

Curso Python en 8 clases

Clase 1: Introducción y visión general

Autor: Sebastián Bassi



Versión: 1.0

Licencia: Creative Commons BY-NC-SA 2.5. ([ver texto completo](#))



Introducción

Un programa es un conjunto de instrucciones diseñadas para ordenar a la computadora a hacer algo.

Es similar a una receta de cocina, que consiste en una lista de ingredientes e instrucciones paso a paso donde se usan dichos ingredientes.

Ejemplo de programa:

```
seq1 = 'Hola'
seq2 = ' mundo!'
total = seq1 + seq2
print total
```

El resultado es (previsiblemente!):

```
Hola mundo!
```

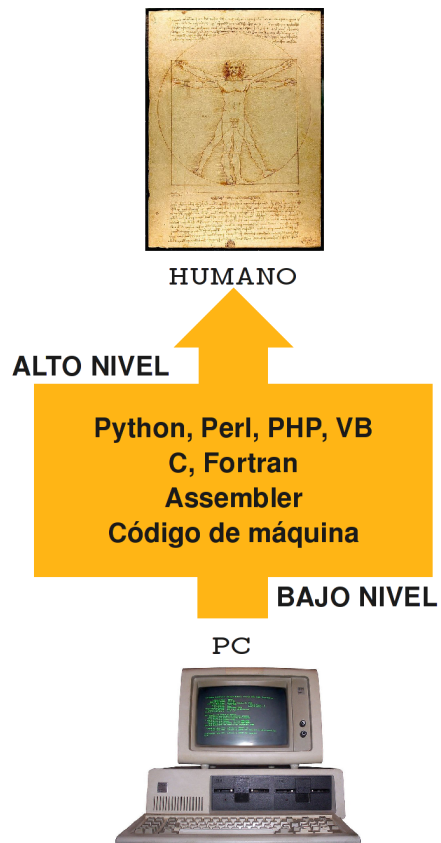
Niveles de abstracción

A medida que las instrucciones dadas a la computadora están codificadas en un formato mas cercano al que entiende la máquina, se dice se trabaja en "bajo nivel". En sentido inverso, a medida que las instrucciones son mas comprensibles para el humano se habla de "alto nivel".

Los lenguajes de "alto nivel" son mas comprensibles para los programadores aunque suelen tener menor performance. Por el contrario los lenguajes de bajo nivel son mas complejos pero aprovechan mas los recursos informáticos. La principal desventaja (ademas de la complejidad) es que el código de bajo nivel suele ser mas dependiente de la plataforma que los de alto nivel (menos portable).

Para la mayoría de las aplicaciones, con la velocidad de los procesadores actuales, la diferencia de performance es despreciable, especialmente si se tiene en cuenta también los tiempos de programación (hora-hombre) y no solo los tiempos de ejecución (hora-máquina). El valor de la hora-máquina baja constantemente mientras que el costo de la hora-hombre suele aumentar.

Python es considerado un lenguaje de alto nivel.



Ejemplo de alto y bajo nivel

Bajo nivel:

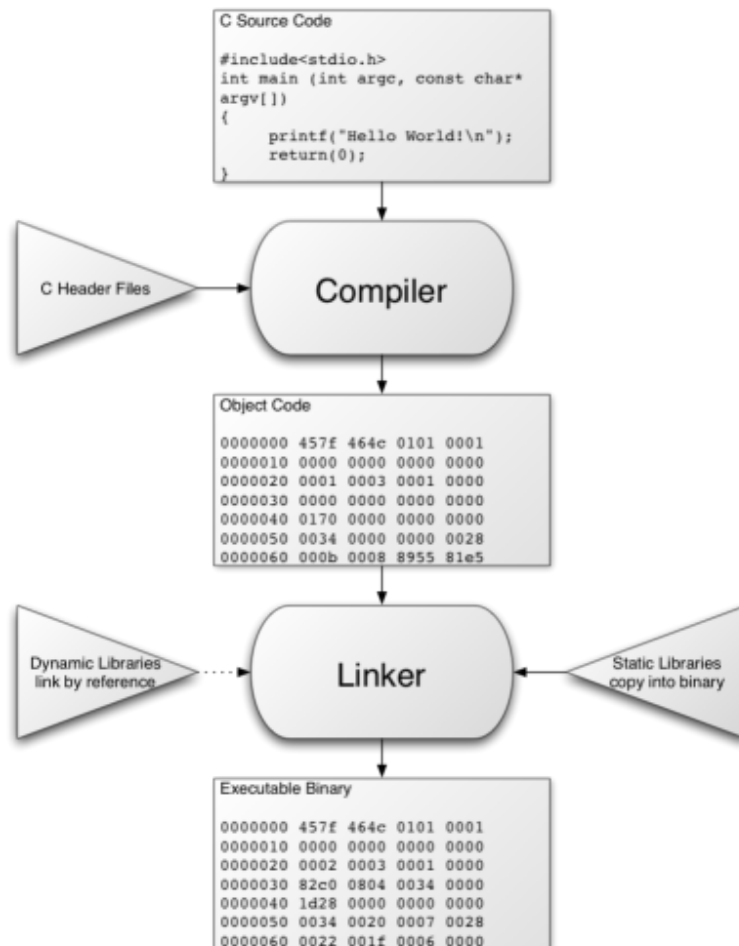
```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD98B
C84AEBF1 5BC3
```

