Authors: Santiago Badia, Alberto F. Martin

1. A brief overview of programming languages and the elective modules: `R`, `Python`, `MATLAB` and `Mathematica` (*20 min*).

A programming language consists of instructions for computers. Like natural languages, programming languages are formal languages which conform to rules for syntax and semantics. They allow scientists and engineers to implement algorithms.

1.1. A taxonomy of programming languages: static versus dynamic languages

There are different ways to classify programming languages. In this brief introduction, we only consider a taxonomy of programming languages accordingly to the way data types of variables are handled. This taxonomy will allow us to motivate the programming languages that we have selected for SCI1022. Accordingly to this taxonomy, a language can be either *statically-typed* or *dynamically-typed*, or *static* or *dynamic*, respectively, for short. A *static* language compels the developer to declare the type of any variable. For example, if we want to declare integer variables in our `C` code we would write:

```
int age1;
int age2;
int diff;

age1 = 10;
age2 = 20;
diff = substract(age2,age1); // Function call that substracts age1 to age2
```

When one declares a function, e.g., `substract` in the previous code, the type of all arguments and the type of the output must also be explicitly declared:

```
int substract(int num1, int num2)
{
   int result;
   result = num1-num2;
   return result;
}
```

All this information is used by a **static compiler**. Once you have finished your code, you must compile it. Compilers transform the code written in a static language into machine code to create an executable program. Compilers can exploit different types of optimisation to generate highly efficient code. Thus, static languages are used in the industry and academy when performance is a requirement, e.g., when implementing computationally intensive algorithms. The most important static languages are C, C++ and Fortran; Fortran was the first commercially available programming language, developed from inception for mathematical computations, but its usage is continuously decaying.

However, the performance of static languages comes with a price. The learning curve is very steep and the productivity of developers can be very low. First, the compilation step can take a considerable amount of time, and second, the need to declare all types and interfaces makes the code development quite tedious.

On the other side of the spectrum, we have the *dynamic languages*. Dynamic languages (e.g., Python, R, MATLAB, Julia) often do not require type declaration and do not involve a pre-compilation step. Instead, the code is **interpreted** at run-time using an special program called **interpreter**. One could write the previous C code in Python as

```
age1 = 10
age2 = 20
def diff(num1,num2): return num1-num2
diff(age2,age1)
```

These languages are more expressive, drastically increasing the productivity of developers. The price to pay is a (in many cases unacceptable) performance hit that can easily be of the order of hundreds.

In this unit, we will focus on dynamic languages, since the performance hit is not an issue, in general, in undergraduate scientific projects and some scientific tasks. In any case, one can combine the productivity of dynamic languages with the performance of static languages by using pre-compiled external libraries (written in C, C++, or Fortran) for the computationally intensive kernels. One ubiquitous approach is to combine Python code with the C library NumPy for array computations. In order for this approach to work, Python code must involve vectorisation, e.g., operations are applied to whole arrays instead of individual entries.

The static vs. dynamic paradigms, code productivity and performance are serious issues to be considered when creating scientific software projects. There is no *the best* alternative, since it is very case-dependent. Analogously, it is hard to decide which is *the best* static or dynamic programming language, since it depends on the type of work to be performed. In fact, new programming languages are still being created nowadays to solve the drawbacks of existing approaches. Julia deserves special mention in the frame of scientific coding. It is a dynamic

language originated in 2011 at MIT that does not suffer the performance hit.

1.2. The four elective modules

For all these reasons, SCI1022 includes four different workshops on three dynamically-typed languages, namely, Python, MATLAB, R, and an untyped language, Mathematica. Below, we provide a description of the four different languages that would help students to pick the ones that better fit their requirements. You can do your own research, in order to decide which is the right language for you. There are also different indices that measure the *popularity* of programming languages, e.g., the TIOBE index. You can find the rankinghere.

1.2.1. Python

Python is one the most popular languages for scientific coding, especially for data science. It is widely used both in academia and industry because of its simplicity and flexibility. Python is an object-oriented language, and probably the best one for learning this programming paradigm. Python is general purpose and can be used in any scientific discipline, from computational mathematics and physics to data science. Python is a community-driven open source project, with thousands of freely available packages in almost any discipline.

1.2.2. MATLAB

MATLAB is a commercial numerical computing environment that is easy to learn and use. It is very popular at undergraduate university programmes, especially in mathematics and some engineering disciplines. It might be easier for beginners. However, it is a proprietary language and one needs a license to use it. The basic package comes with linear algebra, data processing and plotting tools. Extra functionality is provided by Mathworks through toolkits (at an extra cost). Monash has a MATLAB site-license that allows Monash students to install it on their personal computers.

1.2.3. R

R is an open source language and environment for statistical computing and graphics. R provides a wide variety of statistical and graphical techniques, and is highly extensible. It is widely used in data science and applications, e.g., in biology or chemistry. In the field of data science, one can find a freely available package for almost anything.

1.2.4. Mathematica

Wolfram Mathematica is a proprietary program which is particularly well-suited for symbolic computations. It is an untyped language. It can be of interest for undergraduate students in mathematics and physics that require to make intensive use of symbolic computation. Mathematica is less common in scientific computing and data science fields. On the downside, one needs a Mathematica license to use it. Monash has aMathematica site-license that allows Monash students to install it on their personal computers.

Paul Cally will be the workshop leader for Mathematica. He has shared with us a brief description about Mathematica:

- It is a very flexible highly-integrated high-level language ideally suited to modern technical computing, covering symbolic algebra, calculus, statistics, data analysis, sophisticated graphics, easy access to extensive databases, and many other fields. (See https://www.wolfram.com/mathematica/)
- It allows you to produce very sophisticated complex calculations with just a few lines of code.
- Its Documentation Center contains over 150,000 examples covering Core Language & Structure, Data Manipulation & Analysis, Visualization & Graphics, Machine Learning, Symbolic & Numerical Computation, Higher Mathematical Computation, Strings & Text, Graphs & Networks, Images, Geometry, Sound & Video, Knowledge Representation & Natural Language, Time-Related Computation, Geographical Data & Computation, Scientific & Medical Data & Computation, Engineering Data & Computation, Financial Data & Computation, Social, Cultural & Linguistic Data, and more.
- It seamlessly implements parallel computation.