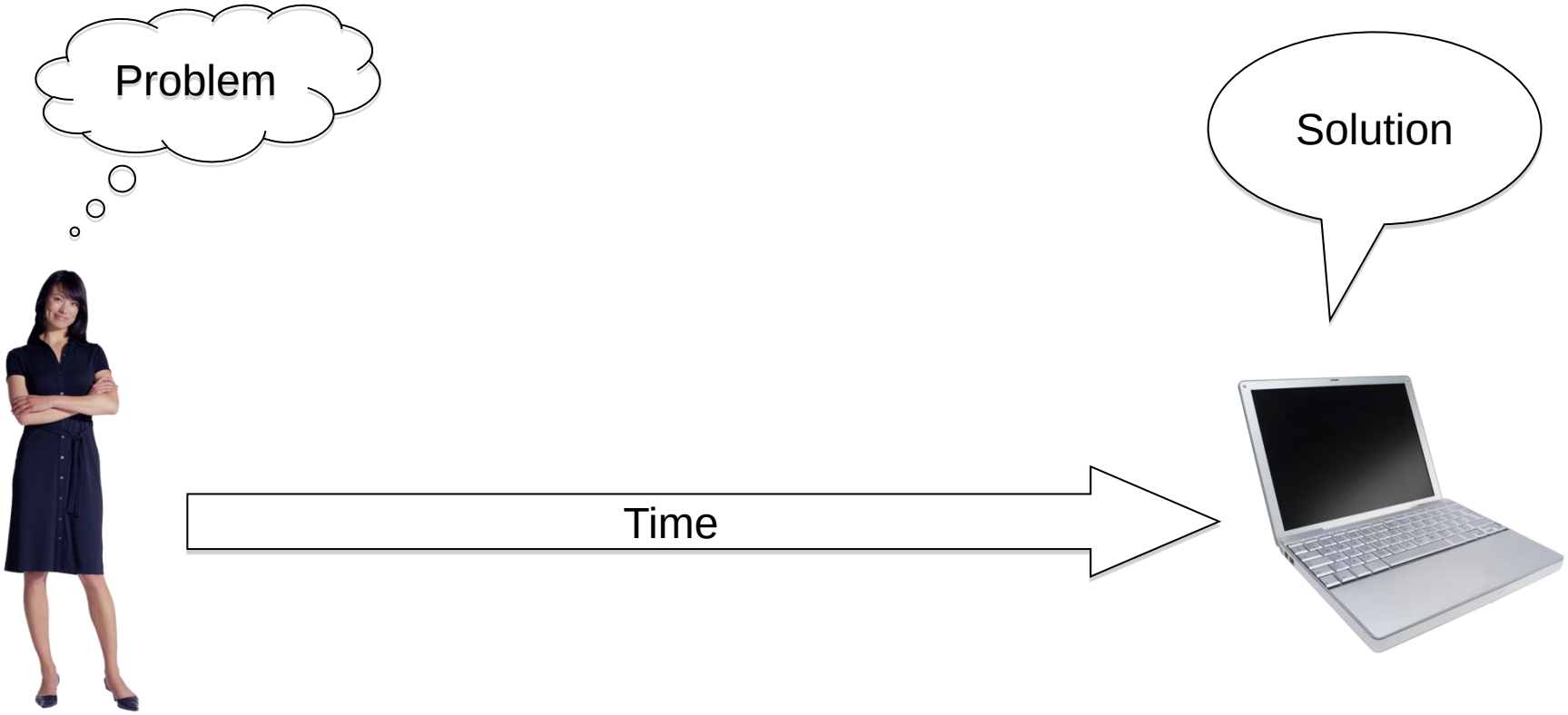
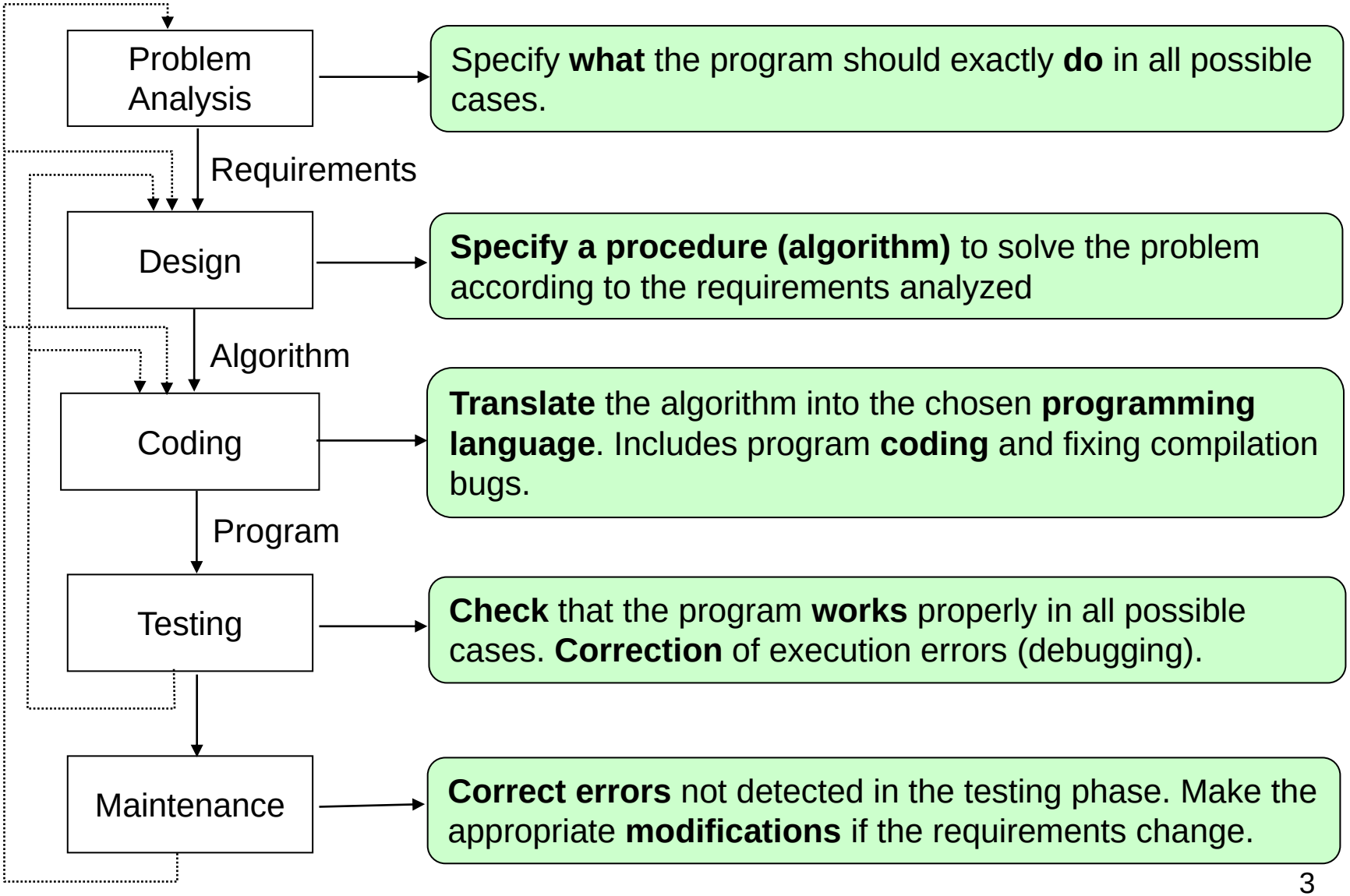
Lesson 2.
Problem solving: Introduction to algorithms and programming

