Python como primer lenguaje de programación

Ariel Ortiz Ramírez
Departamento de Tecnologías de Información y Computación
División de Ingeniería y Arquitectura
Tecnológico de Monterrey, Campus Estado de México
ariel.ortiz@itesm.mx

30 de junio, 2010.

Resumen

El lenguaje de programación Python se ha convertido en años recientes en una herramienta muy valiosa para la enseñanza de la programación. Este documento propone utilizar dicho lenguaje en el primer curso de programación de los planes de estudio 2011 de las carreras profesionales del Tecnológico de Monterrey. Con el fin de justificar esta propuesta, se analizarán las características de Python que lo hacen una alternativa superior a otros lenguajes al momento de aprender a programar. Así mismo, se discutirán las investigaciones que se han hecho en diversas universidades sobre el uso de Python en la educación, así como las experiencias que ha tenido el Campus Estado de México en este mismo sentido.

1 Introducción

En tiempos recientes, el primer curso de programación, tal como se enseña en muchas instituciones de educación superior en México y el mundo, ha tomado un fuerte énfasis hacia la enseñanza de lenguajes de programación como C, C++, Java o C#. Dichos lenguajes son muy relevantes para la construcción de software de producción, pero no fueron diseñados para enseñar a programar. Lamentablemente esto ha provocado que muchos alumnos tengan la percepción de que la computación es una disciplina árida y con un alto nivel de dificultad técnica. Quizás esto no sea un gran problema cuando se está en una época de bonanza en cuanto a captación, pero cuando la matrícula se contrae esta percepción negativa puede tener un efecto devastador que no puede ser ignorado [28].

1.1 Objetivo de un primer curso de programación

Para comenzar nuestra discusión, primero debemos responder a la pregunta: ¿cuál es el propósito de un primer curso de programación? Este es el objetivo general de dicha materia, correspondiente a los planes de estudio 2011 del Tecnológico de Monterrey [23]:

> *Al finalizar este curso el alumno será capaz de aplicar la lógica para generar algoritmos que permitan resolver problemas.*

Es importante notar que no se menciona en el objetivo que el alumno tenga que aprender un lenguaje de programación. De alguna forma está implícito que sí tendrá que aprender alguno con el fin de poder probar si los algoritmos que generó realmente resuelven los problemas planteados. Sin embargo lo que en verdad pesa aquí es desarrollar la capacidad para resolver problemas. Necesitamos promover el uso de herramientas que sean parte de la solución, y no parte del problema.

1.2 Complejidad accidental y esencial

A mediados de los años ochenta, Brooks argumentó que todo proyecto de software afronta dos tipos de complejidades [4]:

Complejidad accidental. Es aquella que tiene que ver con las situaciones que surgen en el proceso de desarrollo de software pero que no tienen que ver con el problema siendo resuelto. Normalmente ocurre por la manera en que se decide abordar el problema.

Complejidad esencial. Es aquella que es inherente al problema en cuestión y no hay forma de evitarla.

Estos dos tipos de complejidades están también presentes a la hora de enseñar y aprender a programar. Muy a menudo hemos presenciado cómo nuestros alumnos batallan, literalmente por horas, con situaciones que no son esenciales al problema siendo resuelto. Por ejemplo, la siguiente porción de código en lenguaje C busca calcular el factorial de un número entero n. El código parece correcto, pero tiene un error muy sutil que sólo puede ser detectado a simple vista por alguien con un buen colmillo o utilizando un depurador:

```
int factorial(int n) {
    int i, resultado = 1;
    for (i = 2; i <= n; i++); {
        resultado *= i;
    }
    return resultado;
}
```

El programa compila correctamente, pero devuelve un resultado incorrecto. Esta situación no es exclusiva del lenguaje C. El mismo código en Java, C++ y C# se comporta igual. A decir verdad, el compilador de C# marca una advertencia que podría revelar el origen del problema[1]:

```
Factorial.cs(6,9): warning CS0642: Possible mistaken empty statement
Compilation succeeded - 1 warning(s)
```

El mensaje es bastante críptico, especialmente para un programador novato. Pero alguien con más experiencia seguramente se percataría ahora que la instrucción `for` tiene un punto y coma sobrante, implicando un cuerpo de ciclo vacío. El bloque que está inmediatamente después del `for` parece ser el cuerpo del ciclo, pero el compilador no lo considera así. Esto es complejidad accidental, y resulta bastante frustrante para un principiante.

Como académicos, resulta de suma importancia que procuremos minimizar la complejidad accidental, sobre todo en el primer curso de programación. Esto permitirá que los alumnos se enfoquen a desarrollar su capacidad para resolver problemas. El lenguaje de programación Python es una herramienta que podemos utilizar para eliminar éste y otros casos de complejidad accidental que surgen al momento de enseñar a programar.

2 El lenguaje de programación Python

Python es un lenguaje de programación moderno creado por Guido van Rossum a inicios de los años noventa. La implementación canónica, conocida como CPython, está bajo una licencia de software libre y se puede descargar del sitio oficial [26]. El que sea una tecnología abierta y libre tiene ventajas importantes sobre tecnologías propietarias. La principal es que se puede usar sin tener que cubrir costos de licencias. Esto quiere decir que un alumno puede seguir usando Python gratuitamente fuera del Tecnológico de Monterrey, por ejemplo para escribir software en un entorno comercial, o para continuar con sus estudios de posgrado en alguna otra universidad.

