PROGRAMAR EN PYTHON

Resumen

La elección del primer lenguaje de programación es un debate recurrente entre los docentes universitarios de ingenierías informáticas. La Universitat Jaume I ha optado por una solución poco convencional: en el primer curso de dos titulaciones de ingeniería informática se aprende a programar con Python y C. Python es un lenguaje que está en auge en el mundo del software libre y que presenta una serie de características que lo hacen muy atractivo para enseñar a programar. Como material de apoyo hemos escrito un libro de texto (accesible gratuitamente) y desarrollado un sencillo entorno de programación multiplataforma para Python que se distribuye con licencia GPL: el entorno PythonG, formado por un intérprete interactivo, un editor, un depurador sencillo y una ventana con salida gráfica. Con el material docente elaborado se facilita la formación autodidacta para cualquiera que quiera aprender a programar desde cero.

En este artículo reflexionamos sobre la idoneidad de Python como primer lenguaje de programación, describimos la experiencia docente de enseñar Python y C en primer curso y presentamos el entorno de programación PythonG.

1. Introducción

Hace años había un claro consenso en el mundo académico acerca del lenguaje de programación con el que enseñar a programar: Pascal. Era considerado elegante y sencillo, a la vez que ofrecía soporte para el paradigma de programación imperante: la programación estructurada. Son pocos los que aún consideran seriamente que Pascal sea adecuado y no faltan razones: las deficiencias del Pascal estándar, que obligan a usar variantes dialectales incompatibles entre sí; la ausencia de modularidad para ayudarse en el desarrollo de software de medio y gran tamaño; la falta de apoyo a paradigmas de programación como la programación orientada a objetos; su escasísima presencia en el mundo empresarial (exceptuando la de algún derivado de Pascal, como Delphi) o en el desarrollo de software libre...

En el entorno académico se opta hoy por diferentes lenguajes para introducir a los estudiantes en la programación. Es corriente optar por C, C++ o Java, y más raramente por otros como Modula-2, Ada o Scheme. Quienes optan por lenguajes como Modula-2 o Ada lo hacen principalmente por su elegancia y por el soporte que dan a ciertos aspectos de la programación: modularidad, chequeo

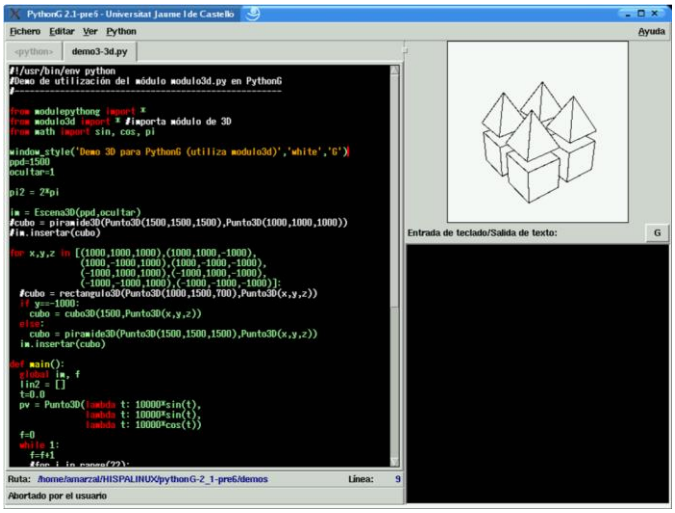

Figura 1: El entorno de programación PythonG.

Estático de tipos, etc. Scheme, un derivado de Lisp, forma parte del curr´ıculum de algunas universidades estadounidenses (en buena medida gracias al libro de Abelson *et al.* [1]) y permite una aproximación funcional a la programación, aunque los paradigmas imperativo y orientado a objetos sean hoy dominantes. Mucho se puede criticar de C (y cualquiera que haya programado en C puede escribir un tratado sobre ello), pero frente a una visión puramente académica de la programación, C contrapone su fuerte presencia en *((*el mundo real*))*. Quienes consideran que C no resulta un buen primer lenguaje de programación pero siguen con la mirada puesta en *((*el mundo real*))*, se decantan generalmente por C++ o Java, atractivos por su orientación a objetos y fuerte implantación en la industria.

Dos titulaciones de informática de la Universidad Jaume I hacen una apuesta diferente: en primer curso no se ensena un lenguaje de programación, sino dos, Python y C.[1] Python no solo es un lenguaje académicamente interesante, muy expresivo y con una sintaxis limpia y sencilla: es, además, un lenguaje ya presente y con mucho futuro en *((*el mundo real*))*. Los lenguajes de *script* (lenguajes interpretados de muy alto nivel, como Perl y Python) gozan de creciente importancia e implantación en el mundo del software libre. Pero Python no es suficiente para abordar el contenido de muchas asignaturas que requieren un lenguaje de programación de sistemas, ni para colmar los conocimientos prácticos de programación en el curr´ıculum de un informático. Una vez han aprendido a programar con Python, los alumnos pasan a estudiar el lenguaje C. La gran ventaja de esta aproximación es que, al haber aprendido ya a programar, las peculiaridades de C pueden estudiarse como lo que son: peculiaridades (por emplear una expresión amable).