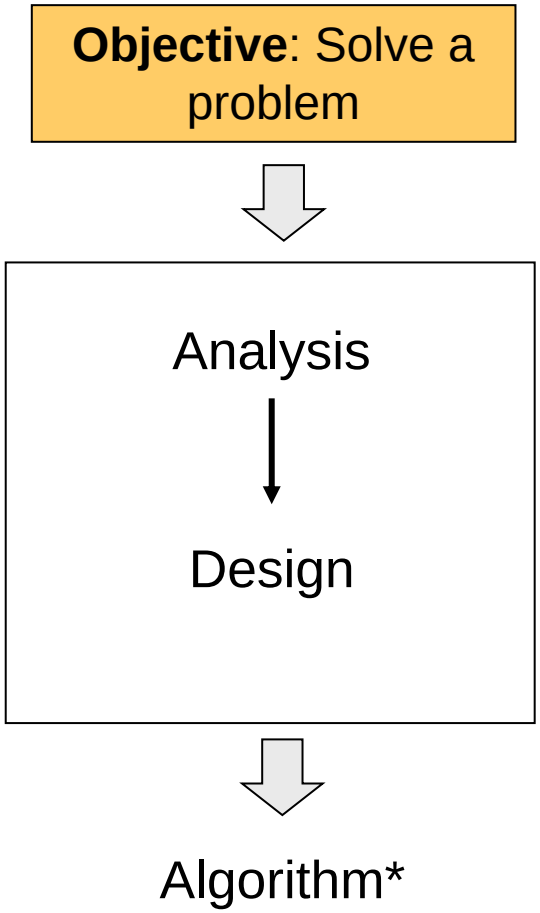
Phases in problem solving



Software life cycle



Problem solving: Concept of Algorithm



Definition of **Algorithm**:

Accurate and **unambiguous** description of the actions to be taken to solve a **well-defined** problem in a **finite time**.

Accurate: The order of each step must be indicated.

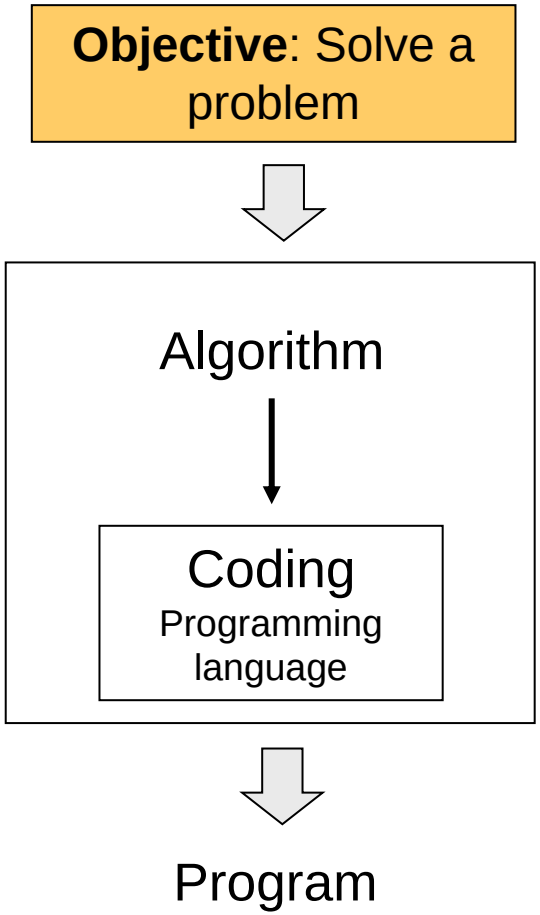
Unambiguous: There is only one interpretation.

Well-defined: it always gets the same result.

Finite time: It ends in a finite number of steps.

* It derives from the Latin translation of the word Alkhôwarîzmi, an Arab mathematician who wrote an essay on the manipulation of numbers and equations in the 9th century

Algorithms and Programs



Definition of **Coding:**

Writing the algorithm using a **programming language.**

Definition of **Programming language:**

A set of precise **instructions** that can be **interpreted** and **executed** by a computer.

Definition of **Compiler:**

Program that **translates** the source code into **machine language**.

Definition of **Program:**

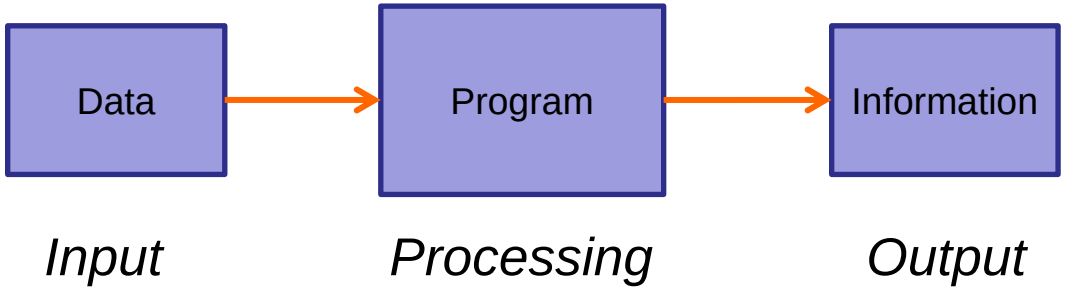
Coding of the **algorithm** by using a specific **programming language**.

Analysis of the Problem

Analysis: Understand **WHAT** the customer wants

Clear definition of what the program should do and the desired outcome.

