Análisis de nuevos lenguajes para la enseñanza de programación imperativa en los primeros años de las carreras de Informática de la Universidad Nacional del Noroeste de la Provincia de Buenos Aires

Germán L. Osella Massa[1], Claudia Russo[1], Mónica Sarobe[1], Sabrina Pompei[1]

[1] Universidad Nacional del Noroeste de la Provincia de Buenos Aires (UNNOBA), Roque Saenz Peña 456 (Sede Junín), Argentina.
german.osella@nexo.unnoba.edu.ar, {crusso,monicasarobe,sabrinapompei}@unnoba.edu.ar

Resumen: A partir de la necesidad de actualizar el lenguaje y las herramientas utilizadas para el dictado de las materias relacionadas con la programación imperativa (pertenecientes a los primeros años de las carreras de Informática) se realizó una investigación de tipo descriptiva transversal evaluando lenguajes de programación alternativos. El universo de estudio estuvo constituido en total por doce lenguajes, de distinto tipo y características. A partir de las conclusiones obtenidas, se elaboró una propuesta metodológica de integración de contenidos. Los profesores valoraron la propuesta como muy adecuada y se recomendó su puesta en marcha para el próximo año lectivo.

Palabras clave: Programación, Lenguajes, Metodologías, Actualización.

1 Introducción

Simultáneamente desde la Escuela de Tecnología de la UNNOBA y desde las cátedras a cargo de la enseñanza de programación imperativa en asignaturas del primer año de las carreras de Informática ha surgido la necesidad de replantear la metodología que se venía empleando. El uso de PASCAL como lenguaje de base para la enseñanza de conceptos fundamentales ha traído en los últimos años un creciente número de inconvenientes que se desea subsanar. Algunos de ellos están asociados a dificultades técnicas (compiladores para plataformas obsoletas, falta de herramientas de apoyo modernas, etc.) mientras que otros están asociados a la percepción que los propios alumnos tienen del lenguaje (cuya resolución es más compleja). En razón de esto, se decidió estudiar y comparar un conjunto de lenguajes de programación, buscando extraer las virtudes y debilidades de cada uno en lo que respecta a la enseñanza de la programación imperativa. Los lenguajes estudiados fueron: C, C++, C#, D, Go, Java, Javascript, Objective-C, Pascal, Python, Ruby y Scala.

En el resto del artículo, se mencionarán las razones que llevaron a descartar la mayoría de los lenguajes evaluados como herramientas para enseñar programación imperativa y se detallará la nueva metodología diseñada a partir de los lenguajes elegidos. Finalmente, se describirá los resultados esperados tras el cambio.

2 Lenguajes analizados y descartados

Los lenguajes analizados poseen características muy diferentes: algunos son procedurales, otros orientados a objetos e incluso multiparadigma. Pueden separarse en interpretados y compilados. Algunos son muy recientes (Go se anunció en el 2009) mientras que otros han marcado la historia (como es el caso de C). En el resto de esta sección se explicará que llevó a descartar a muchos de los lenguajes vistos.

2.1 Pascal

Si bien Pascal es un lenguaje estructurado pensado para enseñar los conceptos básicos de la programación imperativa, lamentablemente ha ido caído en desuso (tanto en la industria como en la academia), trayendo como consecuencia la falta de herramientas de apoyo modernas (entornos de desarrollado y depuradores) y de comunidades activas que lo promuevan. Delphi [12] y FreePascal [13] son excepciones que han encontrado su nicho y aún continúan desarrollándose. Delphi, sin embargo, al tratarse de un producto comercial, impone ciertas restricciones que limitan su atractivo: no es multiplataforma, requiere hardware relativamente moderno para ser usable y tiene asociado un costo por cada licencia de uso. FreePascal, por su parte, no sufre de estos inconvenientes y el proyecto Lazarus, que busca implementar un ambiente similar a Delphi, podría ser un entorno de desarrollo conveniente para que los alumnos utilicen. No obstante, la percepción por parte de los alumnos (y el mensaje trasmitido a ellos por alumnos más avanzados de la carrera) es que nunca usarán Pascal en su desarrollo profesional. Esto causa desmotivación y falta de interés en el lenguaje, que va en detrimento de los conceptos fundamentales que quieren enseñarse. El uso de un lenguaje que resulte más atractivo para el alumno, ya sea por su aplicación comercial o por su "popularidad", puede fomentar el interés por aprenderlo, traduciéndose en una mejor asimilación de los tópicos a enseñar.

