OOP - Python

Jaime A. Riascos

Potsdam University

Institution University of Envigado (IUE)

Elemental Concepts- Algorithm

Where did everything start?

The word algorithm comes from the nickname of a 9th century Persian mathematician, Al-Khwarizmi, who was recognized for enunciating step by step the rules for basic mathematical operations with decimals (addition, subtraction, multiplication and division).

Elemental Concepts- Algorithm

Where did everything start?

The word algorithm comes from the nickname of a 9th century Persian mathematician, Al-Khwarizmi, who was recognized for enunciating step by step the rules for basic mathematical operations with decimals (addition, subtraction, multiplication and division).

Elemental Concepts- Algorithm

The word algorithm comes from the nickname of a 9th century Persian mathematician, Al-Khwarizmi, who was recognized for enunciating step by step the rules for basic mathematical operations with decimals (addition, subtraction, multiplication and division).

Elemental Concepts- Algorithm

In mathematics and computer science, an algorithm is a finite sequence of instructions, which are well-defined and can be implemented in a computer. Typically, these instructions are used to solve a type of problem or perform a calculation.

Elemental Concepts- Algorithm

Objective: to calculate the average age of workers in an office.

Sequence of steps:

Step 1: Collect the age of member 1

Step 2: Collect the age of member 2

Step 3: Collect the age of member 3

Step 4: Collect the age of member 4

Step 5: Collect the age of member 5

Step 6: Collect the age of member 6

Step 7: Add the age of members 1, 2, 3, 4, 5 and 6

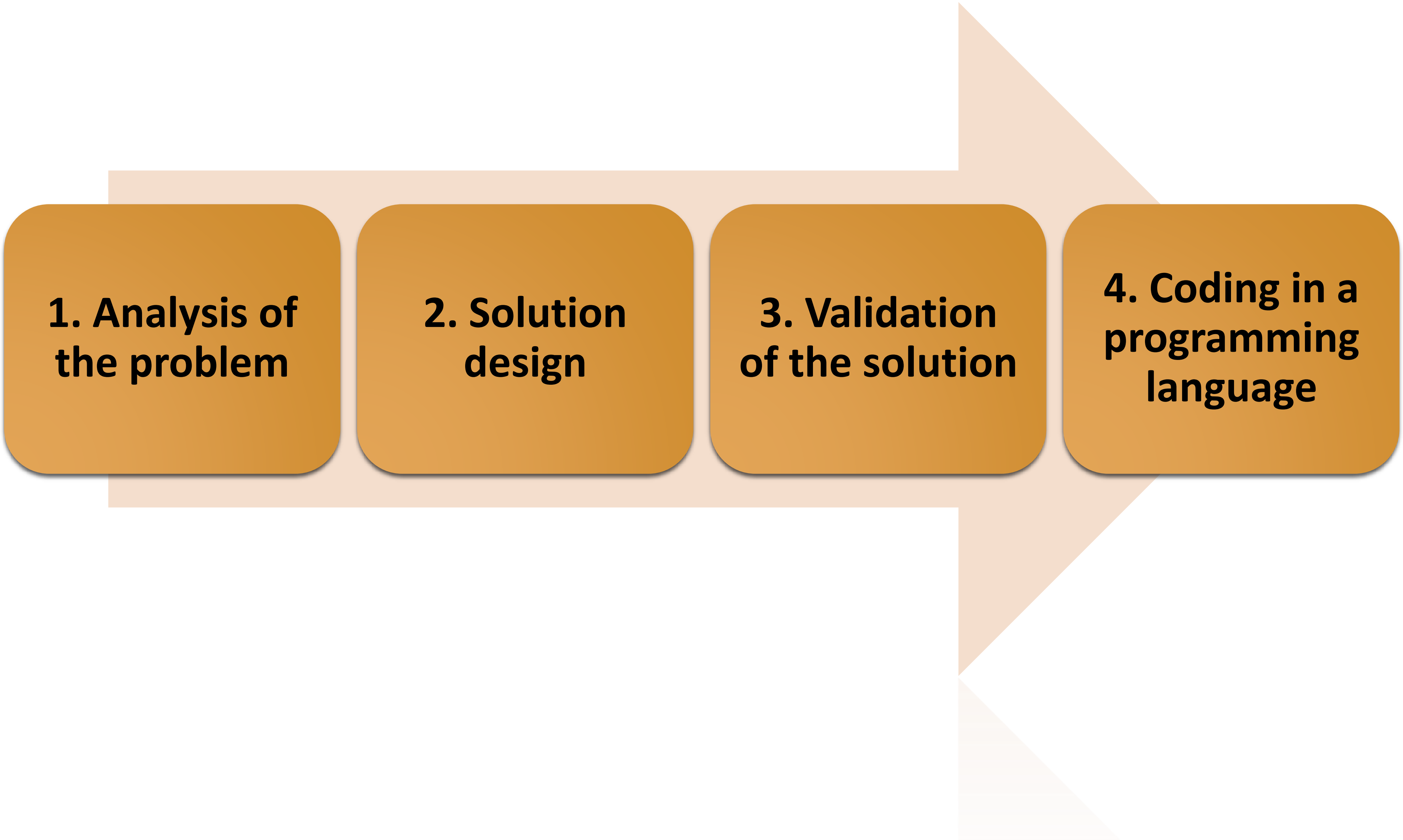
Step 8: Divide the total of the previous sum by 6

12 **18** **23** **40** **20** **23**

12 + 18 + 23 + 40 + 20 + 23 = 136

136 / 6 = 22.66

Methodology of working with algorithms



1. Analysis of the problem

2. Solution design

3. Validation of the solution

4. Coding in a programming language

1. Analysis to design an algorithm

Ask the following questions:

What is the objective sought?

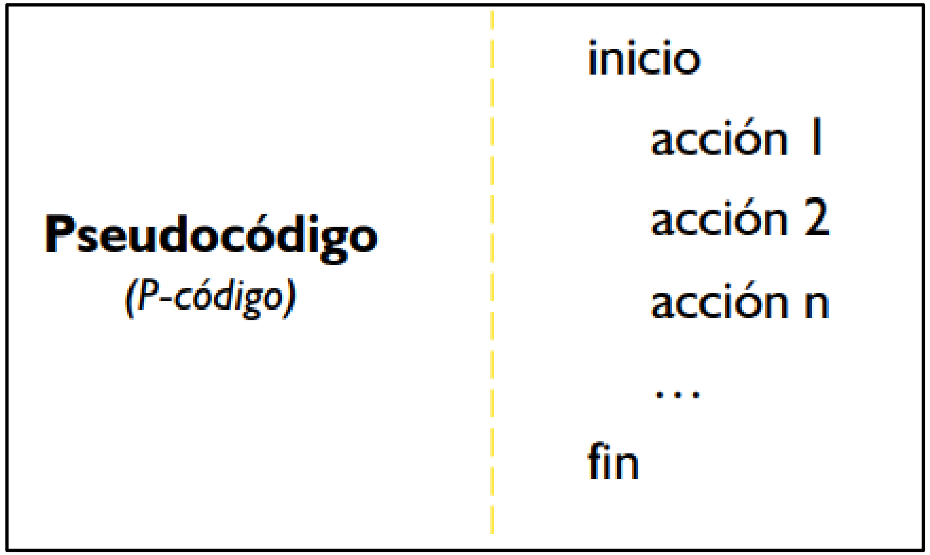
What is the input data?

What calculations/processes should be carried out?

What is the output data?

2. Solution design

- **Objective:** to calculate the average age of workers in an office.
- **Entry data:** member age 1, member age 2, member age 3, member age 4, member age 5, and member age 6.
- **Processes / calculations:** add the ages of members 1, 2, 3, 4, 5, and 6. And divide the total by 6.
- **Output data:** average age of the 6 members.



Pseudocódigo

(P-código)

```
inicio
    acción 1
    acción 2
    acción n
    ...
fin
```

3. Validation of the solution

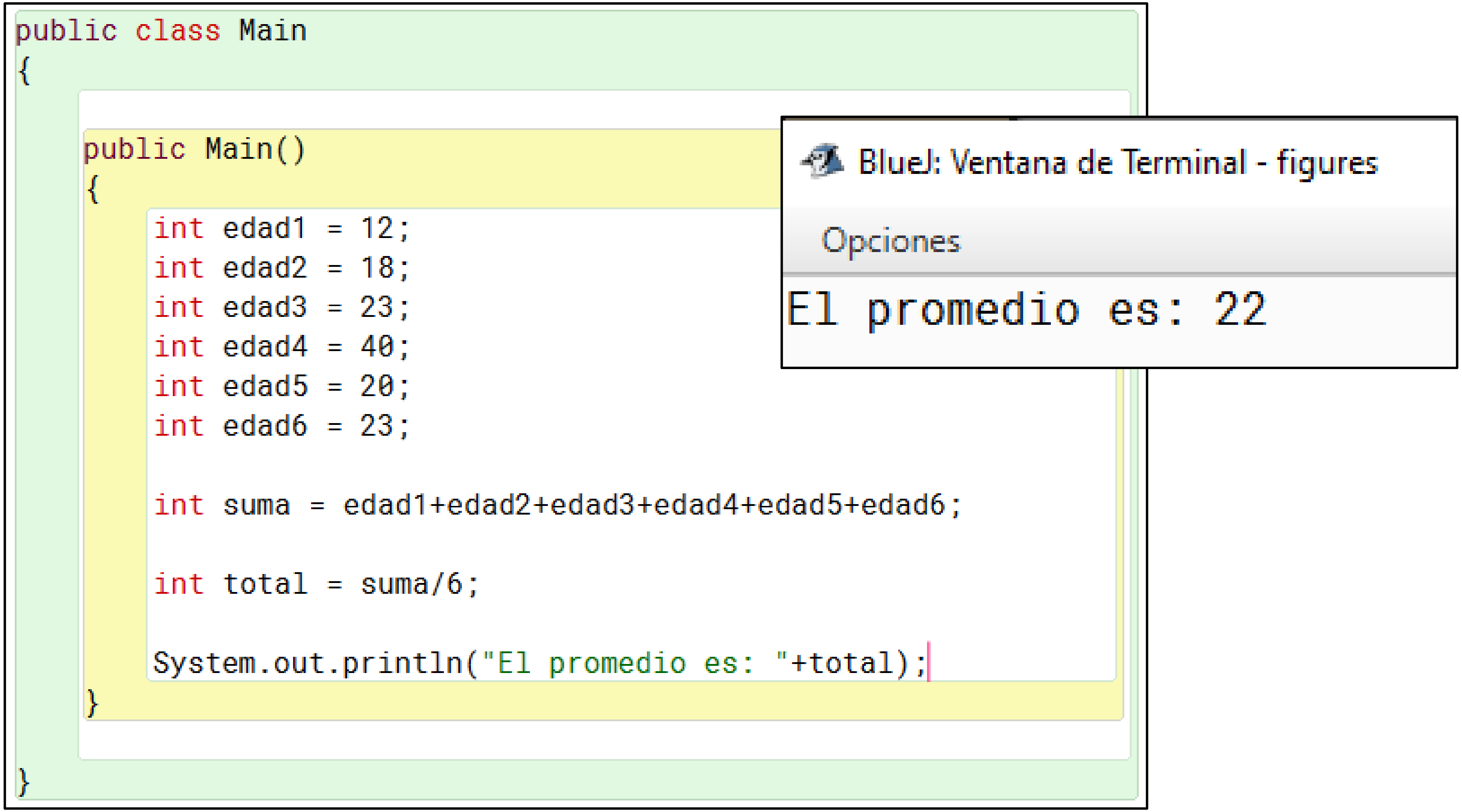
- To validate the solution of an algorithm, desktop tests (which do not require the use of computers) can be performed. Or tests can also be performed on computers, for which it is necessary to have the algorithm coded in a programming language.

- Manual validation (desktop test):

Determine the final values of the variables assuming that you have the following instructions:
age1 = 19
age2 = 34
total = age1 + age2
totalRare = total - age1

	Age1	age2	total	totalRare
1	19			
2	19	34		
3	19	34	53	
4	19	34	53	34

4. Coding in a programming language



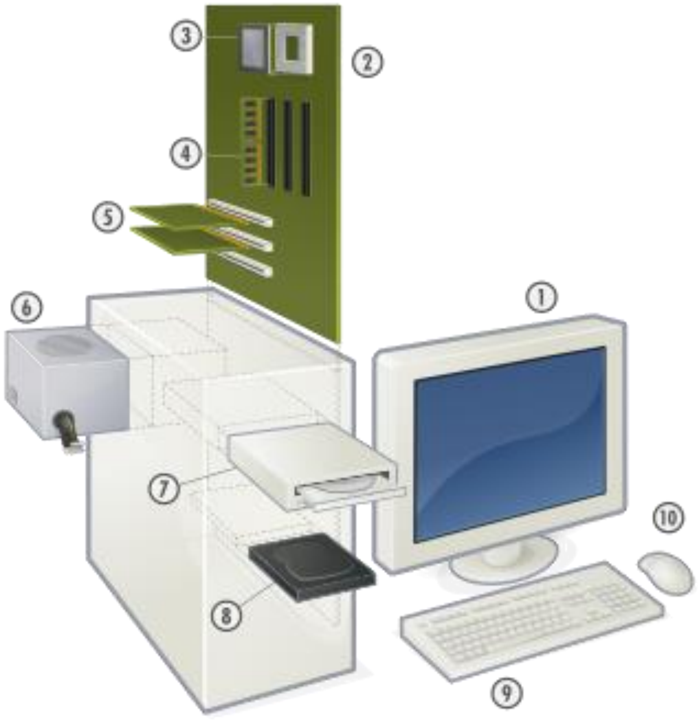
```java
public class Main
{

    public Main()
    {
        int edad1 = 12;
        int edad2 = 18;
        int edad3 = 23;
        int edad4 = 40;
        int edad5 = 20;
        int edad6 = 23;

        int suma = edad1+edad2+edad3+edad4+edad5+edad6;

        int total = suma/6;

        System.out.println("El promedio es: "+total);
    }

}
```

```
El promedio es: 22
```

Hardware

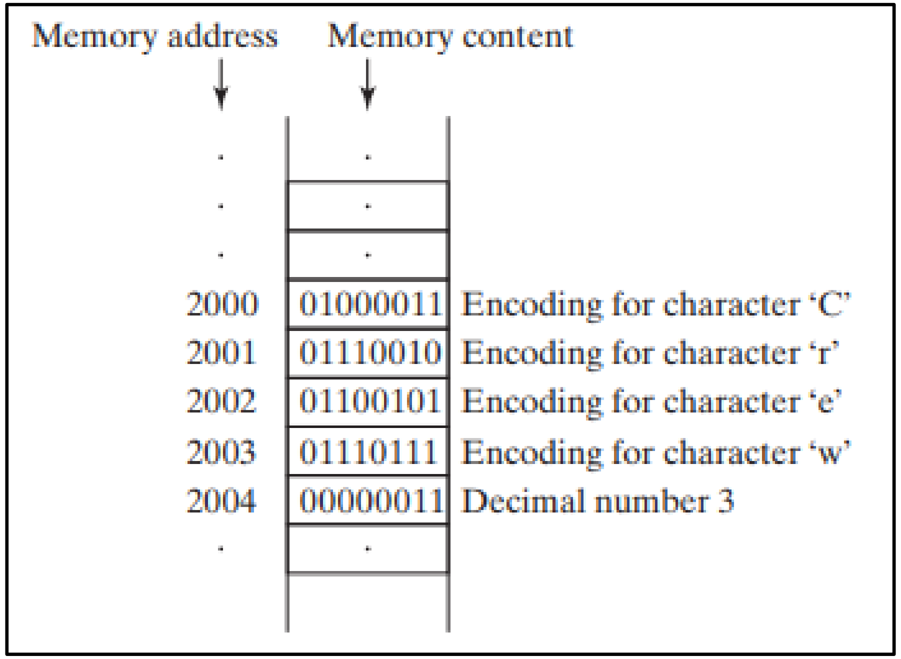
In general a computer contains the following hardware components:

A central processing unit (CPU)
Memory
Storage devices (such as disks and CDs)
Input devices (such as mouse and keyboard)
Output devices (such as monitors and printers)
Communication devices (such as network cards).



Memory

- A computer's memory consists of an ordered sequence of bytes for storing programs as well as the data the program is working with.
- You can think of memory as the working area of the computer for running a program.
- A program and its data must be moved to the computer's memory before the CPU can execute them.

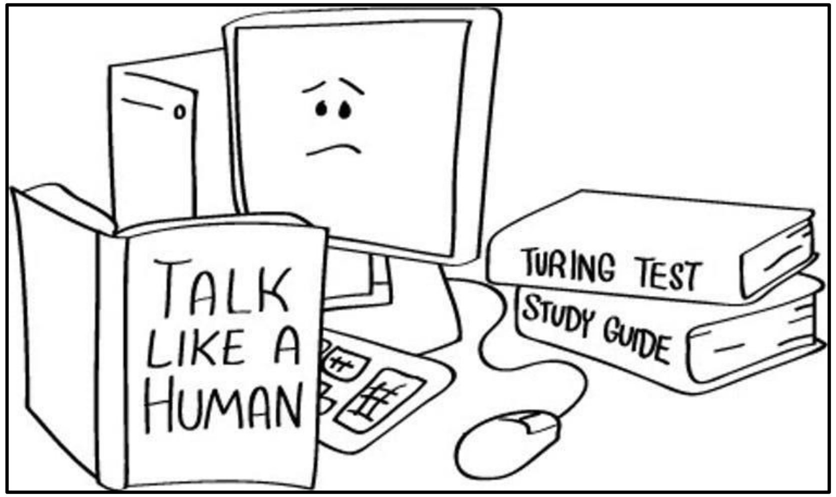


Memory address	Memory content	
.	.	
.	.	
.	.	
2000	01000011	Encoding for character ‘C’
2001	01110010	Encoding for character ‘r’
2002	01100101	Encoding for character ‘e’
2003	01110111	Encoding for character ‘w’
2004	00000011	Decimal number 3
.	.	

Programming languages

What are programming languages?
Why are they necessary?

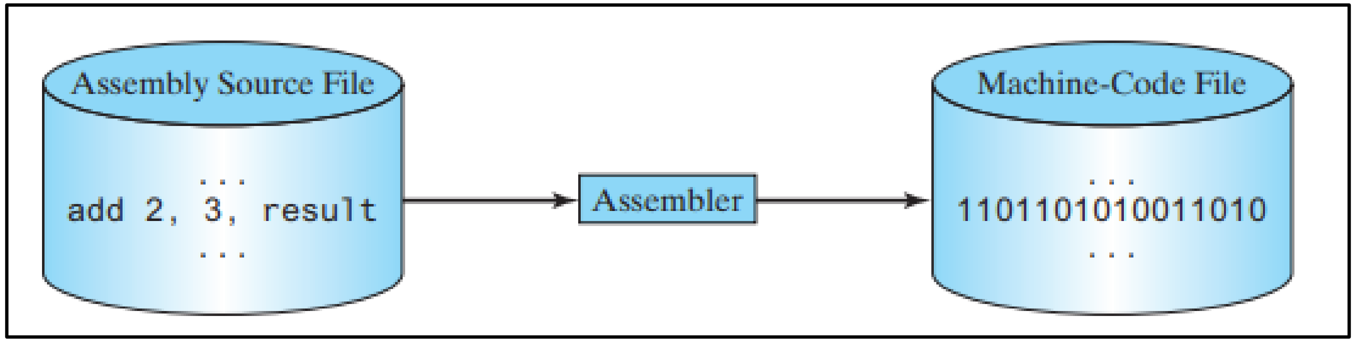
- Computers do not understand human languages, so programs must be written in a language that a computer can use.
- There are hundreds of programming languages, and they were developed to make the programming process easier for people. However, all programs must eventually become instructions that computers can execute.



Language Types - I

Machine language
It is the native language of a computer. Which is made in the form of binary code.

Assembly Language
Programming at the tip of ones and zeros is very tedious, therefore at the beginning of the computer age, the assembly language was invented. assembly language uses short descriptive words to represent a set of machine instructions



Language Types - II

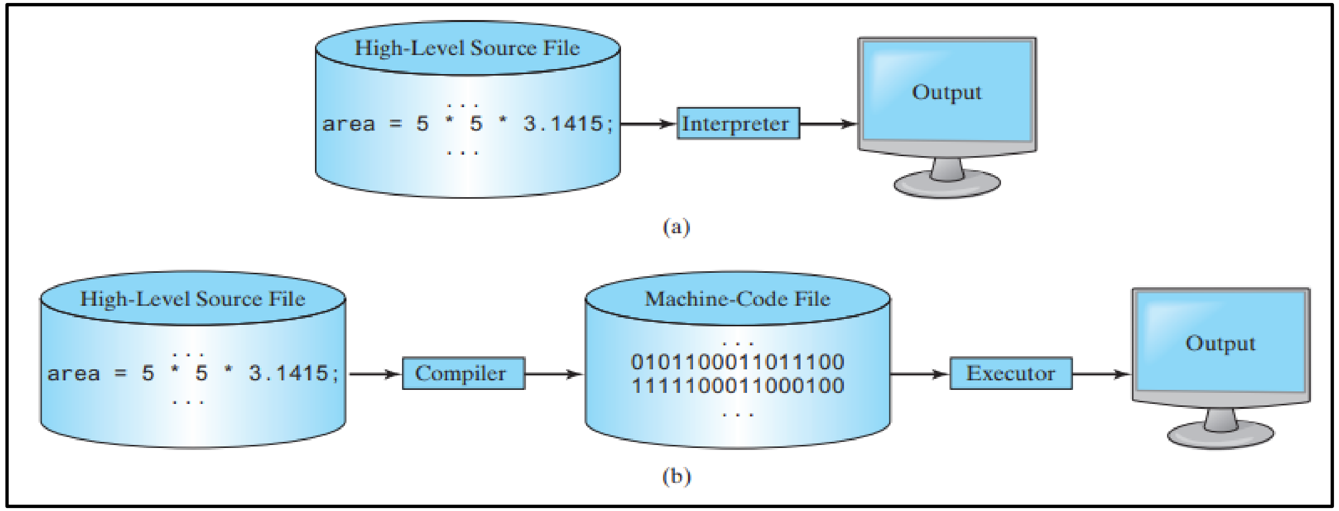
High-level language

In the 1950s, a new generation of programming languages known as "high-level languages" emerged. These languages are platform-independent, meaning that you can write a program in a high-level language and run it on different types of machines. High-level languages are similar to the "English" language, and are easier to learn and use.

Java

python

php



```
...
area = 5 * 5 * 3.1415;
...
```

```
...
0101100011011100
1111100011000100
...
```

Python

Python is a very powerful and flexible general-purpose programming language, as well as simple and easy to learn, and was created in the early nineties by Guido van Rossum in the Netherlands. It is a high-level language, which allows you to easily process all kinds of data structures, both numerical and text. It is free software, and is implemented on all common platforms and operating systems.

Python

The features of the Python programming language are summarized below:

It is an **interpreted**, uncompiled language that uses **dynamic**, strongly **typed typing** (the value type does not change suddenly).

Python

The features of the Python programming language are summarized below:

It is an **interpreted**, uncompiled language that uses **dynamic**, strongly **typed typing** (the value type does not change suddenly).

It is cross-platform, which is advantageous for making its source code executable between various operating systems.

Python

The features of the Python programming language are summarized below:

It is an **interpreted**, uncompiled language that uses **dynamic**, strongly **typed typing** (the value type does not change suddenly).

It is cross-platform, which is advantageous for making its source code executable between various operating systems.

It is a multiparadigm programming language, which supports various programming paradigms such as object orientation, structured, imperative programming and, to a lesser extent, functional programming.

Python

The features of the Python programming language are summarized below:

It is an **interpreted**, uncompiled language that uses **dynamic**, strongly **typed typing** (the value type does not change suddenly).

It is cross-platform, which is advantageous for making its source code executable between various operating systems.

It is a multiparadigm programming language, which supports various programming paradigms such as object orientation, structured, imperative programming and, to a lesser extent, functional programming.

It is cross-platform, which is advantageous for making its source code executable between various operating systems.

In Python, code formatting (e.g., indentation) is structural.

Python - Advantages

- **Simplified and fast:** This language greatly simplifies programming and "makes you adapt to a programming language mode, Python proposes a pattern". It's a great language for scripting, if you require something fast (in the sense of language execution), with a few lines it's already solved.
- **Elegant and flexible:** The language gives you many tools, if you want lists of various types of data, you do not need to declare each type of data. It's such flexible language you don't care so much about details.
- **Healthy and productive programming:** Programming in Python becomes a very healthy style of programming: it is simple to learn, directed to the perfect rules, it makes you dependent on improving, complying with the rules, the use of lines, variables. " It is also a language that was made with productivity in mind, that is, Python makes you more productive, allows you to deliver in the times that require me.