Para impartir las prácticas de la asignatura se ha desarrollado un sencillo entorno de programación, llamado PythonG (ver figura 1), con un editor de textos

[1] Dedicamos a cada 45 horas de teoría/problemas y 30 horas de prácticas a cada lenguaje.

orientado a Python, interprete interactivo de ´ordenes Python, terminal gráfico y depurador. El software desarrollado se distribuye con licencia GPL.

El mercado editorial ofrece infinidad de títulos para aprender programar en C, C++ y Java. No ocurre lo mismo con Python, al menos no en España. Este problema se ha superado escribiendo un curso completo que cubre el temario de la asignatura. Se trata de un libro de apuntes que supone en el alumno unos conocimientos previos prácticamente nulos y que ha sido escrito con un estilo expositivo muy próximo al de una clase presencial [6]. El libro puede utilizarse para el aprendizaje autodidacta de la programación y que se distribuye gratuitamente en la red.

Este artículo está estructurado como sigue. En la sección 2 se presentan las características que, a nuestro juicio, debe reunir un lenguaje de programación utilizado para aprender a programar. En la sección 2.1 se examina Python bajo esta luz y se justifica su idoneidad como primer lenguaje de programación. El apartado 2.2 resume las ventajas de aprender C una vez se sabe programar con Python. Los apartados 3 y 4 presentan el entorno de programación PythonG y el libro de texto de la asignatura. Finalmente, en el apartado 5 se apuntan algunas conclusiones.

2. La cuestión del primer lenguaje

La primera pregunta que hemos de hacernos es ¿qué requisitos debe reunir un lenguaje de programación para que pueda considerarse un buen lenguaje de iniciación? Debe tenerse en cuenta que el objetivo de un curso introductorio a la programación no es la enseñanza en profundidad de *un* lenguaje de programación concreto, sino la asimilación de una serie de estrategias para el diseño y desarrollo de soluciones a problemas que usan, como vehículo de expresión, un lenguaje de programación. El lenguaje de programación es instrumental y lo ideal es que *interfiera* lo menos posible en la implementación de los algoritmos.

El primer aspecto a tener en cuenta es la *sintaxis*. El lenguaje debe tener cierto sentido de la economía en el uso de símbolos auxiliares y sus estructuras deben seguir unos principios sencillos que permitan generalizaciones efectivas. Muchos errores de programación de los aprendices se deben a la omisión o uso incorrecto de terminadores y delimitadores de bloque (que frecuentemente pasan inadvertidos al compilador por no provocar errores sintácticos o semánticos). Estos errores no siempre se deben a un mal diseño del algoritmo por parte del estudiante, sino al pobre soporte que ofrece el lenguaje de programación para una expresión concisa y clara del algoritmo. Esto nos lleva a una segunda característica deseable: el lenguaje debe ser *expresivo*, es decir, debe poder *((*decir mucho con poco*))*. Para ello, el lenguaje debe ofrecer *estructuras de control* flexibles y presentar una colección de *tipos y estructuras de datos* que permita expresar relaciones complejas entre datos con una notación sencilla (por ejemplo, dando soporte sintáctico a estructuras secuenciales como las listas). La *semántica* del lenguaje también debe ser sencilla. Contribuye a ello que el lenguaje sea muy *ortogonal*, es decir, si una construcción o método funciona con una estructura de datos, debe funcionar de modo similar con aquellas otras que guardan alguna semejanza (si cierta función o método calcula la longitud de una cadena, por ejemplo, debería calcular también la longitud de una lista, pues ambas son secuencias).

Programar es una actividad que, especialmente en fases tempranas del aprendizaje, se basa en el método de prueba y error. Es deseable que el lenguaje vaya acompañado de un entorno de programación que facilite un ciclo de edición ejecución rápido. Resulta crítico, además, que se detecten y señalen correctamente los errores de todo tipo (léxicos, sintácticos, semánticos estáticos *y de ejecución*): nada hay más frustrante para un estudiante que un programa dado por bueno por un compilador y que, *((*inexplicablemente*))*, falla en la ejecución con un mensaje tan parco como *((*violación de segmento*))*, sin indicar siquiera en que línea se produjo el error.

Las características citadas son, a nuestro entender, fundamentales. Un lenguaje que la presente menguada o que no las presente en absoluto no es un buen candidato. Hay otras características que, aunque deseables, son secundarias. Entre ellas tenemos, por ejemplo, su presencia en el *((*mundo real*))*, aunque teniendo en cuenta que está fuertemente sometido al dominio de las modas y la mercadotecnia. Hace años, C reinaba absolutamente; al poco, irrumpió con fuerza C++; hoy parece que Java domina buena parte del mercado; y Microsoft se ha empeñado en que C# sea lo más. Pero, si examinamos el mundo de la pequeña y mediana empresa, Visual Basic es el amo. Sin comentarios. En cualquier caso, no perdamos de vista la creciente presencia de Perl y Python en el mundo de la programación. El entorno LAMP (Linux-Apache-MySQLPerl/PHP/Python) [4] ha introducido estos lenguajes en el currículo de muchos programadores profesionales. Otra característica deseable es la existencia de un rico conjunto de módulos o librerías que facilite la programación de ciertas tareas: entrada/salida, funciones y constantes maten áticas, expresiones regulares... y, por qué no, aplicaciones web, serializaci´on de objetos, comunicación entre ordenadores, interfaces gráficas,... También resulta deseable que el entorno de programación esté disponible en el mayor número posible de plataformas (incluso en las que no son libres, ya que un efecto secundario de este aspecto es que facilita la migración de los estudiantes a la plataforma GNU/Linux; pero eso es otra historia).