2.2 C

C [1] es un lenguaje relativamente pequeño y goza de una muy buena popularidad en cuando a lenguaje de sistema se refiere. Permite aplicar muchos de los contenidos que se desea enseñar en programación imperativa aunque muchas veces resulta ser de muy bajo nivel, causando que los conceptos se pierdan entre las particularidades del lenguaje. Para mencionar un ejemplo sencillo, una simple concatenación de cadenas de caracteres se transforma en una tarea sumamente compleja y propensa a errores, involucrando manejo de punteros, asignación dinámica de memoria o comprobaciones de dimensiones para evitar desbordamientos de memoria. C obliga a entender claramente el manejo de la memoria tanto por parte del sistema operativo como del programa, siendo propenso a que errores difíciles de detectar causen la terminación inesperadamente del programa. Por ese motivo se descartó a C como lenguaje, siendo probablemente un lenguaje más apropiado para ir aprendiendo a la par de un lenguaje de ensamblador (en donde la distancia entre ambos sería relativamente pequeña y los conceptos manejados estarían más relacionados).

2.3 Objective-C

Objective-C [11] es un lenguaje popular en las plataformas impulsadas por la empresa Apple: Es uno de los lenguajes de base en Mac OS X (Macs) y iOS (iPhone, iPad y iPod). El lenguaje es un híbrido entre C y Smalltalk. Aumenta al lenguaje C agregando clases y objetos usando el modelo propuesto en Smalltalk, empleando una sintaxis similar (aunque permite usar una notación similar a C como syntactic sugar). En el siguiente código puede verse fácilmente que parte corresponde a lo que es el lenguaje C y que parte a Smalltalk, ya que lo segundo se escribe entre corchetes:

```
 1 #import <stdio.h>
 2 #import "Fraction.h"
 3
 4 int main( int argc, const char *argv[] )
 5 {
 6    Fraction *frac = [[Fraction alloc] init];
 7    [frac setNumerator: 1 denominator: 3];
 8    printf("The fraction is: %d\n", [frac value]);
 9    [frac release];
10    return 0;
11 }
```

Lamentablemente, si no se necesitan incorporar los conceptos de programación orientada a objetos (sólo es de interés enseñar programación imperativa), lo que queda es básicamente el lenguaje C, que ya fue rechazado por los motivos explicados.

2.4 C++

C++ [2], como evolución del lenguaje C, fue considerado como el lenguaje ideal para enseñar a programar, ya que permite utilizar construcciones más avanzadas que simplifican muchas de las tareas que en C requerirían un esfuerzo importante. Por otra parte, todo lo incorporado a C++ es opcional: pueden definirse clases o no, puede usarse sobrecarga de operadores o no, pueden definirse funciones o clases genéricas o no, etc. La filosofía siempre fue que si algo no se usa, no se paga un costo adicional (en memoria o velocidad) por tener la posibilidad de usarlo. Sin embargo, C++ es un lenguaje enorme y no fue ideado para ser usado como primer lenguaje de programación. El hecho que C++ soporte casi todo el lenguaje C y además agregue conceptos como clases, operadores, sobrecarga, espacio de nombres, programación genérica, plantillas y más, lo convierten en un lenguaje monstruoso. Explicar el siguiente programa trivial a un alumno inexperto no es una tarea sencilla:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6  cout << “Hola, mundo!” << endl;
7   return 0;
8 }
```

La línea 1 requiere entender el uso de archivos de cabecera y del preprocesador (ambos heredados del lenguaje C). La línea 2 involucra el concepto de espacio de nombres. La línea 4 requiere explicar que es una función y el por qué es necesario devolver un `0` en la línea 7. La línea 6, involucra tantos conceptos que resulta difícil enumerarlos sin olvidar ninguno: `cout` es un objeto de la clase `std::ostream`, "`<<`" es un operador sobrecargado para trabajar con un "`char *`" y explicar el mecanismo detrás de "`endl`" le daría dolores de cabeza a más de uno. Obviamente, es posible aplicar la metodología de "*No se preocupe, lo va a entender más adelante. Por ahora, copie eso textualmente que va a funcionar*" pero como estrategia pedagógica no es la mejor, causando desconcierto y frustración en el alumno cuando algo no funciona de la forma esperada. Por todo esto es que C++ fue descartado.

2.5 Java y C#

Java [8] y C# [3] son dos lenguajes similares y de amplio uso. Ambos son, en principio multiplataforma, empleando una máquina virtual para abstraerse del sistema operativo sobre el que corren. En el caso de Java, Oracle provee máquinas virtuales que corren tanto sobre Windows, Linux, Solaris y OS X. En el caso de C#, Microsoft provee solamente para Windows la plataforma .NET (plataforma para la cuál C# compila) pero existe el proyecto Mono [4] que busca desarrollar una plataforma compatible con .NET (en la medida de lo posible) y provee versiones para Windows, Linux, Solaris y OS X, permitiendo también compilar para iOS y Android. El principal inconveniente que ambos lenguajes tienen es que solamente soportan el paradigma de la programación orientada a objetos y casi rechazan completamente el paradigma imperativo. Es imposible en ambos lenguajes definir una función independiente; sólo pueden definirse métodos en una clase. Esto conduce a una sobrecarga de estructuras innecesarias en un programa donde solamente se pretende enseñar programación imperativa, como puede verse en los siguientes ejemplos de un "Hola, mundo" escritos en Java y C# respectivamente.

Java

