Curso Python en 8 clases

Clase 1: Introducción y visión general

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. Esta *receta* en programación se le dice *programa*. Este programa debe estar escrito en un lenguaje determinado para que quien la vaya a ejecutar pueda entenderla. Si en una receta aparece la frase "cocinar a baño María" es porque ya está establecido previamente que significa dicho procedimiento. En programación ocurre algo similar, existen lenguajes específicos solo diseñados para comandar a la máquina para realizar una determinada acción. La multitud de lenguajes de programación existentes se debe a una combinación de cuestiones historicas, científicas y comerciales.

De todos los lenguajes en este libro veremos Python.

Ejemplo de programa en Python:

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

El resultado es (¡previsiblemente!):

```
Hola mundo!
```

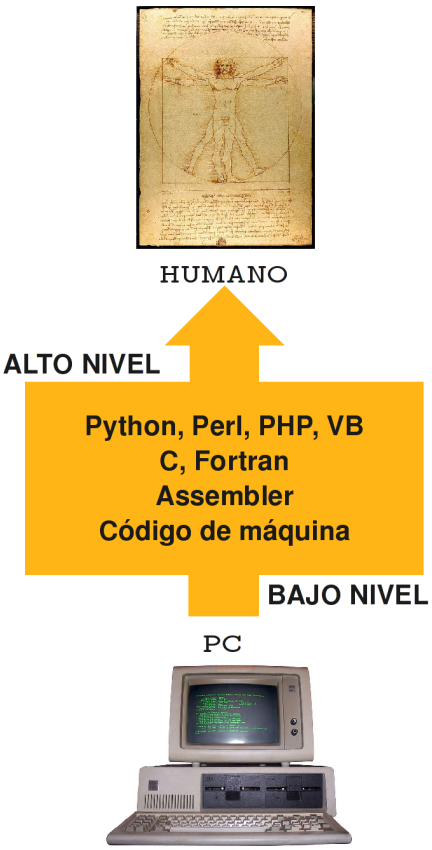
Niveles de abstracción

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

Los lenguajes de "alto nivel" son mas comprensibles para los programadores aunque suelen tener menor performance (ejecución mas lenta). Por el contrario los lenguajes de bajo nivel son mas complejos para programar pero aprovechan mejor los recursos informáticos. La principal desventaja (además de la complejidad) es que el código de bajo nivel suele ser mas dependiente de la plataforma que los de alto nivel y por lo tanto, 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 tiende a aumentar.

Python es considerado un lenguaje de alto nivel.



Ejemplo de alto y bajo nivel

Bajo nivel:

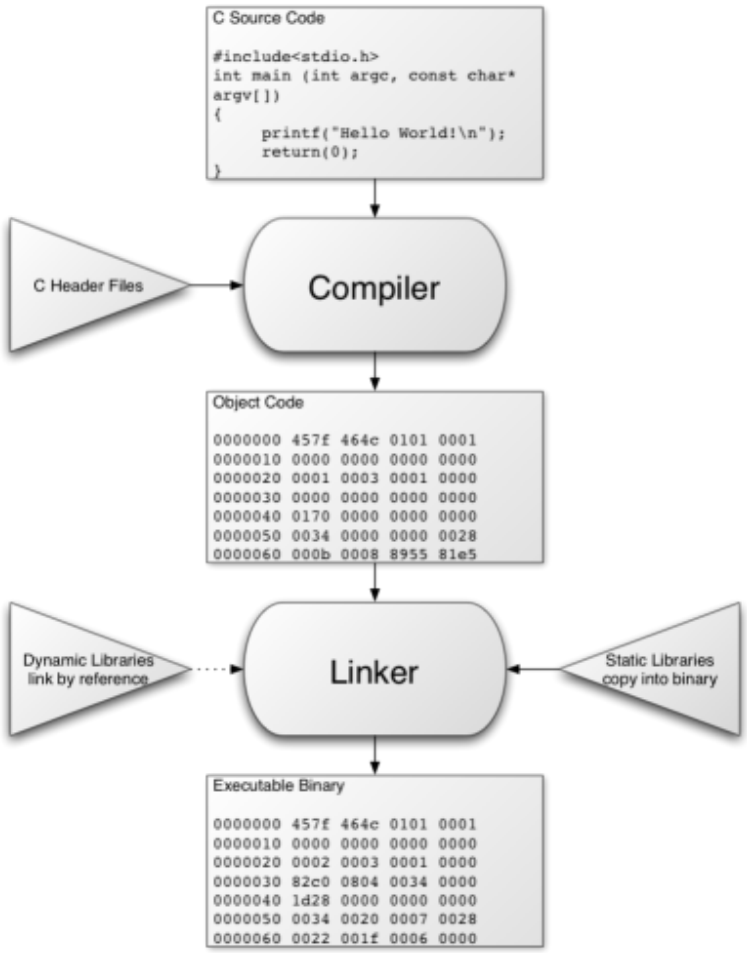
```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD98B
C84AEBF1 5BC3
```