2.1 Principales características

A continuación se listan algunas de las características más sobresalientes del lenguaje Python [26]:

- Es orientado a objetos, pero soporta también los estilos de programación procedural y funcional.
- Corre en múltiples plataformas, incluyendo Windows, Mac OS y Linux.
- Su sintaxis y semántica es sencilla y consistente.
- Utiliza tipos dinámicos.
- Es adecuado tanto para programar *scripts* como aplicaciones de gran tamaño.
- Es muy modular.
- Cuenta con administración automática de memoria a través de recolección de basura.
- Incluye una poderosa y extensa biblioteca de clases.
- Cuenta con una gran comunidad que se dedica a promover su desarrollo y adopción.
- Por su naturaleza interactiva, resulta ideal para llevar a cabo programación experimental y desarrollo rápido.

2.2 Ejemplos

La simplicidad del lenguaje Python se puede demostrar con algunos ejemplos relativamente sencillos. En las siguientes subsecciones se presentan tres problemas distintos con sus respectivas soluciones tanto en Python[2] como en Java. El objetivo es poder comparar las soluciones. Estos problemas se pueden considerar típicos de un primer curso de programación.

2.2.1 Hola mundo

En primera instancia, el clásico programa *hola mundo* se puede escribir en Python en una sola línea de código:

```
print('Hola mundo!')
```

Lo único que se le tiene que explicar al alumno en este momento es que existe una función predefinida llamada `print` y que recibe una cadena de caracteres como argumento. El ejemplo es claro y conciso, y tiene un mínimo de complejidad accidental.

El mismo programa en Java resulta bastante más elaborado:

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

Tan sólo en este ejemplo los siguientes conceptos entran en juego: clases, métodos estáticos, modificador de acceso `public`, parámetros, arreglos de cadenas de caracteres, la clase predefinida `System`, el campo estático `out`, el método `println` y la cadena de caracteres que se manda como argumento. Y claro, también está la cuestión de dónde se deben usar llaves, paréntesis y puntos y comas. En la práctica, un profesor que enseña Java en un primer curso de programación normalmente no explica todos estos conceptos desde un inicio, sino que les dice a sus alumnos que acepten todo esto por el momento como "acto de fe" y que las cosas se irán aclarando conforme las vaya explicando a su debido tiempo. Como se puede observar, la mayor parte del programa no tiene absolutamente nada que ver con el problema que se está intentando resolver (imprimir "Hola mundo!" en la pantalla). Este ejemplo en Java demuestra una alto nivel de complejidad accidental que simplemente resulta en una carga intelectual innecesaria para nuestros alumnos. Y esto no es un problema exclusivo de Java. Lo mismo ocurre con otros lenguajes como C++ y C#.

2.2.2 Desviación estándar

Como segundo ejemplo tomemos un problema de estadística. Veamos la manera de calcular la desviación estándar σ de una población. La fórmula correspondiente es:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

En este caso n es el total de elementos de la población; $x_1, x_2, \ldots, x_n$ son los elementos individuales; y $\overline{x}$ es la media aritmética, la cual se calcula así:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Supongamos que se desea leer cada uno de los elementos desde la entrada estándar. El programa correspondiente en lenguaje Python podría quedar así:

```
from math import sqrt

def leer_datos():
    n = int(input('Cuantos elementos deseas leer? '))
    x = []
    for i in range(n):
        elemento = float(
            input('Dame el elemento {0}: '.format(i + 1)))
        x.append(elemento)
    return x

def media_aritmetica(x):
    suma = 0
    for e in x:
        suma += e
    return suma / len(x)

def desviacion_estandar(x):
    suma = 0
    media = media_aritmetica(x)
    for e in x:
        suma += (e - media) ** 2
    return sqrt(suma / len(x))

# El programa inicia su ejecucion aqui.
x = leer_datos()
s = desviacion_estandar(x)
print('Desviacion estandar = {0:.4f}'.format(s))
```

La solución en Java podría quedar como sigue:

```
import java.util.Scanner;

public class Desviacion {

    public static double[] leerDatos() {
        Scanner entrada = new Scanner(System.in);
        System.out.print("Cuantos elementos deseas leer? ");
```

```
        int n = entrada.nextInt();
        double[] x = new double[n];
        for (int i = 0; i < n; i++) {
            System.out.printf("Dame el elemento %d: ", i + 1);
            x[i] = entrada.nextDouble();
        }
        return x;
    }

    public static double mediaAritmetica(double[] x) {
        double suma = 0;
        for (double e: x) {
            suma += e;
        }
        return suma / x.length;
    }

    public static double desviacionEstandar(double[] x) {
        double suma = 0;
        double media = mediaAritmetica(x);
        for (double e: x) {
            suma += Math.pow(e - media, 2);
        }
        return Math.sqrt(suma / x.length);
    }

    public static void main(String[] args) {
        double[] x = leerDatos();
        double s = desviacionEstandar(x);
        System.out.printf("Desviacion estandar = %.4f%n", s);
    }
}
```

Al correr cualquiera de los dos programas, la entrada y la salida podrían ser:

```
Cuantos elementos deseas leer? 6
Dame el elemento 1: 4
Dame el elemento 2: 8
Dame el elemento 3: 15
Dame el elemento 4: 16
Dame el elemento 5: 23
Dame el elemento 6: 42
Desviacion estandar = 12.3153
```

Para este problema en particular, las dos soluciones resultan muy parecidas. Ambas usan estructuras de control y operadores similares. Sin embargo hay dos diferencias importantes a notar:

1. En Java los bloques de una clase, método o enunciado `for` se delimitan con las llaves "{" y "}". En Python la indentación cumple esta función. Esto es algo positivo, ya que obliga a los alumnos a adquirir prácticas de escritura de código legible.

2. Java utiliza declaraciones explícitas y estáticas de tipos de datos asociados a las variables, parámetros y tipos de regreso de los métodos. En Python no se requieren declaraciones; basta asignar un valor a una variable para poderla utilizar. Las variables pueden ser reasignadas a otros valores de cualquier tipo. En Java, los errores de incompatibilidad de tipos (por ejemplo intentar dividir un número entre un arreglo), se detectan normalmente a tiempo de compilación. En Python este tipo de errores se detectan a tiempo de ejecución.

