Introducción a la Programación

en Python

Mario González
Facultad de Ingeniería y Ciencias Aplicadas
Universidad de las Américas, Quito, Ecuador

Agosto, 2021

Is Computer Science a Natural Science?

- Computer Science is no more about computers than astronomy is about telescopes.
- The computer is a tool in a lot of regards to when people study computation (which is science) unless they have solely applications in mind (which is not necessarily scientific).
- Computation is what is at the heart of computer science: Computing Science.
- Computer Science is formal science because of its roots in mathematics, i.e. Algorithm design.
- Some branches of CS can relate to natural science: theory of computing around genomic sequences and how natural computing occurs, or how the brain processes certain things.

Why Computer Science? I

- Computation is having a significant **impact on the way other disciplines do their work.** And for almost every field X, there is a new interdisciplinary field **“Computational X”**.
  - The CMS conference is an annual meeting associated with the journal of Computational Management Science published by Springer. The aim of this conference is to provide a forum for theoreticians and practitioners from academia and industry to exchange knowledge, ideas and results in a broad range of topics relevant to the theory and practice of computational methods, models and empirical analysis for decision making in economics, engineering, finance and management.
  - These include computational economics, finance and statistics, energy, scheduling, supply chains, design, analysis and applications of optimization algorithms, deterministic, dynamic, stochastic, robust and combinatorial optimization models, solution algorithms, learning and forecasting such as neural networks and genetic algorithms, models and tools of knowledge acquisition, such as data mining, and all other topics in management science with the emphasis on computational paradigms.

Why Computer Science? II

- **Programming develops your ability to solve problems.** Because machine languages are so simplistic, you have to tell the computer everything it needs to do in order to solve a problem.
- The single most important skill for a computer scientist is problem solving. Problem solving means the ability to **formulate problems, think creatively about solutions, and express a solution clearly and accurately.**
- Computer science **develops your ability to understand systems**: simulation.
- **Programming languages are tools for creation:** they let you build cool things.

Why Python?

- Programming in Python is simply a lot of fun and more productive.
- Python is interpreted, beginners can pick up the language and start doing neat things almost immediately without getting lost in the problems of compilation and linking.
- Python comes with a large library of modules that can be used to do all sorts of tasks ranging from web-programming to graphics.
- Python can also serve as an excellent foundation for introducing important computer science concepts.
- Python's most remarkable features is its broad appeal to professional software developers, scientists, researchers, artists, and educators.
- What about C++ or MATLAB?

My first program in Python!

- C++ Hola Mundo

```
#include <iostream.h>

   void main()
   {
     cout << "Hello, world." << endl;
   }
```

- Python Hola Mundo

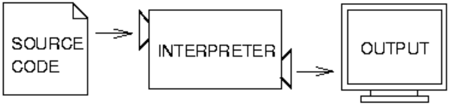
```
print("Hola, Caracola!")
```

The Python programming language I

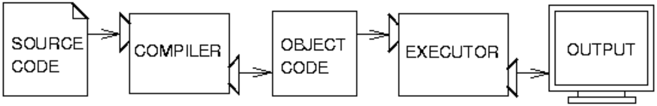
- Python is a **high-level language** like C, C++, Perl, and Java.
- There are also **low-level languages**, sometimes referred to as "machine languages" or "assembly languages."
  - Computers can only execute programs written in low-level languages. Thus, programs written in a high-level language have to be processed before they can run.
  - This extra processing takes some time, which is a small disadvantage of high-level languages.
  - It is much easier to program in a high-level language, takes less time to write, programs are shorter and easier to read, and they are more likely to be correct.
  - High-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications.
  - Low-level programs can run on only one kind of computer and have to be rewritten to run on another.

The Python programming language II

- Two kinds of programs process high-level languages into low-level languages: **interpreters and compilers.**
  - An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing computations.



  - A compiler reads the program and translates it completely before the program starts running. The high-level program is called the **source code**, and the translated program is called the **object code** or the **executable**. Once a program is compiled, you can execute it repeatedly without further translation.



