**Introducción:**

En este trabajo se propone realizar un análisis comparativo entre los aspectos de programación funcional (FP, por las siglas en inglés) de C++ y de Python. Ambos lenguajes poseen características funcionales, como la mayoría de los lenguajes de programación actuales, pero ¿cuál de los dos es el indicado para la resolución de problemas de manera funcional? ¿Depende del problema o alguno de los dos tiene características definitivamente superiores? Para responder esto, se aplicarán distintas técnicas de FP en uno y otro lenguaje, y se evaluarán puntos como: desempeño; facilidad de escritura, lectura y chequeo de errores; potencial de uso. Luego de cada aspecto, se hará una conclusión parcial de los resultados obtenidos en el experimento. Finalmente, se expresará una conclusión a modo de cierre, en la cual se intentará responder a las preguntas planteadas previamente.

**Análisis:**

Para empezar a responder las preguntas planteadas en la introducción, es necesario primero tener en claro algunos conceptos relacionados a la programación funcional.

En primer lugar, ¿qué es FP? Según la Universidad de Nottingham, FP:

*La programación funcional es un estilo de programación que pone énfasis en la evaluación de expresiones, en lugar de la ejecución de comandos. Las expresiones en estos lenguajes son formadas usando funciones para combinar valores básicos.*[[1]](#footnote-1)

¿Qué quiere decir esto? Esta expresión quiere decir que, a diferencia de la programación imperativa tradicional de C++, se intenta describir a los problemas en términos de funciones, más que en términos de órdenes. Al paradigma FP se lo suele considerar como un paradigma declarativo, en el cual no se le explica a la computadora qué pasos debe seguir para resolver un problema, sino que se definen los parámetros del problema. Así, los programas se estructuran de forma que los pasos que se siguen son evaluaciones de funciones, para las cuales el procesador tiene “una explicación” dada por el programador. Veamos un ejemplo:

Imagen que contiene pantalla, celular, monitor, teléfono

Descripción generada automáticamente

Figura 1:

Aplicación en C++ de la función factorial de forma imperativa y recursiva.

En la Figura 1 se ven dos formas alternativas de escribir una función en C++ que reciba un número entero positivo y devuelva su factorial (producto de todos los números enteros entre 1 y sí mismo, inclusive).

Vemos que, en el primer caso, la función se llama a sí misma. No se le dice expresamente a la función cómo se calcula el factorial de un número, sino que simplemente se le dice que evalúe los valores y siga el *step* hasta que, si se lo programó correctamente, el programa llegará al caso base y se efectuarán los sucesivos *return* hasta salir de la función. Es un estilo de programación claro, conciso y declarativo. Sin embargo, trae consigo un agregado de *overhead*, ya que se deben ir apilando las llamadas a funciones.

En el segundo caso, la función se escribe de forma imperativa, con un *loop* explícito en el cual se van multiplicando los números sucesivos y luego sale, cuando la variable *i* deja de cumplir la condición especificada. Como se puede ver, se debe agregar una variable *tot* que mantenga el valor actual de la cuenta, lo cual no es necesario en la versión FP.

Nota: se dejó de lado el chequeo de errores en el input del número a evaluar, con el fin de simplificar el código.

Retomando el tema de FP, en general, los programas que se escriben suelen seguir ciertas reglas generales:

* Utilizar principalmente funciones puras, que son funciones que sólo pueden usar (pero no modificar) los argumentos que se les pasan con el fin de calcular el resultado.[[2]](#footnote-2) Esto quiere decir que una función llamada con los mismos argumentos debería devolver siempre el mismo valor, independientemente del estado del programa que la está llamando.
* Mantener el valor de las variables: también conocido como inmutabilidad. Las funciones no deben modificar variables externas a ella, o que “no le pertenezcan”. Estrictamente, una función no debería tener mutabilidad incluso dentro de sí misma.

Siguiendo en principio estas reglas, se podría comenzar a escribir con el paradigma de FP código cada vez más corto, fácil de leer, manejable y con un mínimo de puntos propensos a errores. Al definir los problemas en términos de sí mismo, o en términos de evaluaciones de funciones que no modificarán su ambiente externo, la posibilidad de errores se minimiza y la detección de los errores que puedan ocurrir es mucho más veloz.

1. Graham Hutton, “Frequently Asked Questions for comp.lang.functional”, University of Nottingham, Nottingham, Inglaterra, 2002. [↑](#footnote-ref-1)
2. Ivan Čukić, “Functional Programming in C++”, Manning Publications, 2018. [↑](#footnote-ref-2)