```
1 package hola;
2 public class HolaMundo {
3    public static void main(String[] args) {
4       System.out.println("¡Hola, mundo!");
5    }
6 }
```

C#

```
1 using System;
2 namespace hola {
3    public class HelloWorld {
4       public static void Main(string[] args) {
5          Console.WriteLine("¡Hola, mundo!");
6       }
7    }
8 }
```

La cantidad de conceptos involucrados como la definición de una clase, la de un método de la clase, modificadores de visibilidad para ese método y el uso de un

método de un objeto y de una clase para poder producir una salida nada aportan a la enseñanza de la programación imperativa. Por otra parte, Java carece de manejo de punteros y C# los soporta pero su uso es restringido. Por todo lo expuesto, ambos lenguajes fueron descartados para la enseñanza de la programación imperativa.

2.6 Scala

Scala [16] es un lenguaje diseñado para programar expresándose en una forma concisa, elegante y segura (segura desde el punto de vista de que se hace una fuerte comprobación de tipos al momento de compilar). Incorpora características de lenguajes orientados a objetos y funcionales y puede interoperar en forma transparente con Java. Sin embargo, el paradigma imperativo no es uno de sus fuertes, donde la manera más concisa para expresar un "Hola, mundo" es la siguiente:

```
1 object HolaMundo {
2   def main(args: Array[String]) {
3     println("Hola, mundo!")
4   }
5 }
```

Comparado con la versión de Java, la cantidad de conceptos presentes disminuye notablemente y a pesar de ser un código más compacto, es completamente equivalente. De hecho, si se tuviera que optar entre utilizar Java o Scala como lenguaje de base, este último sería una alternativa más que interesante. Lamentablemente, sigue siendo claramente un lenguaje orientado a objetos (donde es imposible definir una función fuera de un objeto o clase), haciendo imposible que se lo pueda hacer pasar por un lenguaje imperativo.

2.7 Go

Go [7] es un lenguaje relativamente nuevo (fue anunciado públicamente en el 2009) creado e impulsado por un grupo de desarrolladores de la empresa Google. Busca ser un lenguaje de sistema simple pero incorpora facilidades que típicamente no se incluían en lenguajes de sistema, como manejo automático de memoria (usando un garbage collector) y soporte para programación concurrente (introduciendo el concepto de goroutines). Se aparta de la sintaxis y las estructuras de control propuestas por el lenguaje C, proponiendo alternativas más simples y potentes. Soporta programación orientada a objetos pero no utiliza el concepto de clases y herencia común en otros lenguajes de programación. Ejemplo de un "Hola, mundo":

