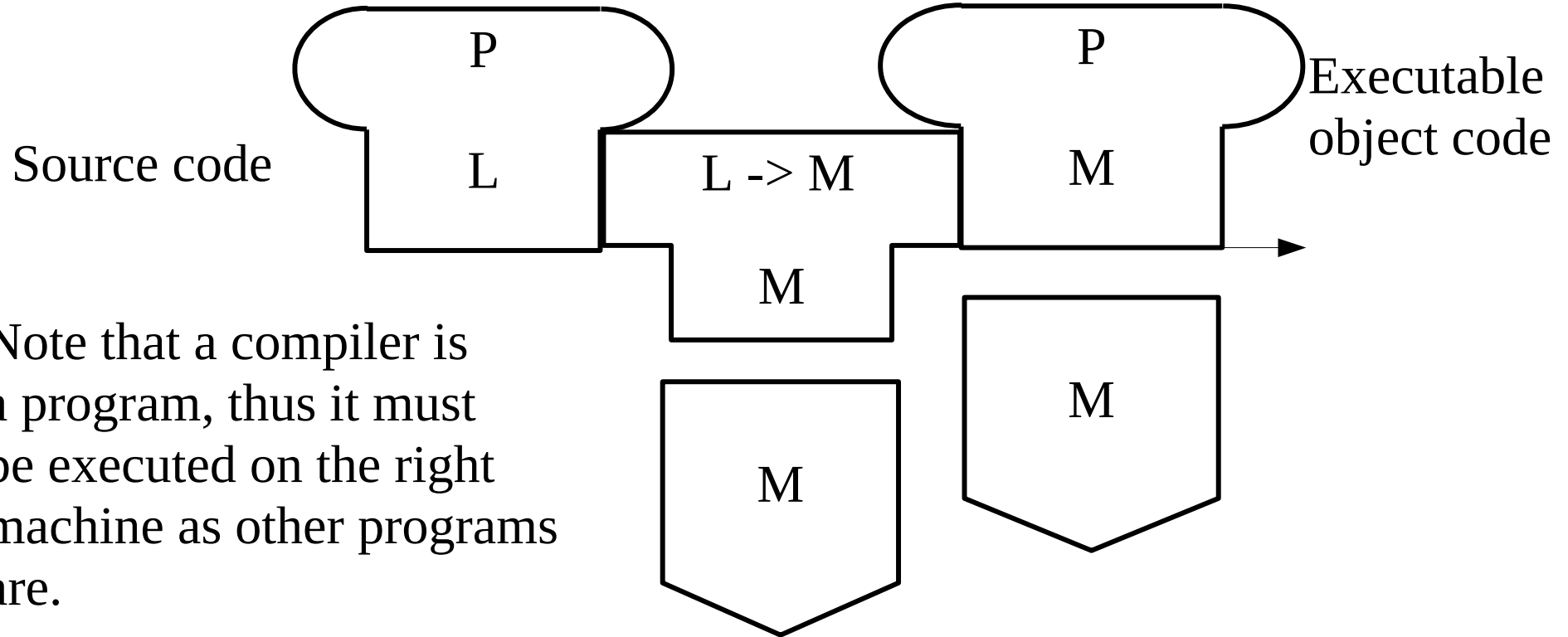
- A compiler is a program which translates source code (usually written in a given high level language) into object code usually written in machine language.

**Compiler Tombstone Diagram**  
A compiler which translates source code written in a language L into object code written in machine language M.

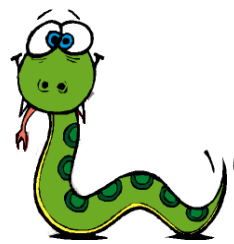




# Using a compiler






# Instructions

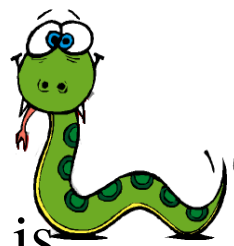


- **input:** Get data from the keyboard, a file, or some other device.
- **output:** Display data on the screen or send data to a file or other device.
- **math:** Perform basic mathematical operations like addition and multiplication.
- **conditional execution:** Check for certain conditions and execute the appropriate code.
- **repetition:** Perform some action repeatedly, usually with some variation.
- **Programming** = the process of breaking a complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with one of these basic instructions.

# Bugs and Debugging!



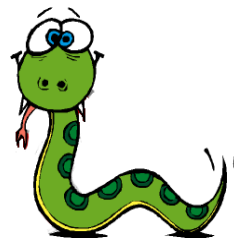
- Bugs 
  - Debugging (the process of tracking down bugs) 
  - Why we introduce this concept here?
    - Is an essential skill associated to computer programming.
-  In principle all the programs may have bugs!
- Bugs are frequent in programming, but they can be solved with debugging!