Python - Advantages

- **Tidy and clean:** The order that Python maintains, is what its users like the most, it is very readable, any other programmer can read it and work on the program written in Python. The modules are well organized, unlike other languages.
- **Portable:** It is a very portable language (either on Mac, Linux or Windows) compared to other languages. The philosophy of batteries included, are the libraries that you need most to the day to day programming, they are already inside the interpreter, you do not have the need to install them additionally with in other languages.
- **Community:** Something very important for the development of a language is the community, the Python community itself takes care of the language and almost all updates are made in a democratic way.

Python – Download and installation

To be able to use Python you have to have it installed on your computer.

There are different ways to do this, but we recommend that you use Anaconda, a Python distribution that incorporates many tools.

Simply choose the version that corresponds to your operating system and install it by following the instructions. Also, make sure you choose the version that comes with Python 3 and not Python 2.

https://www.anaconda.com/

Python – Descarga e instalación

https://www.anaconda.com/

The Anaconda distribution includes many popular tools such as the **Ipython console, Jupyter Notebook, and Spyder IDE.**

Anaconda comes with a package manager called conda, which makes it easy to install and update additional tools (libraries).

Ipython is an enhanced interactive Python interpreter.

Jupyter Notebook is an open source web application that allows you to create and share documents that allow you to easily create documents that combine code, graphics and narrative text.

Spyder is an Integrated Development Environment (IDE) that allows you to write Python scripts and interact with Python software from a single interface.

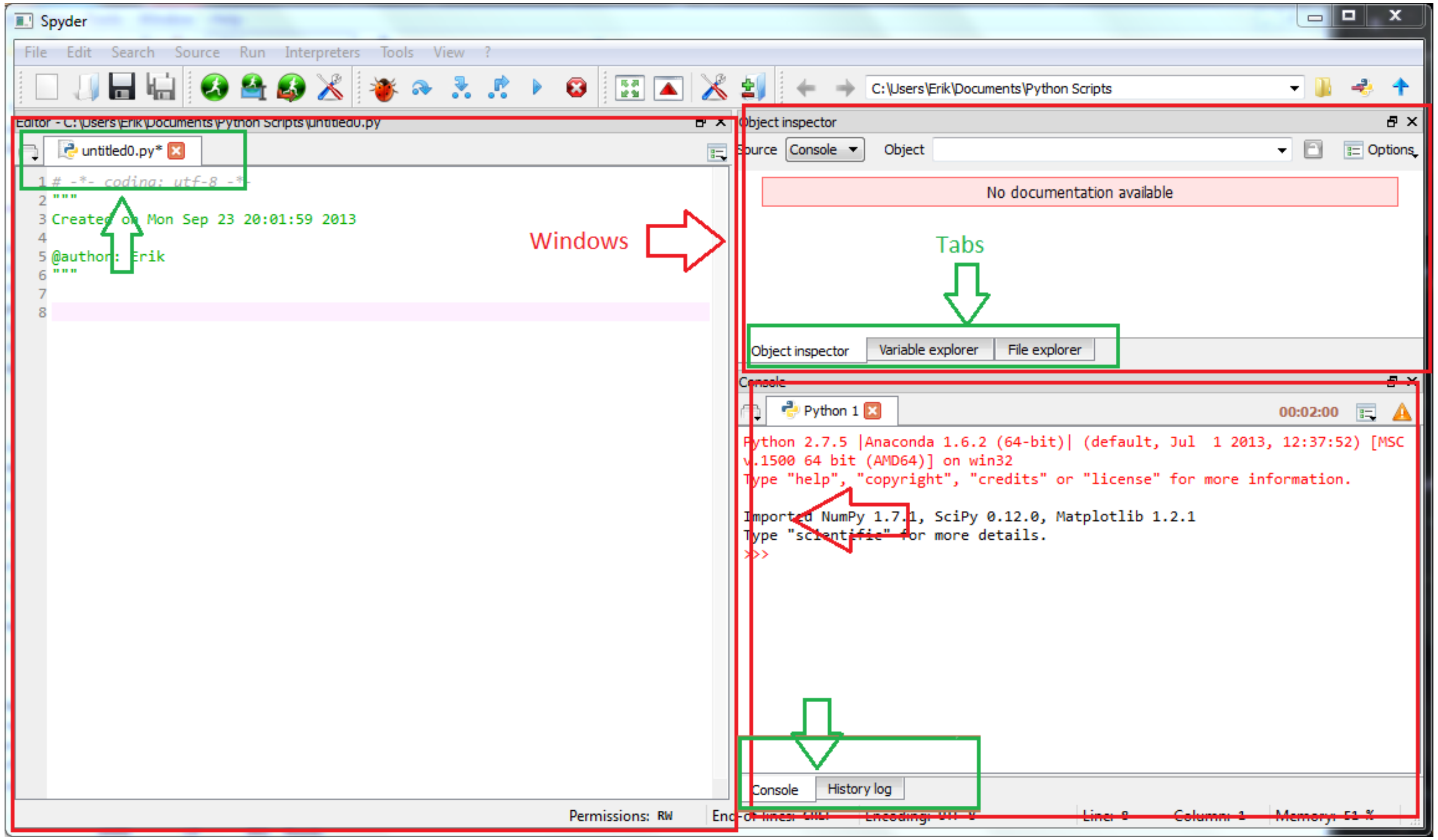
Python – Download and installation

Spyder has advanced editing, interactive testing, debugging and introspection, and a numerical computing environment. Thanks to the support of IPython and popular Python libraries such as Numpy, Scipy or matplotlib (interactive 2D / 3D plotting). Spyder (formerly Pydee) is an open-source, cross-platform integrated development environment (IDE) for scientific programming in the Python language.

SPYDER

https://www.spyder-ide.org/

Spyder interface



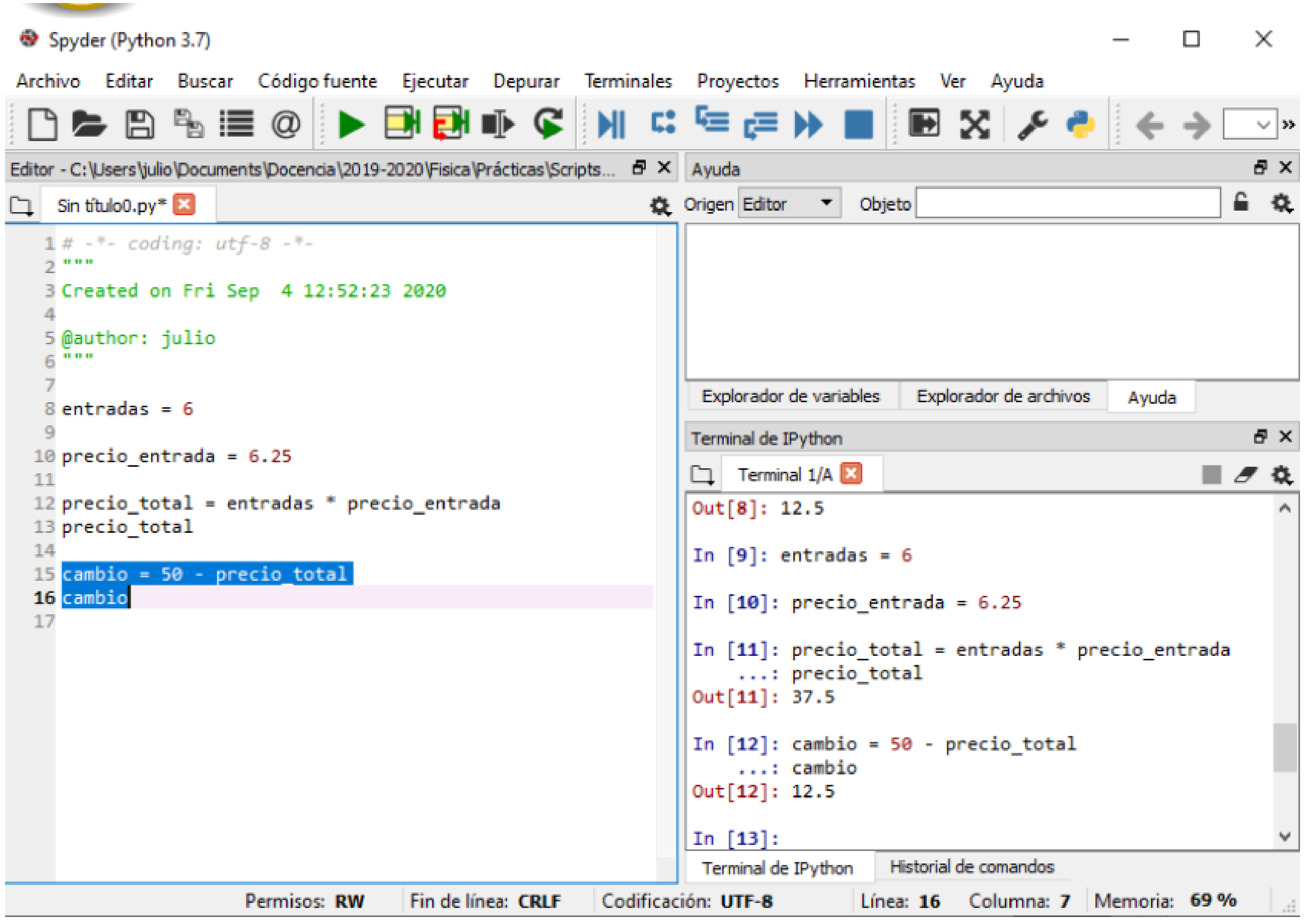
```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 23 20:01:59 2013

@author: Erik
"""
```

```
Python 2.7.5 |Anaconda 1.6.2 (64-bit)| (default, Jul  1 2013, 12:37:52) [MSC
v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

Imported NumPy 1.7.1, SciPy 0.12.0, Matplotlib 1.2.1
Type "scientific" for more details.
>>>
```

Example 1

- A family, made up of 4 children and a married couple, goes to the movies. If each ticket is worth €6.25, how much do they have to pay? If the mother hands over a €50 bill to pay for the tickets, how much will they have to pay back?

Example 1



Editor - C:\Users\julio\Documents\Docencia\2019-2020\Fisica\Prácticas\Scripts...

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep  4 12:52:23 2020

@author: julio
"""

entradas = 6

precio_entrada = 6.25

precio_total = entradas * precio_entrada
precio_total

cambio = 50 - precio_total
cambio
```

```
Out[8]: 12.5

In [9]: entradas = 6

In [10]: precio_entrada = 6.25

In [11]: precio_total = entradas * precio_entrada
    ...: precio_total
Out[11]: 37.5

In [12]: cambio = 50 - precio_total
    ...: cambio
Out[12]: 12.5

In [13]:
```

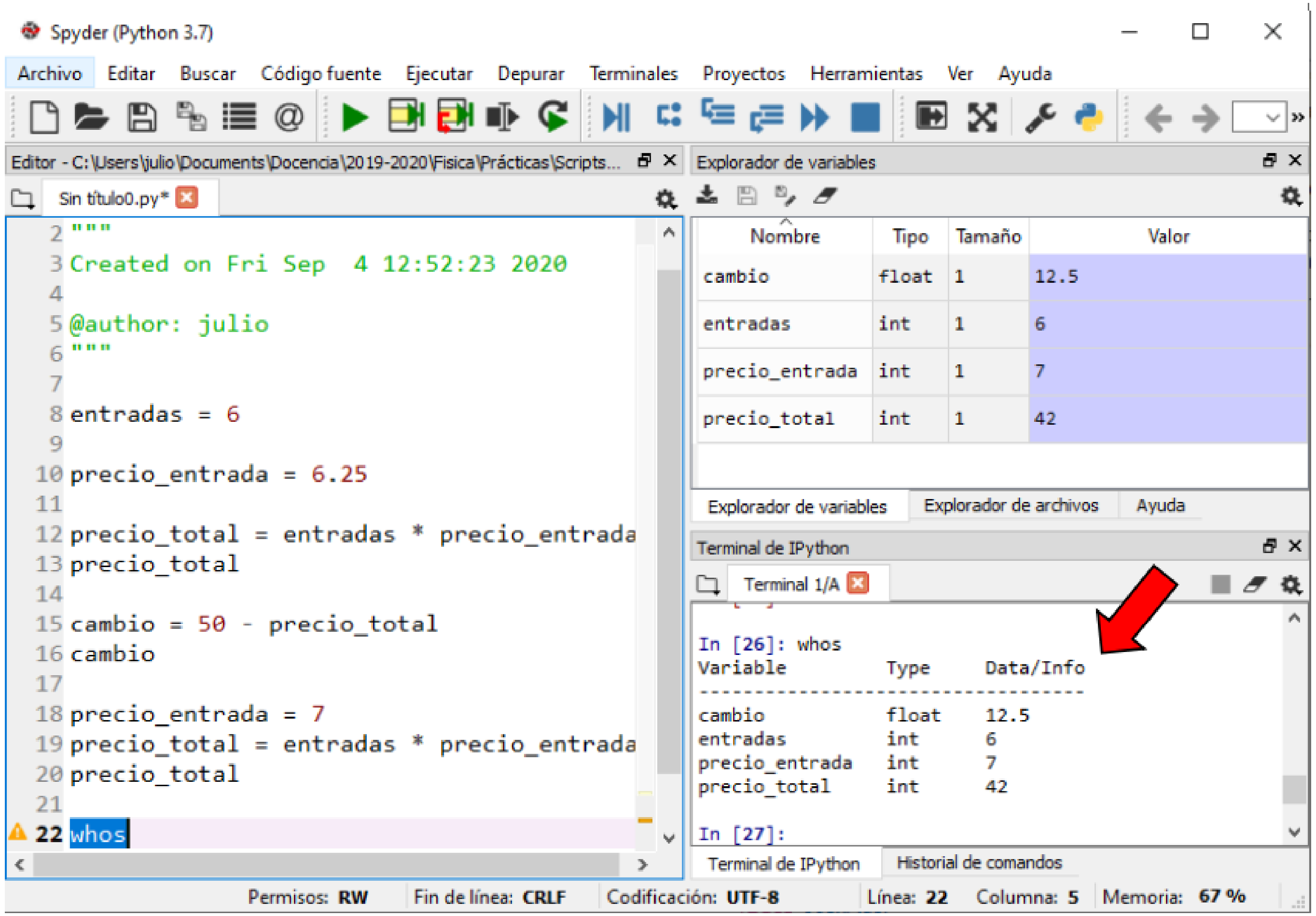
Permisos: RW   Fin de línea: CRLF   Codificación: UTF-8   Línea: 16   Columna: 7   Memoria: 69 %

We can store the information in objects (inputs, precio_entrada, etc.). Once the values are assigned (using =), we can work with the name of the objects. Objects play the role of variables.

The objects

There are many types of objects we can work with. For example, numbers in Python are strongly related to mathematical numbers, but are subject to the limitations of numerical representation in computers. Python distinguishes between integers, floating-point numbers, and complex numbers (more examples later):

Object Identification I


Spyder (Python 3.7)
Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda
Editor - C:\Users\julio\Documents\Docencia\2019-2020\Fisica\Prácticas\Scripts...
Sin título0.py*
Explorador de variables
Explorador de variables Explorador de archivos Ayuda
Terminal de IPython
Terminal 1/A
Terminal de IPython Historial de comandos
Permisos: RW Fin de línea: CRLF Codificación: UTF-8 Línea: 22 Columna: 5 Memoria: 67 %

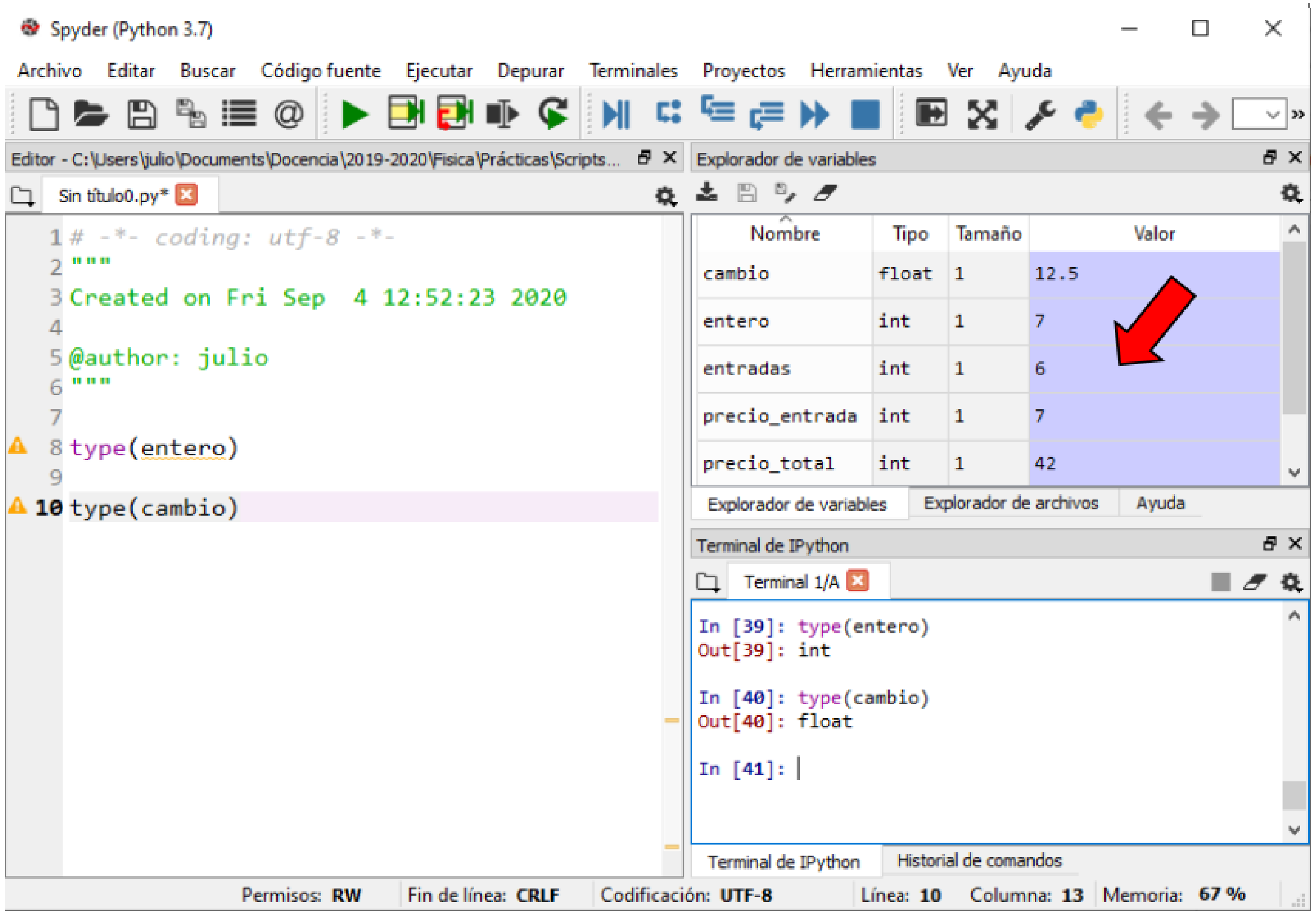
```
"""
Created on Fri Sep  4 12:52:23 2020

@author: julio
"""

entradas = 6

precio_entrada = 6.25

precio_total = entradas * precio_entrada
precio_total

cambio = 50 - precio_total
cambio

precio_entrada = 7
precio_total = entradas * precio_entrada
precio_total

whos
```

Nombre	Tipo	Tamaño	Valor
cambio	float	1	12.5
entradas	int	1	6
precio_entrada	int	1	7
precio_total	int	1	42

```
In [26]: whos
Variable         Type     Data/Info
-----------------------------------
cambio           float    12.5
entradas         int      6
precio_entrada   int      7
precio_total     int      42

In [27]:
```

If we do not remember the names of the variables (or their value), we can consult it in the Variable Explorer (just above the console). Likewise, we can also use the whos function, which returns in the console a table similar to the one that appears in the Variable Explorer.

Object Identification II



```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep  4 12:52:23 2020

@author: julio
"""

type(entero)

type(cambio)
```

Nombre	Tipo	Tamaño	Valor
cambio	float	1	12.5
entero	int	1	7
entradas	int	1	6
precio_entrada	int	1	7
precio_total	int	1	42

```
In [39]: type(entero)
Out[39]: int

In [40]: type(cambio)
Out[40]: float

In [41]:
```

Permisos: RW   Fin de línea: CRLF   Codificación: UTF-8   Línea: 10   Columna: 13   Memoria: 67 %

The Variable Explorer also displays the type of the different objects. Even so, if we wanted to remember it without going to the browser, we can use the type function.

Basic Elements – Strings

In the handling of text, the use of strings (sequence) or also called 'strings' is made

s = 'abc'

Its use is wide and can be reinforced with different methods and functions.

How could we know the number of characters within a string?

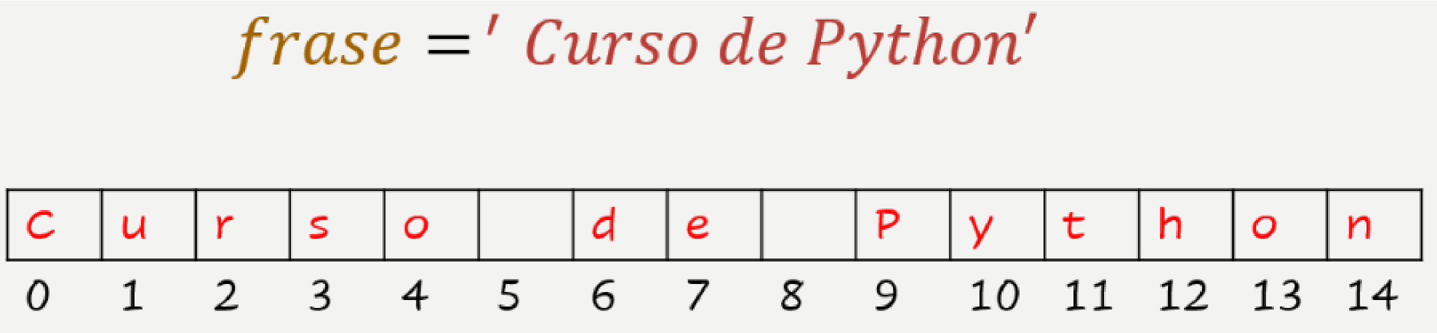
Basic Elements – Strings

The len(string) function allows us to know the number of characters.

len("abc") 3

It is important to know how we can extract the characters within the strings either to explore their content or to process it and use it later. To do this, indexing will be used

Basic Elements – Strings

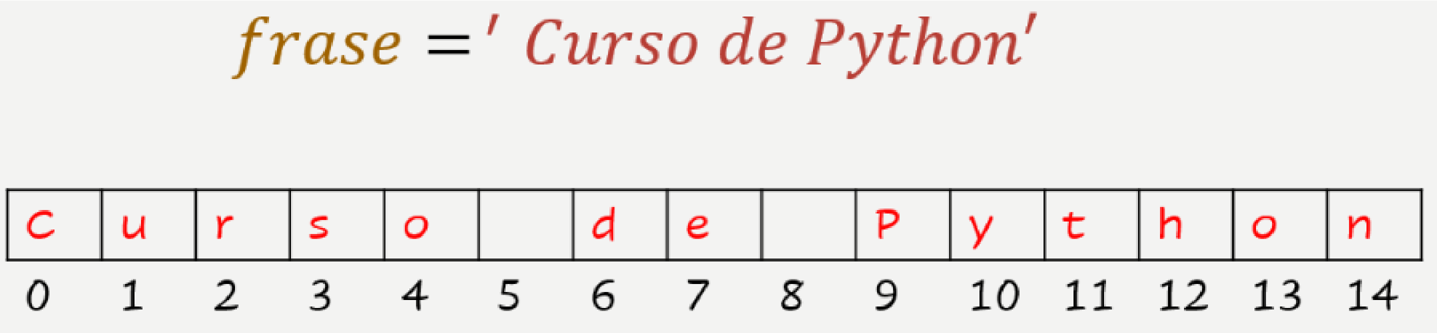


frase =' Curso de Python'

C	u	r	s	o		d	e		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Note that when an object of type string is created, it is stored in small memory positions always starting with the zero position, up to the size of the string, in this case up to 14, having a total of 15 characters.

Basic Elements – Strings



$$frase =' \ Curso \ de \ Python'$$

c	u	r	s	o		d	e		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Now, from the index (position of the characters) we can perform data extraction (split). Note that in Python, the index starts at 0 and the last element will always be -1 of the total size (len -1)

Basic Elements – Strings

Sintax: [start:stop:step]