At the highest level of abstraction, the computational solution would be as follows:



We have to answer:

- What is the desired output? (type and quantity)
- What method produces the output?
- What input is needed? (type and quantity)

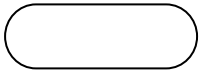
Analysis of the Problem: Example

We want to make a program that reads, via keyboard, the person's birth year and calculates the person's age. Then the program must tell us if the person is of legal age or not.

- What is the desired **output**? (type and quantity)
  - Person's age
  - Message informing if the person is of legal age
- What **method** produces the output?
  - Calculating age as the difference between the birth year and the current year
  - Comparison between age and age of majority (18 years)
- What **input** is needed? (type and quantity)
  - Birth Year

Algorithm Design: Tools

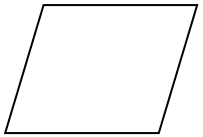
- Flowcharts
  - Graphic representation of an algorithm
  - ANSI Standardized Diagrams



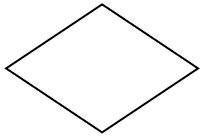
Start / End

Program flow

Action



Input / Output



Conditional

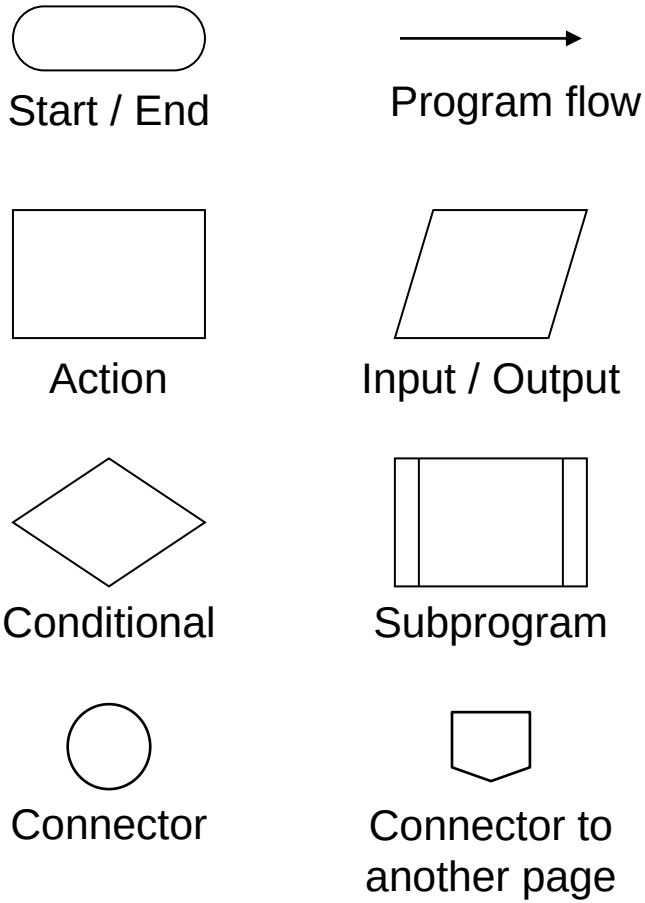
Subprogram

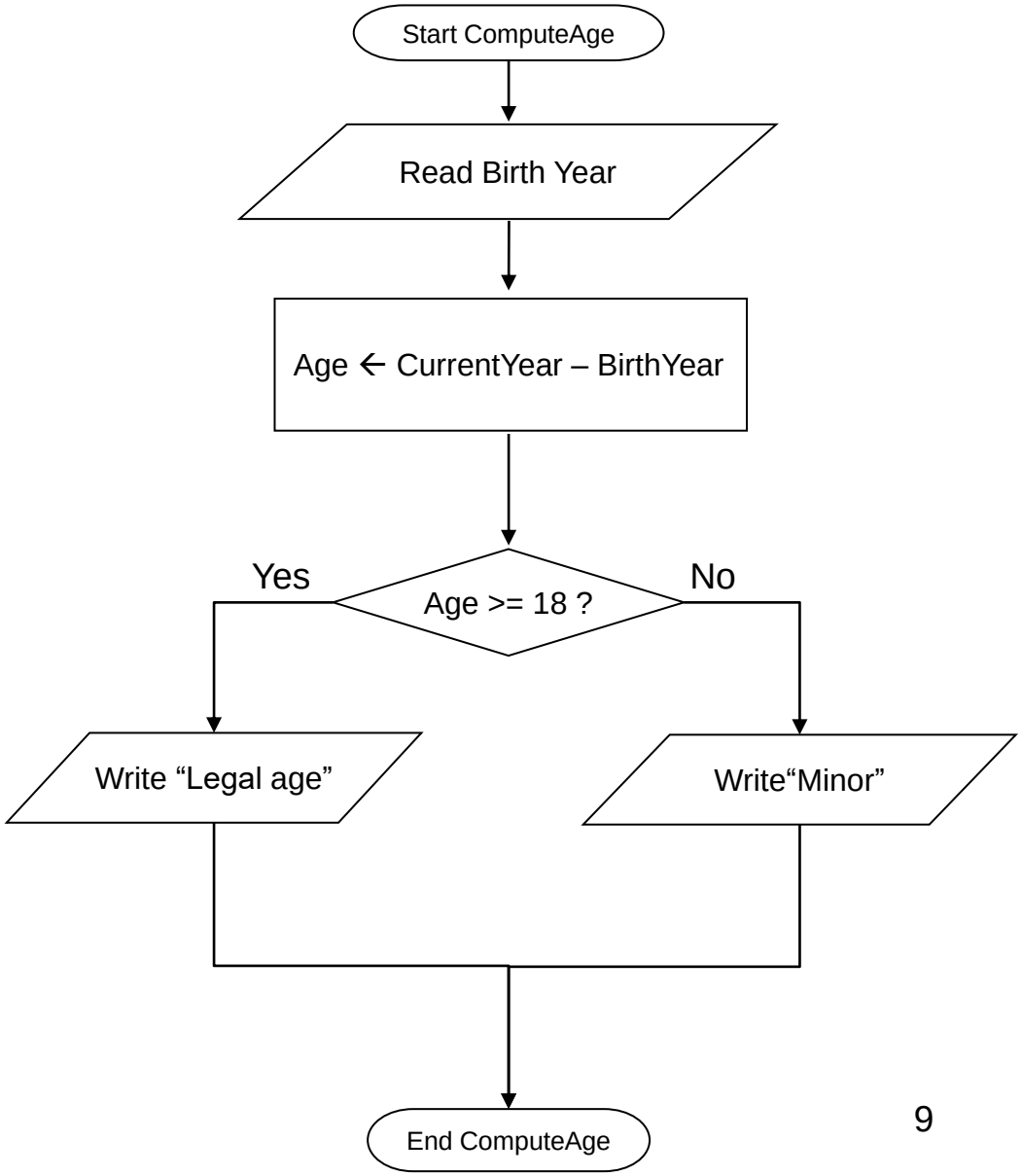
Connector

Connector to another page

- Pseudocode
  - Text programming tool.
  - It is a plain language of specification.
  - The instructions are written in a natural language (english, spanish, french, etc) making use of structures common to all programming languages.