Viendo los programas detenidamente, la versión escrita en Python es más parecida a un pseudocódigo. La eliminación de las llaves, puntos y comas y declaraciones de tipos de datos hace que el código sea más compacto y expresivo.

Tal como sucedió en el ejemplo de la subsección 2.2.1, Java obliga a poner todo el código dentro de una o varias clases. Python permite definir clases también, pero soporta así mismo la definición de funciones libres, las cuales resultan más naturales al momento de implementar código como el de la desviación estándar. Tal como se mencionó anteriormente, Python es un lenguaje orientado a objetos, pero también soporta un estilo de programación procedural más parecido a C o Pascal.

Un último aspecto que conviene comentar aquí es sobre el uso del enunciado `for` de Python, el cual difiere de manera importante del `for` de lenguaje C. En el lenguaje C, la forma común de hacer un ciclo `for` es algo así:

```
for (i = 0; i < 1000; i++) {
    /* Cuerpo del ciclo */
}
```

En realidad, este `for` es lo que se conoce como "azúcar sintáctica", ya que se pudo haber escrito usando un `while`:

```
i = 0;
while (i < 1000) {
    /* Cuerpo del ciclo */
    i++;
}
```

En Python, el enunciado `for` es mas bien equivalente al enunciado `foreach` de C# o al `for` "mejorado" de Java 5 o superior, y se utiliza para iterar sobre los elementos de una secuencia (por ejemplo una lista o una cadena de caracteres). Si se desea un ciclo que vaya del 0 al 999, como en el ejemplo anterior, se tiene que crear una secuencia que contenga a dichos elementos. La función `range` sirve justamente para dicho efecto, y se podría usar así:

```
for i in range(1000):
    # Cuerpo del ciclo
```

La función `range` devuelve un objeto especial que usa un esquema de evaluación perezosa (*lazy evaluation*), en donde se van generando los elementos de la secuencia conforme se van necesitando[3]. Esto es con el fin evitar que todos los elementos de la secuencia estén simultáneamente en memoria.

En casi cualquier lenguaje, uno de los usos más comunes de los ciclos `for` es el de recorrer todos los elementos de un arreglo. La variable con la que se controla el `for` normalmente se utiliza también como índice del arreglo. Por ejemplo, para sumar todos los elementos de un arreglo, el código idiomático en lenguaje C sería algo así:

```
int a[] = {4, 8, 15, 16, 23, 42};
int suma = 0, i;
for (i = 0; i < sizeof(a) / sizeof(a[0]); i++) {
    suma += a[i];
}
```

Traduciendo a Python línea por línea, tenemos:

```
a = [4, 8, 15, 16, 23, 42]
suma = 0
for i in range(len(a)):
    suma += a[i]
```

Sin embargo, el uso de los índices del arreglo es incidental. Lo que realmente nos interesa son los elementos individuales del arreglo. Por tanto, se puede iterar directamente sobre el arreglo para obtener el resultado deseado:

```
a = [4, 8, 15, 16, 23, 42]
suma = 0
for elem in a:
    suma += elem
```

Este código tiene un nivel mayor de abstracción. A parte de ser un poco más breve, esta solución es más idiomática en Python, y por lo tanto preferible.

2.2.3 Contando palabras de un archivo

Como tercer ejemplo, veamos cómo se puede determinar la frecuencia en la que ocurren las palabras dentro de un archivo de texto. Supongamos que el archivo de texto llamado `archivo.txt` contiene lo siguiente:

```
si la sierva que te sirve
no te sirve como sierva
de que sirve que te sirvas
de una sierva que no sirve
```

El programa en Python que resuelve el problema tiene sólo siete líneas de código:

```
with open('archivo.txt') as archivo:
    cadena = archivo.read()
conteo = {}
for palabra in cadena.split():
    conteo[palabra] = conteo.get(palabra, 0) + 1
for palabra, veces in conteo.items():
    print('{0:10}: {1}'.format(palabra, '*' * veces))
```

El programa equivalente en Java es al menos cinco veces más extenso:

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.HashMap;

public class Palabras {
    public static void main(String[] args) throws IOException {
        FileInputStream archivo = null;
        StringBuilder sb = new StringBuilder();
        String cadena;
        try {
            archivo = new FileInputStream("archivo.txt");
            int ch;
            while ((ch = archivo.read()) != -1) {
                sb.append((char) ch);
            }
            cadena = sb.toString();
        } finally {
            if (archivo != null) {
                archivo.close();
            }
        }
```

```
            HashMap<String, Integer> conteo =
                new HashMap<String, Integer>();
            for (String palabra: cadena.split("\\s+")) {
                if (conteo.containsKey(palabra)) {
                    conteo.put(palabra, conteo.get(palabra) + 1);
                } else {
                    conteo.put(palabra, 1);
                }
            }
            for (String palabra: conteo.keySet()) {
                System.out.printf("%-10s : ", palabra);
                for (int i = 0, n = conteo.get(palabra); i < n; i++) {
                    System.out.print("*");
                }
                System.out.println();
            }
        }
    }
```

Para ambos casos, la salida de la corrida es un histograma horizontal con las frecuencias (representadas con asteriscos) de cada palabra distinta hallada en el archivo de entrada:

```
de         : **
una        : *
sirve      : ****
como       : *
no         : **
la         : *
te         : ***
que        : ****
sirvas     : *
sierva     : ***
si         : *
```

La solución de Python es más breve gracias al soporte nativo que tiene para manejar archivos y diccionarios (tablas *hash*). De hecho, la solución de Java es tan elaborada y compleja que normalmente no se cubriría algo así en un primer curso de programación. Sin embargo sí sería posible hacerlo si se usara Python.

El lenguaje Python maneja en muchos sentidos un nivel más alto de abstracción que lenguajes como C++ y Java. Esto impacta positivamente en la productividad de quien lo usa. Esta es la misma razón por lo que hace cincuenta años una gran parte de la industria dejó de utilizar lenguajes de bajo nivel (código maquinal y ensamblador) y comenzó a usar en su lugar lenguajes de alto nivel (por ejemplo Fortran, Cobol y Lisp).

2.3 Cómputo numérico con Python

Python no es un software del mismo estilo que MATLAB, Mathematica o Maple en lo referente a su orientación matemática. Aún así, tiene un comportamiento numérico más regular que lenguajes como C o Java, y esto es algo que un principiante puede apreciar bastante [7]. Las siguientes subsecciones muestran las situaciones en donde Python resulta más adecuado en este sentido.

2.3.1 *Fixnums* y *Bignums*

Los enteros en lenguajes como C/C++ y Java son *fixnums*, es decir usan una representación interna fija, normalmente de 32 o 64 bits en las arquitecturas computacionales modernas. Si en alguna operación aritmética el resultado no es del tamaño esperado, ocurre un desbordamiento silencioso. Por ejemplo, dado el siguiente código en lenguaje C corriendo en una plataforma de 32 bits:

```
int a = 123456;
int b = 100000;
int c = a * b;
printf("%d\n", c); // Imprime -539301888
```

El resultado esperado es 12345600000, pero en su lugar se imprime -539301888. El signo negativo es el primer indicio de que ocurrió un desbordamiento. A esto se le conoce como una *leaky abstraction*, en donde los detalles específicos de una implementación se manifiestan de manera contraproducente, de tal forma que interfieren con la abstracción [32].

El error de desbordamiento de este ejemplo se puede corregir usando tipos de datos de mayor precisión, pero esto puede resultar algo complicado para un principiante. Otra opción es usar *bignums*, es decir, aritmética de precisión arbitraria. En este caso, la cantidad de dígitos de estos valores numéricos está limitada por la memoria disponible en el sistema. Varios lenguajes soportan este concepto de *bignums* de manera nativa, incluyendo Python, Scheme, Haskell y Smalltalk. Otros lenguajes como Java y C# lo soportan a través de bibliotecas, pero debido a esto su uso tampoco resulta intuitivo, es decir, es algo que se tendría que enseñar explícitamente. El ejemplo anterior podría quedar en Java de la siguiente manera:

```
BigInteger a = new BigInteger("123456");
BigInteger b = new BigInteger("100000");
BigInteger c = a.multiply(b);
System.out.println(c); // Imprime 12345600000
```

En Python los enteros se representan internamente de la manera en que resulte más eficiente. Los valores se convierten de *fixnums* a *bignums* y vice versa de manera totalmente transparente. No hay nada especial que explicar aquí a los alumnos. El código de Python se vería como sigue y el resultado no contiene sorpresa alguna:

```
a = 123456
b = 100000
c = a * b
print(c) # Imprime 12345600000
```

2.3.2 División entera

Una de las concepciones erróneas más comunes que tienen los alumnos que aprenden a programar en los lenguajes derivados de C (C/C++, C#, Java) tiene que ver con la manera en que trabaja la división entera. Dado el siguiente código de Java:

```
int a = 1;
int b = 2;
System.out.println(a / b); // Imprime 0
```

La mayoría de los alumnos cree que el resultado que se imprime es 0.5, en lugar de 0. Lamentablemente cuando intentan corregirlo, pueden escribir algo así:

```
int a = 1;
int b = 2;
float c = a / b;
System.out.println(c); // Imprime 0.0
```

El problema es que el operador de división (`/`) está sobrecargado, es decir, el operador puede realizar dos operaciones distintas. Si sus dos operandos son enteros, el resultado es un entero que corresponde al cociente de la división. Para que el resultado dé un número real, uno de los dos operandos debe ser de tipo real. En el siguiente ejemplo se usa un *cast* para lograr este efecto:

```
int a = 1;
int b = 2;
System.out.println((float) a / b); // Imprime 0.5
```

En Python[4] hay dos operadores para realizar la división: `/` (para división real) y `//` (para división entera). A los alumnos les resulta más intuitivo entender que hay dos operaciones distintas, que cada una tiene su propio operador, y que el operador `/` funciona justo como se imaginan. He aquí un ejemplo:

```
a = 1
b = 2
print(a / b)  # Imprime 0.5
print(a // b) # Imprime 0
print(a % b)  # Imprime 1
```

En la última línea podemos ver que el operador `%` (residuo) funciona igual que en otros lenguajes.

2.3.3 Operadores relacionales

Supongamos que deseamos verificar si el valor de una variable x se encuentra entre 10 y 20, de manera excluyente. Matemáticamente, esto se expresaría así: $10 < x < 20$. La traducción a lenguajes como C o Java no resulta muy intuitiva para un alumno novato, pues es necesario incluir el operador `&&`, el cual realiza la conjunción lógica entre dos subexpresiones:

```
10 < x && x < 20
```

En Python, la expresión queda escrita de manera más familiar así:

```
10 < x < 20
```

2.3.4 Número complejos

Python también soporta números complejos de manera nativa y cuenta con una biblioteca especial para soportar este tipo de valores. El número complejo $5 + 3i$ se escribe en Python como la literal `5+3j`. He aquí algunos ejemplos:

```
from cmath import sqrt
x = sqrt(-1)
print(x) # Imprime 1j
y = x + (4+3j)
print(y) # Imprime (4+4j)
```

Fortran y Scheme son otros dos lenguajes de programación que soportan números complejos de forma nativa; muchos otros lenguajes los soportan a través de bibliotecas.

3 Python en la educación

La idea de usar Python para enseñar a programar no es nueva. Desde hace algunos años, universidades de diversas partes del mundo han hecho una transición hacia Python con resultados bastante exitosos.

3.1 Estudio comparativo de lenguajes

En el estudio elaborado por [19], se presenta una lista de 17 criterios para determinar qué lenguaje es el más adecuado para enseñar a programar. Los criterios fueron generados a partir de las decisiones de diseño manifestadas por cuatro autores de lenguajes reconocidos por su enfoque didáctico[5]. Los criterios se clasifican en cuatro grupos: 1) aprendizaje, 2) diseño y ambiente, 3) soporte y disponilidad, y 4) más allá de la programación introductoria. El estudio consistió en usar dichos criterios para comparar 11 lenguajes que comúnmente se usan para enseñar a programar[6]. La conclusión fue que Python e Eiffel son los dos lenguajes más indicados para un curso introductorio de programación.

3.2 Materiales de apoyo

Otro elemento importante que habla del progreso que ha tenido Python en la educación es el número de libros de texto que han aparecido en el mercado. El más antiguo data del 2003 [37], aunque la gran mayoría han surgido en el último par de años. Varios de estos libros se enfocan a un curso tradicional de introducción a la ciencia de la computación con programación [5, 8, 11, 21, 24]. Otros son un poco más avanzados e incorporan temas de estructuras de datos [17, 22, 30]. Ha habido también algunos libros con una orientación hacia la ciencia e ingeniería [15, 18] y otros más hacia dominios específicos como robótica [16], juegos [33] y multimedios [12]. Y todo esto es sin contar las decenas de libros que hay sobre Python dirigidos a gente que ya tiene cierta experiencia programando.

3.3 La experiencia de Georgia Tech

En *Georgia Tech* se ha utilizado Python en el curso titulado *Introduction to Media Computation*. Este es un primer curso de programación que tiene un enfoque hacia la escritura de programas que manipulan imágenes, sonido y video. Los resultados se han documentado en [29, 31] e incluso se ha escrito un libro de texto [12]. En particular, el curso ha sido muy exitoso como medio para atraer a más mujeres a la ciencia de la computación. Se ha desarrollado una versión del mismo curso utilizando el lenguaje de programación Java. Sin embargo, Python se considera una mejor opción ya que permite hacer mucho antes cosas más interesantes gracias a su sencillez sintáctica.

3.4 La experiencia del MIT

Durante muchos años, el *Massachusetts Institute of Technology* (MIT) utilizó el lenguaje Scheme en su curso 6.001 de introducción a la programación ofrecido por el departamento de Ingeniería eléctrica y ciencia de la computación. De este curso incluso surgió el legendario *Wizzard Book* [1]. En el 2006, la institución tomó la controvertida decisión de comenzar a usar Python en lugar de Scheme en este curso [6]. La razón del cambio obedeció a la necesidad de actualizar los planes de estudio con el fin de incorporar temas que resultaran más relevantes a las exigencias ingenieriles de la época actual. Específicamente se optó por usar el lenguaje Python debido a que ofrecía un esquema adecuado para la programación de robots y aplicaciones móviles [34].

3.5 La experiencia del Campus Estado de México[7]

3.5.1 Antecedentes

En el semestre enero-mayo del 2008 se inició en el Campus Estado de México (CEM) un grupo piloto que utilizó Python en la clase de *Fundamentos de programación*. Esta iniciativa nació por la preocupación en el porcentaje de reprobación de dicha materia y los niveles de deserción (cambios de carrera) en los primeros semestres de las carreras de computación, además del impacto indirecto que esto tiene en la captación de nuevos alumnos para las carreras de TI.

En el proceso de diseño de este grupo piloto se evaluó la propuesta que ofrecía el *Institute for Personal Robots in Education* (IPRE) [14]. Este organismo tiene como misión: "Aplicar y evaluar el uso de robots como un contexto para la enseñanza de la ciencia de la computación". El IPRE es un esfuerzo conjunto entre *Georgia Tech* y el *Bryn Mawr College*, además de contar con el patrocinio de Microsoft Research. *Georgia Tech* se destaca entre las universidades norteamericanas por las estrategias que ha desarrollado para cambiar la educación de la computación (*Georgia Computes!*) en los niveles básicos de educación (K-12). Su objetivo es ampliar la participación de los jóvenes en las carreras de computación, así como plantear la importancia de contextualizar la enseñanza de la programación en el nivel universitario, esto es, ubicar la programación (actividades, tareas, proyectos) en un ambiente conocido, relativo al campo de su carrera y atractivo para el alumno.

La propuesta desarrollada por el IPRE consiste en utilizar un paquete que contiene un libro de texto [16] con la estrategia didáctica, contenidos temáticos y ejercicios, además de un robot *Scribbler* y una tarjeta que le añade al robot funcionalidad extra como la capacidad de comunicación bluetooth, tomar fotografías y contar con sensores adicionales. El lenguaje en el que se programa el robot es Python, apoyado con bibliotecas

desarrolladas para usar el robot y un simulador. El simulador permite aprender a programar usando el contexto del robot, aún en el caso de no contar físicamente con los robots.