2.1. Python como primer lenguaje

Aunque Python no goce aún de amplia reconocimiento en el mundo académico nacional, pretendemos demostrar en esta sección que puede compararse muy favorablemente con los lenguajes adoptados en la mayoría de universidades y que, en consecuencia, este reconocimiento puede ser cuestión de tiempo. De hecho, son varias las universidades extranjeras que ya han adoptado Python como parte de su curricular en informática o como herramienta para la introducción a la programación de no informáticos[2].

La sintaxis de Python es extremadamente sencilla. Ilustremos esa sencillez con un ejemplo: el tradicional *((*Hola, mundo.*))* con el que se presentan muchos lenguajes de programación. Un lenguaje como C obliga a incorporar la cabecera de una biblioteca estándar, a definir una función principal que devuelve un valor nulo (obligatorio en C99) y a codificar el salto delinea de una forma criptica (al menos para una persona que no ha visto un programa en su vida):

[2] Universidad de Irvine, California (EEUU); Wartburg College, Iowa (EEUU); Centre College, Kentucky (EEUU); Universidad de San Diego (EEUU); Universidad de Helsinki (Finlandia); Trinity College, Melbourne (Australia)...

```
#include < stdio.h >
int main(void) {
    printf("Hola, mundo.\n"); return 0 ;
}
```

¡Quince componentes léxicos (sin contar la directiva del preprocesador)! La cosa no mejora mucho en C++ (¡y eso sin entrar en la polémica de que cada an˜o, o casi, hay que escribir este programa *((*canónico*))* de una forma diferente!):

```
#include < iostream >
int main(void) {
    std::cout << "Hola, mundo." << std::endl;
}
```

Y en Java, resulta complicado hasta lo inveros´ımil:

```
public class HolaMundo {
    public static void main(String [] args) { System.out.println("Hola, mundo.");
    }
}
```

¿Cómo explicar a un estudiante que no ha visto un programa en su vida el significado de cada uno de los términos sin solicitar un acto de fe tras otro? Python va directo al grano: `print 'Hola, mundo.'`

Naturalmente, el programa *((*Hola, mundo.*))* no es determinante a la hora de escoger un lenguaje de programación, pero si debería suscitar una seria reflexión acera de la excesiva e innecesaria complejidad a la que se somete a los principiantes a la programación.

No podemos desgranar todos los elementos que, a nuestro juicio, hacen de Python un lenguaje de sintaxis sencilla, pero si queremos destacar uno de ellos: los **bloques se marcan con la indentaci´on** del código. He aquí un ejemplo de programa Python con bloques:

```
def f(x): return x**2 - 2*x - 2
def biseccion(a, b, epsilon): c = (a + b) / 2.0 while
   f(c) != 0 and b - a > epsilon:
       if f(a)*f(c) > 0: a = c elif
       f(b)*f(c) > 0: b = c
       c = (a + b) / 2.0 return c print 'x =',
biseccion(0.5, 3.5, 1e-10)
```

Como se puede comprobar, **no hay terminadores de sentencia** (como el punto y coma de C/C++/Java) **ni marcas de inicio/fin de bloque** (como las llaves de esos mismos lenguajes). La indentaci´on como forma de marcar bloques elimina errores propios de los lenguajes citados y que son frecuentes en los estudiantes (¡y también en programadores profesionales!): sentencias condicionales sin acción por añadir un punto y coma incorrecto, bucles con una sola sentencia cuando el alumno cree que hay dos o más (por omisión de llaves con un sangrado inadecuado del programa), sentencias con semántica *((*alterada*))* por usar una coma cuando corresponde un punto y coma o por omitir un punto y coma al

declarar un registro antes de una función, etc. La indentaci´on solo resulta molesta cuando el tamaño de un bloque de cierta profundidad excede del tamaño de la ventana del editor, pero ese caso no es frecuente en los programas de un curso introductorio. El entorno PythonG es un editor adaptado a la programación que elimina las incomodidades de iniciar manualmente cada línea con blancos o tabuladores[3]. Un aspecto interesante de la indentaci´on forzosa es que disciplina a los estudiantes en el sangrado correcto del código *en otros lenguajes de programación*: hemos percibido que el código C de nuestros estudiantes esta mejor endentado si empiezan aprendiendo Python.

Python es un lenguaje interpretado. Los lenguajes interpretados permiten ciclos de desarrollo breves (edición y ejecución) que animan a los estudiantes a experimentar. Python dispone de un entorno de ejecución que ayuda a detectar los errores (incluyendo aquellos que solo se manifiestan en ejecución) señalándolos con mensajes muy informativos. Python ofrece, además, un entorno interactivo con el que es posible efectuar pequeñas pruebas o diseñar incrementalmente las soluciones a los problemas. Nuestro curso sugiere empezar a programar usando el entorno interactivo de Python como una calculadora avanzada. He aqu´ıun ejemplo de una breve sesi´on interactiva de trabajo:

```
>>> 2 + 2
4
>>> from math import sin, pi
>>> pi
3.1415926535897931
>>> for i in range(4): ...      print i,
sin(i*pi/4) ...
1  0.0
2  0.707106781187
3  1.0
4  0.707106781187
```

La contrapartida de que se trate de un lenguaje interpretado es, obviamente, la menor velocidad de ejecución. No obstante, esta menor velocidad no resulta en absoluto importante para los programas propios de un primer curso de programación.

