Introducción a la programación funcional

Taller de Álgebra I

Verano 2020

Taller de Álgebra I

- ¿De qué se trata el taller de Álgebra I?
  - Dar una introducción a la computación y a la programación, apta tanto para estudiantes de computación como para estudiantes de matemática.
  - Programar los algoritmos que se ven en las teóricas y prácticas...
  - ...usando programación funcional.
    - ¿qué? ¿por qué?
- 6 horas semanales de clase teórico-práctica (modalidad taller).
- Miércoles y Viernes de 19 a 22
- En el Laboratorio 7 (Turing) del DC

Régimen de cursada y evaluación

- Régimen de evaluación:
  - Un parcial individual a mitad del curso.
  - Un trabajo práctico en grupos (de tres alumnos) al final del curso.

  - **Importante:** En las instancias de evaluación solo podrán utilizarse las técnicas y herramientas vistas en clase.

Listas de correo del taller

- Si te inscribiste por el SIU, deberías haber recibido un mail de bienvenida.
- **Importante**: Si no te llegan los mails o necesitás usar otra dirección, avisanos.

- Anuncios para alumnos $\longrightarrow$ `algebra1-alu (arroba) dc.uba.ar`
  - Anuncios importantes sobre la cursada (¡leé tu mail seguido!).
  - Entre ustedes: para buscar grupo de TP, armar grupos de estudio, etc.
- Consultas para docentes $\longrightarrow$ `algebra1-doc (arroba) dc.uba.ar`
  - **No enviar consultas a** `-alu`.
  - Cuando la respuesta es de interés general, nosotros lo re-enviamos a `-alu`.
  - Es mejor enviar tu consulta a `-doc` que a un docente en privado.

Página del Campus

- Allí pueden encontrar información general de la materia, y los slides de las clases
- También publicamos novedades en su portada
- Se encuentra enlazada desde la página de Álgebra I del DM
- **URL (Acortada):** http://bit.do/talgebra-v2020

Laboratorios

- Fuera de los horarios de clase, los laboratorios están disponibles para poder ejercitar o ponerse al día.
- Puede ser que este particularmente esté ocupado por otra clase, pero siempre se busca que haya alguno disponible.
- Se puede consultar a algún conservador.

Contenidos del taller

Herramientas computacionales que permitan trabajar sobre los problemas vistos en la práctica.

¿Qué problemas?

- Relaciones
- Funciones
- Inducción
- Conjuntos
- Combinatoria
- Ecuaciones de congruencia
- Ecuaciones diofánticas
- Polinomios
- Y más..

¿Qué herramientas?

- Diferencia entre Problema, Algoritmo y Programa
- Introducción a la programación funcional
- Tipos de datos
- Evaluación de funciones
- Reducción y recursión
- Recursión sobre números
- Recursión sobre listas
- Pattern matching
- Y más..

Problema $\rightarrow$ Algoritmo $\rightarrow$ Programa

Resolviendo problemas con una computadora

- La resolución de un problema en computación requiere de al menos los siguientes pasos:
  - Identificar y aislar el problema a resolver.
  - Pensar una descripción de la solución (algoritmo) para resolver el problema.
  - Implementar el algoritmo en un lenguaje de programación y en una plataforma determinada, obteniendo así un programa ejecutable.

Ejemplo

- **Problema:** Dados dos números naturales representados en base decimal, encontrar la suma de los números.
- Podemos pensar varios algoritmos para resolver este problema.
  - **Algoritmo escolar**: Sumo las unidades del primero a las del segundo, después las decenas, etc ("llevándome uno de acarreo" cuando hace falta)
  - **Algoritmo sucesor**: Voy sumando 1 al primero y restando uno al segundo, hasta que el segundo llegue a 0.
  - **Algoritmo posta**: Entro a google.com, escribo el primer número, luego el signo "+", luego escribo el segundo y luego aprieto enter.

Ejemplo

- Los tres algoritmos difieren en las herramientas primitivas utilizadas para expresarlos:
  - sumar números de un dígito cada uno, contemplar el acarreo, concatenar resultados, etc.
  - sumar y restar 1, y comparar contra el 0.
  - usar teclado y mouse de una computadora conectada a Internet.
- ¿Cuál algoritmo tiene más sentido/conviene usar?

Programas

- Un programa es la descripción de un algoritmo en un lenguaje de programación.
  - Es una descripción precisa, de modo tal que pueda ser ejecutada por una computadora.
  - Un lenguaje de programación tiene una sintaxis y una semántica bien definidas.
- Cuando se implementa un programa, es importante preguntarse
  - si el programa es correcto,
  - si implementa adecuadamente el algoritmo propuesto,
  - si puede pasar que el programa no termine,
  - qué datos son válidos para ejecutar el programa,
  - cuánto va a tardar la ejecución,
  - si está bien resuelto el problema original.

Halting problem

- **Algoritmo sucesor para sumar dos números:** Voy sumando 1 al primero y restando uno al segundo, hasta que el segundo llegue a 0. Pero, ¿qué pasa si sumamos 15 y -5?
- ¿Se puede hacer un algoritmo que indique cuándo otros programas están por colgarse?
- Independientemente Turing y Church en 1936/7 (antes que existan las computadoras!) probaron que no puede existir. El problema es *indecidible* (no tiene solución).
  - Dieron una definición precisa de algoritmo.
  - Turing desarrolló las Máquinas de Turing y probó que el problema de la detención (*halting problem*) no es decidible.
  - Church desarrolló el cálculo lambda y mostró que no existe algoritmo capaz de decidir si dos funciones son equivalentes.
  - Estos enfoques son equivalentes y capturan la noción de los problemas que son computables.