```
s = "abcdefgh"
s[3:6] “def", just like S[3:6:1]
s[3:6:2] Delivery “df"
s[::] delivery "abcdefgh", as well as s[0:len(s):1]
s[::-1] delivery "hgfedbca",
like s[-1:-(len(s)+1):-1]
S[4:1:-2] delivery “ec"
```

Basic Elements – Strings

Exercise 1.

From the following string Clase de Strings en Python, 25 de enero ', extract the following using indexing:

- Strings
- Python
- 25
- enero
- esalC
- 'oeee 2,otPn git deaC'

Basic Elements – Strings

Strings have different methods associated with their data type. Among many, we can highlight:

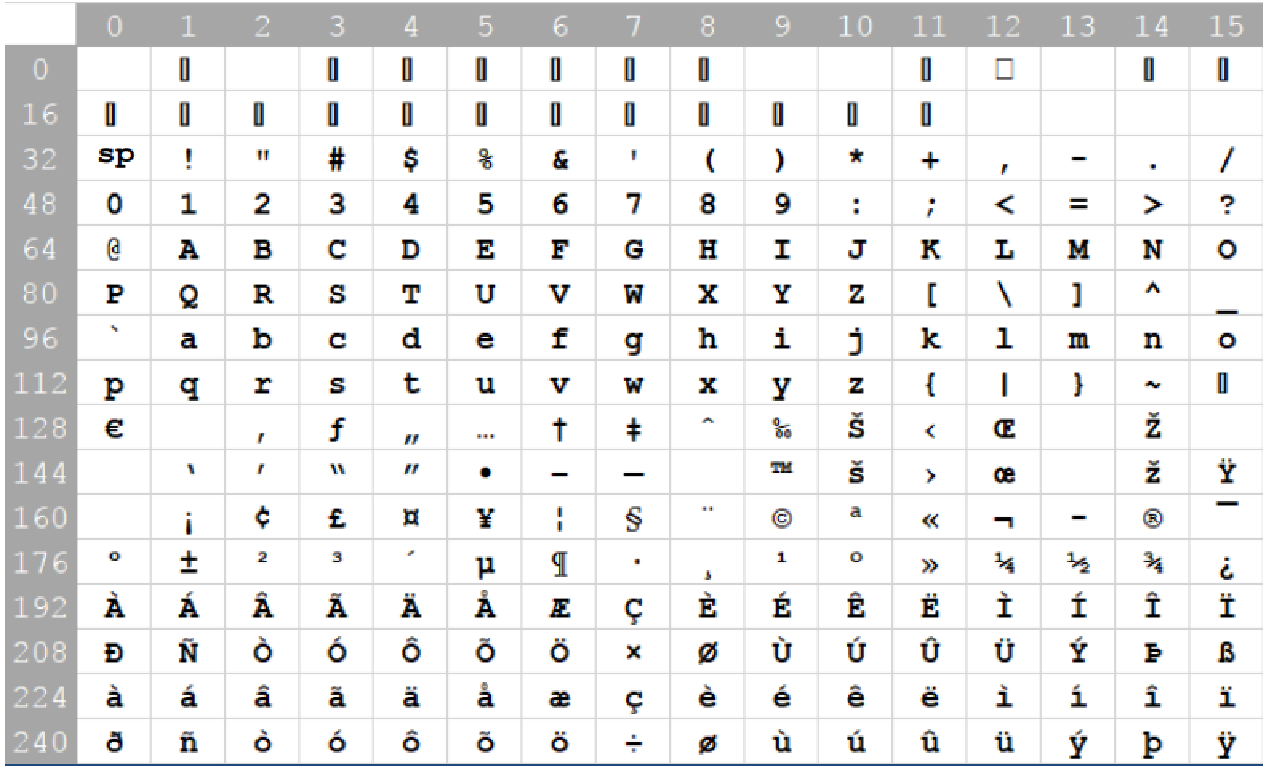
count()	Returns the number of times a character is repeated in the string
find	Searches for a substring or character within the character string
replace(str1,str2)	Replace one substring with another

upper()	Capitalize the entire string
lower()	Set the entire string to lowercase
capitalize()	Capitalize the entire first
title()	Capitalize the entire first character of each word

https://docs.python.org/3/library/stdtypes.html#string-methods

Basic Elements – Strings

The ASCII Table gathers the first 127 bits is common to all Western languages. The rest are language-specific. numeric that corresponds to the selected character.

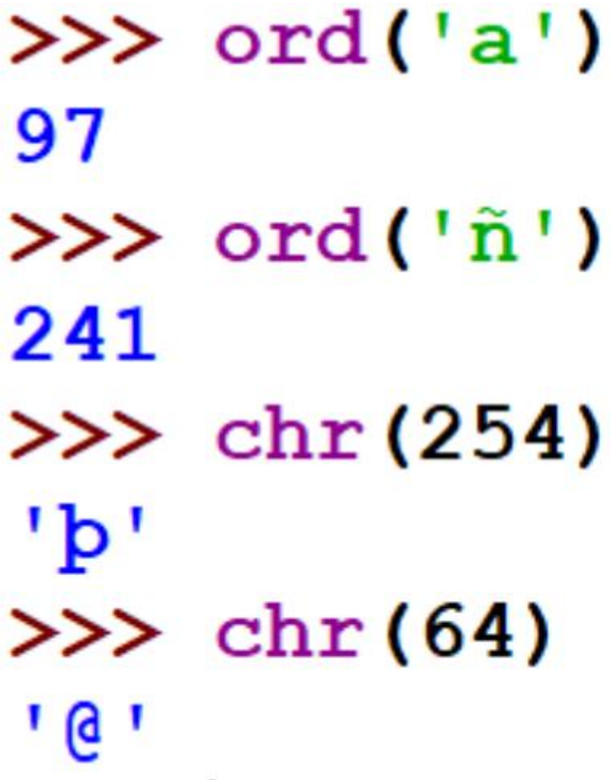


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0		▯		▯	▯	▯	▯	▯	▯			▯	□		▯	▯
16	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯	▯				
32	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	▯
128	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
144		‘	’	“	”	•	–	—		™	š	›	œ		ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Basic Elements – Strings

The "ord(character)" function indicates the numeric ASCII code that corresponds to the selected character.

The function "chr(value)" displays the character that is associated with the specified ASCII value:



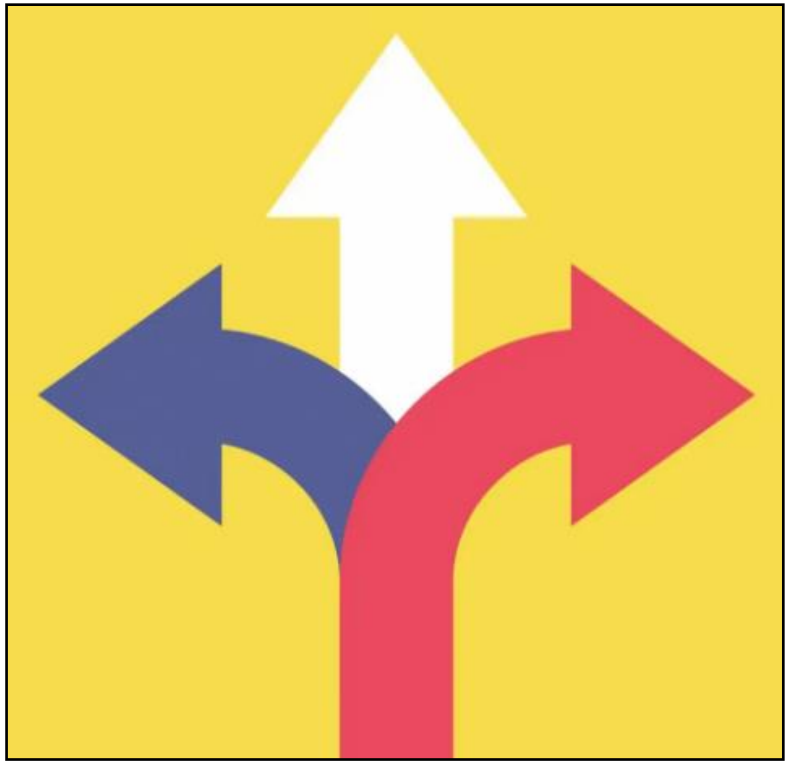
```
>>> ord('a')
97
>>> ord('ñ')
241
>>> chr(254)
'þ'
>>> chr(64)
'@'
```

Selection structures – I

- So far we have seen that programs run in an order that we define when encoding each line. In our case, the order of execution usually starts from within a method called "main".
- Regardless of the data we enter (or that we ask the user for on the screen), the order (control flow) is the same.
- This means that we do not have a different answer or a different order when different inputs or values are entered.
- But what if we can define a different order, depending on the inputs of the users, or the values of certain variables.

Selection structures – II

- In this case we could, for example: (i) execute the program in a normal order, if the value of a radius is greater than zero, or (ii) execute another path if the value of the radius is less than zero.
- To be able to give our code this ability. We need to use "selection structures".



Comparison operators

- How do I know if a radius is: (i) greater than 0, (ii) equal to 0, or (iii) less than 0?.
- Python provides six relationship operators (also known as comparison operators). Let's look at the following table and find the result assuming that the value of the variable "radius" is equal to 5.

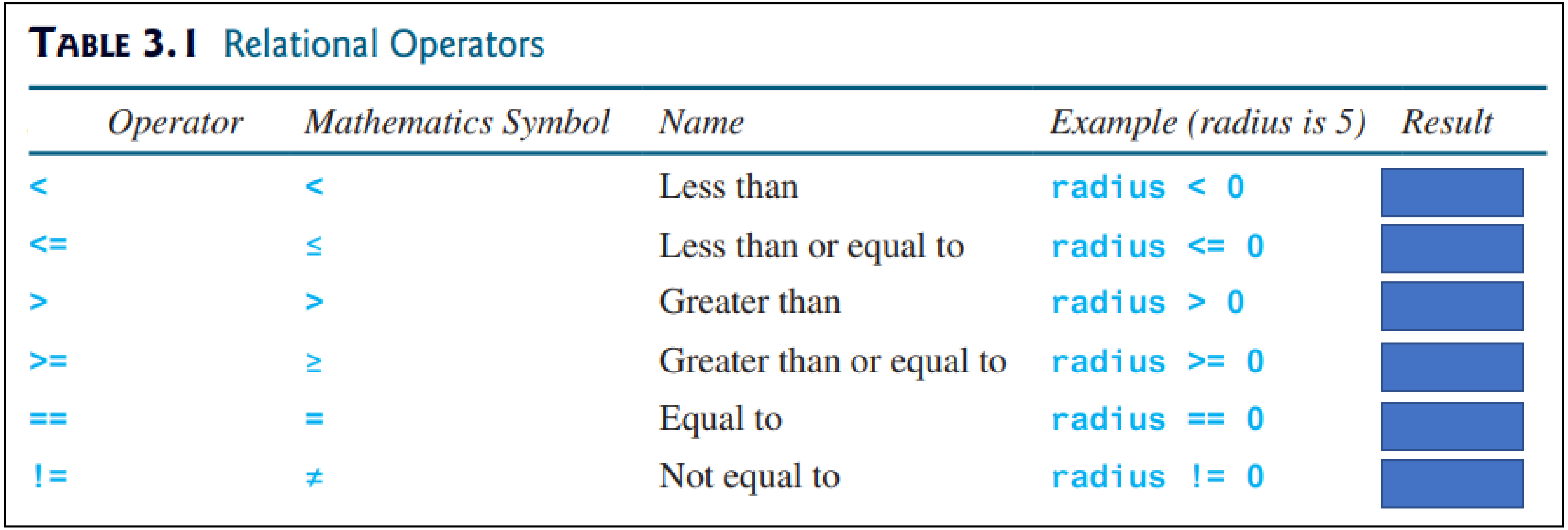


TABLE 3.1 Relational Operators

Operator	*Mathematics Symbol*	*Name*	*Example (radius is 5)*	*Result*
`<`	<	Less than	`radius < 0`	🟦
`<=`	≤	Less than or equal to	`radius <= 0`	🟦
`>`	>	Greater than	`radius > 0`	🟦
`>=`	≥	Greater than or equal to	`radius >= 0`	🟦
`==`	=	Equal to	`radius == 0`	🟦
`!=`	≠	Not equal to	`radius != 0`	🟦

Estructura if-then

It is the most basic of the selection structures.

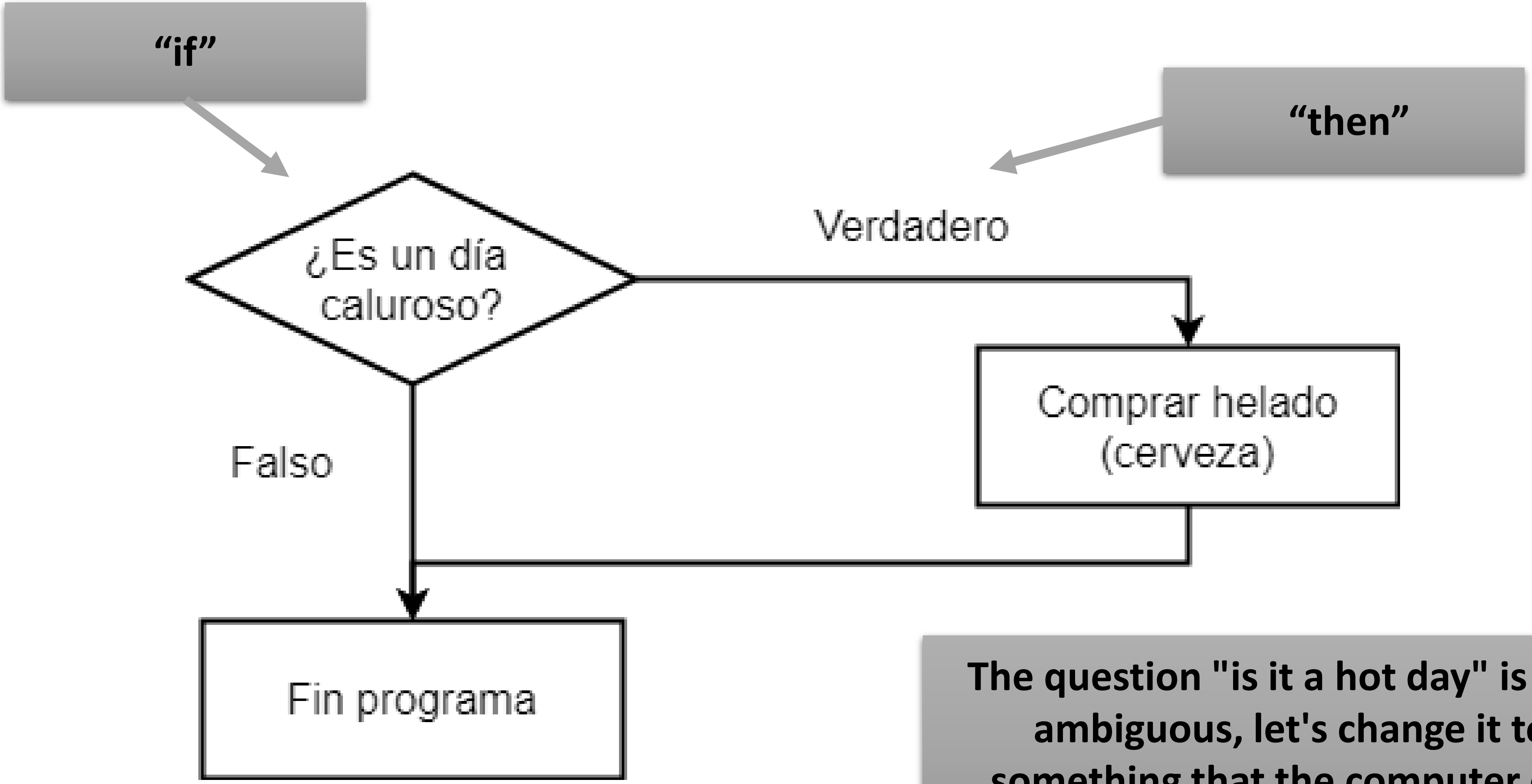
Its operation is as follows:

If a particular condition is true, then the "then" portion is executed; otherwise, the "then" portion is not executed.

Example:

Similar to natural languages where you can say: Is it a hot day? So, I buy an ice cream (a beer).

Estructura if-then – II



The question "is it a hot day" is very ambiguous, let's change it to something that the computer can understand.

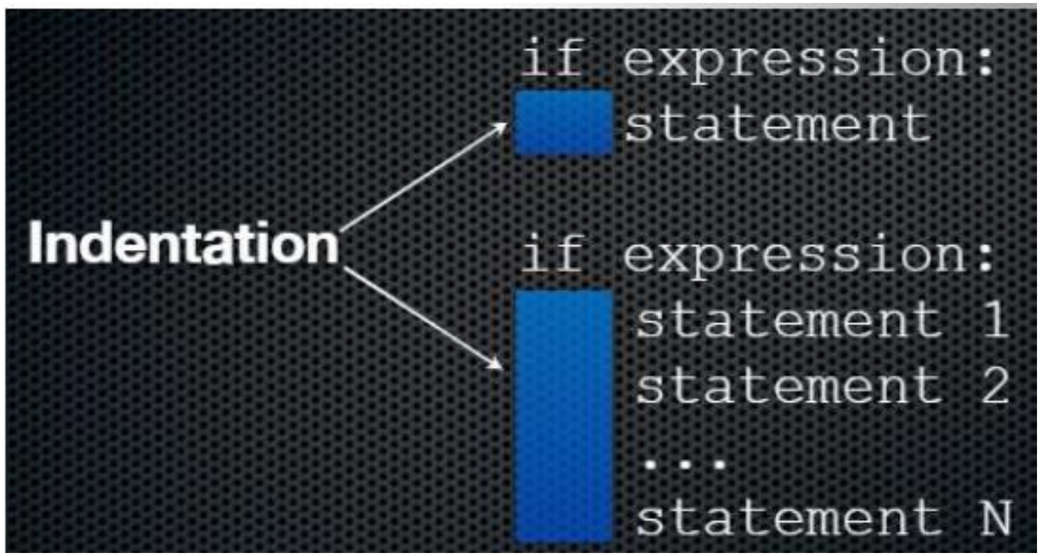
Pseudo-Code ice cream example

Pseudo-code program

```
Read temp
If temp > 27 then
    Print "Buy ice cream (beer)"
Print "End of program"
```

Sintax Python

- The tab is the space that tells the program what is or is not part of the If condition.
- All expressions that are indented will be included within this condition.



```
if expression:
    statement

if expression:
    statement 1
    statement 2
    ...
    statement N
```

Python Code

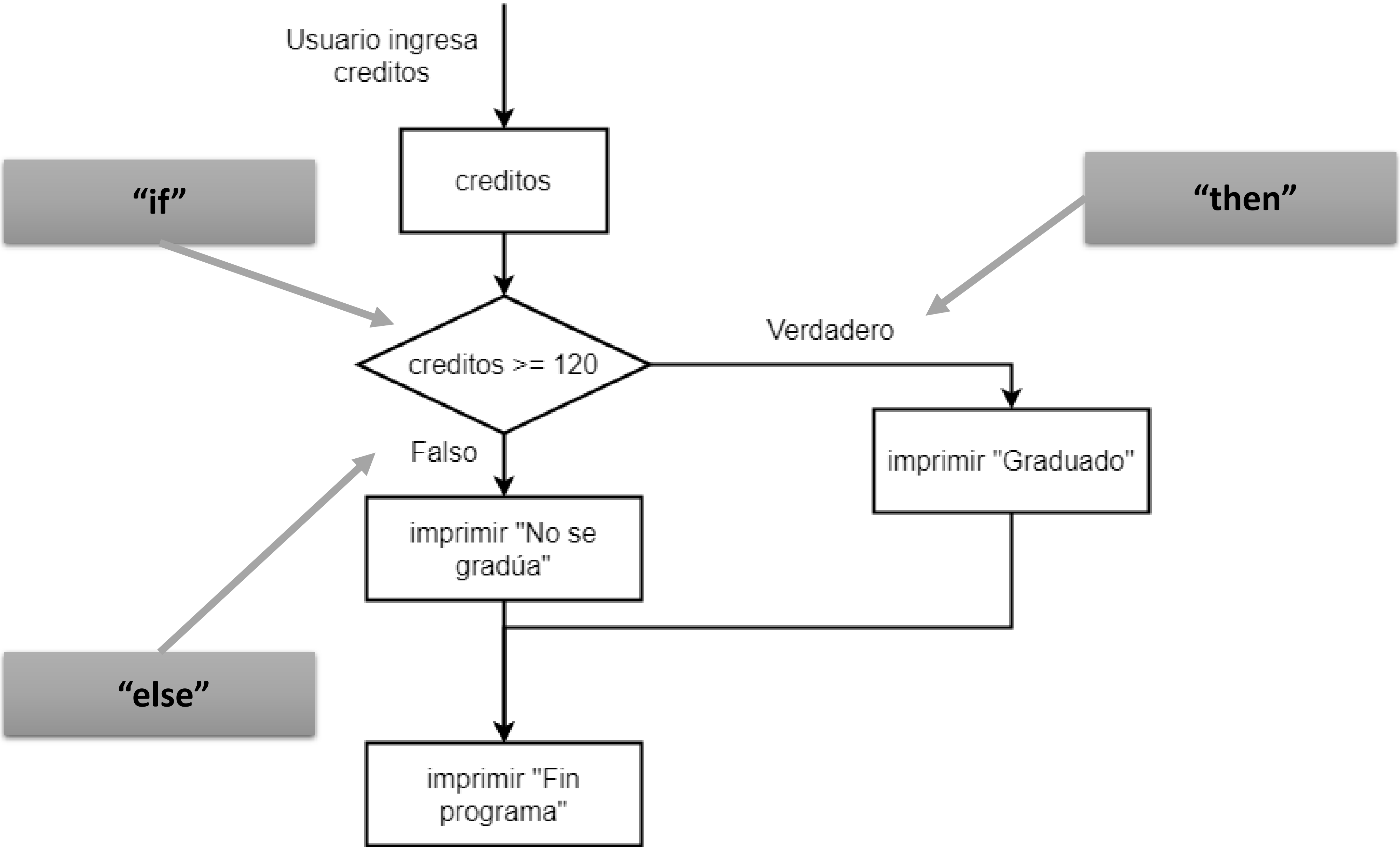
Structure If-then-else

- The If-then structure is very necessary to change the flow control of a program (offer several paths).
- However, what happens if the program is required to do something in both alternatives: true case and false case.
- Example: if the total number of approved credits is greater than or equal to 120, the person graduates; otherwise he does not graduate.

Pseudo-Code example

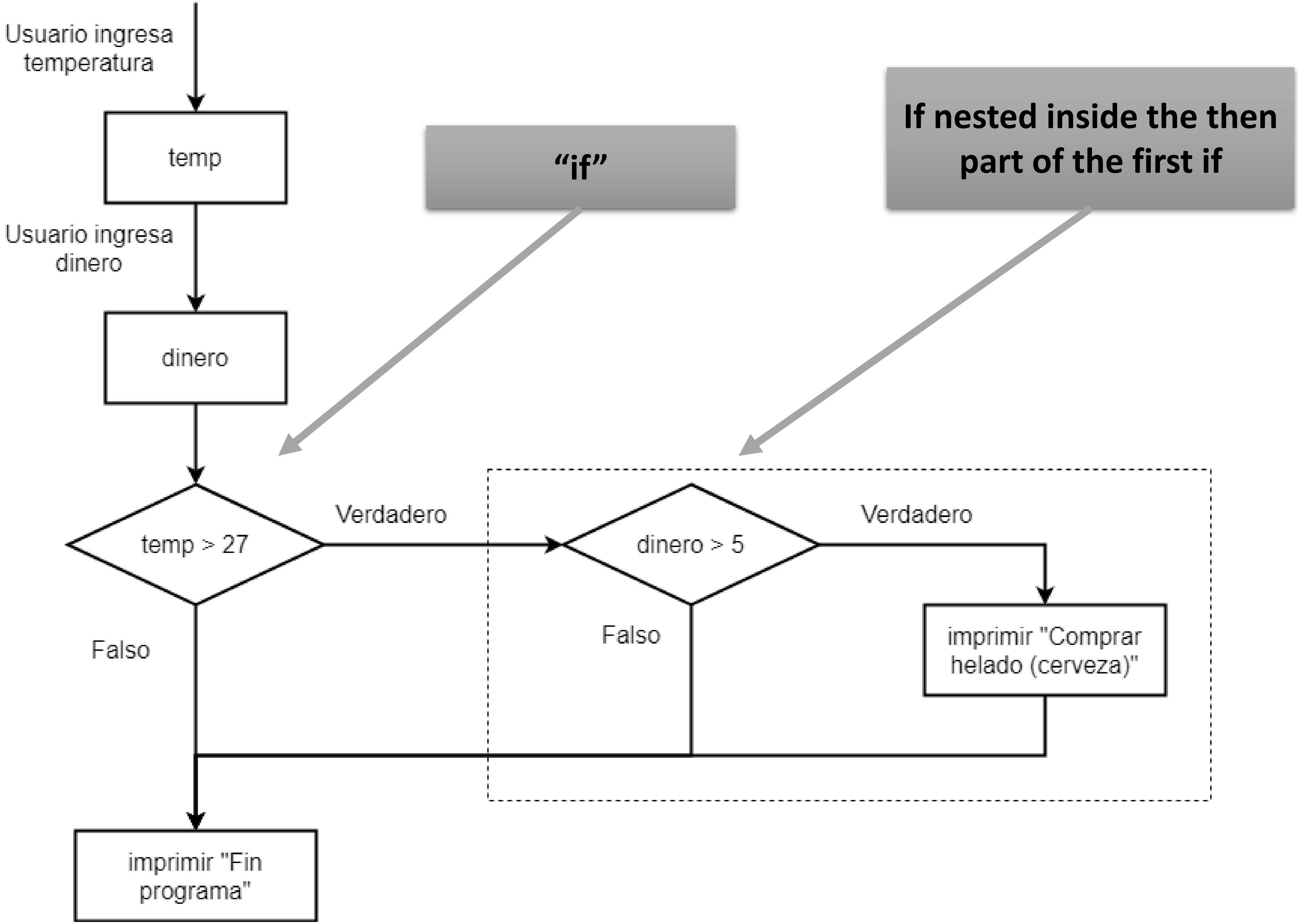
Pseudo-code program

