Analysis of suitable languages to teach Procedural Programming at the Universidad Nacional del Noroeste de la provincia de Buenos Aires

Germán L. Osella Massa

Claudia Russo
Mónica Sarobe

Sabrina Pompei

Escuela de Tecnología – Instituto de Investigación y Transferencia de Tecnología, Universidad Nacional del Noroeste de la provincia de Buenos Aires (UNNOBA), Roque Sáenz Peña 456 (Junín), Argentina.

ABSTRACT
Based on the need to update the language and tools used to teach Procedural Programming during the first year of computer science careers, a cross descriptive research was conducted by assessing different programming languages. The analysis involved a total of twelve languages with different characteristics. A new methodology was proposed based on the conclusions obtained from this work. Teachers have considered this proposal highly appropriate and its implementation was recommended for the next year.
Keywords: Programming Languages, Comparison, Methodology.

1. INTRODUCTION
Both the Escuela de Tecnología at UNNOBA and the staff in charge of teaching Procedural Programming to first year students in the Computer Science courses saw the need to rethink the methodology that was being used. The use of PASCAL as an introductory language to teach fundamental concepts has given rise to an increasing number of difficulties that are needed to be solved. Some of these issues are related to technical difficulties (i.e., compilers and IDE for obsolete platforms, lack of modern support tools, etc.) whereas others are related to the way in which students see this language. Therefore, it was decided to conduct a study to compare a set of twelve programming languages with the aim of identifying their strengths and weaknesses as regards teaching procedural programming. In the rest of this article, the reasons that led to discard most of the languages assessed as a tool to teach procedural programming will be mention and the new methodology designed based on the chosen languages will be analyzed. Finally, the results expected from the change will be described.

2. ANALYZED AND DISCARDED LANGUAGES
The languages under analysis were C, C++, C#, D, Go, Java, Javascript, Objective-C, Pascal, Python, Ruby and Scala. They all have different characteristics: some of them are procedural languages, others are object-oriented and some are even multiparadigm ones. They can be broken down into interpreted and compiled languages. Some of them are quite new (Go was announced in 2009) whereas others have made history (such as C). In this section, the reasons why most of the languages were discarded will be discussed.

Pascal
Although Pascal is a structured language designed to teach the fundamentals of procedural programming, it unfortunately has fallen into disuse (both in the industry and in academia) and, as a result, there are no modern support tools (development environments and debuggers) or any active communities promoting it. Delphi [12] and FreePascal [13] are exceptions which have found their niches and are still being developed. Delphi, however, as it is a commercial product, has certain constraints that make it less attractive: it is not multiplatform, it requires relatively new hardware to be usable and there is an associated cost per use license. On the other hand, FreePascal does not present any of these inconveniences and the Lazarus project, which seeks to implement an environment similar to Delphi, could be a convenient development environment. However, students believe that they will never use Pascal and they receive the same message from senior students. This causes demotivation and lack of interest in the language thus having a negative impact on the teaching of the curricula. The use of a language that the students consider more attractive, either because of its commercial application or because of its popularity, can boost interest in learning it and, in consequence, all the topics involved in the courses.

C
C [1] is a relatively small and very popular system programming language. It supports mostly all of the procedural programming topics required, although sometimes it turns out to be a very low level language, causing concepts to become blurred by this. To mention an example, concatenating two strings of characters becomes an extremely complex and prone to error task, involving pointers, dynamic memory allocation and tests to avoid memory overflows. C makes it necessary to understand how memory is managed both by the operating system and the program. It allow to write code with errors that sometimes are very hard to detect and may cause the program to terminate unexpectedly. Therefore, C was discarded. It is probably more suitable to learn it along with an assembly language (where the distance between both of them would be relatively small and the concepts would be more related).

Objective-C
Objective-C [11] is a popular language on Apple platforms: It is one of the basic languages in Mac OS X (Macs) and iOS (iPhone, iPad and iPod). This language is a hybrid of C and Smalltalk. It expands C language adding classes and objects using the model proposed in Smalltalk, with a similar syntax. The difference between the code related to C and the code related to Smalltalk can be clearly seen in the following code since the latter is written in square brackets.

```
#import <stdio.h>
#import "Fraction.h"

int main( int argc, const char *argv[] )
{
  Fraction *f = [[Fraction alloc] init];
  [f setNumerator: 1 denominator: 3];
  printf("Fraction = %d\n", [f value]);
  [f release];
  return 0;
}
```

Unfortunately, as there is no need to teach the contents related to object oriented programming (the aim is to teach procedural programming only) what is left is basically C, which was rejected for the reasons discussed before.