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. Un problema asociado a los lenguajes compilados es que para cada cambio que se quiera probar hay que recompilar 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. Publicado en este libro con permiso del autor.

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 suele ser clasificado como "multiparadigma" debido a que puede ser usado tanto de manera estructurada 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 coloquilmente se la llama "Python" a secas. Se puede decir que CPython es la implementación de referencia de Python. En este curso cada vez que nos referiremos a Python en realidad lo hacemos a CPython. Otra implementaciones importantes son: Jython, Stackless Python, IronPython y PyPy. Por ejemplo PyPy es una implementación de Python hecha en Python, mientras que IronPython funciona utilizando la máquina virtual de .net o .mono.

Características de Python

Las siguientes son algunas de las características en la que se destaca Python:

- 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 Visual Basic (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 trata 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.

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

Biblitecas externas

Hay miles de bibliotecas externas con software muy diverso. Algunos ejemplos:

- Bases de datos: MySQL, PostgresSQL, 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: cálculos de alta performance

Para ver una lista completa de bibliotecas externas, consultá la página de PyPI (Python Package Index) en http://pypi.python.org/pypi.

Práctica en interprete interactivo

Las siguientes instrucciones son para ser ejecutadas en el interprete interactivo (IDLE o via terminal de línea de comando). Para arrancar dicho interprete hay que hacer doble-click sobre el ícono de Python o tipear *python* desde una terminal (esto último es válido solo para Linux y MacOSX). Existen reemplazos para el interprete interactivo como **IPython** (http://ipython.scipy.org) y **bpython** (http://www.bpython-interpreter.org) que el lector puede probar por su cuenta.

La ventaja del interprete interactivo es que podemos ejecutar código Python y obtener un resultado de manera inmediata. Normalmente se usa para experimentar y aprender. Tambien se lo usa para *debuguear* programas, esto es, encontrar y solucionar errores de programación. Aunque hay programadores que se arreglan para hacer gran parte de sus tareas diarias usando solo el interprete interactivo. En estos casos suelen usar reemplazos mas potentes como el ya mencionado IPython. Nosotros lo usaremos en este libro mas que nada como una herramienta exploratoria para conocer al lenguaje.

Probá ejecutando estas instrucciones en el interprete interactivo:

```
>>> 2+2
4
```

```
>>> _*4
16
>>> _*2
32
```

¿Qué significa el guión bajo (_)? La división entre enteros produce enteros:

```
>>> 10/3
3
```

Si al menos uno de los miembros de la división es un número de coma flotante, el resultado también lo es:

```
>>> float(10)/3
3.3333333333333335
>>> 10.0/3
3.3333333333333335
```

La función *int()* retorna un entero, mientras que *round()* redondea:

```
>>> int(2.1)
2
>>> int(2.9)
2
>>> round(2.9)
3.0
>>> int(round(2.9))
3
```