```
Read credits
If credits ≥ 120 Then
    Print "Graduated"
But
    Print "Not graduating"
Print "End of program"
```


Usuario ingresa creditos
creditos
"if"
"then"
creditos >= 120
Verdadero
Falso
imprimir "Graduado"
imprimir "No se gradúa"
"else"
imprimir "Fin programa"

Nested if

It is also possible to contain ifs, within other ifs. Let's look at the following example

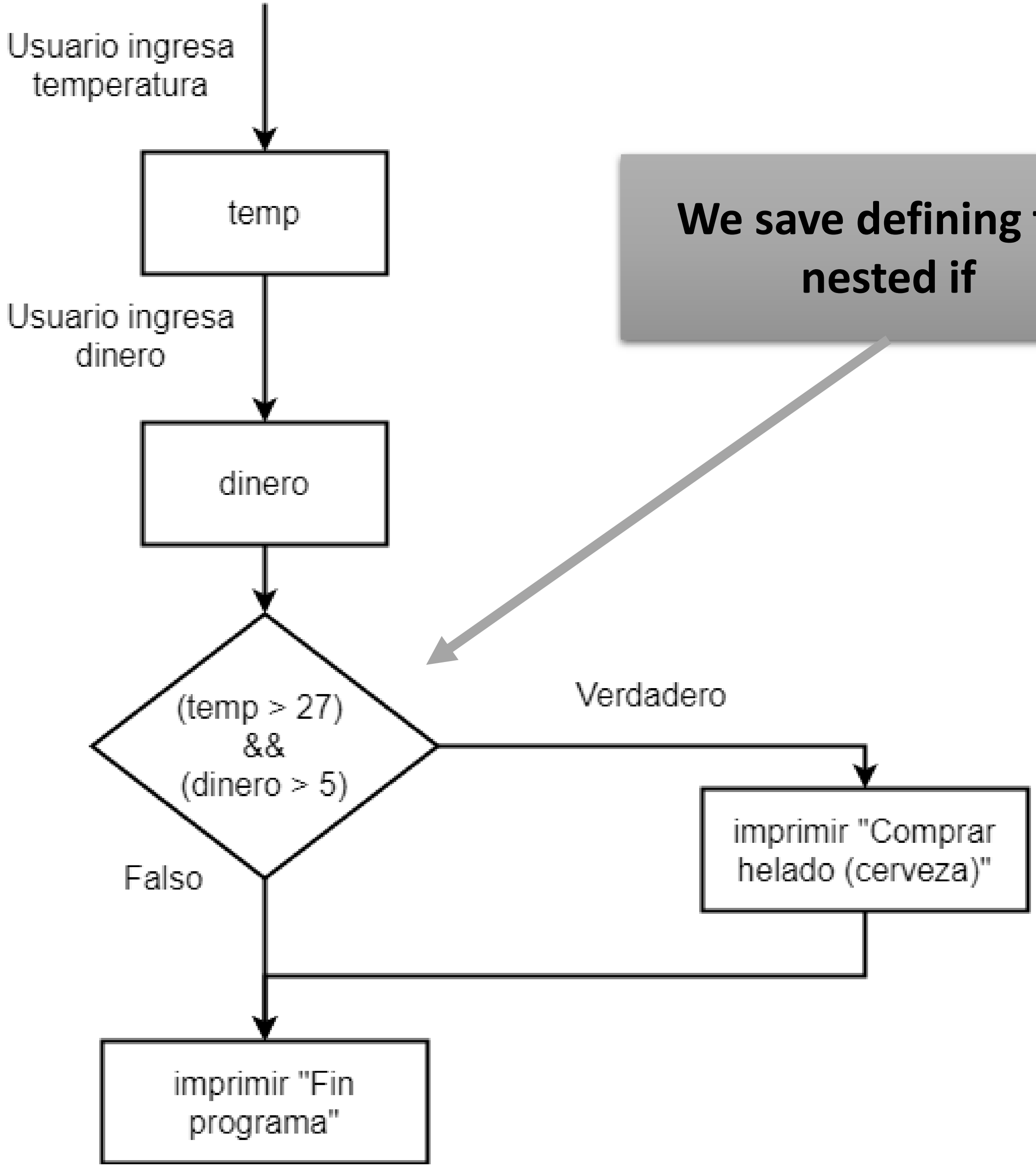


"if"

If nested inside the then part of the first if

Logical operators

- In many cases, nested ifs can be "summarized" by using logical operators.
- Logical operators are used for Boolean algebra operations, that is, to describe logical relationships, expressed as true or false.
- Java provides three fundamental logical operators, called: "and" (&&), "or" (||), and "not" (!).
- Let's see how to use them to reduce the number of lines of code in the previous example.


Usuario ingresa temperatura
temp
Usuario ingresa dinero
dinero
(temp > 27) && (dinero > 5)
Verdadero
Falso
imprimir "Comprar helado (cerveza)"
imprimir "Fin programa"

We save defining the nested if

Table of truth

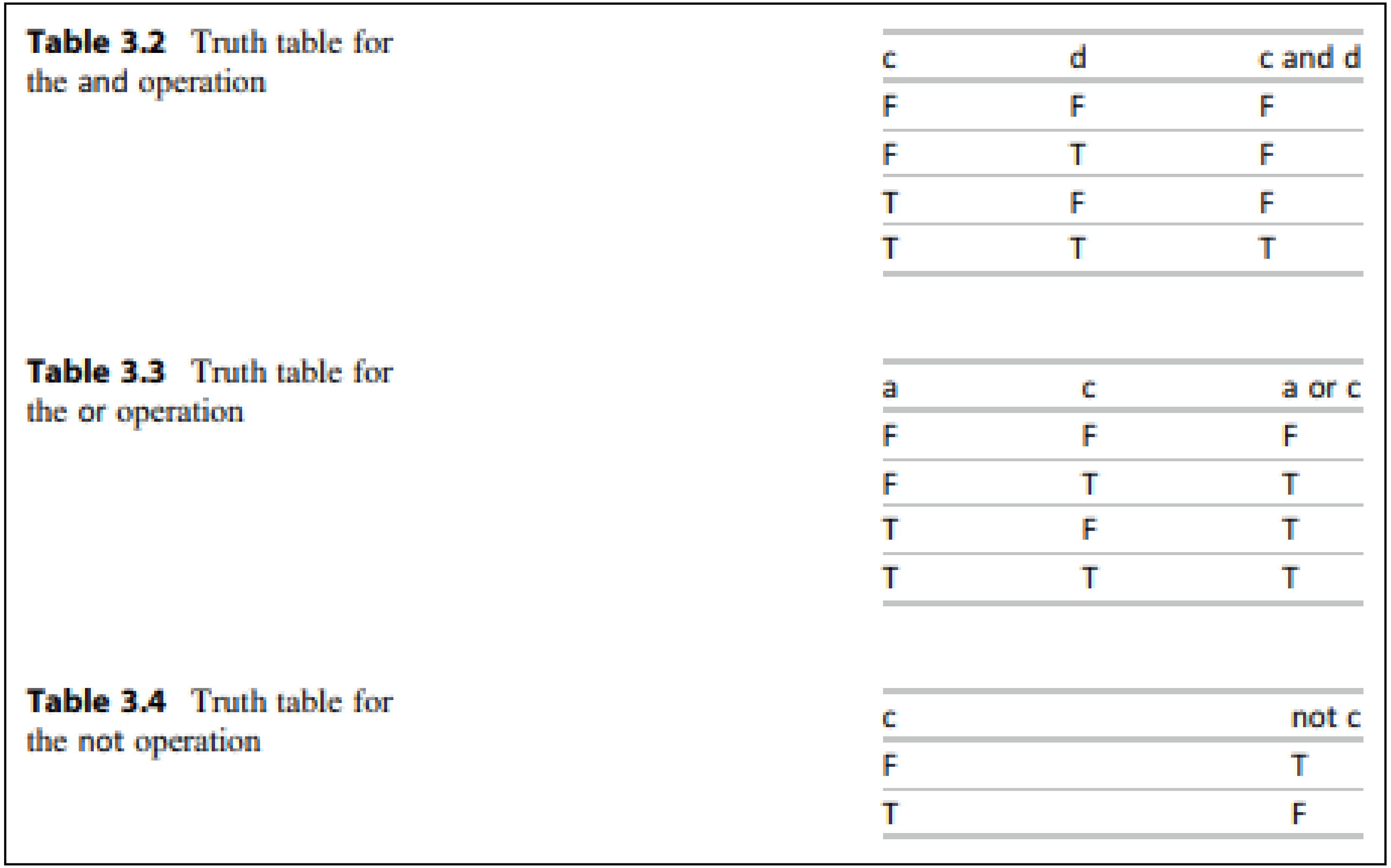


Table 3.2 Truth table for the and operation

c	d	c and d
F	F	F
F	T	F
T	F	F
T	T	T

Table 3.3 Truth table for the or operation

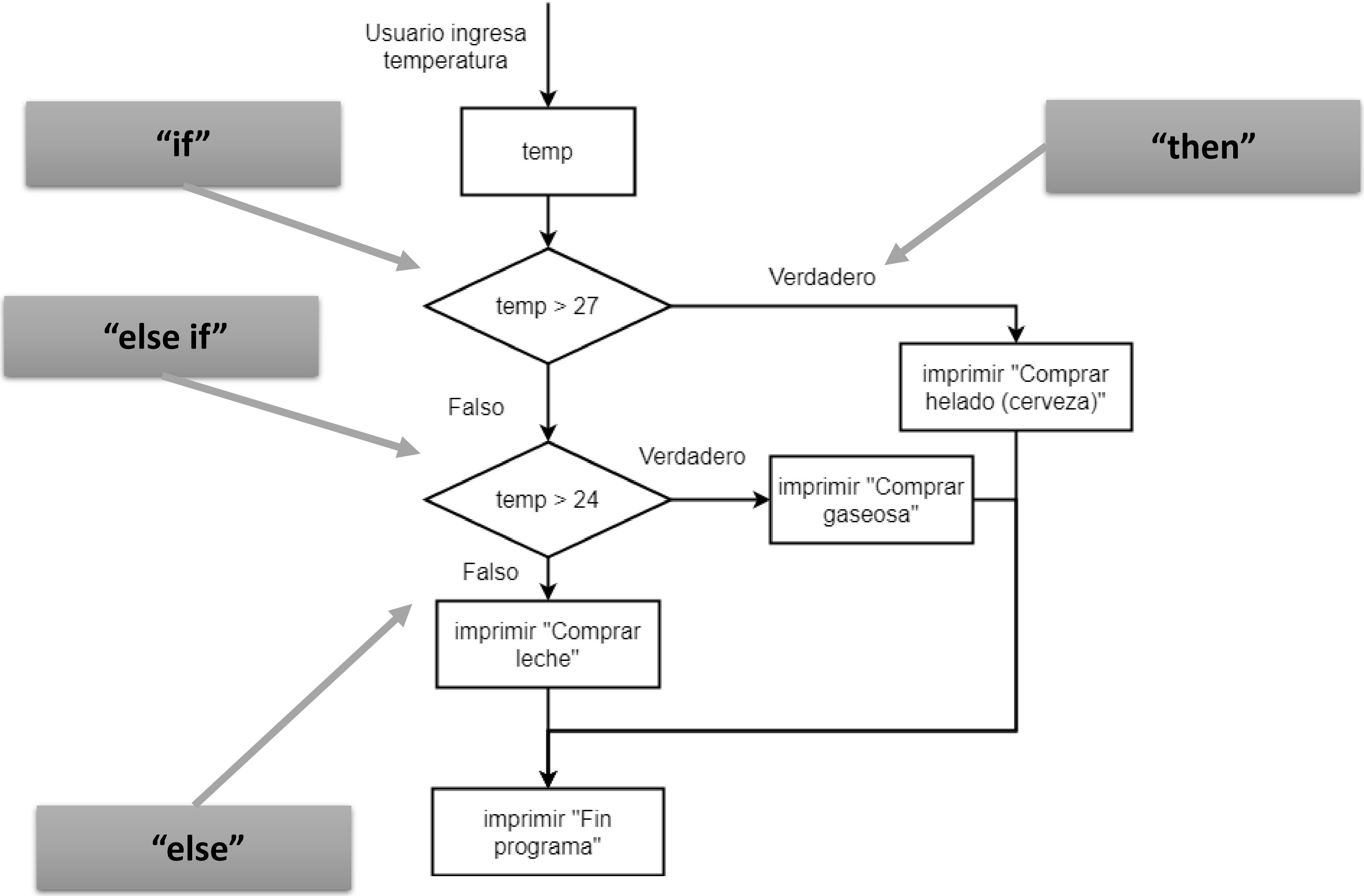
a	c	a or c
F	F	F
F	T	T
T	F	T
T	T	T

Table 3.4 Truth table for the not operation

c	not c
F	T
T	F

Structure If-then-else if-...

- The If-then-else if-... structure It allows us to offer multiple paths depending on multiple checks.
- The else if is used to add a new condition in case the previous condition is not met (is false).
- An if statement can include any number of else if blocks.
- "else if" means (otherwise, if ...).
- At most, a single else if is executed, and that else if will be executed only if all of the above conditions were evaluated as false.
- Example: if the temperature is higher than 27 we buy ice cream (beer); otherwise, if the temperature is higher than 24 we buy soda; Otherwise, we buy milk.


Usuario ingresa temperatura
temp
temp > 27
Verdadero
imprimir "Comprar helado (cerveza)"
Falso
temp > 24
Verdadero
imprimir "Comprar gaseosa"
Falso
imprimir "Comprar leche"
imprimir "Fin programa"
"if"
"then"
"else if"
"else"

Exercise 2

Prompt the user for a password and verify that it has the following characters

First capital letter

At least one numeric digit

At least one of these symbols: @"$%&

At least 10 digits

Basic Elements II – Lists

Similar to a string that can store a group of characters, an array can be used to store a group of numbers of type int or double.
Arrays can not only store numbers, but can also be used to store a group of strings, objects, and even other arrays.
The arrangements are extremely useful for:

Group data.
Reduce the definition of variables.
Among others.

Basic Elements II – Lists

- Suppose you want to store the value of 100 whole numbers (ages 100 people).
- With what has been seen so far, we would have to define 100 int variables, and store in each of them the value of each age of each person.
- With an array or Lists in Python, we can define a single variable, which has a size of 100, and store in it all 100 ages.

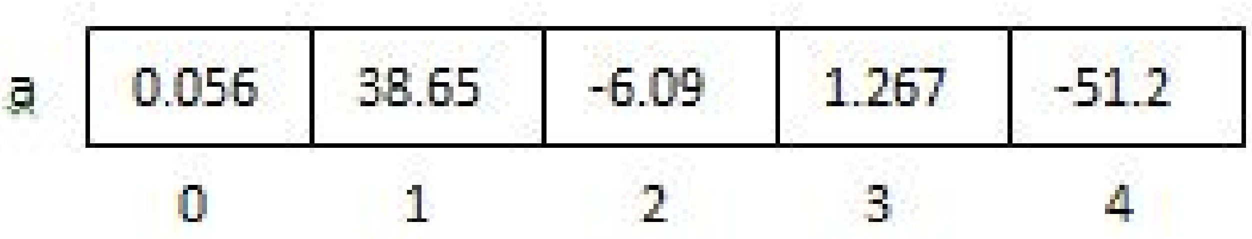
Basic Elements II – Lists

- Mutable and organized set of elements.
- It is declared and represented by enclosing the members in square brackets.
- The items in a list are variable, and you can add and remove items at any time.
- Its elements can be modified.
- Range of elements: 0 to length-1
- May have repeated items

Basic Elements II – Lists

```
a = [0.056,  38.65,-6.09,1.267,-51.2]
```

Representación interna:



a	0.056	38.65	-6.09	1.267	-51.2
	0	1	2	3	4

Basic elements II – Lists – Functions and methods

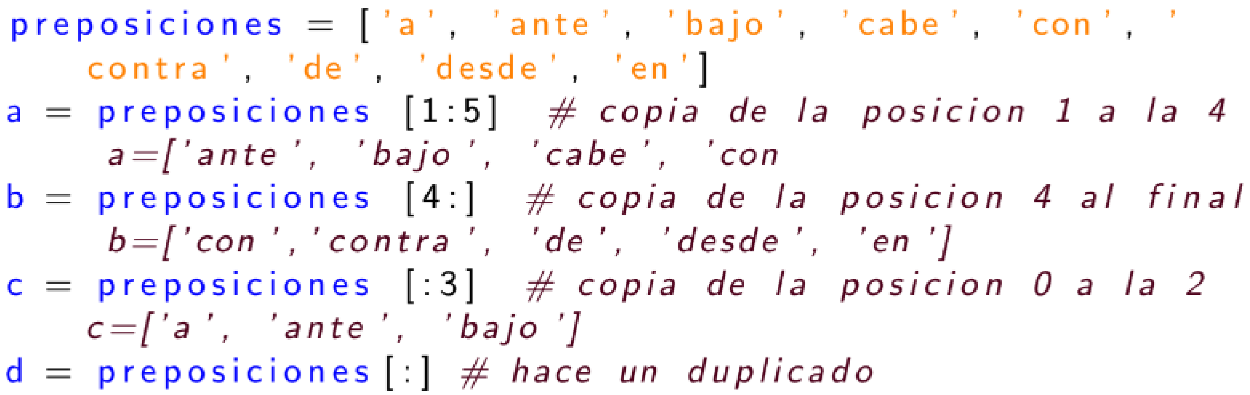
- lista.index(elem) - Returns the position of the item in the list. If it is not, an exception is thrown.
- list[i] - Access the item at position i.
- elem in list - Determines whether or not an item is in the list.
- lista.count(elem) - Number of times an item appears in the list.

Basic elements II – Lists – Functions and methods

- len(list) - returns the size of the list (num. elements).
- list.append(element) - A˜nade an element at the end of the list.
- L1.extend(L2) - a˜nade to L1 all elements of L2.
- list1 + list2 - produces a list with the items of both.
- list.insert(pos,elem) - inserts an element into a position.
- lista.remove(elem) - Deletes the first appearance of the element.
- lista.sort() - Sorts the list
- lista.reverse() - Reverses the order of the items in the list
- lista.pop() - Deletes and returns the last item
- + (concatenate) and * (repeat): Ex. list1 = list2 * 3

Basic elements II – Lists – Slicing

- As in Strings, it is a shorthand way to obtain subsequences from sequences
- Existing
- Syntax: sequence [start:end:break]
- If start is not specified, it is understood from 0, if no end is specified,
- is understood until the end of the sequence
- Examples:



```
preposiciones = ['a', 'ante', 'bajo', 'cabe', 'con', '
    contra', 'de', 'desde', 'en']
a = preposiciones [1:5]  # copia de la posicion 1 a la 4
     a=['ante', 'bajo', 'cabe', 'con
b = preposiciones [4:]  # copia de la posicion 4 al final
     b=['con','contra', 'de', 'desde', 'en']
c = preposiciones [:3]  # copia de la posicion 0 a la 2
    c=['a', 'ante', 'bajo']
d = preposiciones[:] # hace un duplicado
```

Exercise 1

From the list

L1=['abc', 123, [456, 'd', 'e', 'f', [7, 8, 9],['g','h','i']], 10, 'j', 'k'],

create

L2=[['abc', 456, 7], 'def', [8, 9], 'g', 'h', [10, 'j', 'k']]

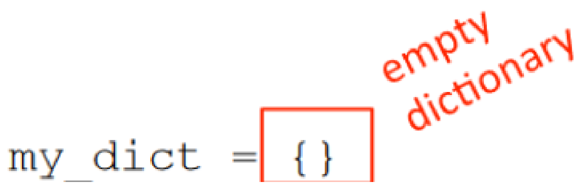
Using Indexing Methods (Slicing)

Basic Elements II – Dictionaries

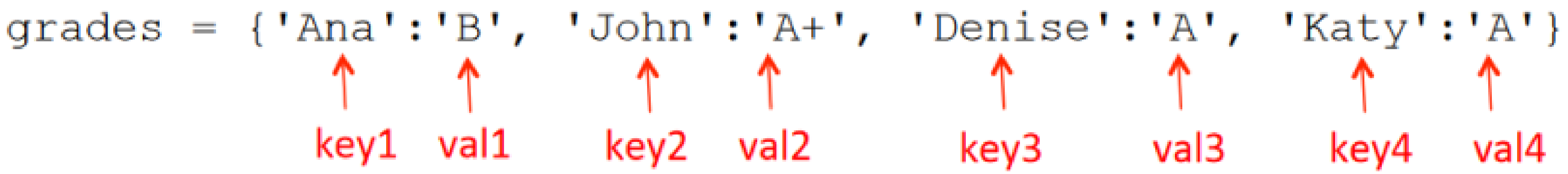
- A data structure with no order between elements, and in which access is determined by a unique key that is associated with each value.
- The key is often a text string, although it can be any of Python's immutable types.
- Key types: boolean, integer, float, tuple, string ...
- Dictionaries are mutable: you can add, delete, and change their items

Basic Elements II – Dictionaries

- To create a dictionary, square brackets ({}) are used around comma-separated key:value pairs.
- The simplest dictionary is a vacío dictionary, which contains no key or value at all:
- Example