Python puede considerarse pseudoc´odigo ejecutable. Muchos cursos de iniciación a la programación empiezan por presentar nociones básicas con seudocódigo, es decir, con un lenguaje de programación inexistente que aporta, eso sí, la flexibilidad suficiente para expresar cómodamente algoritmos. De este modo se evita tener que lidiar con la infinitud de detalles propios de lenguajes de programación tradicionales. Pero Python es muy expresivo y su sintaxis sencilla interfiere poco en la implementación de algoritmos, así que resulta un buen

[3] No es el único editor adaptado a la sintaxis de Python. Editores como vim o Emacs/Xemacs también están preparados para la edición de programas Python. El entorno IDLE, que se distribuye con el intérprete, también facilita el desarrollo de programas.

sustituto del seudocódigo, con la ventaja de que los algoritmos codificados en Python si son ejecutables.

Python es un lenguaje tapado dinámicamente. Cada dato es de un tipo determinado (a diferencia de otros lenguajes de script, como Tcl, en los que muchos tipos son, en el fondo, cadenas) y solo se puede operar con ´el de formas bien definidas. La *ventaja* es que no hay que declarar variables antes de su uso. Esto, que en ciertos ambientes se considera sacrílego, resulta de gran ayuda para el que empieza a programar: las sutiles diferencias entre declaración y definición, por ejemplo, se obvian en Python. Cuando el estudiante se ha acostumbrado a usar variables y ha comprendido el concepto de tipo de datos, puede transitar fácilmente a C y entender con mayor facilidad las ventajas de la declaración de variables en un lenguaje compilado. Curiosamente, quienes critican opciones como Python por no ser un lenguaje estáticamente tapado aceptan de buen grado opciones como C o C++, lenguajes en los que *((*todo vale*))* cuando se apunta a memoria. Esta característica, que resulta útil en la programación de sistemas, se presta a enorme confusión en el principiante. No es que Python evite el trabajo con punteros, al contrario, en Python toda la información se maneja vía punteros (referencias a memoria), sino que la memoria apuntada mantiene información de tipo sobre los datos almacenados, evitando problemas de acceso a ellos con tipos erróneos.

Hay una vertiente negativa en la no necesidad de declarar variables: los errores derivados de teclear incorrectamente identificadores de variables o de atributos de objetos. Si se comete un error al teclear el nombre de una variable o atributo en la parte izquierda de una asignación, se crea una nueva variable. El mismo problema, cuando ocurre en la parte derecha, es detectado por el entorno de ejecución, así que no resulta tan grave.

Python facilita la detección y gestión de errores mediante excepciones. Las excepciones forman parte ya de los lenguajes de programación modernos (Java las incorporo desde el principio y C++ lo ha hecho más recientemente) y eliminan la excesiva complejidad de la detección y tratamiento de errores con lenguajes de programación que, como C, fuerzan a detectarlos con valores especiales de retorno y que no ofrecen un modelo claro de interrupción de rutinas para localizar su tratamiento en un solo punto.

Python ofrece un rico conjunto de estructuras de datos flexibles. El tipo lista de Python (un vector dinámico heterogéneo) permite introducir con naturalidad el concepto de secuencia y presentar los algoritmos básicos de manejo de secuencias. Que la indexación empiece siempre en 0 ayuda a dar el salto a C, C++ o Java. El entorno de ejecución proporciona comprobación de validez de los ´índices, eliminando así una de las principales fuentes de problemas de C y C++. El hecho de que las listas sean redimensionarles elimina al estudiante la necesidad de tomar decisiones acerca de la longitud máxima de los vectores demasiado pronto. Es el camino que ha adoptado últimamente C++ con la STL (aunque, ¿alguien se atreve a presentar la STL en las primeras semanas de formación de un programador, con todas sus sutilezas y esa sintaxis endemoniada?). Por otra parte, Python es un lenguaje muy *ortogonal*: una vez se ha aprendido a manejar listas, por ejemplo, se sabe manejar cadenas, ya que ambos tipos son secuenciales y presentan conjuntos de operadores con igual nombre y semántica. Además de

listas y cadenas, Python ofrece tupas (listas inmutables) y diccionarios (vectores asociativos).

Python simplifica la gestión de memoria. El modelo de memoria de Python es sencillo: todo valor reside en el *((*heap*))* y toda variable contiene una referencia a su valor. A ello se suma un sistema de **recogida automática de basura** (*garbage collection*) que evita los punteros colgantes (*dangling pointers*), las fugas de memoria, las violaciones de segmento, etc. Estos errores de ejecución, habituales en lenguajes como C o C++, son difíciles de detectar y convierten el desarrollo de programas en una actividad más frustrante de lo que es razonable, especialmente para el principiante.