La iniciativa del IPRE ha sido adoptado por diversas instituciones a nivel mundial [13]. El CEM presentó su experiencia en la implantación de dicha iniciativa en la XVIII Reunión Nacional de Directores de Escuelas y Facultades de Informática y Computación ANIEI 2009 [3] en la ciudad de Nuevo Vallarta, Nayarit, ante profesores y directivos de instituciones educativas de nivel superior públicas y privadas del país. A raíz de esta presentación, algunas de estas universidades comenzaron a implantar también la propuesta del IPRE.

3.5.2 Implantación y resultados

Dado que el programa oficial de la materia de *Fundamentos de programación* en el Tecnológico de Monterrey está enfocado al lenguaje de programación Java, se decidió utilizar en el primer parcial la propuesta del IPRE para introducir a la lógica de programación y en el resto del semestre continuar con el lenguaje Java.

Dentro de los resultados obtenidos a la fecha utilizando Python (cinco semestres en total) se ha observado que los alumnos ahora enfocan su esfuerzo a resolver los problemas; la sintaxis del lenguaje ya no es un obstáculo que los distrae del objetivo principal. Utilizando Python en el primer parcial, los alumnos se familiarizan con los siguientes conceptos: variables, literales, operadores, expresiones, funciones, parámetros y estructuras de control (`if`, `for` y `while`). Estos conceptos facilitan continuar después con el paradigma orientado a objetos. En el CEM se ha probado, además de usar Python con el contexto de robots, el enfoque de multimedios y el desarrollo de juegos sencillos con la biblioteca PyGame [25].

La implementación de estas estrategias, donde Python se ha vuelto un jugador central, ha permitido desarrollar actividades que favorecen el aprendizaje activo y comprometen más al alumno con las tareas, sin demeritar la exigencia académica de la materia de *Fundamentos de programación*. Un ejemplo de las actividades de la clase es:

> Escribe una función llamada `cantar`. Ésta debe hacer que el robot toque una melodía. La función debe recibir un parámetro que indique la duración de una nota negra.

En el semestre de enero-mayo del 2009 se realizó una encuesta a los alumnos de uno de los grupos piloto para medir el grado de satisfacción y cambios de percepción hacia la computación [20]. Los resultados fueron bastante positivos. Los alumnos manifestaron sentirse más seguros sobre su capacidad para resolver problemas. Así mismo, consideraron que la programación de robots con Python los había entusiasmado a tal grado que incluso llegaron a dedicar tiempo adicional por gusto propio a ciertas actividades. Finalmente, se les preguntó si habían disfrutado la clase. El 100% de los alumnos respondió que estaban "de acuerdo" o "completamente de acuerdo".

4 Obstáculos percibidos

A menudo se presentan obstáculos al momento en que una institución educativa intenta utilizar Python en su primer curso de programación [36]. A continuación se discute dicha problemática.

4.1 Ausencia de detección de errores a tiempo de compilación

Normalmente los programas de Python son ejecutados por un intérprete, aunque también existe la posibilidad de compilarlos. Algunos de los errores que ocurren en Java o C++ (por ejemplo la omisión de un punto y coma) simplemente desaparecen al momento de usar un lenguaje más sencillo. Los errores sintácticos que aún pudieran estar presentes en un programa en Python son detectados cuando éste se carga en memoria. Los errores semánticos, como es el caso de incompatibilidad de tipos, son detectados a tiempo de ejecución, y no a tiempo de compilación, como ocurre con C++ y Java. Esto podría verse como una fuerte desventaja por parte de Python, ya que hay un principio en ingeniería de software que establece que se debe hacer todo lo posible por detectar y corregir errores en las fases más tempranas. Sin embargo, el problema no es tan crítico como pudiera parecer inicialmente.

El desarrollo típico de programas en Python involucra usar un ciclo de dos fases: edición e interpretación. Por otro lado, en lenguajes que usan un compilador el ciclo es de tres fases: edición, compilación y ejecución. En principio, menos fases significa menos complejidad accidental. Aunque el uso de un IDE (ambiente integrado de desarrollo) podría dar la ilusión de que las fases de compilación y ejecución son una sola, la realidad es que no se puede correr un programa sino hasta que se le hayan eliminado todos los errores de compilación.

Al momento de aprender a programar, el ciclo de interpretación ofrece algunas ventajas importantes. Por ejemplo, al correr un programa, los errores se van reportando uno a la vez. El alumno no tiene que confrontar un listado con N errores de compilación que ni siquiera caben en la pantalla. Cuando un programa es interpretado es posible correrlo y obtener al menos algunos resultados parciales aún ante la presencia de errores. Por otro lado, una desventaja del ciclo de compilación es que éste puede brindar una falsa sensación de seguridad. No es raro toparse con un alumno que cuando escribe un programa mide su avance a partir del número de errores de compilación que le faltan corregir. Desafortunadamente cree que cuando haya eliminado todos esos errores su programa automáticamente estará correcto. Aún no alcanza a distinguir que hay diferentes tipos de errores. Mucho menos sabe que los errores de compilación son los más fáciles de corregir, mientras que los difíciles son los errores de lógica y diseño.

4.2 Desempeño pobre a tiempo de ejecución