Flowcharts


Start / End
Program flow
Action
Input / Output
Conditional
Subprogram
Connector
Connector to another page


Start ComputeAge
Read Birth Year
Age ← CurrentYear – BirthYear
Yes
Age >= 18 ?
No
Write “Legal age”
Write“Minor”
End ComputeAge

Pseudocode (algorithmic language)

```
algorithm ComputeAge

var
        BirthYear: integer
endvar

Read (BirthYear)

Age ← CurrentYear – BirthYear

if (Age >= 18)
    write("Legal age")
Else
    write("Minor")
endif

endAlgorithm
```

Unique point of input / output!

Declaration of variables

sections

Coding: Programming Languages

```
algorithm   ComputeAge

var
        BirthYear: integer
endvar

Read (BirthYear)

Age ← CurrentYear – BirthYear

if (Age >= 18)
   write(“Legal age”)
Else
   write(“Minor”)
endif

endAlgorithm
```

```
/* Program to compute a person’s age */

#include <stdio.h>

void main()
{
    int birthYear, age;

    scanf (“%d”, &birthYear);

    age = retrieveCurrentYear() - birthYear;

    if (age>= 18)
        printf(“Legal age \n”);
    else
        printf(“Minor \n”);
}
```

C Programming language

Coding: Programming Languages

```
algorithm   ComputeAge

var
        BirthYear: integer
endvar

Read (BirthYear)

Age ← CurrentYear – BirthYear

if (Age >= 18)
   write(“Legal age”)
Else
   write(“Minor”)
endif

endAlgorithm
```

```
"""
Program to compute a person’s age

"""“
from datetime import date

birthYear = int(input(“Write your Birth year:"))

age = date.today().year - birthYear

if (age >= 18):
    print("Legal age \n")
else:
    print(“Minor \n");
```

Python Programming language

Coding: Programming Languages

```
algorithm   ComputeAge

var
        BirthYear: integer
endvar

Read (BirthYear)

Age ← CurrentYear – BirthYear

if (Age >= 18)
   write(“Legal age”)
Else
   write(“Minor”)
endif

endAlgorithm
```

```
/* Program to compute a person’s age  */

program ComputeAge
var
    integer birthYear, age;
begin
    readln(birthYear);

    age := fgetCurrentYear() - birthYear ;

    if (age >= 18) then
        writeln(‘Legal age.’);
    else
        writeln (‘Minor.’);
end.
```

Pascal Programming language

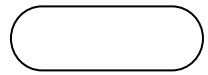
Flowchart Examples

- Summation of **n**

$$\sum_{k=1}^{n} a_k = a_1 + a_2 + \ldots + a_n$$

- Is **n** a Positive, negative, zero value?

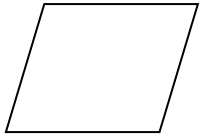
Flowcharts examples: Summation of n



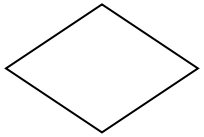
Start / End

Program flow

Action



Input / Output

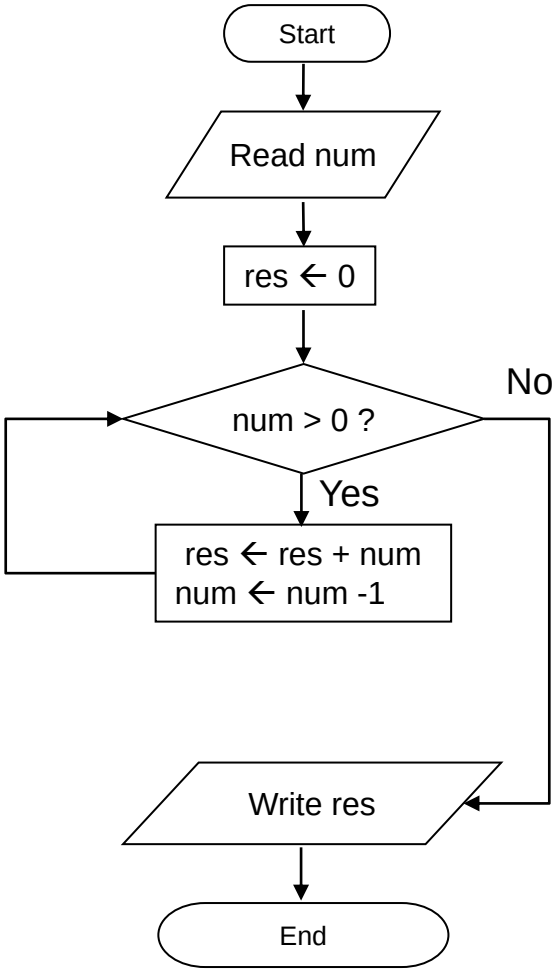


Conditional

Subprogram

Connector

Connector to another page

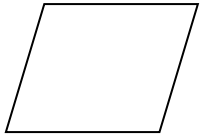


Flowchart examples: n is Positive/Negative/Zero?