Python ofrece una amplísima colección de módulos (bibliotecas). Hay módulos para cualquier actividad imaginable: escritura de CGI, gestión de correo electrónico, desarrollo de interfaces graficas de usuario, análisis de documentos HTML o XML, acceso a bases de datos, trabajo con expresiones regulares, etc. No es que haya que presentar al estudiante todos estos módulos (no hay tiempo); pero sí es posible organizar actividades alrededor de la asignatura de programación (seminarios, talleres,...) que introduzcan diferentes campos de aplicación. Acostumbrar al estudiante a consultar la documentación de las bibliotecas disponibles desde bien temprano ayuda a hacer de ellos programadores eficientes.

Una ventaja adicional de Python es, pues, que hace posible organizar seminarios sobre temas *((*modernos*))* y que alumnos de primer curso los sigan con aprovechamiento: CGI, análisis de texto con expresiones regulares, interfaces gráficos de usuario, etc.

El mecanismo de paso de parámetros es único. Los parámetros se pasan a funciones y métodos por referencia a objeto. En la práctica, se comporta de forma similar al paso de parámetros de Java: el paso de objetos básicos (de tipo escalar) se realiza con efectos similares al paso por valor y el de objetos más elaborados (vectores, diccionarios, instancias de clase, etc.) por referencia.

```
def parametros(a, lista): a += 1
    lista.append(10)
x = 1
l = [1 ,2, 3] parametros(x, l) print 'x no se
ha modificado:', x
print 'pero a l se le ha a~nadido el elemento de valor 10:', l
```

Si bien el paso de parámetros de C es más flexible (incluye una forma de paso de parámetros por referencia basada en el paso por valor de un puntero), el comportamiento por defecto con respecto al paso de variables de tipo básico y vectorial es análogo al de Python, haciendo sencilla la migración.

Python es orientado a objetos. A diferencia de Java, Python permite una programación puramente procedimental. La orientación a objetos, aunque perfectamente soportada, es opcional (a menos, naturalmente, que se recurra a ciertos módulos en los que se definen clases). El soporte a la programación orientada a objetos es similar al de lenguajes como Smalltalk: la resolución de los nombres de método y atributos es dinámica. Ello elimina la necesidad de

complicadas jerarquías de herencia (aunque Python soporta la herencia múltiple), clases virtuales e interfaces. Cuestiones como el diseño de contenedores gen éricos esta también resuelta, pues el sistema de tipos dinámico ofrece la flexibilidad suficiente.

```
class Pila:
    def __init__(self, n): # M´etodo constructor.
        self.index = -1 self.buffer =
        [None] * n
    def push(self, value): # Apila un elemento.
        if self.index >= len(self.buffer):
            raise "Error: Pila llena."
        self.index += 1 self.buffer[self.index] = value
    def top(self): # Consulta la cima de la pila.
        if self.index < 0:
            raise "Error: Pila vac´ıa."
        return self.buffer[self.index]
    def pop(self): # Extrae la cima de la pila.
        if self.index < 0:
            raise "Error: Pila vac´ıa."
        self.index -= 1 return
        self.buffer[self.index+1]
p = Pila(5) p.push(3)
p.push(2) print
p.top() print
p.pop() print
p.pop()
```

(Hemos de decir que el primer año que impartimos el curso incluimos un tema de orientación a objetos. Los resultados no fueron satisfactorios, quizá en buena medida porque la celeridad con que impartimos el temario de Python (un cuatrimestre) no permite alcanzar la madurez necesaria para asimilar los conceptos propios de esta metodología. En la actualidad explicamos los tipos de datos compuestos mediante registros.)

2.2. C como segundo lenguaje

Aprender C una vez se sabe programar con otro lenguaje resulta más sencillo. El estudiante ya conoce los conceptos fundamentales (tipo de dato, variable, bucle, selección condicional, etc.) y tiene cierta soltura en el diseño de algoritmos. La introducción al nuevo lenguaje puede plantearse inicialmente en términos de traducción de programas Python cuyo comportamiento es bien conocido por los estudiantes. El discurso sobre los abundantes detalles de C se puede plantear pues, en el marco de las abstracciones que ya conocen. Python permite introducir todos los conceptos con una sintaxis minimalista y preservando suficiente abstracción como para que los conceptos se aprendan en su esencia. El estudio de C obliga a repasar todos estos conceptos y a examinarlos desde una ´óptica ligeramente (a veces radicalmente) diferente. El resultado es una visión doble de los conceptos y, por tanto, más sólida. Una lista, por ejemplo, es una secuencia de

elementos sobre la que es posible implementar ciertas operaciones. Esa es la esencia. Como se implementa en C (bien con punteros a bloques contiguos de memoria dinámica, bien con registros enlazados) es una cuestión relacionada con las características propias del lenguaje.[4]

Haber aprendido Python ofrece la ventaja añadida de haber disciplinado al estudiante en cuestiones elementales de legibilidad del código, como ya hemos apuntado antes. La indentaci´on, por ejemplo, es un concepto ya interiorizado. Los tipos y estructuras de datos se presentan en C como versiones más pobres de aquellas que ya conocen por Python: los vectores, por ejemplo, no presentan operaciones *((*nativas*))* como la concatenación o el cálculo de su longitud. Un ejercicio interesante es explotar estas limitaciones para proponer al estudiante que implemente operaciones y métodos cuyo comportamiento conoce bien gracias a Python (inversión, búsqueda, ordenación, extracción de cortes, etc.).