```
my_dict = {}
```



```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

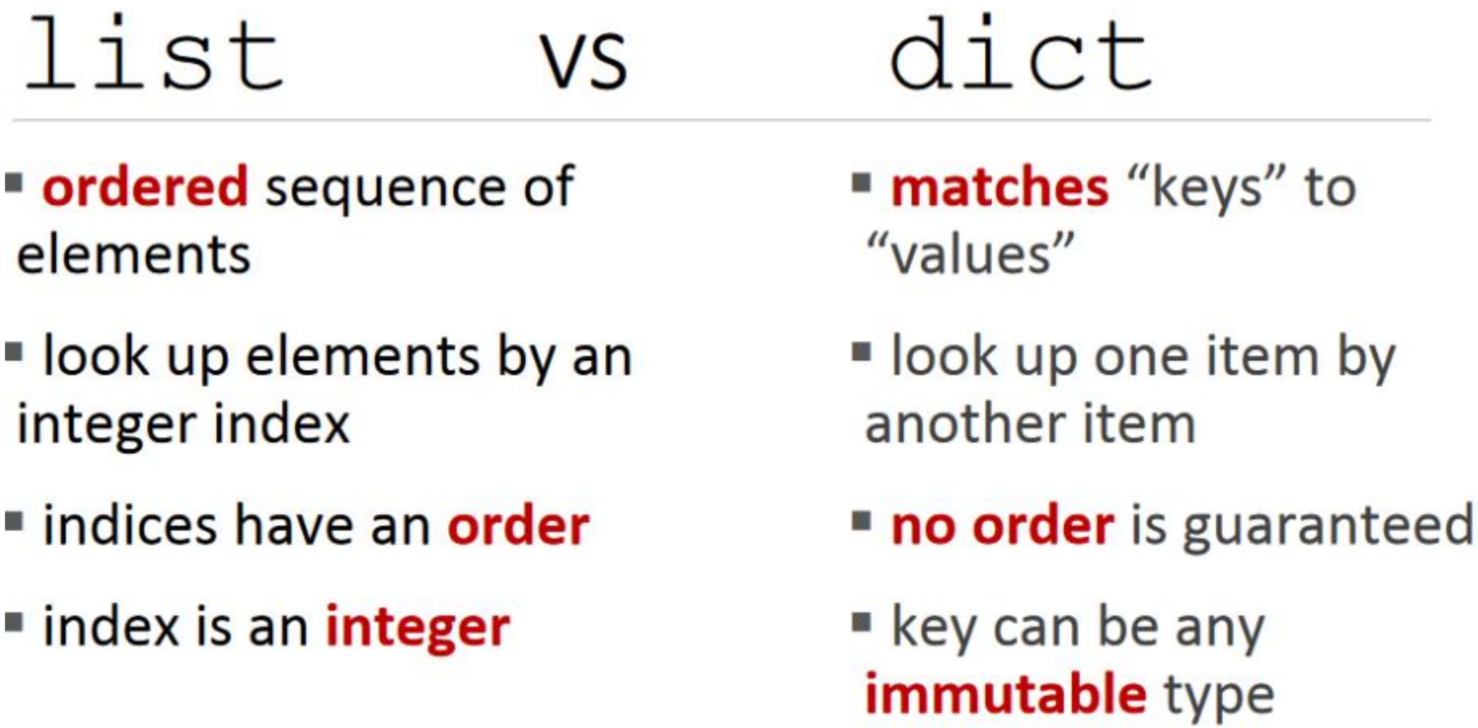
Basic elements II – Dictionaries - Operations

- Add an element: simply refer to thé ıtem by its key and assign it a value.
- If the key was already in the dictionary, the existing value is replaced with the new one, if the key is new, it is added to the dictionary with its value.
- Impossible to err by specifying an out-of-range index.
- Delete an item: use from. Example: del bierce['dia']
- Restart by deleting all items: clear. Example: bierce.clear()
- Key membership: in. Example: 'Ana' in grades (example above)
- Number of elements: len(dictionary)
- Replica: new dictionary = dictionary.copy()

Basic elements II – Dictionaries - Operations

- Key list: keys. Example: grades.keys()
- Returns an object of type dict keys, which is an iterable form of list. You can convert to list with list().
- List of values: values. Example: grades.values()
- Returns an object of type dict values, an iterable form of list. Example: for item in x.valueus(): print item
- List of items: like values and keys, it is iterable: dictionary.items()

Basic elements II



`list` vs `dict`

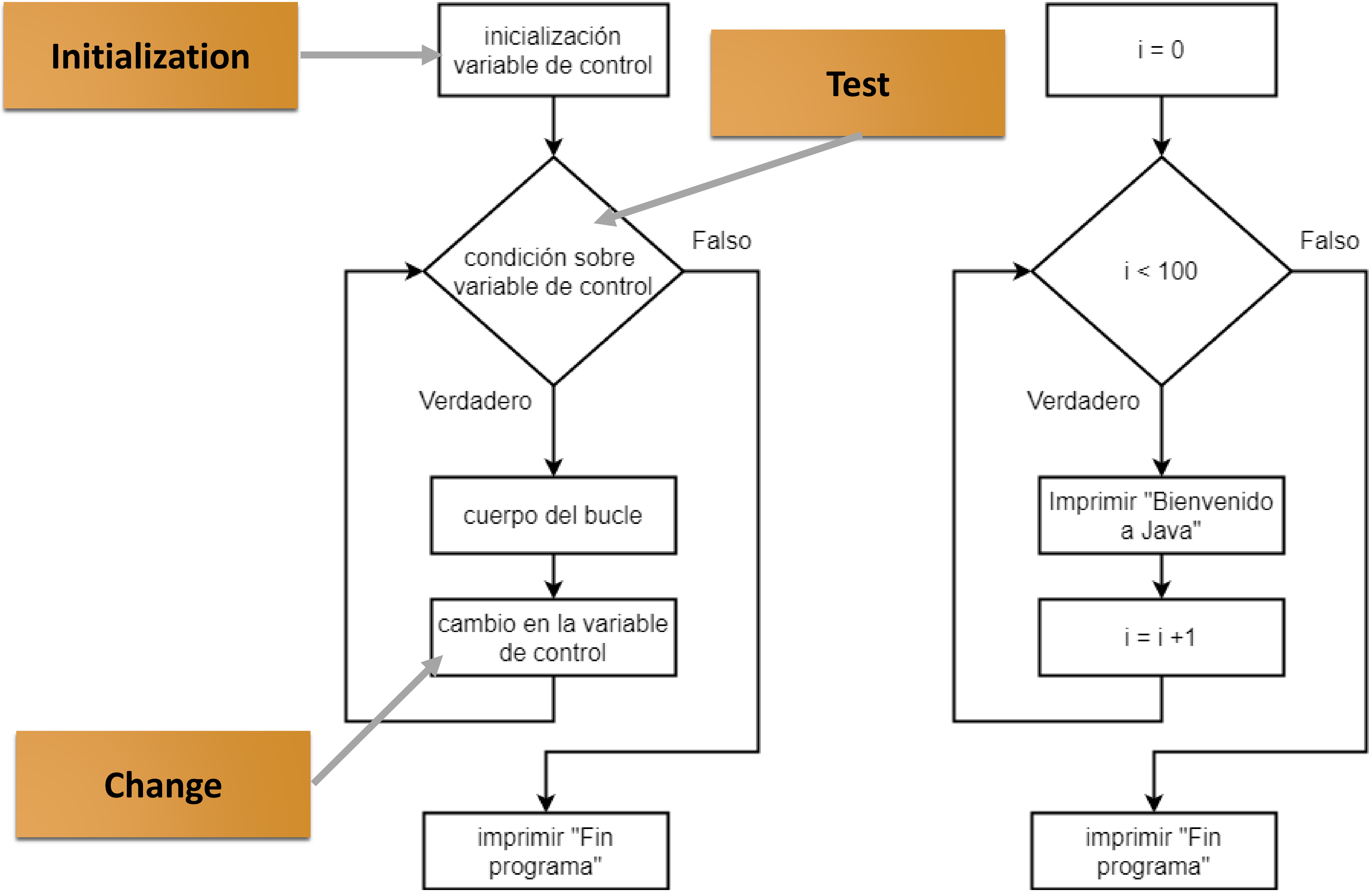
ordered sequence of elements	**matches** “keys” to “values”
look up elements by an integer index	look up one item by another item
indices have an **order**	**no order** is guaranteed
index is an **integer**	key can be any **immutable** type

Basic Elements II – Cycles

- Suppose you are asked to display the message "Welcome to Python" 100 times.
- How do we solve it with what we have seen so far?
- print("Welcome to Python")
- print("Welcome to Python")
- print("Welcome to Python")
- print("Welcome to Python")
- ...

Basic Elements II – Cycles

- Fortunately, Python provides structures called "loops".
- Loops allow you to control how many times an operation is performed, or a sequence of operations.
- Using a loop, the computer can be told to display a text a hundred times without having to encode "the print statement" a hundred times. Let's see an example:
- for i in range(100):
-         print("Welcome to Python")



Initialization

Test

Change

inicialización variable de control
condición sobre variable de control
Falso
Verdadero
cuerpo del bucle
cambio en la variable de control
imprimir "Fin programa"
i = 0
i < 100
Falso
Verdadero
Imprimir "Bienvenido a Java"
i = i +1
imprimir "Fin programa"

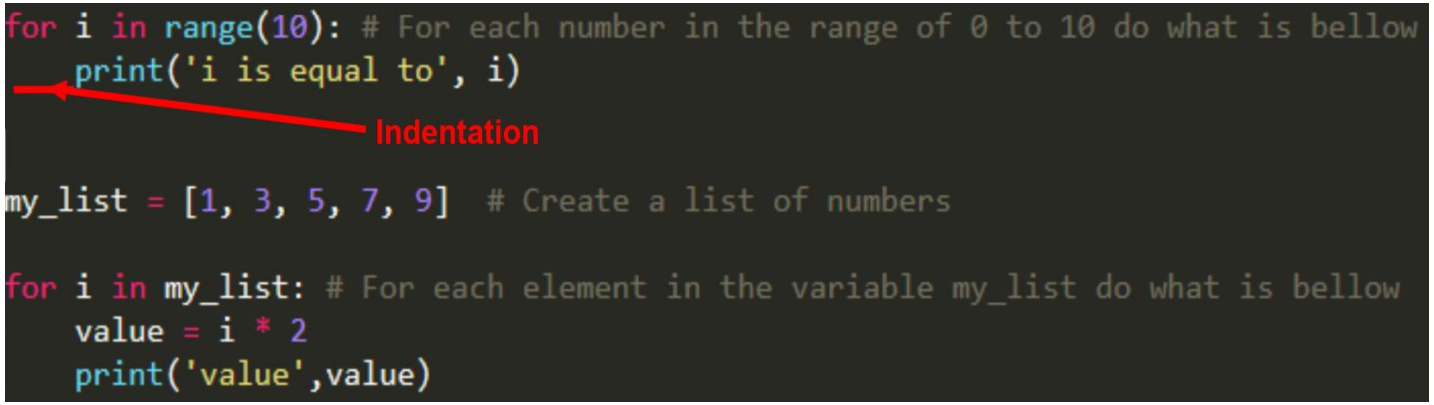
Basic Elements II – Cycles

- In Python we will have two types of cycles: For and While. The syntax for each is as follows:

```
for <variable> in range(<some_num>):
        <expression>
        <expression>

while<condition>:
        <expression>
        <expression>
```

Elementos básicos II – Ciclo for

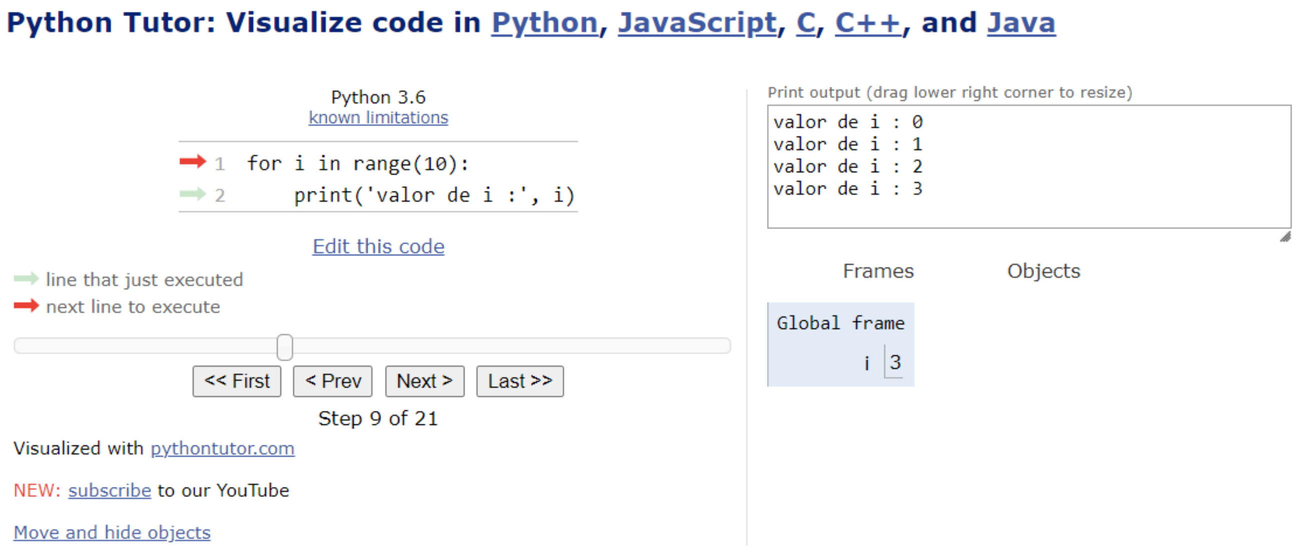
For the range(start,stop,step):
start = 0 and step = 1 (optional) by default and stop -1 to end the cycle.



```
for i in range(10): # For each number in the range of 0 to 10 do what is bellow
    print('i is equal to', i)

my_list = [1, 3, 5, 7, 9]  # Create a list of numbers

for i in my_list: # For each element in the variable my_list do what is bellow
    value = i * 2
    print('value',value)
```

Elementos básicos II – Ciclo for

Visualize each step:
https://pythontutor.com/python-debugger.html#mode=edit



```
for i in range(10):
    print('valor de i :', i)
```

```
valor de i : 0
valor de i : 1
valor de i : 2
valor de i : 3
```

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java
Python 3.6
known limitations
Edit this code
line that just executed
next line to execute
<< First
< Prev
Next >
Last >>
Step 9 of 21
Visualized with pythontutor.com
NEW: subscribe to our YouTube
Move and hide objects
Print output (drag lower right corner to resize)
Frames
Objects
Global frame
i
3

Basic elements II – While cycle

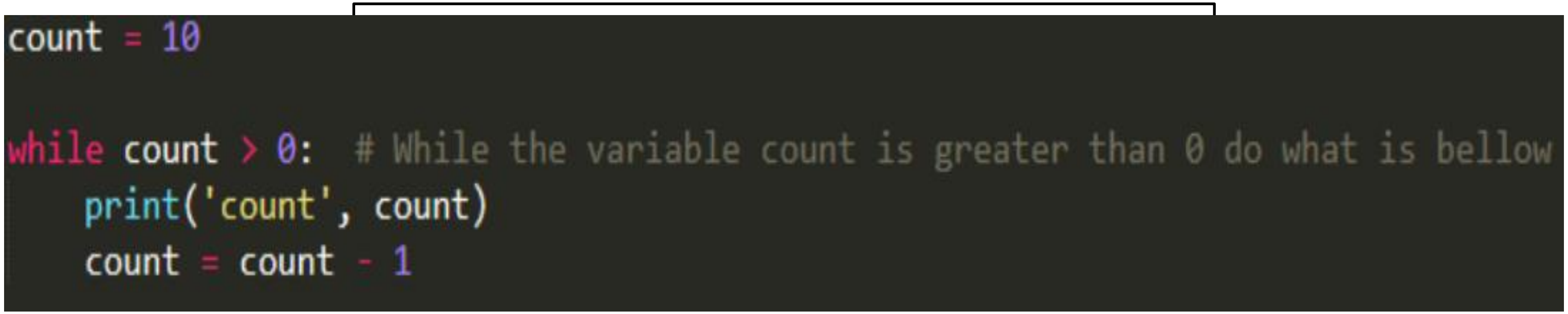
A while loop executes instructions repeatedly while the condition is true.

```
initialization-variable-control
while (condition-variable-control) {
  body of the while
  change-variable-control
}
```

It evaluates <condition> (Boolean), if it <condition> is True, executes all the steps inside the while block. Evaluate <condition> again and repeat until <condition> False

Basic elements II – While cycle

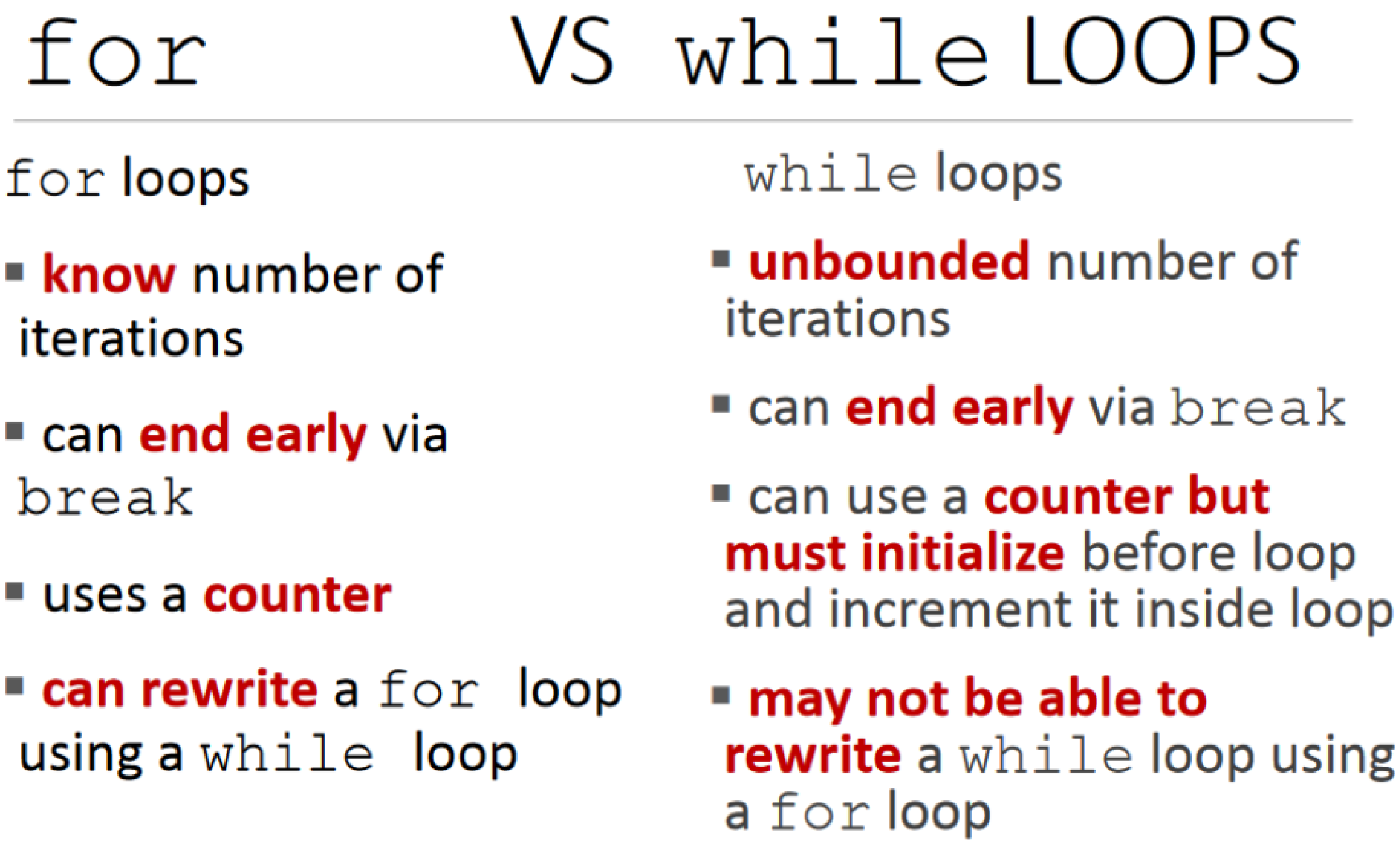
A while loop executes instructions repeatedly while the condition is true.



```python
count = 10

while count > 0:  # While the variable count is greater than 0 do what is bellow
    print('count', count)
    count = count - 1
```

It evaluates <condition> (Boolean), if it <condition> is True, executes all the steps inside the while block. Evaluate <condition> again and repeat until <condition> False

Basic Elements II – Cycles



`for` VS `while` LOOPS

`for` loops

- **know** number of iterations
- can **end early** via `break`
- uses a **counter**
- **can rewrite** a `for` loop using a `while` loop

`while` loops

- **unbounded** number of iterations
- can **end early** via `break`
- can use a **counter but must initialize** before loop and increment it inside loop
- **may not be able to rewrite** a `while` loop using a `for` loop

Exercises

Using While and For cycles

print odd numbers from 1 to 19

Print even numbers 20 through 2.

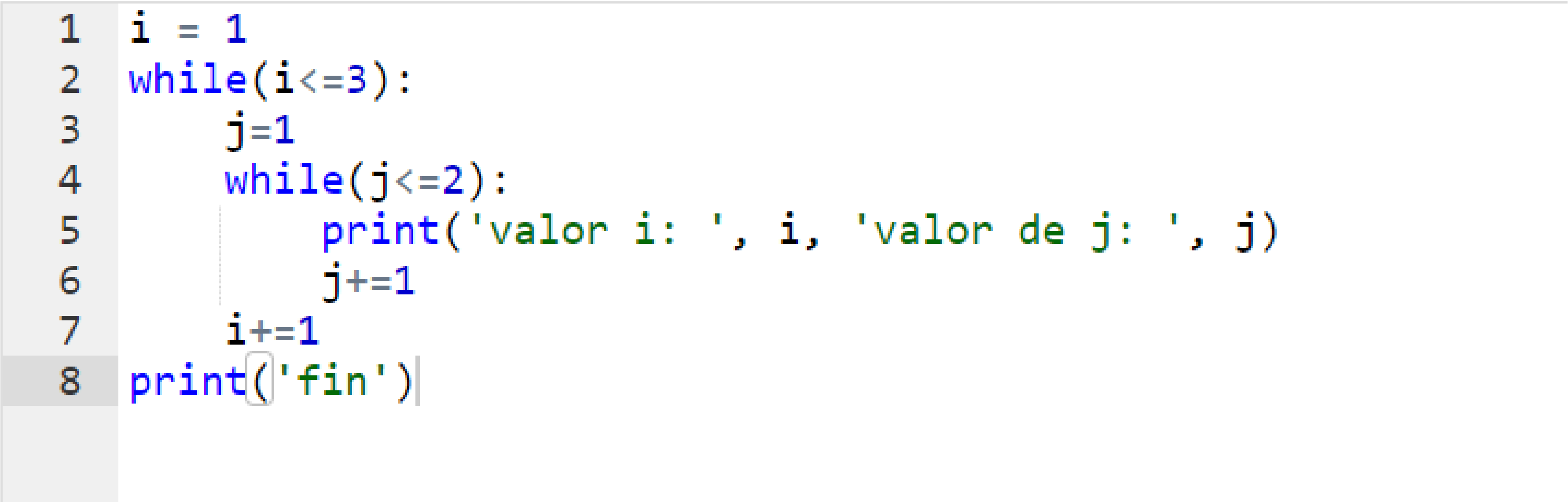
Exercises*

- Develop code that prompts the user for a word.
- If the word is "sum", ask the user for two numbers and then add them up (print the result on the screen).
- If the word is "subtract", ask the user for two numbers and then subtract them (print the result on the screen).
- If the word is "square", ask the user for a number and then find the square (print the result on the screen).
- If the word is "exit", finish the execution.

Nested cycles

- Like conditionals, cycles can also be nested.
- Even a cycle can be nested inside a conditional, and vice versa.
- When nesting a cycle within another cycle, certain aspects must be considered.
- Let's look at the following example.