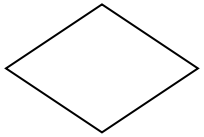
Start / End

Program flow

Action



Input / Output

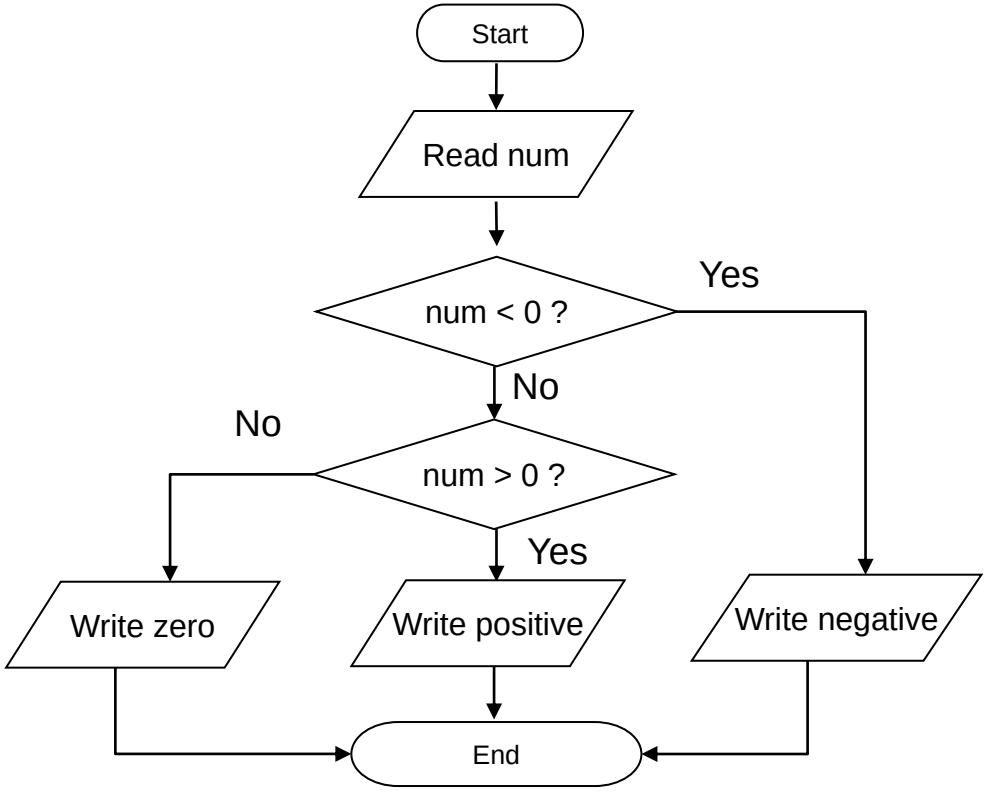


Conditional

Subprogram

Connector

Connector to another page



Analysis of the Problem: Example

If we invest 500€ in a monthly income fund, with a compound interest of 6% per year, for one year, we want to obtain a table that contains the monthly and accumulated return.

- What is the desired **output**? (type and quantity)
  - Cumulative return per month
  - Net return per month
- What **method** produces the output?
  - Calculation of the capital generated by the compound interest
  - Difference between the capital generated between two consecutive months
- What **input** is needed? (type and quantity)
  - Initial capital
  - Compound interest
  - Number of settlements