# Different kinds of errors

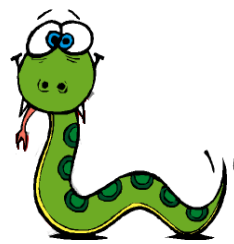
- **Syntax errors:** Python can only execute a program if the syntax is correct; otherwise, the interpreter displays an error message. Syntax refers to the structure of a program and the rules about that structure. If there is a single syntax error anywhere in your program, Python will display an error message and quit.
- **Runtime errors:** these errors do not appear until after the program has started running. These errors are also called **exceptions** because they usually indicate that something exceptional (and bad) has happened.
- **Semantic errors:** If there is a semantic error in your program, it will run successfully (without error messages), but it will not do the right thing. It will do something else. The meaning of the program (its semantics) is wrong. Identifying semantic errors can be very tricky.

# Debugging



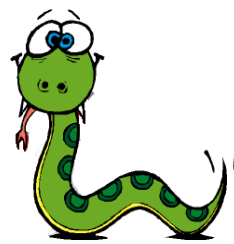
- **Debugging is like detective work:** you are confronted with clues, and you have to infer the processes and events that led to the results you see.
- **Experimental science;**
  - you have an idea about what is going wrong,
  - you modify your program and try again.
  - If your hypothesis was correct, then you can predict the result of the modification, and you take a step closer to a working program.
  - If your hypothesis was wrong, you have to come up with a new one.
- Sherlock Holmes “*When you have eliminated the impossible, whatever remains, however improbable, must be the truth.*”
- **Programming => debugging:** programming is the process of gradually debugging a program until it does what you want.

# Formal languages



- **Natural languages** are the languages people speak, such as English, Spanish, and French. They were not designed by people they evolved naturally.
- **Formal languages** are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols.
- **Programming languages** are formal languages that have been designed to express computations.

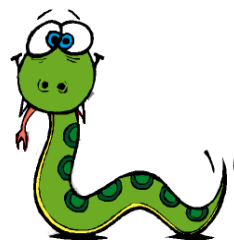
# Syntax



- Formal languages tend to have strict rules about syntax.
  - For example,  $3 + 3 = 6$  is a syntactically correct mathematical statement, but  $3+ = 3\$6$  is not.
- Two levels of **syntax rules**:
  - **Tokens** are the basic elements of the language, such as words, numbers, and chemical elements. One of the problems with  $3+ = 3\$6$  is that  $\$$  is not a legal token in mathematics.
  - **Structure of a statement**: the way the tokens are arranged. The statement  $3+ = 3$  is illegal because even though  $+$  and  $=$  are legal tokens, you can't have one right after the other.
- **Parsing**: is the process that analyse a sentence finding and understanding its structure.

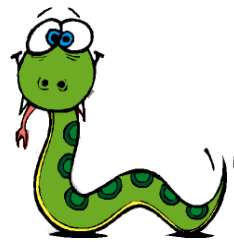


# Formal Languages vs Natural Languages



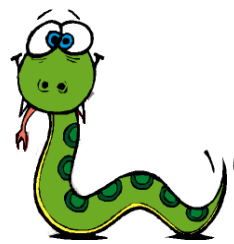
- Although formal and natural languages have many features in common—tokens, structure, syntax, and semantics—there are some differences:
  - **ambiguity:** Natural languages are full of ambiguity, which people deal with by using contextual clues and other information. Formal languages are designed to be nearly or completely unambiguous.
  - **redundancy:** In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose. Formal languages are less redundant and more concise.
  - **literalness:** Natural languages are full of idiom and metaphor. Formal languages mean exactly what they say.

# Semantics



- Semantics: concerns the meaning of instructions, thus the meaning of the constructs of the language.
- Semantics of programming languages:
  - Informal semantics (manual).
  - Formal semantics (computer scientists):  
useful for building programming languages processors.

# The python interpreter

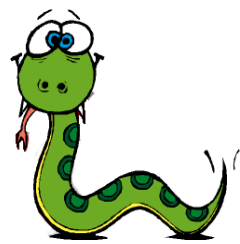


- We will use version 3.7.
- Python is an interpreted language. There are two ways to use the interpreter:
- **Interactive mode:** you type Python programs and the interpreter displays the result:

```
>>> 1 + 1
```

```
2
```

- **Script mode:** you can store code in a file and use the interpreter to execute the contents of the file (script). By convention, Python scripts have names that end with `.py`.



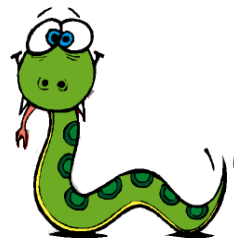
# Script execution

- To execute the script, you have to tell the interpreter the name of the file. If you have a script named `hello.py` and you are working in a UNIX command window, you type:

```
python hello.py
```

- In other development environments, the details of executing scripts are different. Environment based on graphical user interfaces usually have a specific “run” button or menu item to execute the script.
- You can find instructions for your environment at the Python website <http://python.org>

# The first program



- Traditionally, the first program you write in a new language is called “Hello, World!” because it just prints this sentence.
- In Python, the implementation is very simple:  

```
print('Hello, World!')
```
- This is an example of a **print** statement, which displays a value on the screen (**it is an output statement**).
- The quotation marks in the program mark the beginning and end of the text to be displayed.