Habrá quien piense que impartir dos lenguajes de programación en primer curso ha de hacerse, necesariamente, en detrimento de la cantidad de conceptos aprendidos. No es así. El temario propuesto considera todos los aspectos relevantes de un primer curso de programación: tipos de datos, estructuras secuenciales, estructuras de control, funciones, ficheros y una introducción al análisis de algoritmos.

Ensenar dos lenguajes de programación tiene una ventaja añadida: acostumbra al estudiante a una realidad con la que se ha de enfrentar: la diversidad de herramientas que deben formar parte de su *((*banco de trabajo*))*. ¿Se concibe un programador eficaz que use un solo lenguaje de programación? ¿Se es productivo desarrollando software solo en C? Cada problema demanda una forma de trabajo y cada herramienta se adapta a un tipo de problema.

3. El entorno PythonG

Como material de apoyo al curso hemos desarrollado un entorno de programación para Python: PythonG. El objetivo fundamental es ofrecer un entorno sencillo y cómodo de usar. PythonG [5] (ver figura 1) incluye un editor de texto dirigido por la sintaxis (coloreado automático, sugerencia automática de indentaci´on), una consola de entrada salida, un intérprete interactivo, un terminal de salida grafica (con rutinas de acceso a teclado y ratón) y un depurador muy sencillo. El entorno de distribuye con licencia GPL y esta accesible en la dirección http://marmota.act.uji.es/MTP.

El editor de texto tomo como punto de partida el que ofrecía el entorno IDLE. Se ha procurado emular el comportamiento de Emaús/Semas: las combinaciones de teclado para las acciones básicas de edición son las mismas (aunque se han añadido algunas que los estudiantes echaban en falta, como las comunes en operaciones de cortar/copiar/pegar). De este modo facilitamos al estudiante la transición, durante el segundo semestre, al editor que usamos en las sesiones practicas: XEmacs.

PythonG es multiplataforma, al igual que el entorno estándar de Python, pues utiliza la biblioteca estándar para diseño de interfaces, Tkinter (un recubrimiento de la biblioteca Tk). El estudiante puede descargar el software e instalarlo

[4] No decimos que de igual hacerlo de un modo u otro. En el mismo curso, más adelante, se ensena al estudiante a escoger en función del coste computacional de cada opción.

fácilmente en cualquier distribución de Linux o cualquier otro sistema operativo en el que corra el intérprete de Python.

Una característica reseñable y que hace atractivas las practicas es el acceso que facilita a funciones graficas básicas y de interacción con teclado y ratón. De este modo los estudiantes pueden empezar, desde bien temprano, a diseñar programas con salida gráfica y cierto nivel de interacción sin la necesidad de aprender antes Programación Orientada a Objetos y algún marco conceptual excesivamente sofisticado. Creemos que estas posibilidades son muy motivadoras para los estudiantes: el profesorado está acostumbrado a interactuar con su software a través de la consola o la línea de comandos, pero la imagen que tienen los estudiantes del ordenador es fundamentalmente gráfica, así que suelen mostrar aversión por los programas de consola y una cierta inclinación por los programas gráficos. Orientar un curso entero a programas de consola es, en opinión de muchos estudiantes, frustrante. En los dos últimos cursos, los estudiantes han podido mejorar su calificación final realizando trabajos voluntarios atractivos: un juego interactivo donde unos robots dan caza a un personaje (una versión de Daleks), un módulo para trazar gráficos de tortuga (similar a la funcionalidad del lenguaje Logo), una versión completa de Tetris (ver figura 2) y un rompecabezas con Pentominos. (Los enunciados están accesibles en la página web reseñada más arriba).

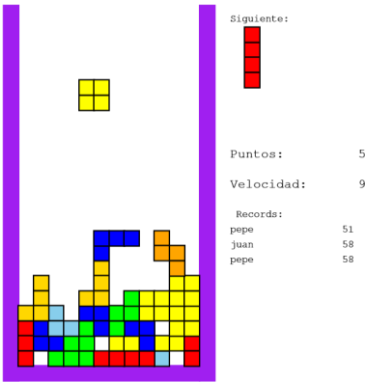

Figura 2: Pantalla del juego de Tetris que implementan los estudiantes en el entorno PythonG.

Las funciones gráficas y de acceso a teclado y ratón están disponibles en un módulo independiente, de modo que los programas desarrollados no necesitan ejecutarse desde el entorno de programación. Es posible, pues, implementar programas portables con salida grafica sencilla.

