# Métodos Numéricos para la Ciencia e Ingeniería:

# Informe Tarea n° 4

María Constanza Flores V.

21 de octubre de 2015

## 1. Pregunta 1

En esta sección de la tarea, se pide que se respondan una serie de preguntas acerca de un tutorial desarrollado por *Software Carpentry*.

- **Describa la idea de escribir el `main driver` primero y llenar los huecos luego. ¿Por qué es buena idea?**

  Escribir el `main driver` primero sirve para poder darle una estructura previamente determinada al código, la cual consiste básicamente en tres partes: una parte donde se obtienen los parámetros de la línea de comando, una parte en donde se corre una simulación, y por último una parte en donde se reportan los resultados. Esto es una buena idea pues al tenerlo estructurado es más fácil de leer y también de corregir si es que tiene errores.

- **¿Cuál es la idea detrás de la función `mark_filled`? ¿Por qué es buena idea crearla e vez del código original que reemplaza?**

  La idea detrás de la creación de la función `mark_filled` es,, en primer lugar, hacer más exlícito el código; en segundo lugar, hace que los comentarios acerca de los errores obtenidos sean más significativos de lo que serían los mensajes genéricos entregados por Python al detectar un error del código original (que está siendo reemplazado), lo cual hace más fácil poder detectar errores; y por último, al crear la función se asegura que los valores tengan la coherenca buscada, evitando errores que Python no podría detectar.

- **¿Qué es *refactoring*?**

  *Refactoring* se refiere al hecho de reorganizar el código, cambiando su estructura pero no su funcionalidad, de modo de que sea más legible y más fácil de corregir al momento de ser testeado.

- **¿Por qué es importante implmentar tests que sean sencillos de escribir? ¿Cuál es la estrategia usada en el tutorial?**

  Es necesario implementar test sencillos para así prevenir posibles errores en ellos; los test son útiles para encontrar errores, por lo que si ellos en sí mismos presentan errores no estarían cumpliendo su función.
  La estrategia utilizada en el código para testear el código se basa en "dibujar.[el] caso que se quiere utilizar para el test. Esto se refiere a escribir strings con el caso a utilizar, por lo que puede ser reutilizable para diferentes tests.

- **El tutorial habla de dos grandes ideas para optimizar programas, ¿cuáles son esas ideas? Descríbalas.**

  Por un lado se da como idea *"trade memory for time"*, es decir, priorizar el ahorro de tiempo por sobre el de memoria, guardando información importante con el objetivo de no tener que recalcular, y así hacer más rápido el código. (También se puede hacer al revés, es decir priorizar el ahorro de memoria por sobre el de tiempo).
  Por otro lado se menciona la idea *"trade human time for higher machine performance"*, la cual hace referencia a mejorar el rendimiento del programa, pero haciéndolo más complicado de comprender.

- **¿Qué es `lazy evaluation`?**

  `lazy evaluation` es una forma común de optimizar. En ella, no se calculan valores hasta que sean necesarios, lo que hace que el código sea más rápido, pero más complicado de comprender.

- **Describa la *other moral* del tutorial (es una de las más importantes a la hora de escribir buen código).**

  La *other moral* dice que si se quiere obtener un código rápido, se debe partir por escribir un código simple, luego debe mejorarse testeándolo por partes, reusando los tests utilizados para checkear el trabajo en cada etapa.

# 2. Pregunta 2

## 2.1. Introducción

En esta pregunta se busca encontrar

## 2.2. Procedimiento