Python, por ser un lenguaje normalmente interpretado, tiene una velocidad de ejecución menor (alrededor de un factor de 10) que los programas generados a partir de un lenguaje compilado. Sin embargo esto no es problema en un primer curso de programación, ya que los alumnos no estarán escribiendo código de producción. Normalmente sus programas serán ejecutados sólo unas cuantas veces, por lo que es irrelevante en la práctica si el programa tarda 0.2 o 2.0 segundos en correr. Lo que sí es muy importante es la cantidad de tiempo en que se tardan en escribir un programa. Python definitivamente lleva las de ganar en este aspecto gracias a la rapidez de su ciclo de edición/interpretación, la ausencia de declaraciones y a las abstracciones de alto nivel que soporta.

4.3 Los alumnos requieren aprender un lenguaje comercial

Aquí vale la pena hacer una distinción entre nuestros estudiantes. Algunos de ellos estudian una carrera en el área de tecnologías de información y electrónica (TIE). Ellos sí necesitan conocer varios lenguajes comerciales (como C/C++, C# o Java), pero no necesitan aprenderlos en el primer curso. Ya llevarán más cursos en semestres posteriores y podrán cubrir ahí esta necesidad. Tal como lo han planteado diversos autores [2, 10, 28, 29], Python resulta un excelente lenguaje para introducirse a la ciencia de la computación. Cuando este autor ha introducido Python a alumnos de semestres superiores, la reacción más común ha sido: "¿Por qué no nos enseñaron esto desde el principio?".

Otro tipo de alumnos que estarían tomando un primer curso de programación son los de las carreras de ingeniería, ciencias y medios digitales. En principio ellos no se dedicarán a programar profesionalmente, por lo que no es realmente indispensable que aprendan un lenguaje comercial. Lo que sí necesitan es desarrollar su capacidad para resolver problemas usando la computadora. Lo que tenemos que hacer aquí es brindarles las mejores herramientas para su desarrollo académico y profesional. Hay evidencia anecdótica documentada en [9] que sustenta la idea de que enseñarles Python a este grupo de alumnos les permite resolver problemas fuera de clase que difícilmente hubieran intentado con un lenguaje como C++.

Aunque Python no es tan popular como C++ o Java, sí tiene usos importantes en la industria. En Python se han escrito aplicaciones como BitTorrent, Ubuntu Software Center, YUM, GNU Mailman, Plone, Django y Zope. Así mismo, Python se usa como lenguaje de *script* en aplicaciones como Maya, Nuke, GIMP y Blender [35]. El lenguaje Python está siendo utilizado para fines comerciales por las siguientes compañías: Google, Autodesk, NASA, reddit, Yahoo!, YouTube e Industrial Light & Magic [27]. A la luz de la evidencia, podemos llegar a la conclusión de que Python también puede considerarse un lenguaje comercial.

4.4 Falta de familiaridad

Algunos docentes podrían objetar al uso de Python como primer lenguaje de programación debido a que les resulta poco familiar. Muchos lenguajes modernos (por ejemplo C++, C#, D, Objective-C, Java y JavaScript) tienen una notación sintáctica derivada del lenguaje C. No es así en el caso de Python. Pero esto no es un problema real. La sintaxis y semántica de Python es muy sencilla, y se puede aprender en cuestión de días o incluso horas por alguien que ya tiene experiencia programando en lenguajes convencionales [36].

Tal como se puede observar en el ejemplo de la subsección 2.2.2, los programas de Python no son tan diferentes a los de Java, específicamente en cuestión de estructuras de control y operadores, los cuales constituyen la esencia misma de cualquier algoritmo. Esto es importante ya que alguien que aprende a programar con Python no tendrá mucho problema en aprender después otros lenguajes como Java o C++ [9].

5 Conclusiones

Python es una de las mejores opciones disponibles para enseñar a programar a estudiantes de las áreas de ingeniería, computación y medios digitales. Para un primer curso de programación, este lenguaje ofrece ventajas importantes sobre otros lenguajes. Gracias a la simplicidad sintáctica y semántica que ofrece Python, los alumnos pueden dedicar más tiempo a la resolución de problemas.

En la actualidad, un número creciente de instituciones de educación superior alrededor del mundo están usando Python y obteniendo resultados muy interesantes. Han aparecido también varios libros de texto que usan Python para enseñar a programar con diferentes enfoques; el problema principal es que todos están en inglés.

Estamos en este momento ante una gran oportunidad, similar a la que tuvimos a inicios de los años ochenta cuando el Tecnológico de Monterrey fue pionero en México al utilizar el lenguaje Pascal en sus cursos de introducción a la programación. Cambiar de herramienta y usar un nuevo lenguaje de programación involucra salirnos de nuestra zona de confort. Definitivamente no es algo fácil. Requiere de esfuerzo y dedicación, pero es algo que tenemos que hacer si deseamos mantener el liderazgo educativo en nuestro país. Debemos buscar la forma de servir mejor a nuestros alumnos y trascender positivamente en su formación académica.

6 Agradecimientos

Se agradece el apoyo moral, así como las observaciones y contribuciones realizadas durante la elaboración de este documento por parte de Alma Patricia Chávez Cervantes y Humberto Cárdenas Anaya del Campus Estado de México, y Román Martínez Martínez del Campus Monterrey.

Notas

[1] Compilador utilizado: Mono C# 2.4.4.0.

[2] Todos los programas de este documento fueron escritos y probados en Python versión 3.1.1.

[3] El comportamiento descrito sólo aplica para Python 3.x. En Python 2.x se debe utilizar la función `xrange` para lograr el mismo efecto.

[4] Las versiones de Python 2.x tienen el mismo problema que tiene C en cuanto al uso del operador de división. Los operadores `/` y `//` que se describen aquí corresponden a la versión de Python 3.0 y posterior.

[5] Dichos autores de lenguajes son: Seymour Papert (creador de Logo), Niklaus Wirth (creador de Pascal), Guido van Rossum (creador de Python), y Bertrand Meyer (creador de Eiffel).