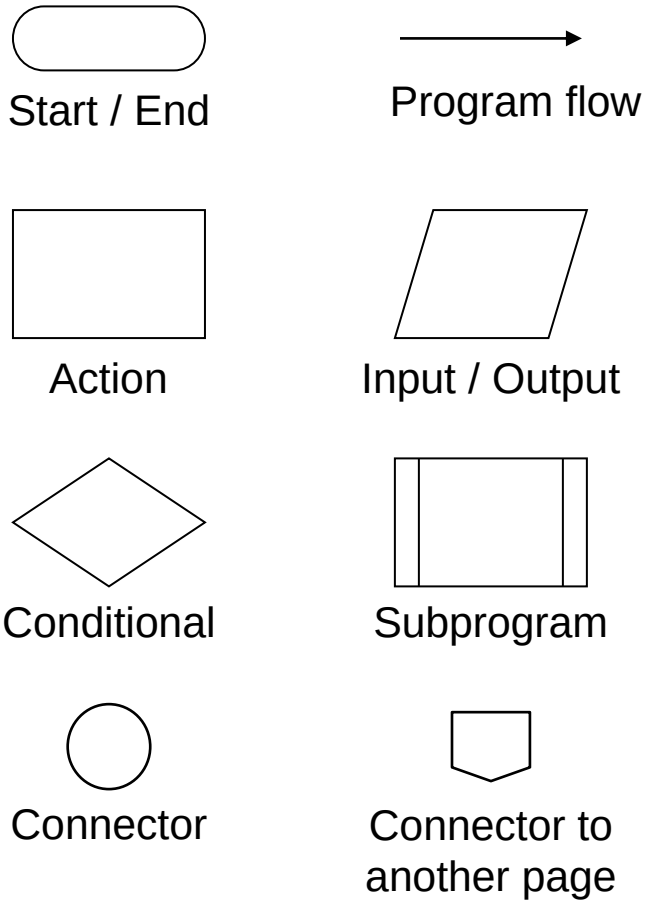
COMPOUND INTEREST

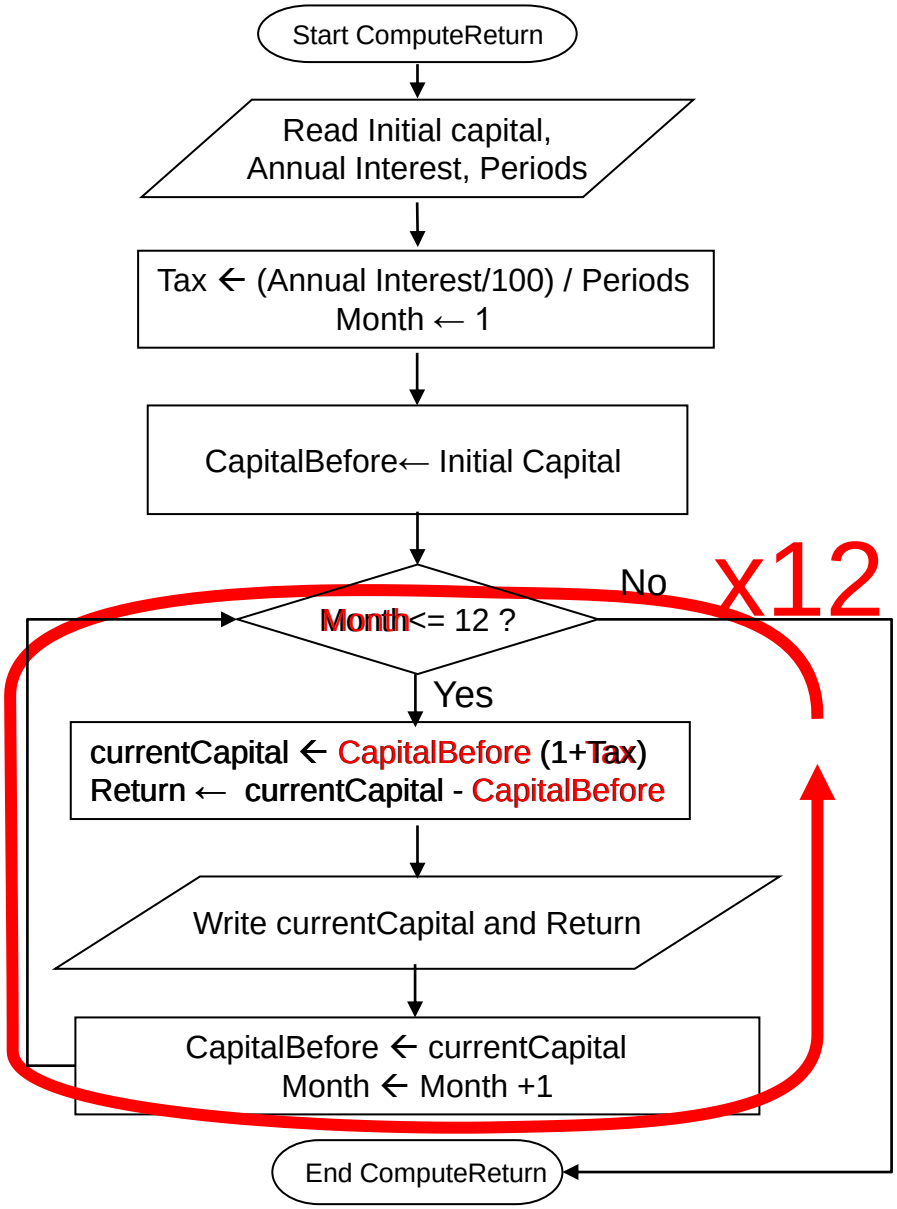
$$C_n = C_i\,(1+ t / k)^{m\,k}$$

C_i = Initial capital
C_n = Final capital
t = interest /100
k = settlement period, m=1

Month	1	2	3	4	5	6	7	8	9	10	11	12
Capital	€ 502,50	€ 505,01	€ 507,54	€ 510,08	€ 512,63	€ 515,19	€ 517,76	€ 520,35	€ 522,96	€ 525,57	€ 528,20	€ 530,84
Return (compound interest)	€ 2,50	€ 2,51	€ 2,53	€ 2,54	€ 2,55	€ 2,56	€ 2,58	€ 2,59	€ 2,60	€ 2,61	€ 2,63	€ 2,64

Flowchart


Start / End
Program flow
Action
Input / Output
Conditional
Subprogram
Connector
Connector to another page


Start ComputeReturn
Read Initial capital, Annual Interest, Periods
Tax ← (Annual Interest/100) / Periods
Month ← 1
CapitalBefore← Initial Capital
Month<= 12 ?
No
x12
Yes
currentCapital ← CapitalBefore (1+Tax)
Return ← currentCapital - CapitalBefore
Write currentCapital and Return
CapitalBefore ← currentCapital
Month ← Month +1
End ComputeReturn

Pseudocode (algorithmic language)

```
algorithm ComputeReturn
var
        Initial Capital, Current Capital, Capital Before, Annual Interest,
        Tax, Return: float
        Period, Month: integer
fivar

Read (InitialCapital, Annual Interest, Periods)
Tax ← (Annual Interest / 100) / Periods
Month← 1
Current Capital ← Initial Capital
Capital Before  ← Initial Capital

while (Month <=12)
    Current Capital ← Capital Before (1+Tax)
    Return ←  Current Capital - Capital Before

    write(Current Capital, Return)

    Capital Before ← current Capital

    Month ← Month+1
endwhile
endalgorithm
```

Unique point of input / output!

Declaration of variables

sections

Coding: Programming Languages

```
algorithm ComputeReturn
var
    Initial Capital, Current Capital, Capital Before,
    Annual Interest, Tax, Return: float
    Period, Month: integer
fivar

Read (InitialCapital, AnnualInterest, Periods)
Tax ← (Annual Interèst / 100) / Períods
Month ← 1
Current Capital ← Initial Capital
Capital Before ← Initial Capital

while (Month <=12)
   Current Capital ← Capital Before (1+Tax)
   Return ← Current Capital - Capital Before

   write(Current Capital, Return)

   Capital Before ← current Capital

   Month ← Month+1
endwhile
endalgorithm
```

C Programming language

```
/* Program to compute the return of Compound Interest */

#include <stdio.h>

void main()
{
      int Periods, Month;
      Float CapitalInitial, CapitalCurrent, CapitalBefore,
      InterestAnnual, Tax, myReturn;

      scanf ("%f", &CapitalInitial);
      scanf ("%f", &InterestAnnual);
      scanf ("%d", &Periods);

      Tax = (InterestAnnual / 100) / Periods;

      CapitalCurrent = CapitalInitial;
      CapitalBefore = CapitalInitial;

      Month = 1;

      while (Month <= 12)
      {
            CapitalCurrent = CapitalBefore * (1+Tax);
            myReturn= CapitalCurrent - CapitalBefore;

            printf("Return at month %d is %f\n", Month, myReturn);
            printf("Cummulated capital at month %d is %f\n", Month,
            CapitalCurrent);

            Month = Month +1;
            CapitalBefore = CapitalCurrent;

      }
}
```

Coding: Programming Languages

```
algorithm ComputeReturn
var
    Initial Capital, Current Capital, Capital Before,
    Annual Interest, Tax, Return: float
    Period, Month: integer
fivar

Read (InitialCapital, AnnualInterest, Periods)
Tax ← (Annual Interèst / 100) / Períods
Month ← 1
Current Capital ← Initial Capital
Capital Before ← Initial Capital

while (Month <=12)
   Current Capital ← Capital Before (1+Tax)
   Return ← Current Capital - Capital Before

   write(Current Capital, Return)

   Capital Before ← current Capital

   Month ← Month+1
endwhile
endalgorithm
```

Python Programming language

```
"""
Program to compute the return of Compound Interest

"""


CapitalInitial = int(input("Write the initial capital: "))
InterestAnnual = int(input("Write the annual interest: "))
Periods = int(input("Write the settlement periods: "))

Tax = float((InterestAnnual/100) / Periods);

CapitalCurrent = CapitalInitial
CapitalBefore = CapitalInitial

Month = 1;

while (Month <= 12):
    CapitalCurrent = CapitalBefore *(1+Tax)
    myReturn = CapitalCurrent - CapitalBefore
    print("Return at month ", Month ," is ", myReturn)
    print("Cummulated capital at month ", Month , " is ",
            CapitalCurrent)
    Month = Month +1
    CapitalBefore = CapitalCurrent
```