C++

C++ [2], as an evolution of C, was considered the ideal language to teach programming, since it allows more advanced constructions to be used thus making it easier to carry out many of the tasks that would require considerable effort in C. Additionally, all the items added to C++ are optional: classes may be used or not, operators may be overloaded, and generic functions or classes may be defined but are not mandatory, etc. The philosophy has always been that if something is not used, no additional cost is paid (in memory or speed) for the possibility of using it. However, C++ is a huge language and it was not designed to be used as a first programming language. The fact that C++ supports C language almost entirely and also adds more concepts such as classes, operators, overload, name spaces, generic programming, templates and so on, makes it a monstrous language. Explaining the following trivial program to an inexperienced student is not a simple task:

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

The first line requires an understanding of header files and the preprocessor (both inherited from C). The second line involves the concept of name spaces. Line 4 requires to explain what a function is and why it is necessary to return 0 in line 7. Line 6 involves so many concepts that it is difficult to list them without leaving some of them out: `cout` is an instance of the class `std::ostream`, “<<” is an overloaded operator to work with “`char *`” and explaining the mechanism behind “`endl`” will be a nightmare for many people. One can obviously follow the “*Do not worry, you will understand it later*” and “*for now, just type it and it will work*” approach, but this is not the best teaching strategy since it causes confusion and frustration to students when something does not work as expected. For all this, C++ was discarded.

Java and C#

Java [8] and C# [3] are two very similar languages of widespread use. They are both, multiplatform, using a virtual machine to run on different operating systems. In the case of Java, Oracle provides virtual machines that runs on Windows, Linux, Solaris and OS X. In the case of C#, Microsoft provides the .NET platform (for which C# compiles) which is for Windows only, but there is a project called Mono [4], which seeks to develop a .NET compatible platform (if possible) and provides versions for Windows, Linux, Solaris and OS X, and even making it possible to compile for iOS and Android. The main disadvantage of these two languages is that they only support the object-oriented programming paradigm and they almost reject the procedural paradigm altogether. It is not possible to define an independent function in any of them; only methods can be defined in a class. This leads to an overload of unnecessary structures in a program where they only aim is to teach procedural programming, as seen in the following examples of "Hello world" programs written in Java and C# respectively.

Java

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

C#

```
using System;
namespace Hi {
 public class HelloWorld {
  public static void Main(string[] args)
  {
    Console.WriteLine("Hello, world!");
  }
 }
}
```

The definition of a class, a method for a class, visibility modifiers for that method and the use of an instance method and a class to be able to produce an output do not contribute anything to the teaching of procedural programming. In addition, Java lacks pointers and C# supports them but their use is restricted. As a result, both languages were discarded for the teaching of procedural programming.

Scala

Scala [16] is a language designed to program in a concise, elegant and secure way (it is secure because of its static typing). It incorporates certain features of object-oriented and functional languages and it can inter-operate with Java in a transparent way. However, the procedural paradigm is not one of its strong points. The most concise way to write a "Hello world" is as follows:

```
object HolaMundo {
  def main(args: Array[String]) {
    println("Hello, world!")
  }
}
```

As compared to the Java, the number of concepts decreases notably and although it is a more compact code, it is completely equivalent. In fact, if we were to choose between Java and Scala as a base language, the latter would be a very interesting option. Unfortunately, it is clearly an object-oriented language (where it is impossible to define a function other than an object or a class) making it impossible to make use of it as a procedural language.