Nested cycles



```
i = 1
while(i<=3):
    j=1
    while(j<=2):
        print('valor i: ', i, 'valor de j: ', j)
        j+=1
    i+=1
print('fin')
```

Nested cycles

- Perform the above code but with for cycles

Nested cycles

- *Each iteration must have its own control variable.*
- *if you use the same control variable for nested cycles you will most likely end up with infinite loops*

Functions

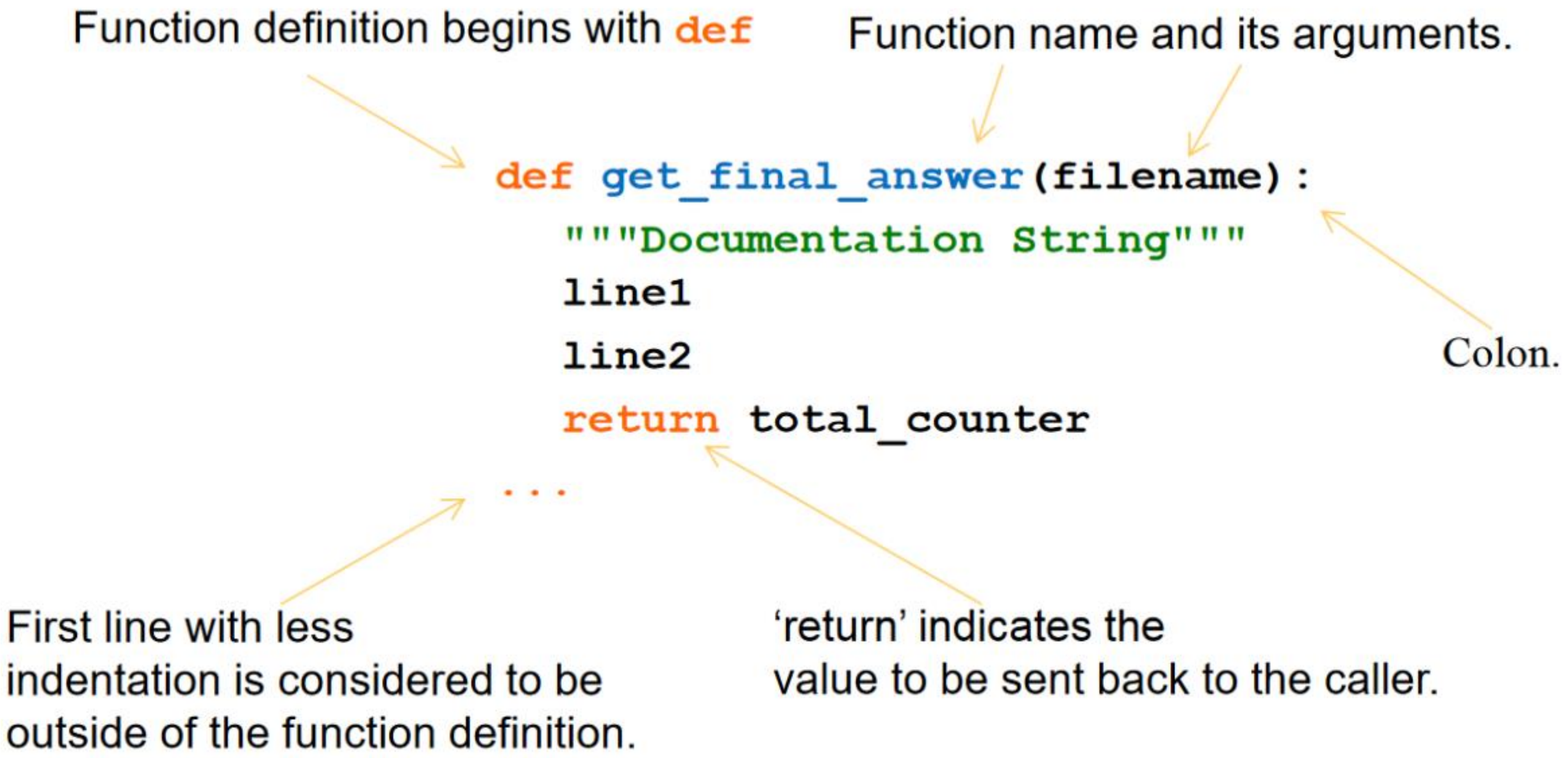
Typing the same instructions multiple times can be tedious. One solution is to join the instructions into a function that behaves like functions in mathematics: giving input arguments yields an answer or output.

A subroutine (also called a procedure, function, routine, method, or subprogram) is a piece of code within a larger program that performs a specific task and is relatively independent of the remaining code.

Functions - Features

- It has a unique name
- It has parameters (from 0 to n)
- Has documentation (optional but recommended)
- Has a body where instructions recur
- Can return something (or be an empty function)

Functions - Definition



Function definition begins with def

Function name and its arguments.

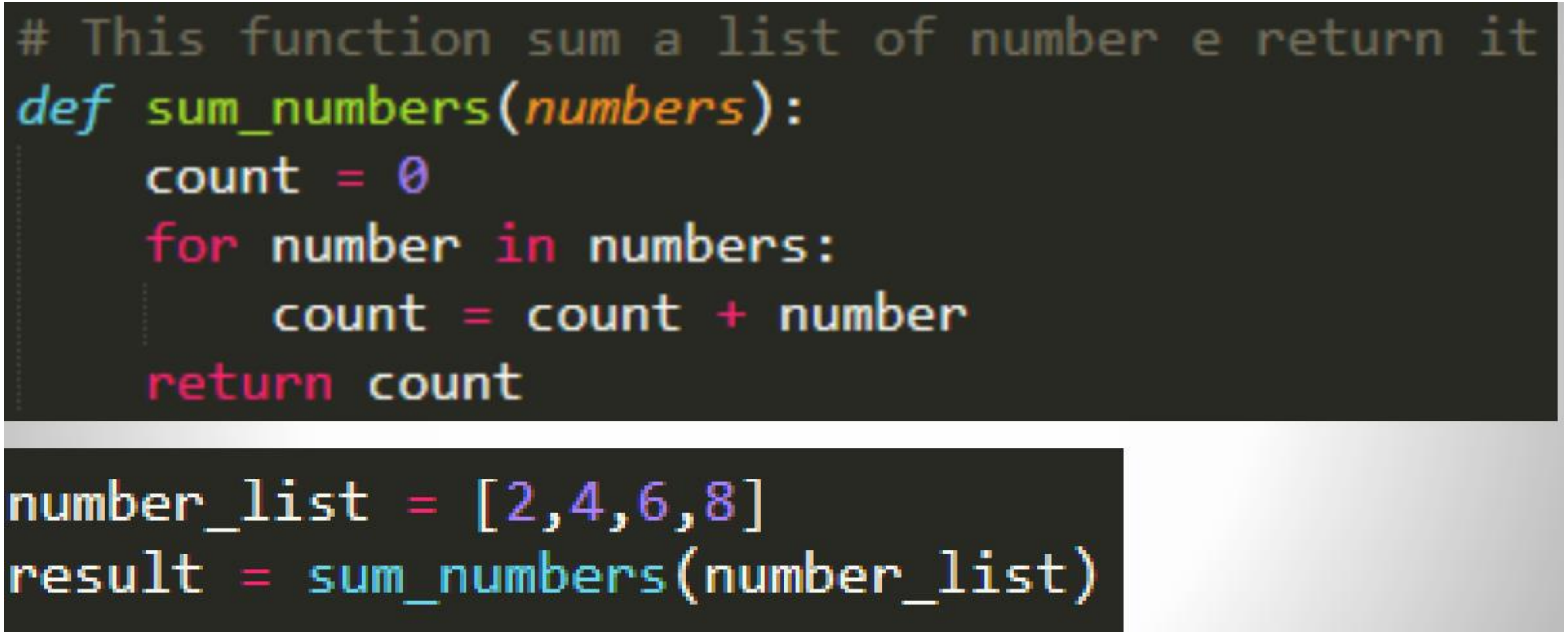
```
def get_final_answer(filename):
   """Documentation String"""
   line1
   line2
   return total_counter
...
```

Colon.

First line with less indentation is considered to be outside of the function definition.

'return' indicates the value to be sent back to the caller.

Functions – Definition – Example



```
# This function sum a list of number e return it
def sum_numbers(numbers):
    count = 0
    for number in numbers:
        count = count + number
    return count
```

```
number_list = [2,4,6,8]
result = sum_numbers(number_list)
```

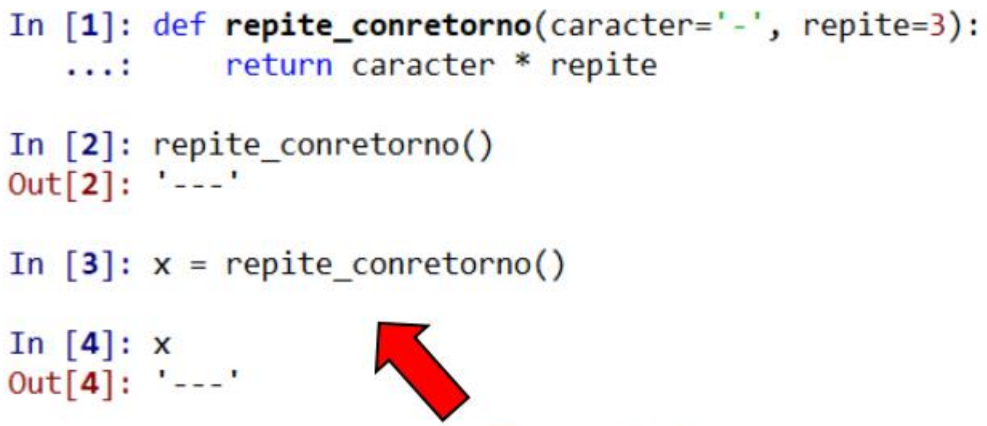
Functions – Exercise 1

- It implements a function that, given a temperature f in degrees Fahrenheit, returns the temperature in degrees Celsius c, that is, $c = 5(f - 32)/9$.

Functions – Exercise 2

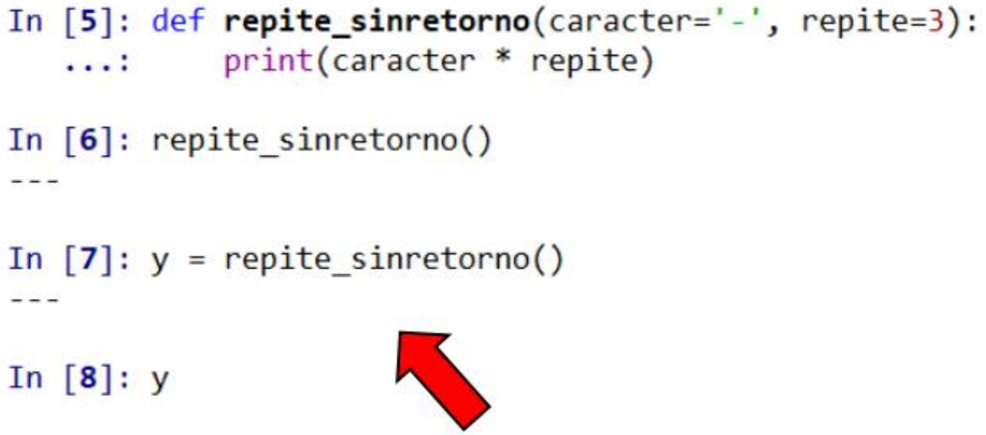
- It implements a function called area_rectangulo(base, height) that returns the area of the rectangle from a base and a height. Calculates the area of a rectangle 15 base and 10 high.

Types of Functions - Return



```
In [1]: def repite_conretorno(caracter='-', repite=3):
   ...:     return caracter * repite

In [2]: repite_conretorno()
Out[2]: '---'

In [3]: x = repite_conretorno()

In [4]: x
Out[4]: '---'
```

Cuando tenemos un return, podemos asignar el valor resultante a un nuevo objeto.

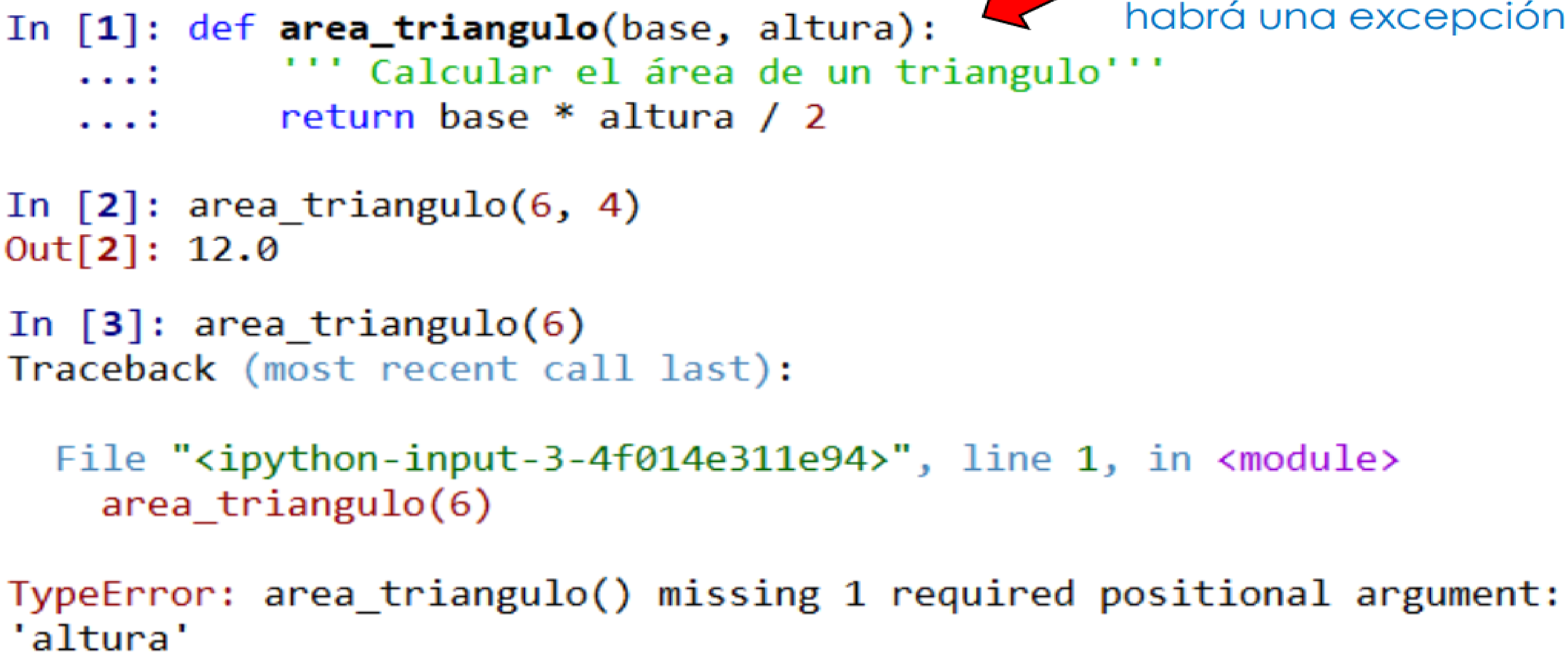


```
In [5]: def repite_sinretorno(caracter='-', repite=3):
   ...:     print(caracter * repite)

In [6]: repite_sinretorno()
---

In [7]: y = repite_sinretorno()
---

In [8]: y
```

Si no tenemos un return, el objeto al que asignamos el valor resultante no guarda ningún valor.

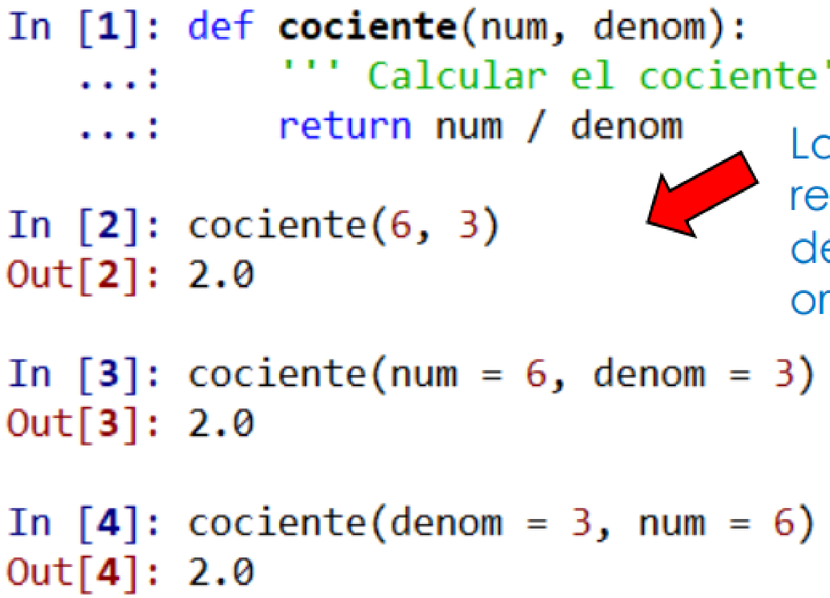
Types of Functions - Arguments

Los dos parámetros son obligatorios. Si alguno falta habrá una excepción.



```
In [1]: def area_triangulo(base, altura):
   ...:     ''' Calcular el área de un triangulo'''
   ...:     return base * altura / 2

In [2]: area_triangulo(6, 4)
Out[2]: 12.0

In [3]: area_triangulo(6)
Traceback (most recent call last):

  File "<ipython-input-3-4f014e311e94>", line 1, in <module>
    area_triangulo(6)

TypeError: area_triangulo() missing 1 required positional argument:
'altura'
```

Types of Functions - Arguments

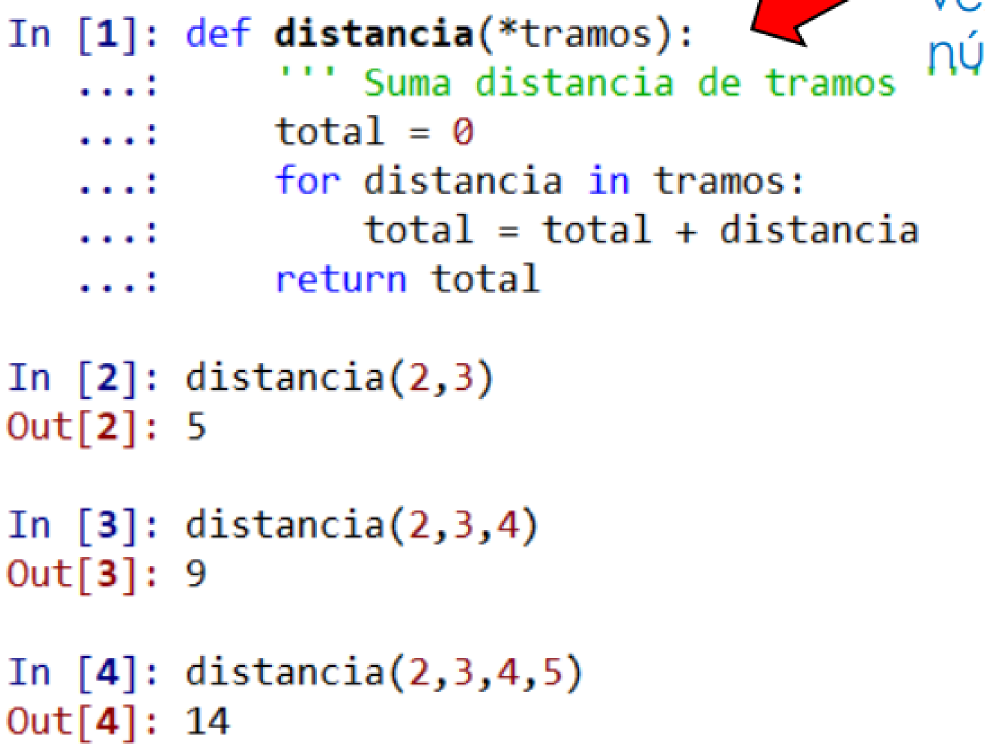


```
In [1]: def cociente(num, denom):
   ...:     ''' Calcular el cociente'''
   ...:     return num / denom

In [2]: cociente(6, 3)
Out[2]: 2.0

In [3]: cociente(num = 6, denom = 3)
Out[3]: 2.0

In [4]: cociente(denom = 3, num = 6)
Out[4]: 2.0
```

La llamada a la función se realiza explicitando el valor de las variables en el mismo orden de la definición.

Pero podemos modificar el orden de las variables en la llamada a la función si especificamos sus nombres.

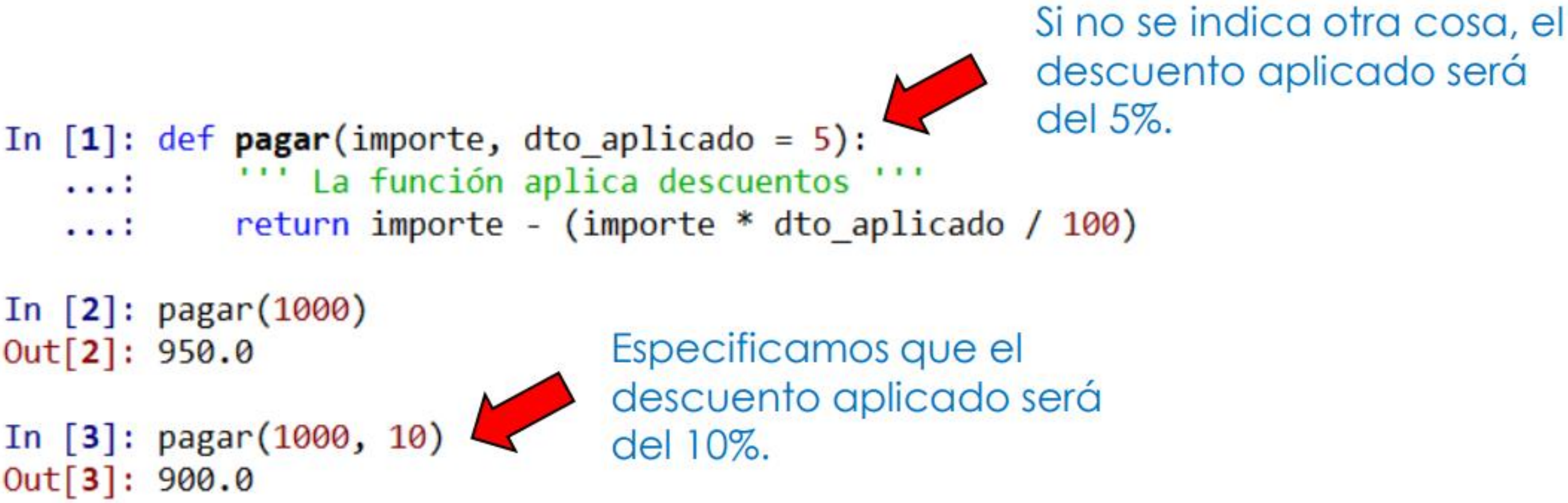
Types of Functions - Arguments



```
In [1]: def distancia(*tramos):
   ...:     ''' Suma distancia de tramos '''
   ...:     total = 0
   ...:     for distancia in tramos:
   ...:         total = total + distancia
   ...:     return total

In [2]: distancia(2,3)
Out[2]: 5

In [3]: distancia(2,3,4)
Out[3]: 9

In [4]: distancia(2,3,4,5)
Out[4]: 14
```

El asterisco indica que cada vez podemos especificar un número distinto de valores.

Types of Functions - Arguments

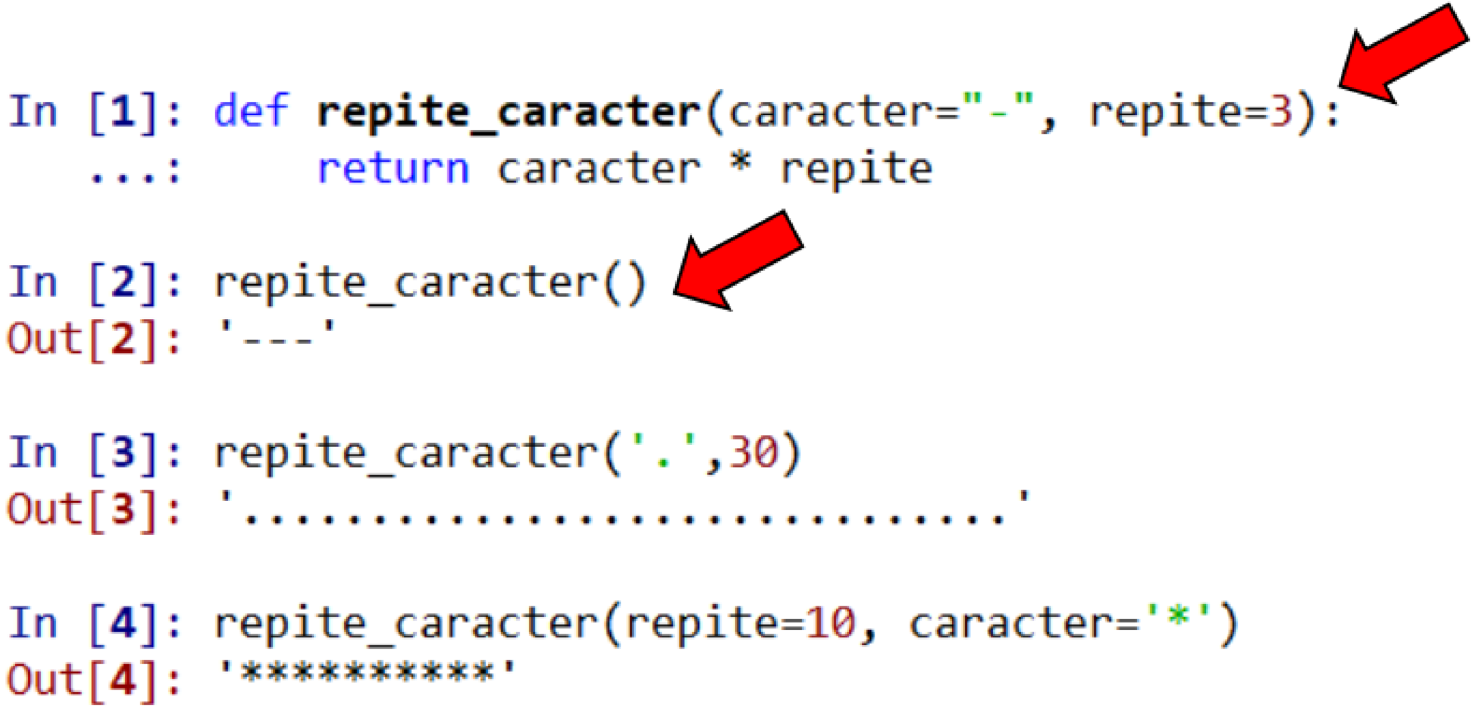


```
In [1]: def pagar(importe, dto_aplicado = 5):
   ...:     ''' La función aplica descuentos '''
   ...:     return importe - (importe * dto_aplicado / 100)

In [2]: pagar(1000)
Out[2]: 950.0

In [3]: pagar(1000, 10)
Out[3]: 900.0
```

Si no se indica otra cosa, el descuento aplicado será del 5%.