```
1 package main
2 import "fmt"
3
4 func main() {
5   fmt.Println("¡Hola, mundo!")
6 }
```

El sistema de tipos empleado en Go usa el concepto de interfaces, en donde si un tipo define un determinado conjunto de métodos, automáticamente ese tipo implementará (se lo desee o no) una determinada interfaz. Sin embargo, carece de tipos genéricos, que puede ser una limitante. Por otro lado, el soporte de herramientas de desarrollo es pobre y la bibliografía es prácticamente inexistente. Si bien hace poco el lenguaje alcanzó la versión 1.0, lo que marca cierta estabilidad, sigue siendo un lenguaje muy nuevo y poco maduro como para tomarlo como lenguaje de base.

2.8 Ruby

Ruby [15] es un lenguaje dinámico muy popular en lo que al desarrollo de sistemas web se refiere. Lamentablemente, posee ciertas características que no lo hacen elegible como lenguaje imperativo. Por un lado, es difícil ocultar el hecho que es orientado a objetos puro (por ejemplo, la manera sugerida para recorrer elemento a elemento una secuencia de datos es usando alguna de las variantes del método each, al que se le pasa un bloque que procesará cada elemento, de la misma forma en que se lo haría en Smalltalk aunque lejos está de ser la manera de hacerlo en un paradigma imperativo). También cuenta con muchas formas alternativas para escribir exactamente lo mismo (existe if y unless, while y until, todos además como estructura de control o como modificador de una sentencia). Ejemplos:

```
if condición then
   puts("Si")
end

if condición
   puts("Si")
end

puts("Si") if condición
```

```
unless condición then
   puts("No")
end

unless condición
   puts("No")
end

puts("No") unless condición
```

Todas las variantes presentadas en cada columna son equivalentes entre sí. Contar con un lenguaje así de expresivo le permite a un programador experto escribir código claro y natural. Sin embargo, a un programador novato sólo le causará confusión y falta de seguridad con respecto a cuál es la manera correcta de escribir código.

2.9 Javascript

Javascript [9, 10] es el lenguaje de facto en Internet. La mayoría de los navegadores actuales (sino todos) lo soportan. Sin embargo, posee características que no lo convierten en un lenguaje ideal para comenzar a programar. En general, aunque hay excepciones, carece se funciones de entrada/salida (como la intención fue que corriera en un navegador, poder leer y escribir archivos de manera arbitraria conllevaba un riesgo de seguridad innecesario). También sufre en cierta medida del mismo inconveniente que Ruby, en el sentido que es difícil ocultar el hecho que se está trabajando con objetos. El sistema de tipos cuenta con conversiones implícitas que más de una vez sorprenden inclusive a programadores avanzados en ese lenguaje.

El resultado de las siguientes expresiones es impredecible si no se conocen a fondo las reglas que rigen las conversiones entre tipos diferentes.

```
> 1 + 2
3

> 1 + "2"
"12"

> []
[]
```

```
> [] + 1
"1"

> [1, 2] + 3
"1,23"

> {}
{}
```

```
> {} + 2
2

> [] + []
""

> {} + []
0
```

```
> [] + {}
"[object Object]"