Go
Go [7] is a relatively new language (announced in 2009) created and sponsored by a group of developers from Google. It aims at being a simple language but adds features such as automatic memory management (using garbage collector) and concurrent programming support (introducing the goroutines concept). It moves away from the syntax and control structures proposed by C, and it suggests simpler and more powerful alternatives. It supports object-oriented programming but does not use the classic class-based model that is common in other programming languages.

```
package main
import "fmt"

func main() {
  fmt.Println("Hello, world!")
}
```

The type system in Go uses the interface concept, where if a type defines a certain set of methods, such type will automatically implement a certain interface (whether or not that was desired). However, it lacks generic types, which can be a constraint. On the other hand, the development tool support is poor and there is practically no literature. Although this language has recently reached its 1.0 version, which marks a certain level of stability, it is still really new and not altogether mature to be used as a base language.

Ruby
Ruby [15] is a very popular dynamic language frequently used on web development. Unfortunately, it has certain features that do not make it eligible as a procedural language. Firstly, the fact that it is purely object-oriented cannot be overlooked (for example, the suggested way to iterate on a sequence is by using one of the variants of the "each" method, on which a block will process each of the elements in the sequence, just as in Smalltalk but far from the way in which this is done in a procedural paradigm). It also has many alternative ways of writing exactly the same thing (there is `if` and `unless`, `while` and `until`) all of them as control structures or statement modifier). Some examples of Ruby, showing in each column variants of exactly the same code are presented next. Having such an expressive language allows expert programmers to write a clear and natural code. However, inexperienced programmers will be confused and doubtful about the right way to write code.

```
if test then               unless test then
   puts("Yes")                puts("No")
end                        end


if test                    unless test
   puts("Yes")                puts("No")
end                        end


puts("Yes") if test        puts("No") unless test
```

Javascript
Javascript [9, 10] is the de facto language on the Internet. Most current browsers (if not all of them) support it. However, it has certain features that do not make it an ideal language to start programming. In general, although there are exceptions, it lacks input/output functions (as it is intended to be run on a browser, the ability to read and write files in an arbitrary manner entailed an unnecessary risk). It also shares one of the problems of Ruby; it is hard to disregard the fact that you are working with objects. The type system has implicit conversions that more than once catch programmers unprepared (expert and novices alike). The result of the following expressions is almost unpredictable if the rules governing conversions among different types are really well known.

```
> 1 + 2                 > 1 + "2"
3                       "12"
> []                    > [] + 1
[]                      "1"
> [1, 2] + 3            > {}
"1,23"                  {}
> {} + 2                > [] + []
2                       ""
> {} + []               > [] + {}
0                       "[object Object]"
> {} + {}
NaN
```

Where `[]` *is an empty array and* `{}` *is an object literal.*

The greatest advantage that JavaScript could offer would be its ubiquity and its almost immediate business application. However, these two reasons are not enough to choose it as a base language.

3. NEW METHODOLOGY PROPOSAL

At UNNOBA, the first contact that students have with procedural programming is during the first term of the first year in the Computer Sciences courses, when they take the subject "Introducción a la Programación Imperativa". They then take "Programación Imperativa" in the second term. Based on this division, our proposal is to use different languages in each subject, each of them with a different approach.
First, the idea is to start with a very high level programming language, which interferes as little as possible with the teaching objectives when starting to program (simple data types, clear control structures, basic input/output) and which can help students acquire good habits when writing code (consistent indentation, documented code, use of test cases). Having an interactive session to test code and obtain an immediate response also turned out to be a very valuable feature: Interaction with an interpreter provides an immediate response, helping the student to experience with language and be able to form a mental model to explain the obtained responses.
Compiled languages need a debugger to be able to carry out a step-by-step trace and the necessary skills to be able to use it. This discourages students from exploring it. **Python** turned out to be a language which met all these requirements and it will be used for students to learn the fundamentals of procedural programming. In addition, it is essential for students to be exposed to concepts that, due to their nature, cannot be easily stated or are simple impossible using Python: Compiled vs. interpreted language, static type checking, pointers and, static vs. dynamic memory management, just to mention a few. To solve this problem, we resorted to another programming language where the student can be in contact with these concepts. In this case, the decision was harder. Several languages that could meet these requirements to different extents were assessed. Finally, the language chosen was **D**, which clearly covers all these concepts and more.

The features that led us to choose Python and D are discussed in the following subsections.