Especificamos que el descuento aplicado será del 10%.

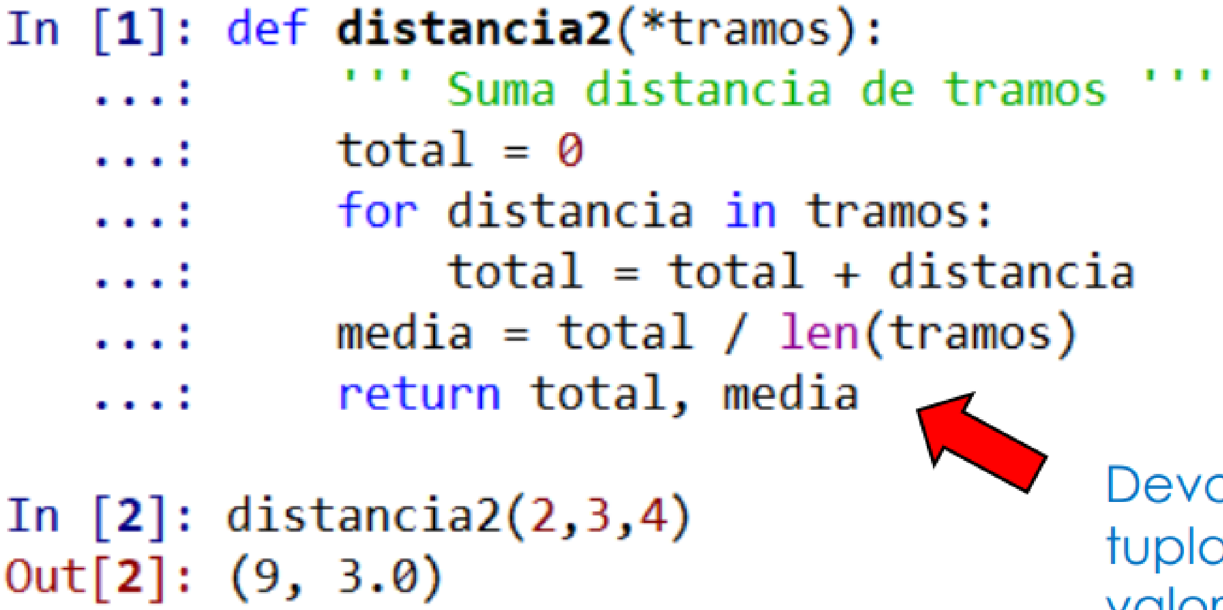
Types of Functions - Arguments



```
In [1]: def repite_caracter(caracter="-", repite=3):
   ...:     return caracter * repite

In [2]: repite_caracter()
Out[2]: '---'

In [3]: repite_caracter('.',30)
Out[3]: '..............................'

In [4]: repite_caracter(repite=10, caracter='*')
Out[4]: '**********'
```

En este caso todos los argumentos tienen valores por defecto.

Types of Functions - Return



```
In [1]: def distancia2(*tramos):
   ...:     ''' Suma distancia de tramos '''
   ...:     total = 0
   ...:     for distancia in tramos:
   ...:         total = total + distancia
   ...:     media = total / len(tramos)
   ...:     return total, media

In [2]: distancia2(2,3,4)
Out[2]: (9, 3.0)
```

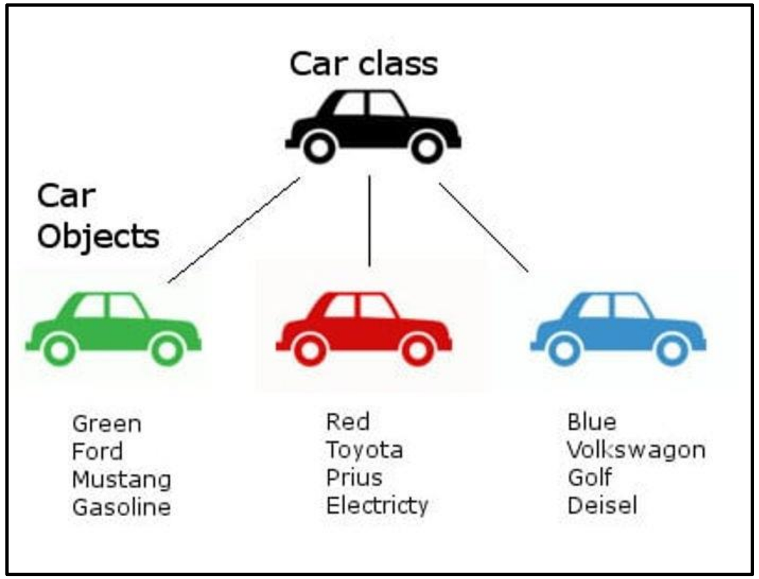
Devolvemos una tupla con dos valores: la suma de los tramos y la distancia media de los tramos.

Introduction to object-oriented programming

Object-oriented programming languages (OOP) are based on a grouping of objects of different classes that interact with each other and, together, get a program to fulfill its purpose. In this programming paradigm we try to emulate the functioning of the objects that surround us in real life. OOP is essentially a way to develop reusable software.

OOP – Features

- OOP organizes programs in a way that represents the interaction of things in the real world.
- In OOP, a program consists of a set of objects.
- In OOP, objects are abstractions of things in the real world.
- In OOP, each object is responsible for some tasks.
- In OOP, each object is an instance of a class.
- In OOP, classes can be organized into an inheritance hierarchy (see below).
- OO programming is a simulation of a model of the universe.



OOP – Features

- Abstraction: An element can be isolated from the rest of the elements and its context to focus interest on what it does and not on how it does it (black box).
- Modularity: A tool can be divided into smaller, independent, reusable parts called modules.
- Encapsulation: It consists of gathering all the possible elements of an entity at the same level of abstraction to increase cohesion, with the possibility of hiding the attributes of an object (in Python, they are only hidden in appearance).
- Inheritance: A class inherits the characteristics of a higher class to obtain similar objects. Both attributes and methods are inherited. The latter can be overwritten to suit the needs of the new class.
- Polymorphism: Similar behaviors, although associated with different objects, always follow the same patterns.

OOP – Objects

- OOP involves programming using objects. An object represents a real-world entity that can be clearly identified.
- For example: a student, a desk, a circle, a house, a car, any specific object.
- Each object has a unique identity, properties (attributes), and behaviors (methods).

OOP – Objects - Properties

Properties (attributes):

Represent data with values belonging to the object.

For example: a student object could have: name, age, gender, email, among others**.**

OOP – Objects - Behaviors

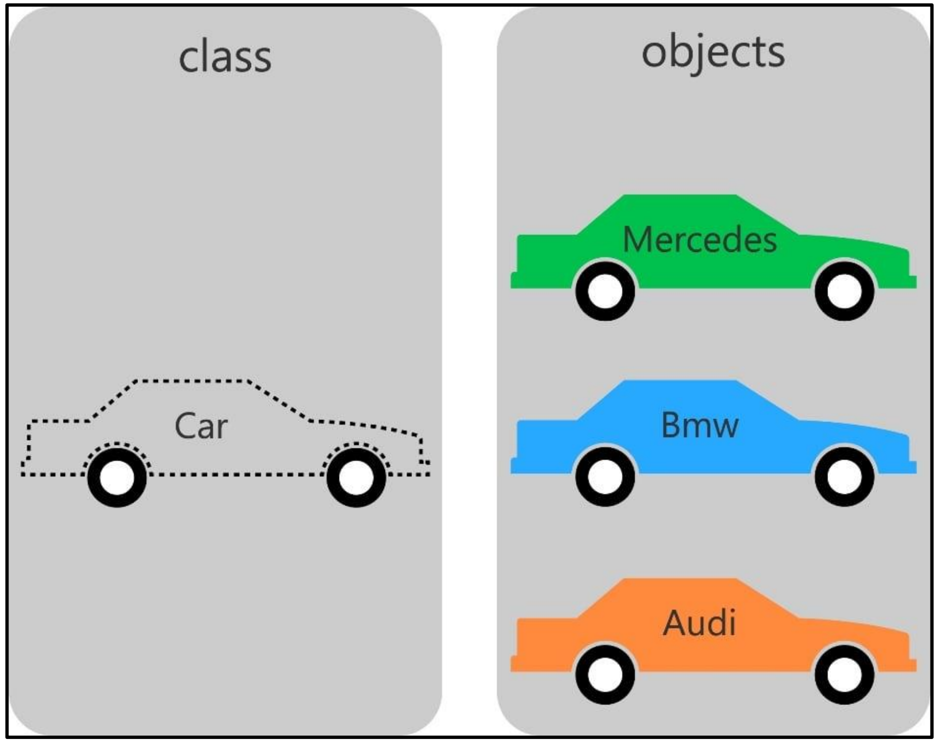
Behaviors (methods):

They represent actions or methods. Each object can invoke or execute different types of actions.

For example: a circle object could invoke methods such as getArea() – to get the area of the circle. Or getPerimeter() to get the perimeter.

OOP – Classes

- Objects of the same type are defined using a common class.
- A class is a template, model, or contract that defines what the attributes and methods of an object will be.
- An object is an instance of a class.
- A developer can create many instances of a class.
- Creating an instance is known as instantiation.

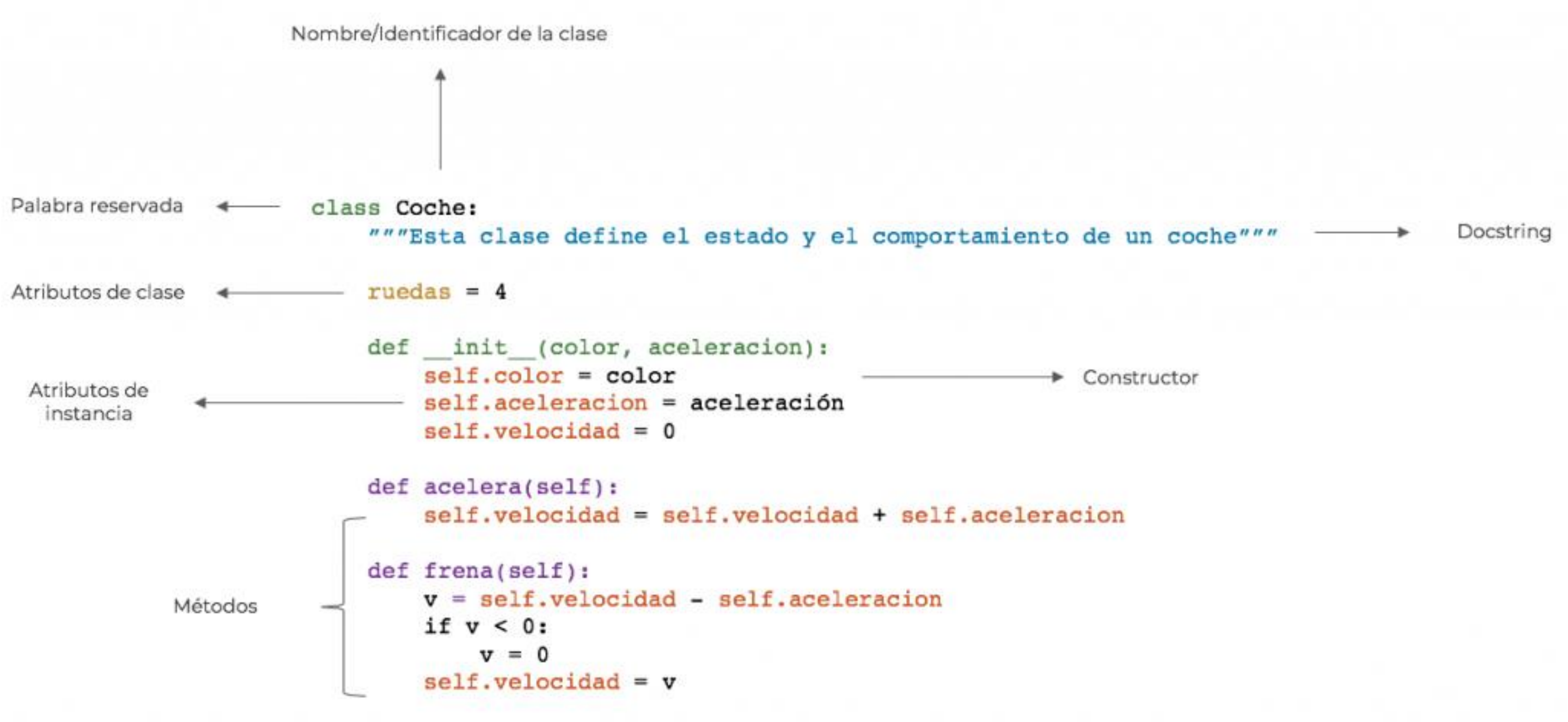


OOP – Classes - Definition

As function definitions begin with the def keyword, in Python, define a class using the *class* keyword. The first string is called docstring and has a brief description about the class. Although not mandatory, it is recommended.

As soon as we define a class, a new class object with the same name is created. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

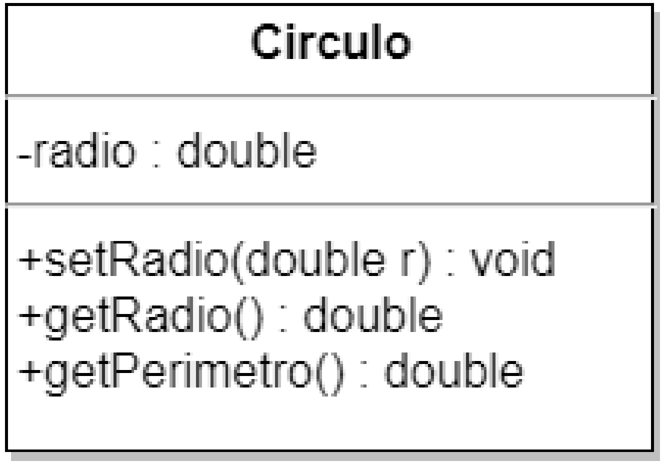
OOP – Classes - Definition



```
class Coche:
    """Esta clase define el estado y el comportamiento de un coche"""

    ruedas = 4

    def __init__(color, aceleracion):
        self.color = color
        self.aceleracion = aceleración
        self.velocidad = 0

    def acelera(self):
        self.velocidad = self.velocidad + self.aceleracion

    def frena(self):
        v = self.velocidad - self.aceleracion
        if v < 0:
            v = 0
        self.velocidad = v
```

OOP – Classes – Class Diagrams (UML)

- Used to represent system classes.
- It allows you to get an overview of my application without having to go looking at 200 files or more.
- It allows you to graphically observe how the different classes of my system are related (see below).
- It allows you to establish a design of my application and present it to other colleagues or stakeholders.



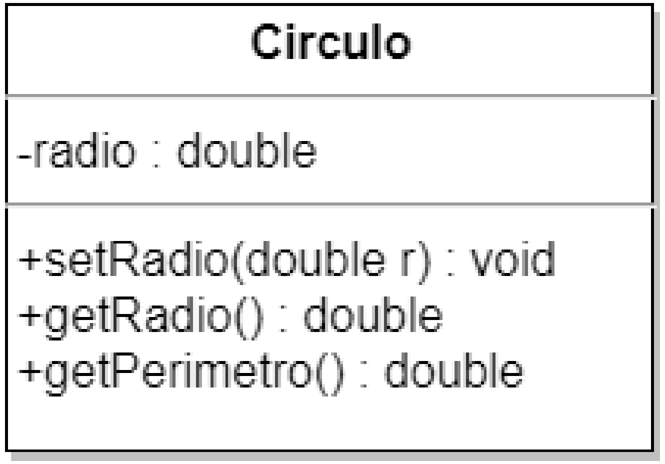
Circulo

-radio : double

+setRadio(double r) : void
+getRadio() : double
+getPerimetro() : double

OOP – Classes – Class Diagrams (UML)

- In its simplest form, a class in UML is drawn as a rectangle divided into up to three sections.
- The top section contains the name of the class (centered and singular).
- The middle section contains the attributes or properties of the class: visibility attributename : type
- The final section contains methods that represent the behavior exhibited by the class: visibility nameMethod(parameters) : typeReturn
- Note: The attributes and operations sections are optional.

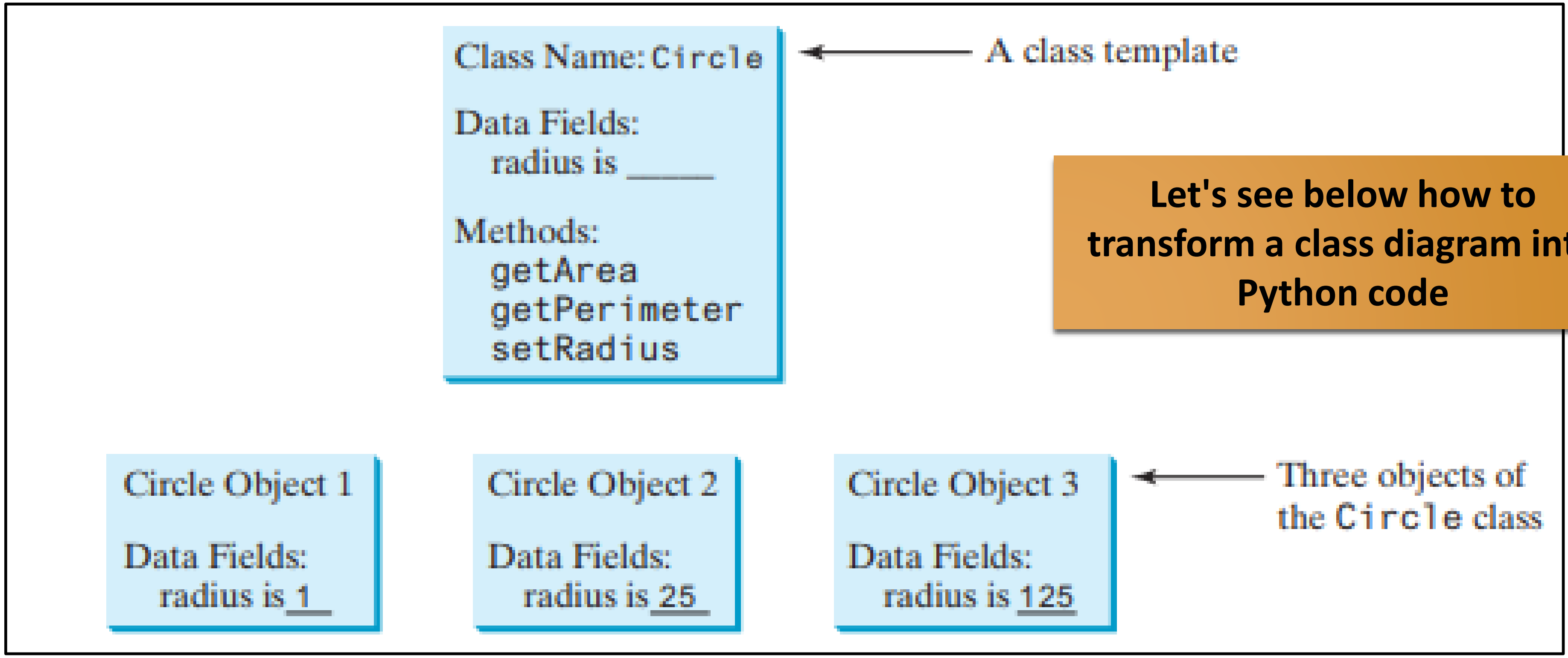


Circulo

-radio : double

+setRadio(double r) : void
+getRadio() : double
+getPerimetro() : double

OOP – Classes – Class Diagrams (UML)



Class Name: Circle

Data Fields:
radius is _____

Methods:
getArea
getPerimeter
setRadius

A class template

Circle Object 1

Data Fields:
radius is 1

Circle Object 2

Data Fields:
radius is 25

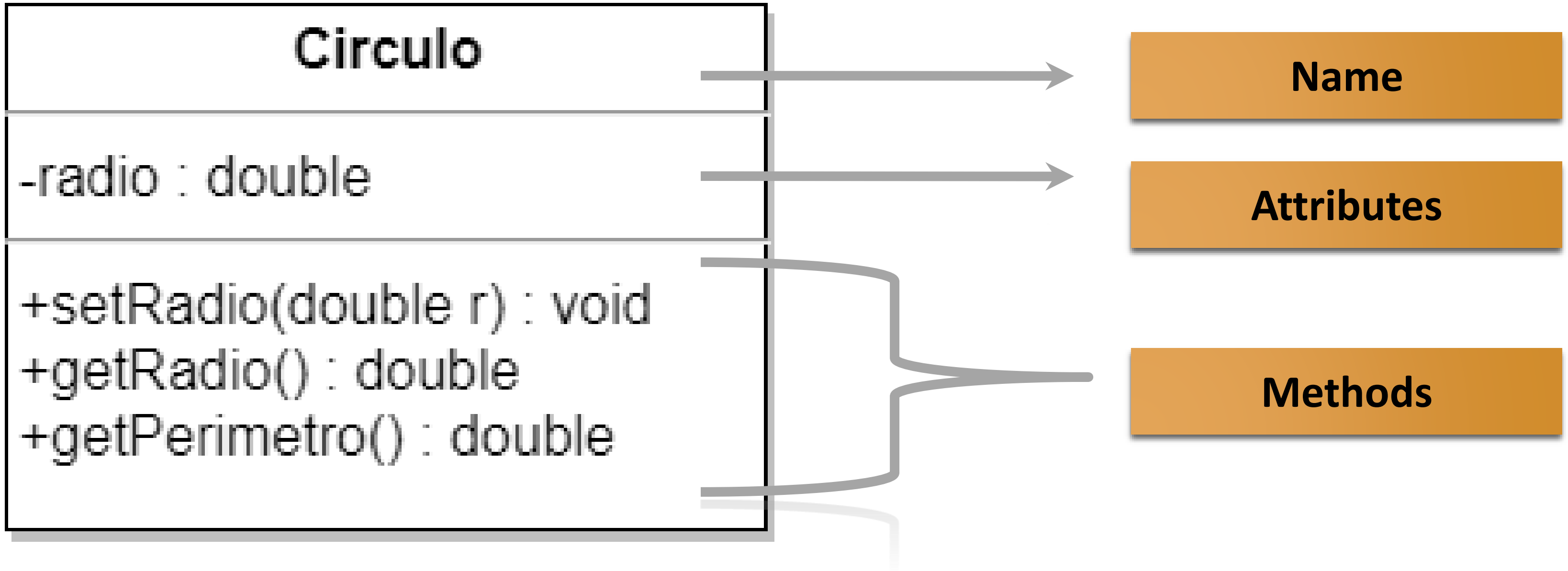
Circle Object 3

Data Fields:
radius is 125

Three objects of the Circle class

Let's see below how to transform a class diagram into Python code

OOP – Classes – Class Diagrams (UML)



Circulo

-radio : double

+setRadio(double r) : void
+getRadio() : double
+getPerimetro() : double

Name

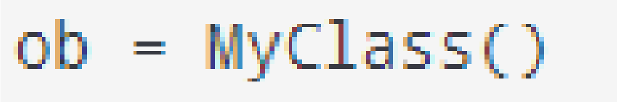
Attributes

Methods

OOP – Objects - Creation

An object (instance) is an "example" of a class. When the class is defined, only the description of the object is defined. Therefore, no memory or storage is allocated.