No es lo mismo ver un valor, que su *representación* (cosa que se logra con *print*):

```
>>> round(2.932224,2)
2.9300000000000002
>>> print round(2.932224,2)
2.93
```

Existe el tipo de datos de cadenas (string) que tiene sus propias operaciones permitidas:

```
>>> "hola" + " mundo!"
'hola mundo!'
>>> ("hola" + " mundo!").upper()
'HOLA MUNDO!'
>>> ' 123'.strip()
'123'
>>> 123.strip()
 File "<stdin>", line 1
  123.strip()
           ^
SyntaxError: invalid syntax
```

Entrada y salida de datos

Los datos pueden ingresar y egresar a los programas dediversas maneras, por ejemplo via la lectura de un archivo en un medio local, de un feed de rss o desde un dispositivo externo como una balanza electrónica. Los datos pueden salir en pantalla, en impresora o ser grabados en un archivo. Esta sección tratará los métodos mas básicos de entrada/salida (I/O) de datos. Para entrada se verá **input** que permite ingresar datos desde el teclado y para salida se mostrará **print**. En los dos casos su comportamiento tendrá diferencias entre las versiones de Python 2.x y 3.x.

Entrada

En Python 2.x existen **input** y **raw_input**. Ambas instrucciones permiten ingresar datos desde el teclado. En el primer caso (input), los datos son procesados antes de ser ingresados mientras que en raw_input lo ingresado es tomado de manera literal (sin procesar).

Ejemplos de uso:

```
>>> input('Ingrese valor: ')
Ingrese valor: 2+2
4
>>> raw_input('Ingrese valor: ')
Ingrese valor: 2+2
'2+2'
```

En Python 3.x *input* funciona como *raw_input* de Python 2.0. El comportamiento de la instrucción *input* original es considerado inseguro. Si pese a esto se quiere emular dicha función en Python 3.x, se puede hacer usando **eval()**.

Salida

Hay varias maneras de sacar datos desde un programa en Python. La mas elemental es usando **print**. En Python 2.x **print** es una instrucción que imprime una expresión en la salida estándar (por defecto la pantalla). En Python 3.x **print** es una función que proveé nuevas opciones. En el uso básico las diferencias son mínimas, basta poner parentesis en el print de Python 3.x para indicar que se trata de una función.

Python 2.x:

```
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56) [GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hola"
Hola
```

Python 3.x:

```
Python 3.1.2 (r312:79147, Apr 15 2010, 12:35:07) [GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola")
Hola
```

Las diferencias mas sustanciales pasan por las nuevas funcionalidades como: Imprimir elementos de una secuencia y cambiar el tipo de separador entre elementos, el caracter de fin de línea y el dispositivo de salida. La función completa (prototipo de la función) es:

```
print([objeto, ...], *, sep=' ', end='\n', file=sys.stdout)
```

Ejemplo de uso:

```
>>> print(1,2,3,sep='\n')
1
2
3
>>> a=[1,2,3] #esto es una lista
>>> print(a,sep=';')
[1, 2, 3]
>>> print(*a,sep=';')
1;2;3
```

Tanto en Python 2.x como en Python 3.x, existe el módulo **pprint** (pretty printer) para imprimir estructuras de datos en un formato que pueda ser entendido por el intérprete y ademas sea agradable a la vista. Veremos su uso luego de haber usado estructuras mas complejas.