The Python programming language III

- Python is an interpreted language because Python programs are executed by an interpreter. There are two ways to use the interpreter: **command-line** mode and **script mode**.

```
$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> print(1 + 1)
2
```

- Alternatively, you can write a program in a file and use the interpreter to execute the contents of the file. Such a file is called a **script.**

```
$ python printTwo.py
2
```

What is a program?

What is a program?

- A program is a sequence of instructions that specifies how to perform a computation i.e.:
  - solving a system of equations or finding the roots of a polynomial
  - symbolic computation: searching and replacing text in a document
  - compiling a program.

What is a program?

- A program is a sequence of instructions that specifies how to perform a computation i.e.:
  - solving a system of equations or finding the roots of a polynomial
  - symbolic computation: searching and replacing text in a document
  - compiling a program.
- A few basic instructions appear in just about every language:

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.
conditionals	Check for certain conditions and execute the appropriate sequence of statements.
repetition	Perform some action repeatedly, usually with some variation.

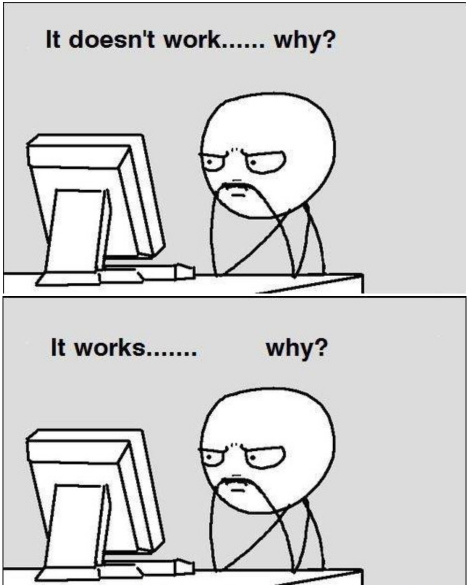
What is debugging?

- Programming errors are called bugs and the process of tracking them down and correcting them is called **debugging**.
- Three kinds of errors can occur in a program:
    - **Syntax errors:** Syntax refers to the structure of a program and the rules about that structure. Python can only execute a program if the program is syntactically correct; otherwise, the process fails and returns an error message.
    - **Runtime errors:** the errors do not appear until you run the program. These errors are also called **exceptions** because they usually indicate that something exceptional (and bad) has happened.
    - **Semantic errors:** the program will run successfully, in the sense that the computer will not generate any error messages, but it will not do the right thing.

Experimental debugging

- One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.
- In some ways, 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.
- Debugging is also like an experimental science. Once you have an idea 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.
- "When you have eliminated the impossible, whatever remains, however improbable, must be the truth." (A. Conan Doyle, The Sign of Four)

Experimental debugging



Formal and natural languages

- Natural languages are the languages that people speak, such as English, Spanish, and French. They were not designed by people (although people try to impose some order on them); 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. Chemists use a formal language to represent the chemical structure of molecules.

Programming languages are formal languages that have been designed to express computations.

Formal an natural languages

Although formal and natural languages have many features in common tokens, structure, syntax, and semantics there are many 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, which means that any statement has exactly one meaning, regardless of context.
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. If I say, ”The other shoe fell,”there is probably no shoe and nothing falling. Formal languages mean exactly what they say.

Course content

1. An informal introduction to Python and the interpreter
2. Data types, variables, compound data types
3. Operators and control sentences: decisions and loops
4. Functions in Python, Practical work
5. Scripts and modules: import and utilize a module
6. Computing with **NumPy** and **SciPy** and render simple plots with **matplotlib**
7. Write a simple class and access methods and attributes
8. Examples of Modeling and Simulation with Python

Evaluación

1. Conjunto 1 de problemas.
2. Conjunto 2 de problemas: morse y words.
3. Conjunto 3 de problemas (numpy y pandas).
4. Conjunto 4 de problemas (POO).
5. Proyecto.

Recursos y trabajo autónomo

- Recursos
  - Introducción a la programación con Python 3, Marzal, Gracia, García.
  - How to Think Like a Computer Scientist.
- Trabajo autónomo:
  - Instale Anaconda en su computador personal. Familiarícese con anaconda y las diferentes herramientas: Jupyter, IPython, Spyder.