We saw that the class object could be used to access different attributes. It can also be used to create new instances of objects (instantiation) of that class. The procedure for creating an object is similar to a function call.



```
ob = MyClass()
```

Exercise 1

Create 3 circles using the previous class, assign them a radius and display the perimeter

OOP - Builders

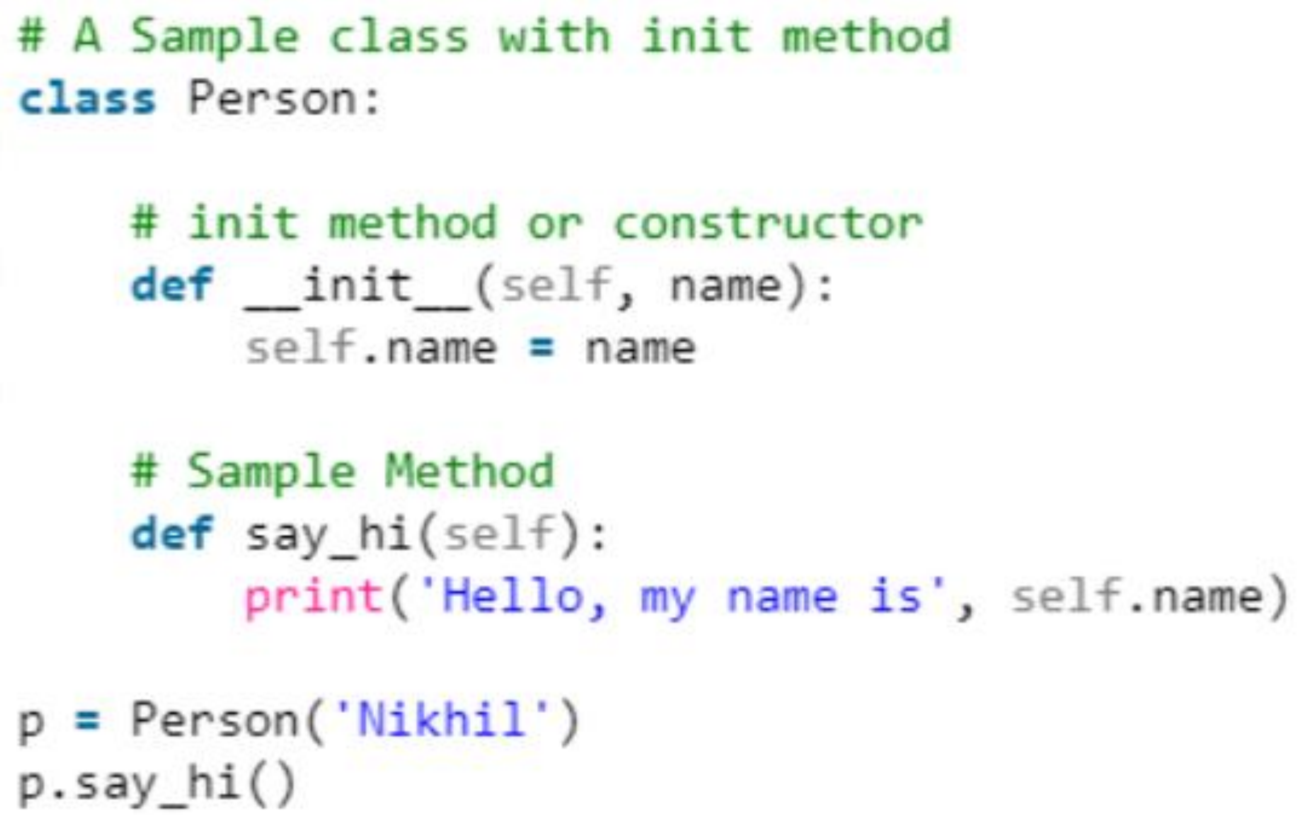
The __init__ method or constructors are used to initialize the state of the object. Like methods, a constructor also contains a collection of declarations (i.e., statements) that are executed at the time of Object creation (an object of a class is instantiated.) The method is useful for doing any initialization. You want to do with your object.

OOP – __init__ Builders

The __init__ method or constructors are used to initialize the state of the object. Like methods, a constructor also contains a collection of declarations (i.e., statements) that are executed at the time of Object creation (an object of a class is instantiated.) The method is useful for doing any initialization. You want to do with your object.

OOP – Method __init__

The __init__ m he state of the
object. Like me on of
declarations (i. e of Object
creation (an ob s useful for
doing any initia



```
# A Sample class with init method
class Person:

    # init method or constructor
    def __init__(self, name):
        self.name = name

    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Nikhil')
p.say_hi()
```

OOP – Self Method

You may have noticed the proper parameter in the function definition within the class, but we call the method simply as ob.func() without any arguments.

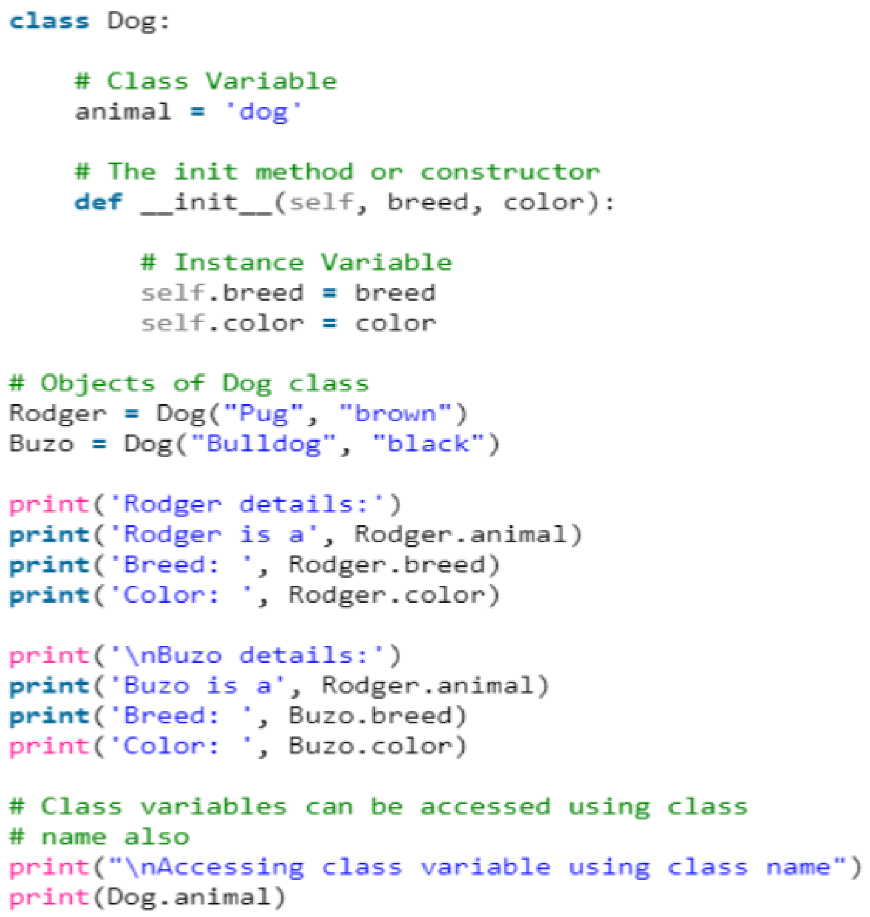
This is because, whenever an object calls its method, the object itself is like the first argument. So, ob.func() translates to MyClass.func(ob)

OOP – Self Method

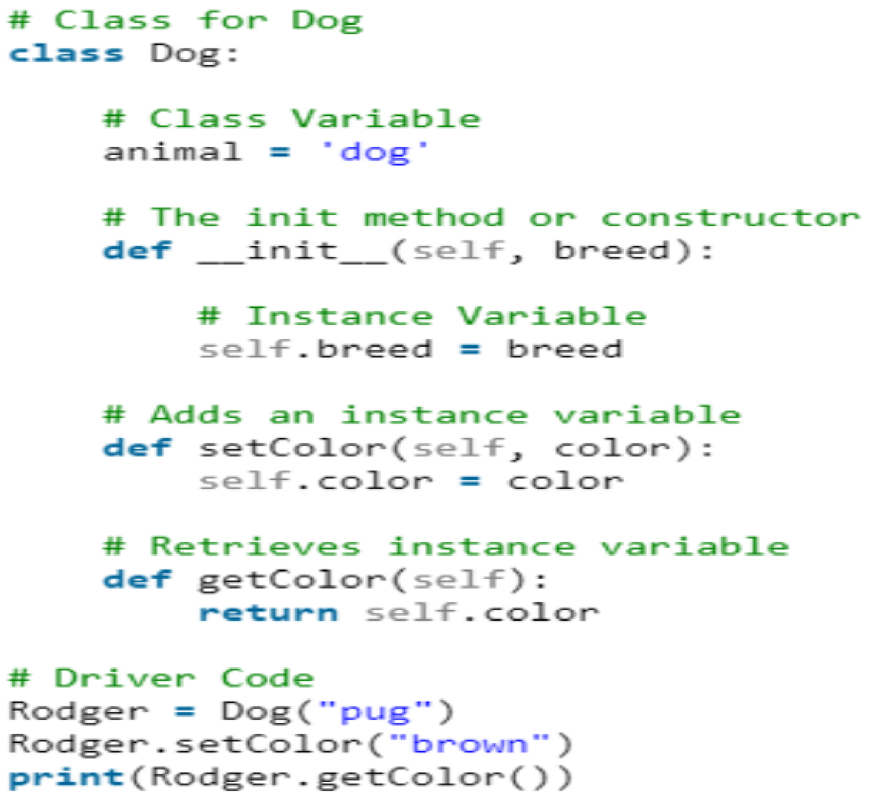
In general, calling a method with a list of n arguments is equivalent to calling the corresponding function with a list of arguments that is created by inserting the method object before the first argument.

For these reasons, the first argument of the function in class must be the object itself. This is conventionally called self.

OOP – Self Method



```python
class Dog:

    # Class Variable
    animal = 'dog'

    # The init method or constructor
    def __init__(self, breed, color):

        # Instance Variable
        self.breed = breed
        self.color = color

# Objects of Dog class
Rodger = Dog("Pug", "brown")
Buzo = Dog("Bulldog", "black")

print('Rodger details:')
print('Rodger is a', Rodger.animal)
print('Breed: ', Rodger.breed)
print('Color: ', Rodger.color)

print('\nBuzo details:')
print('Buzo is a', Rodger.animal)
print('Breed: ', Buzo.breed)
print('Color: ', Buzo.color)

# Class variables can be accessed using class
# name also
print("\nAccessing class variable using class name")
print(Dog.animal)
```



```python
# Class for Dog
class Dog:

    # Class Variable
    animal = 'dog'

    # The init method or constructor
    def __init__(self, breed):

        # Instance Variable
        self.breed = breed

    # Adds an instance variable
    def setColor(self, color):
        self.color = color

    # Retrieves instance variable
    def getColor(self):
        return self.color

# Driver Code
Rodger = Dog("pug")
Rodger.setColor("brown")
print(Rodger.getColor())
```

Exercise 1

Create a class called "Parrot" that uses an init method to
Assign two attributes (name, age). In this class, there are two methods.
to assign behavior to the Parrot object, the functions "sing" and
"dance" prints the name of the Parrot and the actions it performs (if
sing or dance)

OOP - INHERITANCE

Inheritance is a mechanism in object-oriented programming for creating new classes from pre-existing classes. Attributes and behaviors are taken (inherited) from old classes and modified to model a new situation.

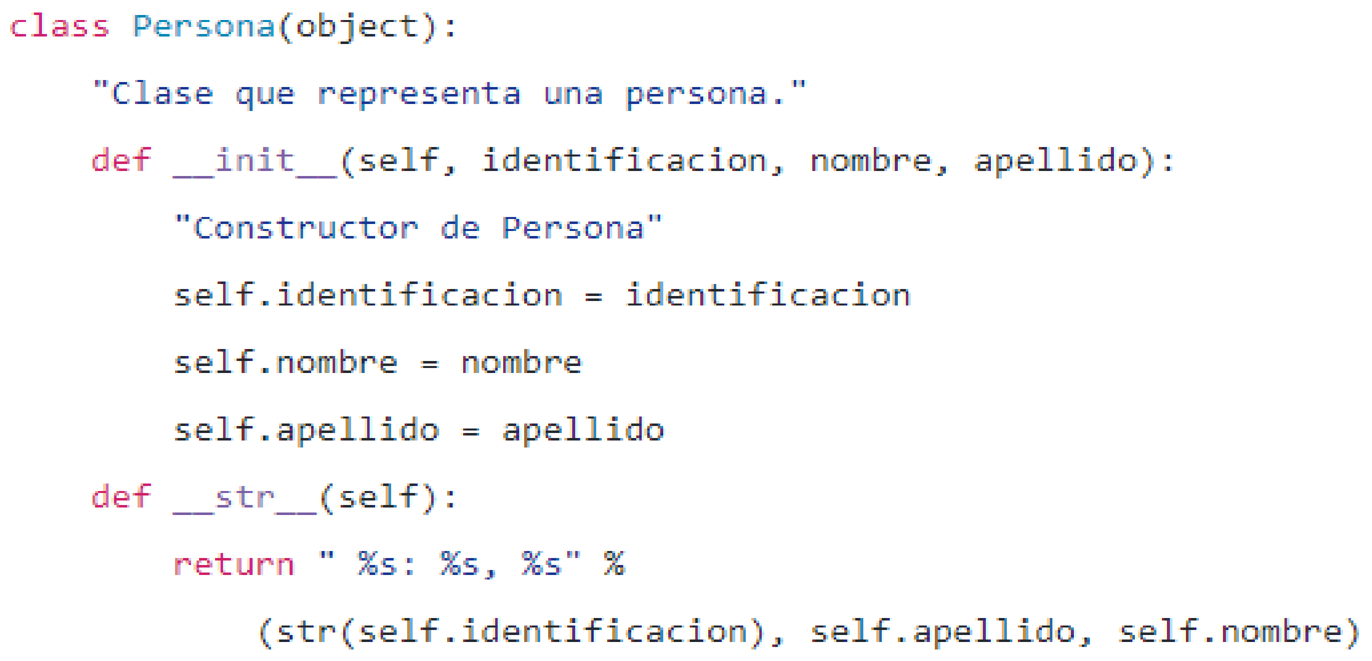
The old class is called the base class and the one built from it is a derived class.

For example, from a Person class (which contains as attributes identification, name, surname) we can build the AlumnoFIUBA class that extends to Person and adds as an attribute the standard.

OOP - INHERITANCE

To indicate the name of the base class, put it in parentheses after the name of the class (instead of the expression object that we put previously; actually object is the name of the generic base class).

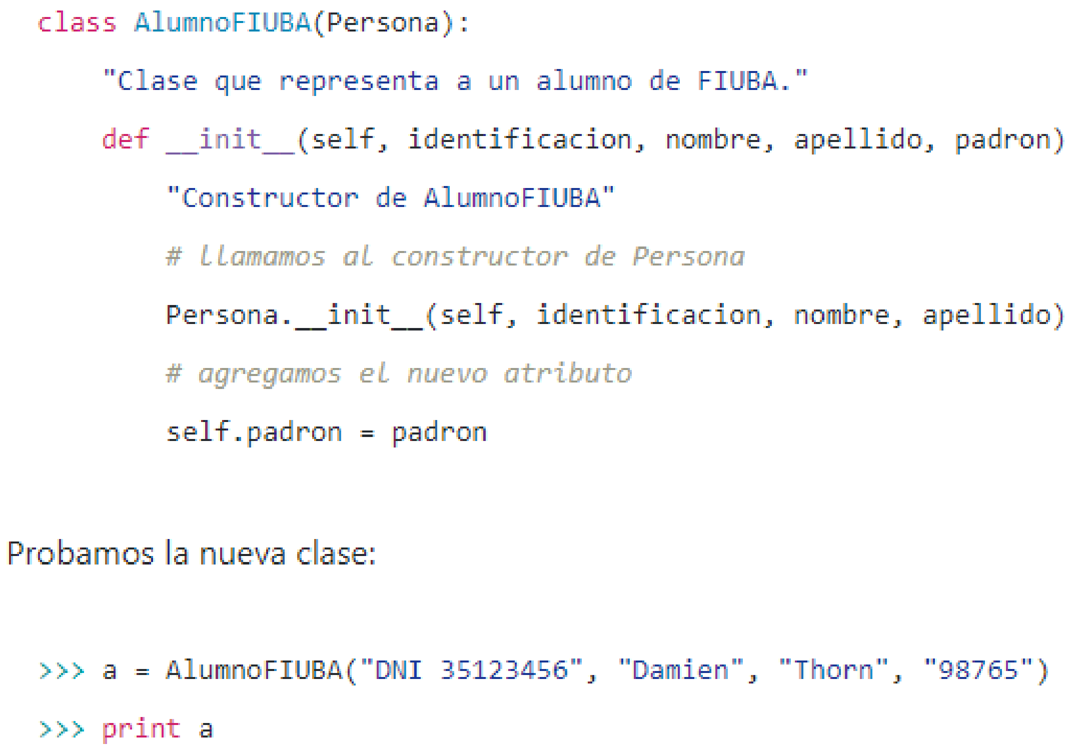
We define person:



```
class Persona(object):
    "Clase que representa una persona."
    def __init__(self, identificacion, nombre, apellido):
        "Constructor de Persona"
        self.identificacion = identificacion
        self.nombre = nombre
        self.apellido = apellido
    def __str__(self):
        return " %s: %s, %s" %
            (str(self.identificacion), self.apellido, self.nombre)
```

OOP - INHERITANCE

Next, we define StudentFIUBA as derived from Person, in such a way that it initializes the new attribute, but in turn uses the initialization of Person for the attributes of the base class:



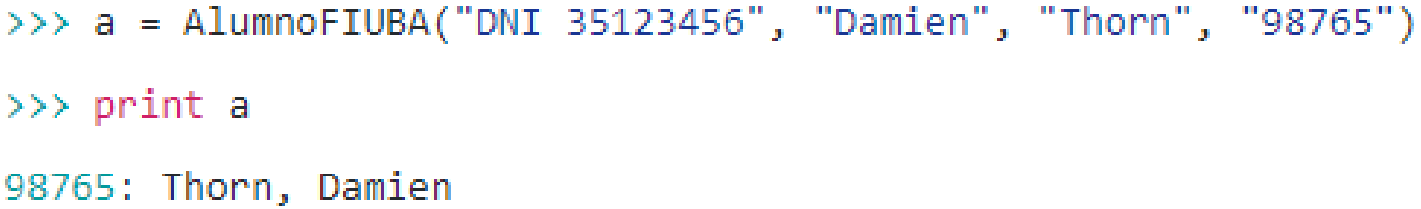
```
class AlumnoFIUBA(Persona):
    "Clase que representa a un alumno de FIUBA."
    def __init__(self, identificacion, nombre, apellido, padron)
        "Constructor de AlumnoFIUBA"
        # llamamos al constructor de Persona
        Persona.__init__(self, identificacion, nombre, apellido)
        # agregamos el nuevo atributo
        self.padron = padron
```

Probamos la nueva clase:

```
>>> a = AlumnoFIUBA("DNI 35123456", "Damien", "Thorn", "98765")
>>> print a
```

OOP - INHERITANCE

- We try again:



```
>>> a = AlumnoFIUBA("DNI 35123456", "Damien", "Thorn", "98765")
>>> print a
98765: Thorn, Damien
```

OOP - INHERITANCE

From a base class many derived classes can be built, just as we have derived students, we could derive teachers, employees, customers, suppliers, or whatever was necessary according to the application we are developing.

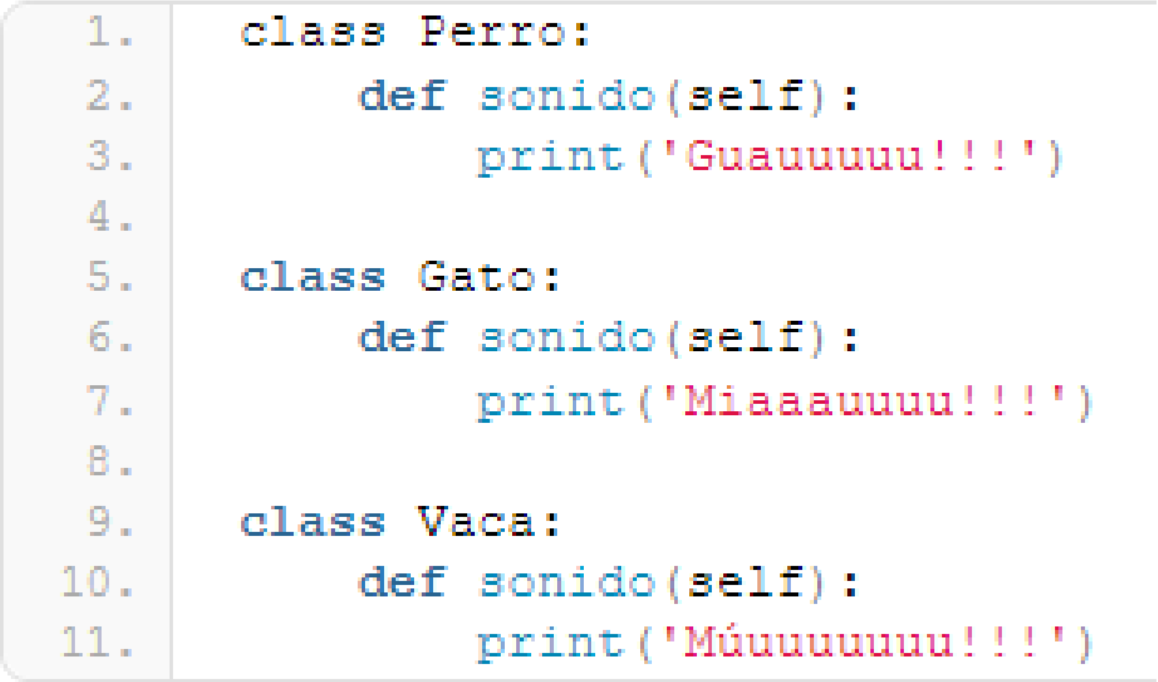
Polymorphism

And we come to the last concept that we will see in this course on object-oriented programming in Python.

Polymorphism is the ability of an entity to reference instances of different classes at run time.

Although this concept sounds strange to you right now, you will understand it with an example. Imagine that we have the following classes that represent animals:

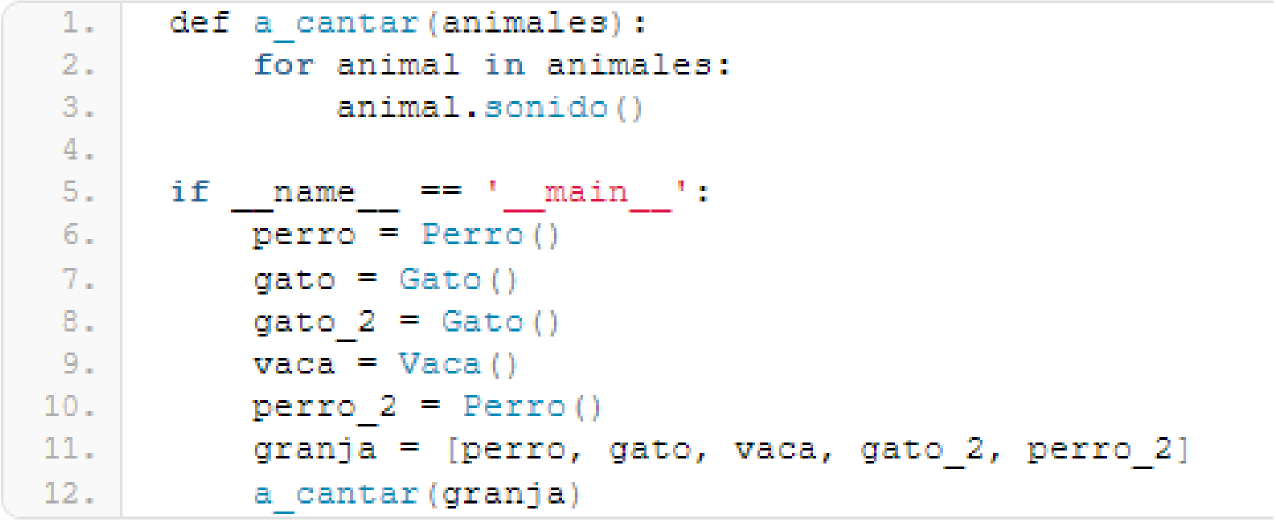
Polymorphism



```python
class Perro:
    def sonido(self):
        print('Guauuuuu!!!')

class Gato:
    def sonido(self):
        print('Miaaauuuu!!!')

class Vaca:
    def sonido(self):
        print('Múuuuuuuu!!!')
```

Polymorphism

- All three classes implement a method called sound(). Now look at the following script:



```
def a_cantar(animales):
    for animal in animales:
        animal.sonido()

if __name__ == '__main__':
    perro = Perro()
    gato = Gato()
    gato_2 = Gato()
    vaca = Vaca()
    perro_2 = Perro()
    granja = [perro, gato, vaca, gato_2, perro_2]
    a_cantar(granja)
```

Polymorphism

A function called a_cantar() has been defined in it. The animal variable that is created within the function's for loop is polymorphic, since at run time it will refer to objects of the Dog, Cat, and Cow classes. When the sound() method is invoked, the corresponding method of the class to which each animal belongs shall be called.