Coding: Programming Languages

```
algorithm ComputeReturn
var
    Initial Capital, Current Capital, Capital Before,
    Annual Interest, Tax, Return: float
    Period, Month: integer
fivar

Read (InitialCapital, AnnualInterest, Periods)
Tax ← (Annual Interèst / 100) / Períods
Month ← 1
Current Capital ← Initial Capital
Capital Before ← Initial Capital

while (Month <=12)
   Current Capital ← Capital Before (1+Tax)
   Return ← Current Capital - Capital Before

   write(Current Capital, Return)

   Capital Before ← current Capital

   Month ← Month+1
endwhile
endalgorithm
```

```
/* Program to compute the return of Compound Interest */

program computeReturn
var
      integer Periods, Month;
      double CapitalInitial, CapitalCurrent, CapitalBefore,
      InterestAnnual, Tax, myReturn;
begin
      readln(CapitalInitial);
      readln(InterestAnnual);
      readln(Periods);

      Tax := (InterestAnnual /100)/Periods;

      CapitalCurrent := CapitalInitial;
      CapitalBefore  := CapitalInitial;

      Month:= 1;

      while (Month <= 12)
      begin
            CapitalCurrent := CapitalBefore *(1+Tax);
            myReturn := CapitalCurrent – CapitalBefore;

            writeln('Return at month ',Month,'is', myReturn);
            writeln ('Cummulated capital at month ' , Month,'is',
            CapitalCurrent);

            Month := Month +1;
            CapitalBefore := CapitalCurrent;

      end
end.
```

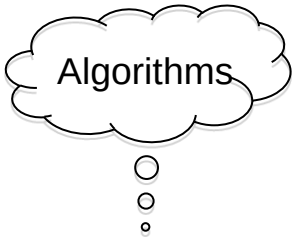
Pascal Programming language

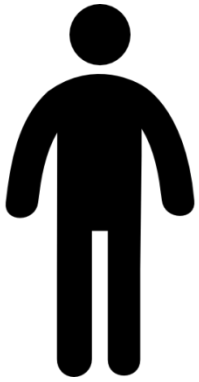
Compiler

```
algorithm   ComputeAge

var
        BirthYear: integer
endvar

Read (BirthYear)

Age ← 2025 – BirthYear

if (Age >= 18)
   write(“Legal age”)
Else
   write(“Minor”)
endif

endAlgorithm
```

```
/* Program to compute a person’s age */

#include <stdio.h>

void main()
{
    int birthYear, age;

    scanf (“%d”, &birthYear);

    age = 2025 - birthYear;

    if (age>= 18)
        printf(“Legal age \n”);
    else
        printf(“Minor \n”);
}
```

```
Mov R2 1966
Mov R3 2025
Sub R3 R2
```





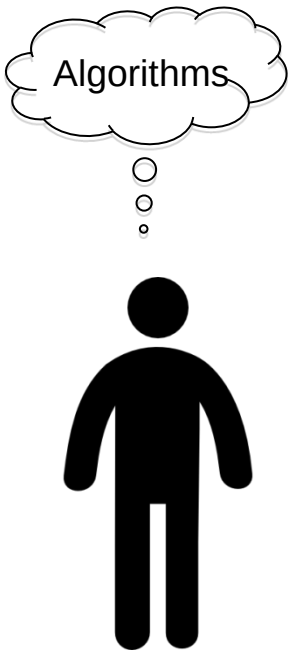
High level language → Compiler → Object files → Linkage → Executable

Libraries → Linkage

Interpreter

```
algorithm   ComputeAge

var
            BirthYear: integer
endvar

Read (BirthYear)

Age ← 2025 – BirthYear

if (Age >= 18)
   write(“Legal age”)
Else
   write(“Minor”)
endif

endAlgorithm
```

```
"""
Program to compute a person’s age

"""

birthYear = int(input(“Write your Birth year:"))

age = 2025 - birthYear

if (age >= 18):
   print("Legal age \n")
else:
   print(“Minor \n");
```