> {} + {}
NaN

[] es un arreglo vacío.
{} es un objeto literal
```

La mayor ventaja que tendría Javascript sobre otros lenguajes sería su ubicuidad y la aplicación comercial casi inmediata que podría dársele, no siendo ambas razones suficientes para elegirlo como lenguaje de base.

3 Nueva metodología propuesta

En el caso particular de la UNNOBA, el primer contacto que los alumnos tienen con la programación imperativa es durante el primer cuatrimestre de las carreras de Informática, en la materia denominada “Introducción a la Programación Imperativa” (IPI), continuando en “Programación Imperativa” (PI) durante el segundo cuatrimestre. A partir de esta división existente, se propone emplear lenguajes diferentes en cada asignatura, cada uno con un enfoque distinto.

Primero, se propone iniciar con un lenguaje de programación de muy alto nivel, que se interponga lo menos posible con los conceptos que se quieren enseñar al comenzar a programar (tipos de datos simples, estructuras de control, entrada/salida) y que además lleve a incorporar buenas costumbres al momento de escribir código (indentación clara, código documentado, uso de casos de pruebas). También, contar con una sesión interactiva para probar código y obtener una respuesta inmediata resultó ser una característica muy valorable: La interacción con un intérprete provee una respuesta inmediata, favoreciendo que el alumno experimente con el lenguaje y pueda formar rápidamente un modelo mental que explique las respuestas que éste arroja. Los lenguajes compilados requieren de un depurador para poder realizar un seguimiento paso a paso y del desarrollado de las habilidades necesarias para usarlo. Esto desanima bastante la exploración por parte del alumno. **Python** resultó ser el lenguaje que cumplió con todas estas condiciones y es el que se empleará para que el alumno adquirirá los conceptos fundamentales de la programación imperativa.

Por otra parte, resulta indispensable que el alumno esté expuesto a conceptos que, por su naturaleza, no se pueden expresar fácilmente en Python (o que es imposible hacerlo): Compilación en contrapartida a la interpretación del código, comprobación estática de tipos, punteros y asignación estática y dinámica de la memoria, por nombrar algunos. Para salvar este problema, se recurrió a un segundo lenguaje de programación, en donde el alumno pueda experimentar el contacto con estos conceptos mencionados. En este caso la elección fue más difícil. Varios fueron los lenguajes evaluados que podían en mayor o menor medida cubrir los requerimientos

expuestos. Finalmente el lenguaje seleccionado para esto fue **D**, abarcando cómodamente con todo lo expuesto y más, como se explicará más adelante.

En las siguientes subsecciones se resaltarán las características de Python y D que los llevaron a ser elegidos.

3.1 Python

Python [14] es un lenguaje claro y minimalista, que busca expresar los conceptos preferiblemente de una única manera. Emplear la indentación para definir bloques fuerza a que el alumno incorpore la buena costumbre de escribir código claro, que se espera arrastre a otros lenguajes donde los bloques se definen usando delimitadores. Si bien Python es un lenguaje orientado a objetos, lo puede disimular muy bien, dando la apariencia de ser un lenguaje completamente imperativo. Además, al ser un lenguaje de muy alto nivel, la cantidad de conceptos que el alumno deberá manejar al comienzo serán pocos. Es un lenguaje fuertemente tipado, dinámico e interpretado. Permite correr una sesión interactiva con ayudas tales como documentación online y completado de código. El sistema de documentación en línea es aplicable tanto sobre una función o método propio del lenguaje como uno escrito por el alumno. Por último, cuenta con la facilidad para escribir casos de pruebas dentro de la documentación del código, denominado *doctests*, usando una sintaxis idéntica a la de la sesión interactiva. La siguiente función posee documentación y casos de pruebas contenidos dentro de la misma documentación.

```
>>> def factorial(n):
...     """
...     Calcula el factorial de n: n*(n-1)*(n-2)*...*2*1
...
...     >>> factorial(1)
...     1
...     >>> factorial(2)
...     2
...     >>> factorial(3)
...     6
...     >>> factorial(4)
...     24
...     """
...     if n > 1:
...         return n * factorial(n - 1)
...     else:
...         return 1
```

3.2 D

El lenguaje D [5, 6] es compatible con un amplio subconjunto del lenguaje C (sintaxis similar, tipos compatibles, las mismas estructuras de control) y, como C++, aumenta esa base incorporando los conceptos de programación orientada a objetos y programación genérica, pero lo hace de una manera más simple y organizada,

aprendiendo de los errores cometidos en C++. No intenta mantener una compatibilidad absoluta con C, proveyendo arreglos asociativos y cadenas de caracteres dinámicas como tipos primitivos, manejo de memoria automática (con garbage collection) y una comprobación de tipos más robusta que la provista por C.

D incorpora conceptos del paradigma de la programación orientada a objetos y del paradigma funcional pero no olvida su origen en el paradigma imperativo. Soporta aserciones y casos de pruebas en forma nativa, de esta forma:

```
 1 int add(int x, int y) { return x + y; }
 2
 3 unittest
 4 {
 5     assert( add(1, 2) == 3 );
 6     assert( add(-7, 3) == -4 );
 7 }
```

El código dentro del bloque unittest se ejecutará solamente en un modo especial de prueba y será ignorado en una ejecución normal. Con esta característica, es fácil incorporar, al igual que en Python, casos de pruebas para el código escrito, fomentando la costumbre de probar los programas de esta manera. Otra característica asociada a ésta son las precondiciones y postcondiciones de una función:

```
 1 double func(double x)
 2 in {
 3     assert( x > 0);
 4 }
 5 out(resultado) {
 6     assert( resultado >=0 && resultado <= 1 );
 7 }
 8 body {
 9     // implementación de func...
10 }
```

Dada la función `func` que recibe un flotante `x` y devuelve otro flotante, el código dentro del bloque `in` verificará que el valor del parámetro `x` sera siempre mayor que 0 (o fallará en caso que no lo sea), luego se ejecutará el cuerpo de `func` y finalmente, en el bloque `out`, el valor retornado por la función (recibido en `resultado` en este caso) se verificará si se encuentra dentro del intervalo [0, 1].

Una problemática que generalmente aparece cuando se quiere implementar un TAD (una lista enlazada simple o una pila, por ejemplo) es el tipo de los valor que se manipularán. En el caso de las listas enlazadas, si se escribe código donde el valor en cada nodo es un entero, no será posible usarlo para crear una lista de cadenas o flotantes o lo que fuera. En D, esto se resuelve mediante tipos genéricos (existen mecanismos similares en C++, Java o C#). Una definición genérica para un nodo de una lista enlazada simple puede escribirse de la siguiente forma:

```
1 struct Nodo(T) {
2     T valor;
3     Nodo *siguiente;
4 }
```

La estructura ahora posee un tipo paramétrico `T`, pudiendo ser instanciada así:

```
Nodo!int n1;        Nodo!double n2;        Nodo!string n3;
```

Las tres definiciones anteriores producirán tres estructuras (`n1`, `n2` y `n3`), cada una especializada para el tipo indicado luego del signo de exclamación. D posee más características interesantes que no se mencionarán por una cuestión de brevedad.

4 Resultados esperados

La nueva metodología a implementarse durante el año 2013 espera motivar a los alumnos a interesarse más por los contenidos de las asignaturas IPI y PI, trayendo un aire renovado en la forma que se aplican los conocimientos impartidos. En particular, Python es un lenguaje popular hoy en día, con muy variadas aplicaciones que van desde la web hasta aplicaciones de escritorio o herramientas del sistema. Por otro lado, se disponen de herramientas modernas que se espera le simplificarán al alumno asimilar los temas tratados en las materias. En particular, se espera que la herramienta Online Python Tutor [17] sea junto con el entorno interactivo de Python dos ayudas invaluables para entender el flujo de control de un programa.

Por su parte, D servirá para aplicar conceptos que anteriormente se describían en papel o se imponían restricciones artificiales para aplicarlos. En Estructuras de Datos, asignatura del segundo año (correlativa de IPI y PI) que se verá afectada por estos cambios, se planea usar D como lenguaje de base en donde implementar en una forma realmente genérica las estructuras y algoritmos que se ven a lo largo de la materia.

Referencias

1. Kernighan, B., Ritchie, D.: The C Programming Language. Prentice Hall; 2nd edition (1988)
2. Stroustrup, B.: The C++ Programming Language. Addison-Wesley Professional; 3er ed. (2000)
3. C# Language Specification, http://go.microsoft.com/fwlink/?LinkId=199552
4. Mono: Cross platform, open source .NET development framework, http://www.mono-project.com/
5. D Programming Language – Official Website, http://dlang.org/
6. Alexandrescu, A.: The D Programming Language. Addison-Wesley Professional; 1st ed. (2010)
7. The Go Programming Language – Official Website, http://golang.org/
8. Java official website at Oracle, http://www.oracle.com/technetwork/java/index.html
9. Flanagan, D.: JavaScript: The Definitive Guide. O'Reilly Media; 6th edition (2011)
10. Crockford, D.: JavaScript: The Good Parts. Yahoo Press; 1st edition (2008)
11. OS X Developer Library: The Objective-C Programming Language, http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/
12. Embarcadero Delphi XE2 - http://www.embarcadero.com/products/delphi/
13. Free Pascal – Advanced open source Pascal compiler for Pascal, http://www.freepascal.org/
14. Python Programming Language – Official Website, http://www.python.org/
15. Ruby Programming Language – Official Website, http://ruby-lang.org/
16. The Scala Programming Language – Official Website, http://www.scala-lang.org/
17. Online Python Tutor, http://www.onlinepythontutor.com/