Alto nivel:

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
```

Compilación

La compilación es la "Traducción" desde el *código fuente* (escrito por el programador) a instrucciones "ejecutables".



Esto implica que los programas deben ser compilados antes de poder ser ejecutados. En algunos lenguajes esta compilación se hace con todo el código mientras que en otros se va compilando línea por línea (lenguajes interpretados). Los lenguajes interpretados son por lo general mas lentos para ejecutarse que los programas compilados en su totalidad. El problema asociado a los lenguajes compilados es que para cada cambio que se quiera probar hay que compilar todo el código.

Una solución intermedia es hacer una compilación previa contra una máquina virtual, este es el caso de Python (junto con Java, C#, VB.net y otros).

Las principales consecuencias de la compilación son:

- Tiempo de compilación
- Aceleración en la ejecución del software
- Software dependiente de una plataforma



Fuente: <http://xkcd.com>. Licencia: CC by-nc 2.5.

Paradigmas de programación

Existen decenas de *paradigmas de programación*, en este instructivo se mencionarán solo los mas importantes:

- Procedural: En la *programación procedural* se le dan ordenes directas a la computadora, que van cambiando el estado del programa. Ejemplo: BASIC
- Estructurada: Es un caso particular de programación procedural, la diferencia es que hay un flujo ordenado (jerárquico) con alguna combinación de los siguientes elementos: Secuencias, selección y repetición. Se reutilizan bloques de código y es relativamente fácil seguir la lógica del programa. Ejemplo: C, Pascal.
- Orientada a objetos: Se usan estructuras de datos que tienen datos y métodos (objetos) con sus interacciones para diseñar programas. Ejemplos: Java, C++
- Lógico: Se declaran los problemas en forma lógica para que la computadora resuelva. Ejemplos: Prolog, Lisp.

Python es clasificado como "multiparadigma" debido a que puede ser usado de manera estructurado como orientado a objetos. Esto depende de las preferencias del programador, no impone uno de los dos métodos.

Diferencia entre especificación e implementación

La *especificación* se define las características del lenguaje. La especificación es única y es hecha por la comunidad de programadores de Python. Se habla de *implementación* de Python a aquel programa que cumple con dicha especificación. Hay varias implementaciones y cualquiera puede hacer la suya. La implementación mas popular es CPython (Python programado en C), a tal punto que coloquialmente se la llama "Python" a secas. Se puede decir que CPython es la implementación de referencia. En este curso cada vez que nos referiremos a Python en realidad lo hacemos a CPython. Otras implementaciones importantes son: Jython, Stackless Python, IronPython y PyPy.

Características de Python

Python se destaca por las siguiente características:

- Fácil de aprender y de programar

- Fácil de leer (similar a pseudocódigo)
- Interpretado (Rápido para programar)
- Datos de alto nivel (listas, diccionarios, sets, etc)
- Libre y gratuito
- Multiplataforma (Win, Linux y Mac)
- Pilas incluidas
- Cantidad de bibliotecas con funciones extras
- Comunidad

Como muestra de la facilidad de programar que se le atribuye a Python, veamos un código en VB y su equivalente en Python:

```
Dim i, j, Array_Used As Integer
Dim MyArray() As String
Dim InBuffer, Temp As String
Array_Used = 0
ReDim MyArray(50)
' Abrir archivo de texto . . .
Do While Not EOF(file_no)
    Line Input #file_no, MyArray(Array_Used)
    Array_Used = Array_Used + 1
    If Array_Used = UBound(MyArray) Then
        ReDim Preserve MyArray(UBound(MyArray) + 50)
    End If
Loop
' Ordeno con metodo de burbuja
For i = Array_Used - 1 To 0 Step -1
    For j = 1 To i
        If MyArray(j - 1) > MyArray(j) Then
            'swap
            Temp = MyArray(j - 1)
            MyArray(j - 1) = MyArray(j)
            MyArray(j) = Temp
        End If
    Next
Next
```

Este código lee un archivo de texto, carga el contenido en una estructura de datos y luego lo ordena de manera alfabética. El mismo programa en Python:

```
# Abrir un archivo de texto . . .
file_object = open(FILENAME)
# Leer todas las lineas del texto en una lista (similar a un array)
lista = file_object.readlines()
# Ordenar la lista
lista.sort()
```

Biblioteca estándar

Los módulos de la biblioteca estándar son aquellos incluidos en Python. Es bueno conocerlos porque proveen de muchos servicios y porque siempre están disponibles donde haya un interprete de Python. Algunos de los temas que nos ayuda la biblioteca estándar:

Servicios del sistema, fecha y hora, subprocesos, sockets, i18n y l10n, base de datos, threads, formatos zip, bzip2, gzip, expresiones regulares, XML (DOM y SAX), Unicode, SGML, HTML, XHTML, email,

manejo asincrónico de sockets, clientes HTTP, FTP, SMTP, NNTP, POP3, IMAP4, servidores HTTP, SMTP, debugger, random, curses, logging, compilador, decompilador, CSV, análisis lexicográfico, interfaz gráfica incorporada, matemática real y compleja, criptografía, introspección, unit testing, doc testing, acceso a SQLite etc., etc...

Para mas información: <http://docs.python.org/library>

Bibliotecas externas

Hay miles de bibliotecas externas con software muy diverso:

- Bases de datos: MySQL, PostgreSQL, MS SQL, Informix, DB/2.
- Interfaces gráficas: Qt, GTK, win32, wxWidgets, Cairo
- Frameworks Web: Django, Turbogears, Zope, Plone, web2py
- Biopython: Manejo de secuencias genéticas
- PIL: para trabajar con imágenes
- PyGame: juegos, presentaciones, gráficos
- SymPy: matemática simbólica
- Numpy: calculos de alta performance

Práctica en interprete interactivo

El siguiente código tiene es para ser ejecutado en el interprete interactivo (IDLE o via terminal de línea de comando). Existen reemplazos como IPython (<http://ipython.scipy.org>) que el lector puede probar por su cuenta.

```
>>> 2+2
4
>>> _*4
16
>>> 10/3
3
>>> float(10)/3
3.3333333333333335
>>> 10.0/3
3.3333333333333335
>>> int(2.1)
2
>>> int(2.9)
2
>>> round(2.9)
3.0
>>> int(round(2.9))
3
>>> round(2.932224,2)
2.9300000000000002
>>> print round(2.932224,2)
2.93
>>> "hola" + " mundo!"
'hola mundo!'
>>> ("hola" + " mundo!").upper()
'HOLA MUNDO!'
>>> ' 123'.strip()
'123'
>>> 123.strip()
```

```
File "<stdin>", line 1
  123.strip()
      ^
SyntaxError: invalid syntax
>>> >>> str(123)
'123'
>>> int('123')
123
```

Ayuda incorporada

Manejar la ayuda online es de gran utilidad:

```
>>> help( )

Welcome to Python 2.6!  This is the online help utility.

If this is your first time using Python, you should definitely check
out the tutorial on the Internet at http://docs.python.org/tutorial/.

Enter the name of any module, keyword, or topic to get help on
writing Python programs and using Python modules.  To quit this help
utility and return to the interpreter, just type "quit".

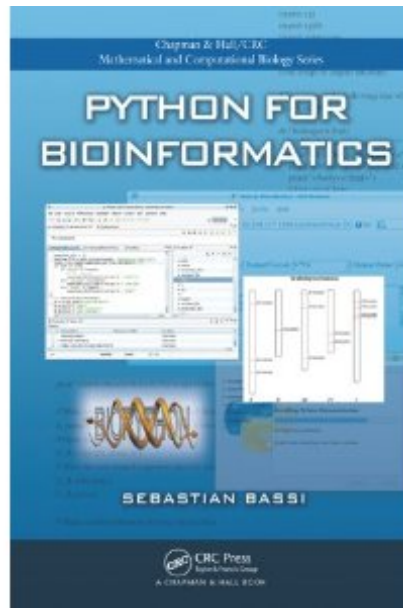
To get a list of available modules, keywords, or topics, type
"modules", "keywords", or "topics".  Each module also comes with a
one-line summary of what it does; to list the modules whose summaries
contain a given word such as "spam", type "modules spam".
```

Mas ayuda

- Manual de Python: <http://pyspanishdoc.sourceforge.net/tut/tut.html>
- Mailing list: <http://python.org.ar/pyar/ListaDeCorreo>
- Mas recursos: <http://python.org.ar/pyar/AprendiendoPython>

Libro

Python for bioinformatics: <http://tinyurl.com/biopython>



Contenido: Programación básica, archivos, funciones, POO, REGEX, Bazaar, Bases de datos, XML y Biopython.