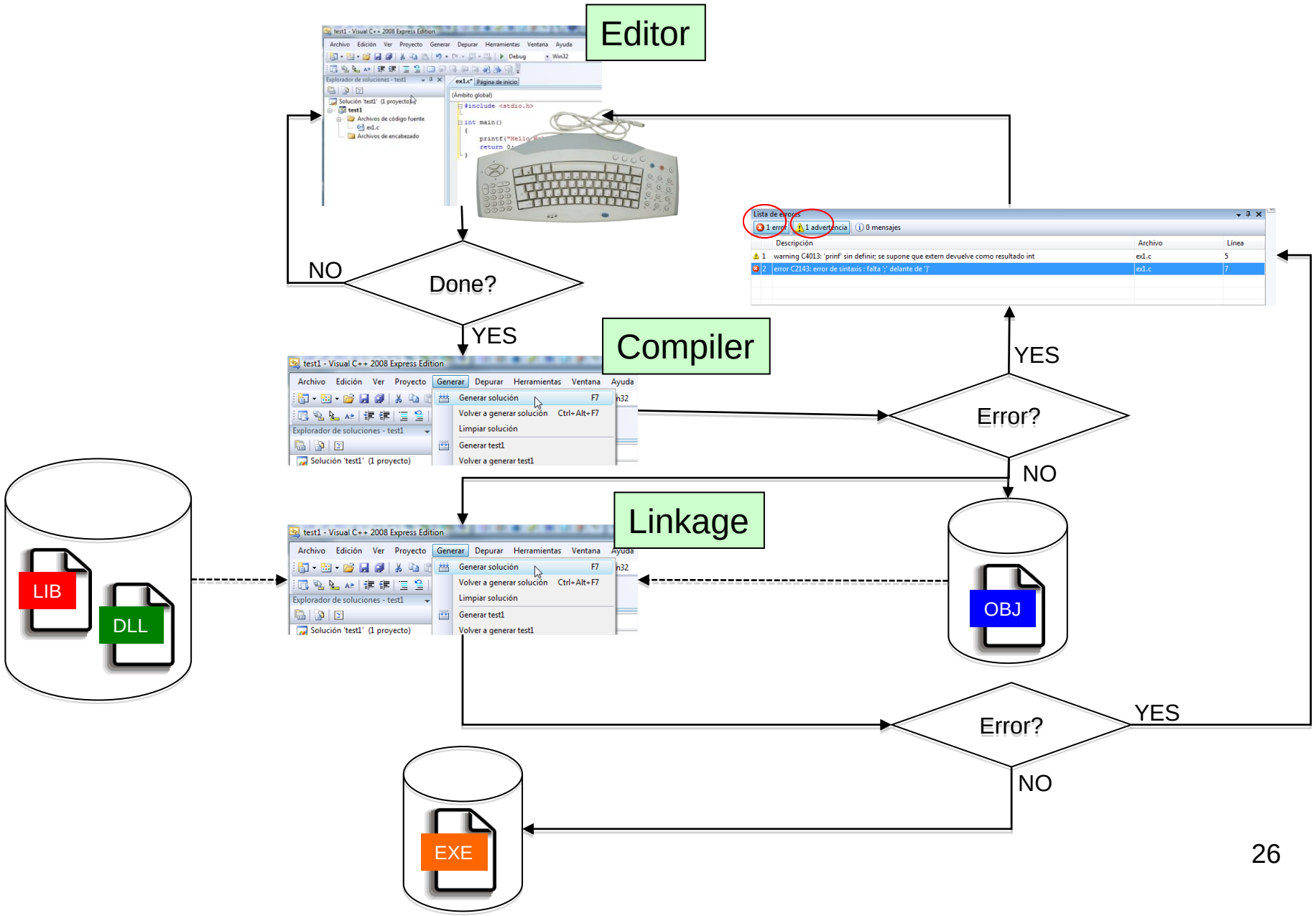
High level language

Interpreter

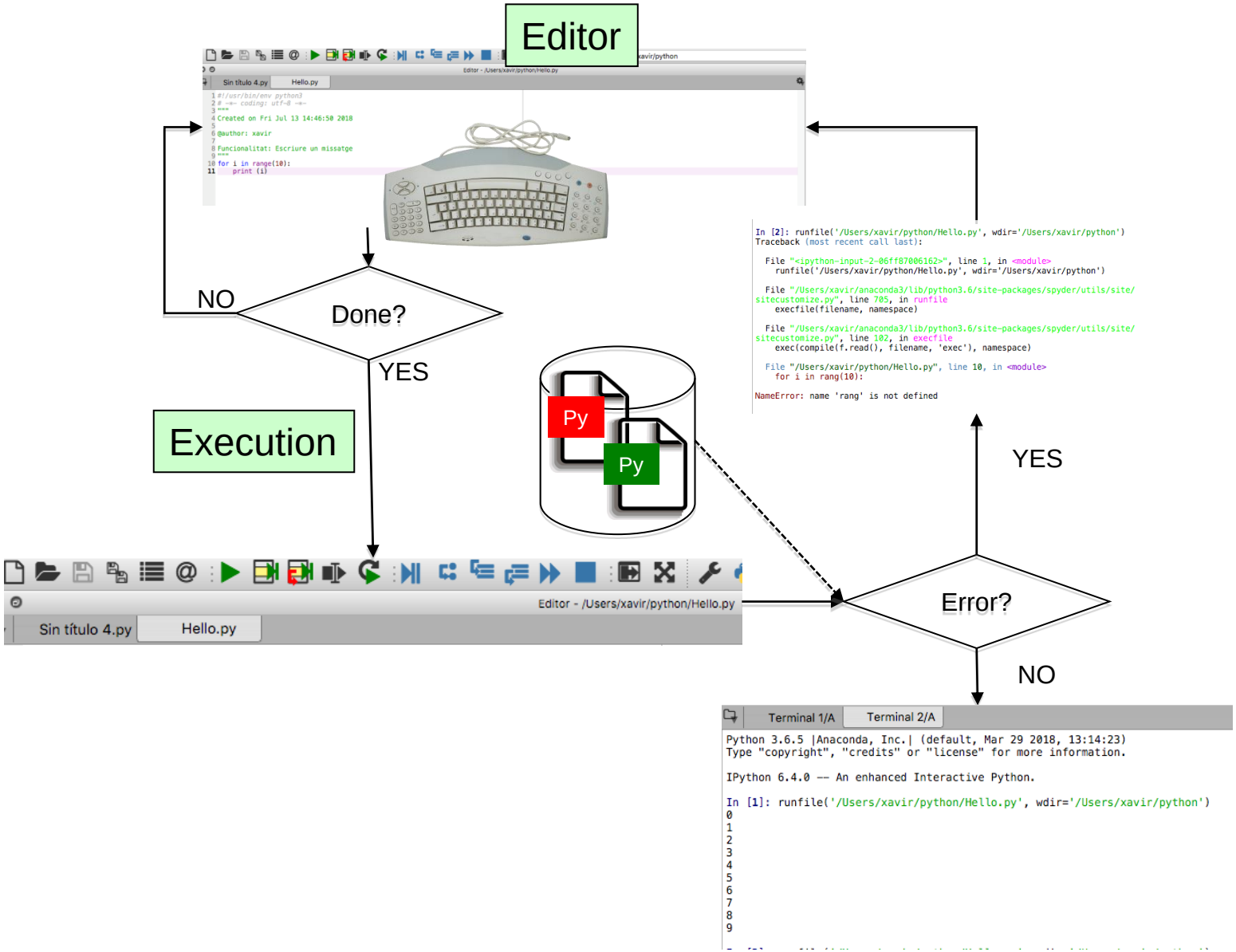
Compiled vs Interpreted

Compiled		Interpreted	
Ready to be executed	They are **not** multiplatform	They are multiplatform	They require an interpreter
Usually faster	Low flexibility	Easier to test	Usually slower
Source code not available	Linkage is needed	Errors are easily detected	Source code is public

Compiler


Editor
NO
Done?
YES
Compiler
Error?
YES
NO
Linkage
LIB
DLL
OBJ
Error?
YES
NO
EXE

Interpreter



Aspects of languages

Construction primitives:

- A (natural) **language**: words
- A **programming language**: operators, numbers, strings

We can create sentences.

Syntax: Proper coordination of words

- Natural language: Bike apple spoon (Incorrect: name name name)
  Bird eats apple (Correct: name verb name)

- Programming language: 9 “apple” (Incorrect: number string)
  9 + 8 (Correct: operator operation operator)

Aspects of languages

Static semantics: syntax can be correct but... it should make sense

(natural) language: She eat apple (Syntax: correct, static semantics: incorrect)

Programming language:
9 + “apple” (Syntax: correct, static semantics: incorrect)
9 + 8 (Correct)

Semantics: Study of the meaning

In any language we can find correct sentences but with more than 1 meaning:

I saw someone on the hill with a telescope

In programming languages, constructions are NOT ambiguous (only one meaning) ... But maybe they don’t do what the programmer wanted.

Errors related to language

In programming there are two type of errors:

1. Syntactic Errors (compiling) :

- We did not follow the exact rules of the writing language.
- As the compiler is a program that automatically scans the code it will warn us that it is not correct.
- Sometimes it helps us with the type of error, but sometimes when the code is written so badly, help messages do not allow us to detect it quickly.

2. Semantic Errors (execution):

- The source code is syntactically correct but our program does not do what we wanted.
- The Integrated Development Environment (IDE) has a debugger tool that allows us to run the program step by step to detect where the error is.