El entorno ofrece un modo de depuración simplificado (figura 4). Cuando se activa, se muestra la línea actual destacada con fondo azul. Una botonera da acceso a las acciones básicas de control de flujo: ejecución paso a paso, ejecución

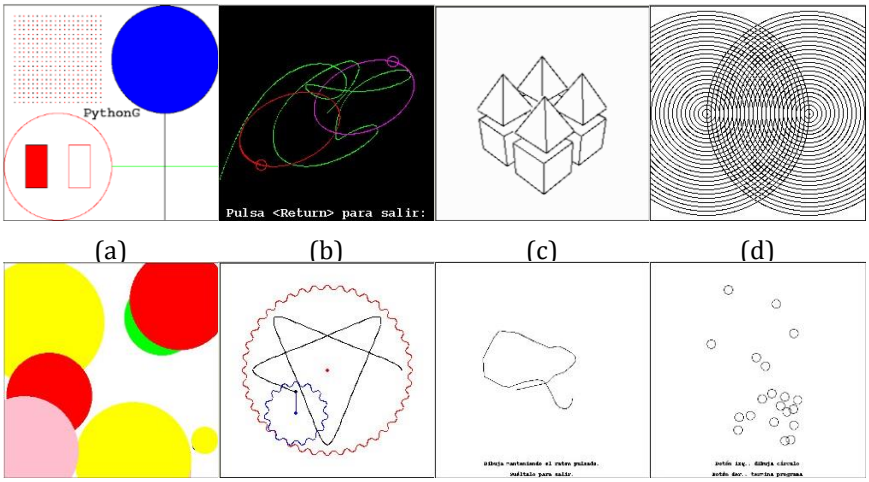

Figura 3: Salida de los programas de demostración que acompañan al entorno PythonG. (a) Ilustración de las posibilidades gráficas. (b) Una simulación gravitacional con tres cuerpos. (c) Un programa de visualización en 3D con ocultación de caras. (d) Una demostración de la capacidad para mover (grupos de) objetos. (e) Una animación. (f) Un programa para dibujar figuras con engranajes. (g) y (h) Demostraciones de interacción con entrada de ratón.

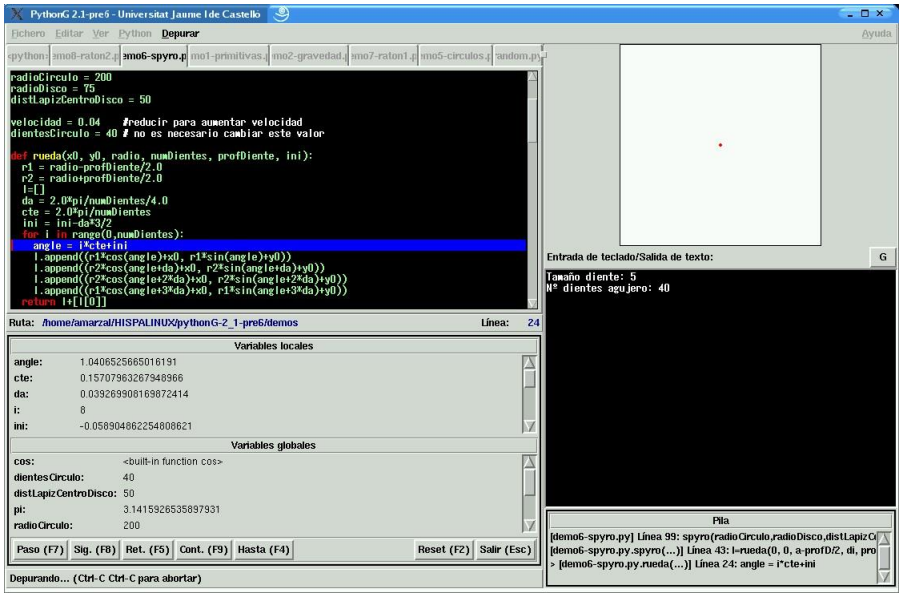

Figura 4: Modo de depuración del entorno PythonG.

Continua hasta la línea del cursor (que actúa como *((*breakpoint*))*), ejecución hasta fin de función, reinicio de la ejecución, etc. Bajo la ventana de edición se muestran

las variables locales y globales en un instante dado y bajo la consola se muestra el estado de la pila de llamadas a función.

4. Un libro de texto

Uno de los problemas con que se enfrenta el profesorado que opta por usar Python como primer lenguaje de programación es la relativa ausencia de libros de texto adecuados. La bibliografía sobre Python no cesa de crecer y se cuenta ya con algunos libros de introducción a la programación. Poems dustcart trees: *((*Python Programming: An Introduction to Computer Science*))* [6] (corridor disponible en web y pendiente de publicaci´on como libro); *((*How to Think Like a Computer Scientist: Learning with Python*))* [2]; y *((*Learn to Program Using Python*))* [3]. Otros libros sobre Python suelen incluir una introducci´on r´apida al lenguaje, pero asumen conocimientos de programación y/o resultan excesivamente breves para un curso como el propuesto.