Python

Python [14] is a clear and minimalist language which seeks to embrace the "*There should be one –and preferably only one– obvious way to do it*" motto. Using indentation to define blocks forces students to acquire the good habit of writing clear code. This habit is expected to be transferred to other languages where blocks are written using delimiters. Although Python is an object-oriented language, it can easily hide it and appear as a completely procedural language. In addition to being a very high-level language, at first, students will need to handle only a limited number of concepts. It is a strongly typed, dynamic and interpreted language. Interactive sessions can be run with helps such as online documentation and code completion. The online documentation system is applicable both on a built-in function of the language or on one written by the students. Finally, test cases, known as *doctests*, can be written in the documentation, using a syntax that is identical to that of the interactive session. The following function has documentation and test cases contained in the same documentation.

```
>>> def factorial(n):
...     """
...     The factorial function, n!
...
...     >>> factorial(1)
...     1
...     >>> factorial(2)
...     2
...     >>> factorial(3)
...     6
...     >>> factorial(4)
...     24
...     """
...     if n > 1:
...         return n * factorial(n - 1)
...     else:
...         return 1
```

D

D [5, 6] is compatible with a large subset of C (similar syntax, compatible types, same control structures) and just as C++, it increases C by incorporating object-oriented and generic programming, but it does this in a simpler and more organized way, learning from the mistakes made in C++. It does not try to be absolutely compatible with C, including associative maps and dynamic strings as primitive types, automatic memory management (with garbage collection) and a more robust type checking than the ones provided by C. D incorporates some of the concepts of the object-oriented and functional paradigms but it does not turn its back to its procedural programming roots. It supports assertions and test cases in a native manner, as follows:

```
int add(int x, int y) {
  return x + y;
}

unittest
{
    assert( add(1, 2) == 3 );
    assert( add(-7, 3) == -4 );
}
```

The code within the `unittest` block will be executed only in a special test mode and it will be ignored in normal execution. With this feature, it is easy to incorporate test cases for the written code, just as in Python, thus fostering the habit of testing programs this way. Another related feature is programming by contract, using preconditions and postconditions of a function:

```
double func(double x)
in {
    assert( x > 0);
}
out(result) {
    assert( result >=0 && result <= 1 );
}
body {
    // func implementation...
}
```

The function `func` receives the `x` parameter which is a floating point value, and returns another floating point value. The code within the `in` block will verify that the value of the `x` parameter is always higher than 0 (otherwise, it will fail), then the body of `func` will be executed and finally, in the `out` block, the value returned by the function (stored in `result`) will be verified to check whether it is within the interval [0, 1].

A problem that usually shows up when implementing a data structure (a linked list or a stack, for example) is the type of the values stored in this structures. In the case of linked lists, an implementation defining the type of the value stored in each node as `int` will not work to create a list of strings or floats or anything else. In D, this is solved using generic types (there are similar mechanisms in C++, Java or C#). A generic definition for a node of a simple linked list can be written as follows:

```
struct Node(T) {
   T    value;
   Node *next;
}
```

The structure now has a parametric type T, that can be instantiated as follows:

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

The three definitions above will result in three different structures (`n1`, `n2` and `n3`), each of them specialized with the type indicated after the exclamation mark. Functions and methods can also be written in this way, making possible to implement truly generic algorithms. D has other interesting features that are not discussed here for the sake of brevity.

4. EXPECTED RESULTS

The new methodology to be implemented in 2013 is expected to motivate students of the courses "Introducción a la Programación Imperativa" and "Programación Imperativa", bringing fresh air to the way in which knowledge is acquired. Python is a very popular language, with many practical uses ranging from the web to desktop applications or games. In addition, there are modern tools that are expected to make it easier for students to learn the contents taught in each subject.

In particular, Online Python Tutor [17] is expected to be together with Python's interactive shell an invaluable tool to understand the control flow of a program. D will be used to apply the concepts that were formerly described on paper or on a concrete language but with restrictions that were artificial imposed.
All these changes will also have an impact on "Estructuras de Datos", a second year course which to take it, requires the students to pass both "Introducción a la Programación Imperativa" and "Programación Imperativa". Plans are made to use D as the language where the structures and algorithms will be studied and implemented in a really generic manner.

5. REFERENCES

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. 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/

Received: December 2012. Accepted: February 2013.