Ayuda incorporada

Manejar la ayuda online es de gran utilidad. Si *help()* se ejecuta sin parámetros, se entra en "modo help", donde se espera que el usuario ingrese el nombre del módulo o tipo de datos del que quiere ayuda. Para salir hay que tipear *quit* o apretar *Control-D*.

```
>>> 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".
```

Otra manera de usar la ayuda es ingresando el módulo, la función o el tipo de dato como parámetro. Por ejemplo:

```
>>> help(len)
Help on built-in function len in module __builtin__:

len(...)
len(object) -> integer
Return the number of items of a sequence or mapping.
>>> help(3)
Help on int object:

class int(object)
 |  int(x[, base]) -> integer
 |
 (...sigue...)
```

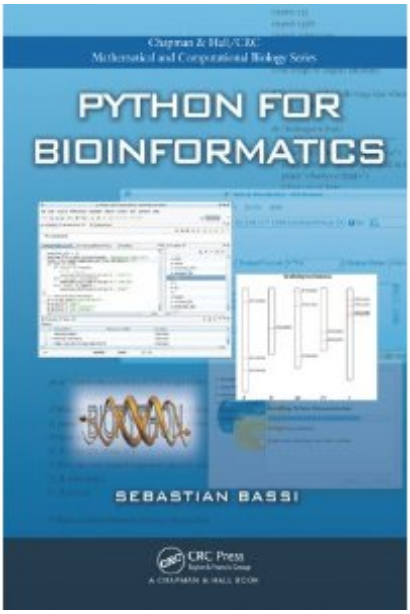
Si la ayuda incluida no es suficiente, recomiendo los enlaces que se encuentran en la sección siguiente (Mas ayuda).

Mas ayuda

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

Algunos libros recomendados

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



Contenido: Programación básica, estructuras de datos, control de flujo, archivos, funciones, POO, REGEX, Bazaar (control de versión), Bases de datos (MySQL y SQLite), XML y Biopython. En inglés.

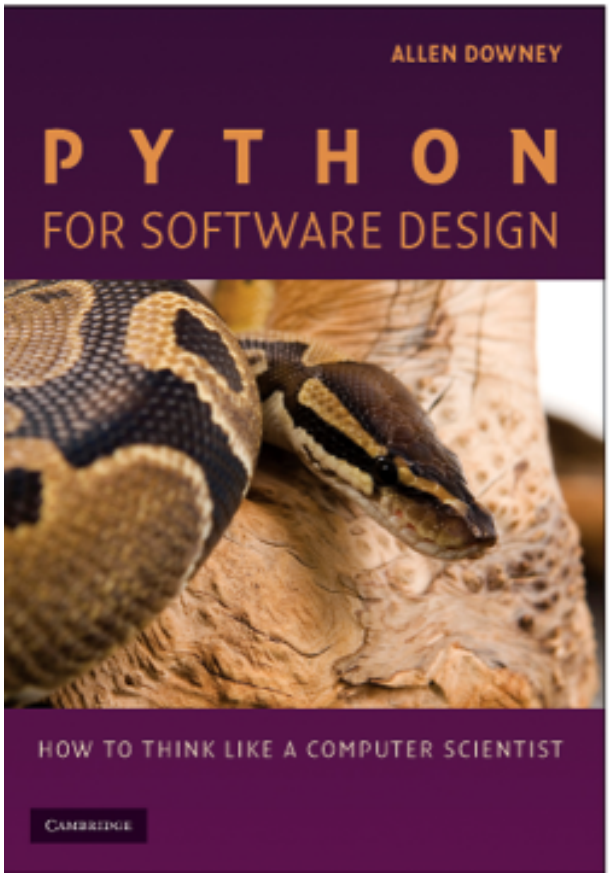
Python para todos: mundogeek.net/tutorial-python

Contenido: Tipos básicos, control de flujo, funciones, objetos, excepciones, REGEX, sockets, threads, bases de datos, documentación, pruebas y distribuir aplicaciones Python.

Python no muerde, yo sí: http://nomuerde.netmanagers.com.ar

Contenido: Para quienes ya conocen la sintaxis de Python y quieren profundizar sobre el mismo.

Python for Software Design: http://greenteapress.com/thinkpython/



Contenido: Como programar, variables, expresiones, funciones, recursión, iteración, cadenas, listas, diccionarios, tuplas, archivos, clases y objetos, funciones y métodos, herencia, tkinter y debugging.