[6] Los lenguajes que se usaron para el estudio fueron: C, C++, Eiffel, Haskell, Java, JavaScript, Logo, Pascal, Python, Scheme y VisualBasic.

[7] Esta sección fue elaborada por el profesor Roberto Martínez Román (rmroman@itesm.mx) del Campus Estado de México.

Referencias

[1] Abelson, H., Sussman, G. J., and Sussman, J. *Structure and Interpretation of Computer Programs*, 2nd ed. The MIT Press, 1996. Libro electrónico disponible de manera gratuita en: http://mitpress.mit.edu/sicp/.

[2] Agarwal, K. K., and Agarwal, A. Python for CS1, CS2 and Beyond. In *The Journal of Computing in Small Colleges* (April 2005), vol. 20, pp. 262–270.

[3] Asociación Nacional de Instituciones de Educación en Tecnologías de Información, A.C. *Sitio web de la ANIEI*. http://aniei.org.mx/ Accedido el 29 de junio del 2010.

[4] Brooks, F. P. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer* (April 1987), pp. 10–19.

[5] Campbell, J., Gries, P., Montojo, J., and Wilson, G. *Practical Programming: An Introduction to Computer Science Using Python*. Pragmatic Bookshelf, 2009.

[6] Daher, W. S. *EECS Revamps Course Structure*. http://tech.mit.edu/V125/N65/coursevi.html Accedido el 29 de junio del 2010.

[7] Donaldson, T. *Python as a First Programming Language for Everyone*. http://www.cs.ubc.ca/wccce/Program03/papers/Toby.html Accedido el 29 de junio del 2010.

[8] Downey, A. B. *Python for Software Design: How to Think Like a Computer Scientist*. Cambridge University Press, 2009.

[9] Enbody, R. J., and Punch, W. F. Performance of Python CS1 Students in Mid-level non-Python CS Courses. In *SIGCSE '10: Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (March 2010), pp. 520–523.

[10] Enbody, R. J., Punch, W. F., and McCullen, M. Python CS1 as Preparation for C++ CS2. In *SIGCSE '09: Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (March 2009), pp. 116–120.

[11] Gaddis, T. *Starting Out with Python*. Addison Wesley, 2008.

[12] Guzdial, M. J., and Ericson, B. *Introduction to Computing and Programming in Python, A Multimedia Approach*, 2nd ed. Prentice Hall, 2009.

[13] Institute for Personal Robots in Education. *Adopters and Testers*. http://www.roboteducation.org/schools.html Accedido el 29 de junio del 2010.

[14] Institute for Personal Robots in Education. *IPRE Website*. http://www.roboteducation.org/ Accedido el 29 de junio del 2010.

[15] Kiusalaas, J. *Numerical Methods in Engineering with Python*, 2nd ed. Cambridge University Press, 2010.

[16] Kumar, D., Ed. *Learning Computing with Robots*. Institute for Personal Robots in Education, 2008. Libro electrónico disponible de manera gratuita en: http://wiki.roboteducation.org/.

[17] Lambert, K. A. *Fundamentals of Python: From First Programs through Data Structures*. Course Technology, 2009.

[18] Langtangen, H. P. *A Primer on Scientific Programming with Python*. Springer, 2009.

[19] Manilla, L., and de Raadt, M. An Objective Comparison of Languages for Teaching Introductory Programming. In *Proceedings of the Sixth Koli Calling Baltic Sea Conference* (November 2006), pp. 9–12.

[20] Martínez, R. *Caso ITESM CEM – Python*. Reporte interno disponible a solicitud del interesado.

[21] Miller, B., and Ranum, D. *Python Programming in Context*. Jones & Bartlett Publishers, 2008.

[22] Miller, B. N., and Ranum, D. L. *Problem Solving With Algorithms And Data Structures Using Python*. Franklin Beedle & Associates, 2005.

[23] Ortiz, A., Marcos, J., Pérez, P. O., Mejorado, A. J., and Hernández, S. F. *Plan analítico del curso de Programación*. Borrador disponible en https://docs.google.com/Doc?docid=0AafT7zP7ykhIZHFucndjaF8yMWNyYzNqNGNi&hl=en Accedido el 29 de junio del 2010.

[24] Punch, W. F., and Enbody, R. J. *The Practice of Computing using Python*. Addison Wesley, 2010.

[25] Pygame Community. *Pygame Website*. http://www.pygame.org/ Accedido el 29 de junio del 2010.

[26] Python Software Foundation. *Python Programming Language – Official Website*. http://www.python.org/ Accedido el 29 de junio del 2010.

[27] Python Software Foundation. *Quotes about Python*. http://www.python.org/about/quotes/ Accedido el 29 de junio del 2010.

[28] Radenski, A. Python First: A Lab-Based Digital Introduction to Computer Science. In *ITiCSE'06: Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (June 2006), pp. 197–201.

[29] Ranum, D., Miller, B., Zelle, J., and Guzdial, M. Successful Approaches to Teaching Introductory Computer Science Courses with Python. In *SIGCSE '06: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (March 2006), pp. 396–397.

[30] Reed, D. M., and Zelle, J. *Data Structures and Algorithms: Using Python and C++*. Franklin Beedle & Associates, 2009.

[31] Rich, L., Perry, H., and Guzdial, M. A CS1 Course Designed to Address Interests of Women. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (March 2004), pp. 190–194.

[32] Spolsky, J. *The Law of Leaky Abstractions*. http://www.joelonsoftware.com/articles/LeakyAbstractions.html Accedido el 29 de junio del 2010.

[33] Sweigart, A. *Invent Your Own Computer Games with Python*, 2nd ed. Albert\Sweigart, 2010. Libro electrónico disponible de manera gratuita en: http://inventwithpython.com/.

[34] Weinreb, D. *Why Did MIT Switch from Scheme to Python?* http://danweinreb.org/blog/why-did-mit-switch-from-scheme-to-python Accedido el 29 de junio del 2010.

[35] Wikipedia. *List of Python Software*. http://en.wikipedia.org/wiki/List_of_Python_software Accedido el 29 de junio del 2010.

[36] Zelle, J. M. *Python as a First Language*. http://mcsp.wartburg.edu/zelle/python/python-first.html Accedido el 29 de junio del 2010.

[37] Zelle, J. M. *Python Programming: An Introduction to Computer Science*. Franklin Beedle & Associates, 2003.