A estas monografías se suma la que hemos escrito como material para clases de teor´ıa y problemas. Se trata de un libro de texto en castellano con una aproximaci´on ligeramente diferente de las seguidas en estos tres libros. El libro, titulado *((*Introducción a la programación. Volumen I: Python*))* [6], disponible gratuitamente en la web (para autodidactas e instituciones públicas de enseñanza), no tiene por objeto presentar la totalidad del lenguaje de programación Python, sino enseñar al estudiante los conceptos básicos de la programación y las estrategias seguidas al desarrollar los programas propios de un primer curso. Se asume que el lector no tiene conocimiento alguno de programación y el nivel de maten áticas necesario para seguir el discurso es el propio de la educación secundaria. Tras una breve introducción a los computadores y la programación en general, se presenta a los estudiantes el entorno interactivo de Python. Con la excusa de su utilización como calculadora avanzada, se introducen las expresiones y los tipos de datos básicos (entero, flotante y cadena), así como algunas funciones de la biblioteca matemática. El siguiente capítulo presenta el concepto de programa y las funciones de entrada salida, a la vez que incluye un breve tutorial del entorno PythonG. El siguiente capítulo se dedica a presentar las estructuras de control (selección condicional y bucle). El estudiante pasa entonces a estudiar listas (en realidad, vectores dinámicos) y a diseñar algoritmos que manejan secuencias de datos. El diseño de funciones se aprende a continuación. El siguiente capítulo presenta registros[5]. El último capítulo se dedica al manejo de ficheros. La próxima edición incorporar ‘a capítulos dedicados al manejo de diccionarios, la programación orientada a objetos y un apéndice dedicado al diseño de interfaces gráficas. Aunque no hay tiempo material para impartir el material adicional en el curso actual, estos nuevos capítulos dotaran de cierta completitud al libro como material de aprendizaje de Python.

El libro está plagado de ejemplos completamente desarrollados y que, en ocasiones, se presentan siguiendo los caminos equivocados que suelen tomar los estudiantes y que conocemos gracias a la experiencia docente de varios años. El objetivo es, precisamente, reflexionar sobre la naturaleza de los errores que suelen cometer. Se han incluido numerosos cuadros flotantes con información

[5] Python no proporciona soporte nativo para registros, sino para clases e instancias, asique los registros se implementan con un módulo extra.

adicional que profundizan en determinados aspectos o que pueden despertar la curiosidad del estudiante por materias que guardan cierta relación con los asuntos tratados en el cuerpo del texto. En la última edición, los programas están apuntados por enlaces desde el documento PDF para facilitar las pruebas y evitar, en la medida de lo posible, erratas en el código presentado.

A lo largo del texto, e intercalados con las explicaciones, se proponen cerca de 480 ejercicios que van desde preguntas simples para constatar que se ha asimilado lo expuesto, a pequeñas aplicaciones o sencillos videojuegos que hacen necesario combinar lo aprendido en diferentes capítulos. Con el fin de hacer más amena su lectura y estimular la curiosidad de algunos estudiantes, el libro se acompaña de numerosas digresiones que abordan cuestiones muy dispares relacionadas con el mundo de la programación.

En 2002 se liberó en la red una primera versión del libro que tuvo muy buena acogida, tanto por estudiantes universitarios y autodidactas españoles como latinoamericanos.

5. Conclusiones

Algunas universidades norteamericanas han empezado a adoptar Python como lenguaje de programación básico. En España, de momento, ya hay una experiencia piloto: la Universidad Jaume I. Durante los cursos 2001/2002 y 2002/2003 se ha puesto en práctica la enseñanza de la programación con Python y C. Creemos que Python es un lenguaje particularmente adecuado como primer lenguaje de programación. Uno de los resultados de la experiencia es el material docente confeccionado: un sencillo entorno de programación, una biblioteca simplificada para la implementación de programas gráficos y un libro de texto. El material se encuentra disponible en Internet y puede utilizarse para el aprendizaje autodidacta de la programación.

La experiencia docente ha sido muy satisfactoria. En un breve plazo de tiempo (un cuatrimestre), los estudiantes aprenden todos los aspectos básicos de la programación. Completamos su formación básica en la materia con la enseñanza del lenguaje C en el segundo semestre. La formación previa con Python es de gran ayuda para asimilar el nuevo lenguaje y entender las cuestiones técnicas que determinan ciertas cuestiones de diseño de C.

6. Referencias

[1] Abelson, H., Sussman, G. J. and Sussman, J.: *Structure and Interpretation of Computer Programs*, Segunda edici´on, 1996, MIT Press/McGraw-Hill.

[2] Allen Downey, Jeff Elkner and Chris Meyers: *How to Think Like a Computer Scientist: Learning with Python*. Green Tea Press. ISBN: 0971677506. http://www.ibiblio.org/obp/thinkCSpy/dist/thinkCSpy.pdf.

[3] Alan Gauld: *Learn to Program Using Python: A Tutorial for Hobbyists, Self-Starters, and All Who Want to Learn the Art of Computer Programming*. Addison-Wesley. ISBN: 0201709384.

[4] LAMP: The Open Source Web Platform. http://www.onlamp.com.

[5]

O'Reilly & Associates.

[6] David Llorens, http://www3.uji.es/~dllorens/PythonG/.

[7] Andr´es Marzal e Isabel Gracia: *Introducción a la programación. Volumen I: Python*. Colecci´on *((*Materials*))*, 147, Servei de Publicacions de la Universitat Jaume I. Disponible en http://marmota.act.uji.es/MTP.

[6] John Zelle: *Python Programming: An Introduction to Computer Science*. Pendiente de publicaci´on. Borrador del libro disponible en http://mcsp.wartburg.edu/zelle/python.

Realizado por rafael gonzalez freites pepelo
Email rafaelfreites@hotmail.com
Azua rep dom