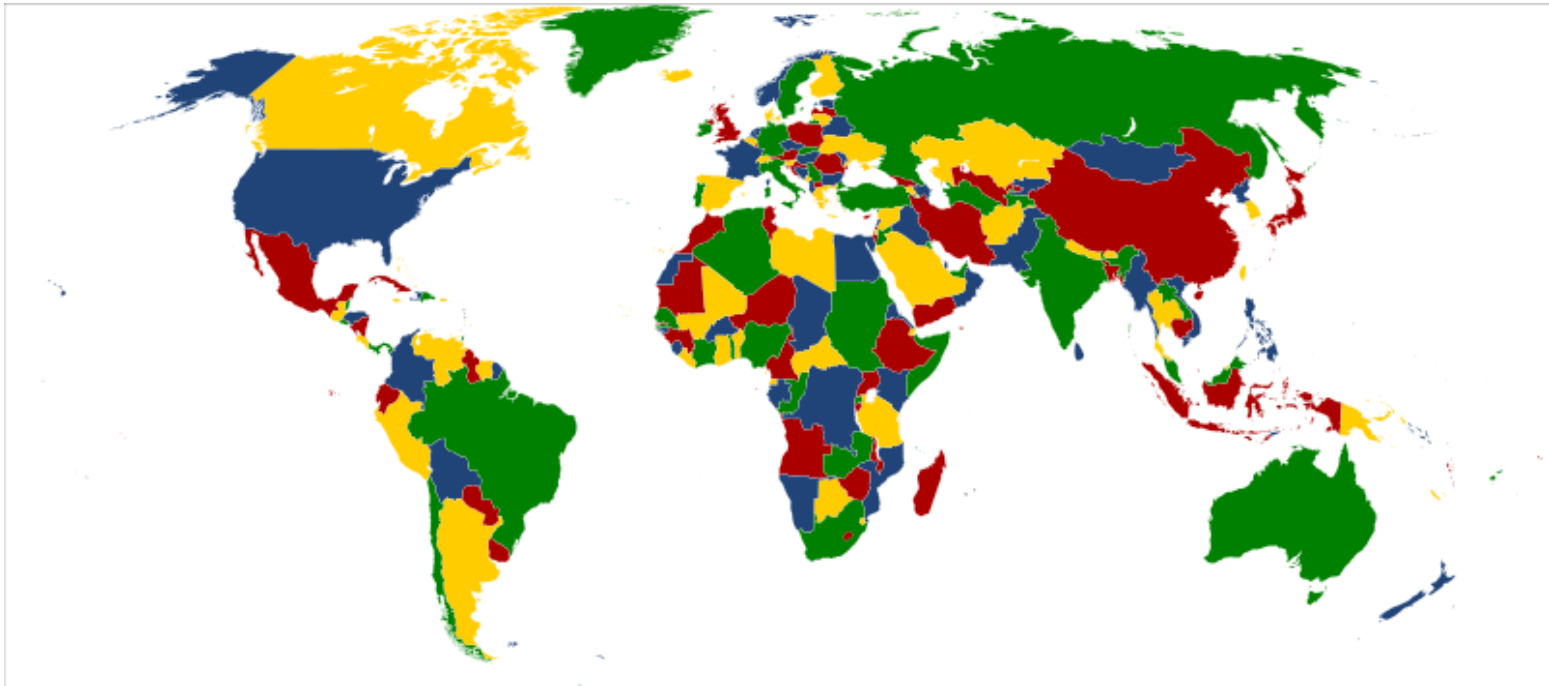
¿Por qué la computación es útil? ¿Y por qué para un matemático?

¿Por qué la computación es útil para un matemático?

- La computación comenzó como una rama de la matemática, como una forma de entender (entre otras cosas) si todos los problemas son resolubles. Para eso, se pensaba en computadoras ideales.
- Con el tiempo, las computadoras se volvieron reales. Y cambiaron el panorama, la forma de hacer matemática. Permiten por ejemplo:
  - hacer cuentas que llevarían años de manera manual
  - probar conjeturas
  - chequear todos los casos en un teorema
  - probar que un teorema está bien demostrado
  - y más
- ¿Y si no hago ninguna de las anteriores? Pensar algorítmicamente permite atacar los problemas desde un punto de vista totalmente distinto y muchas veces permite organizar mejor las ideas.

Veamos un ejemplo: los cuatro colores

¿Cuántos colores hacen falta para colorear un mapa de tal manera que dos regiones limítrofes tengan distintos colores? Ejemplo: el mapa del mundo y sus países se puede colorear con 4 colores.



En 1852, Francis Guthrie le preguntó a su hermano matemático si 4 colores alcanzarían para cualquier mapa. Ni el hermano de Guthrie, ni su profesor (Augustus de Morgan) ni otros matemáticos de la época pudieron demostrar que fuese así (ni encontrar un contraejemplo).

Cuatro colores

En 1879, Alfred Kempe publicó un trabajo donde demostró que era cierto que 4 colores bastaban para cualquier mapa.

En 1890, Percy Heawood probó que la demostración de Kempe estaba mal.

En 1977, Appel y Haken demostraron que era cierto, que 4 colores bastaban para cualquier mapa.

Pero la demostración de Appel y Haken dividió a la comunidad matemática. ¿Por qué?:

- Redujeron los mapas a unos 10.000 casos prototípicos. Chequear que estos eran todos los casos posibles no es sencillo.
- Luego, Appel y Haken probaron que en estos últimos 10.000 casos 4 colores bastaban... pero lo hicieron usando una computadora.

¿Por qué creerle a una computadora?

Un programa, decían los críticos de la computación entre los matemáticos, puede

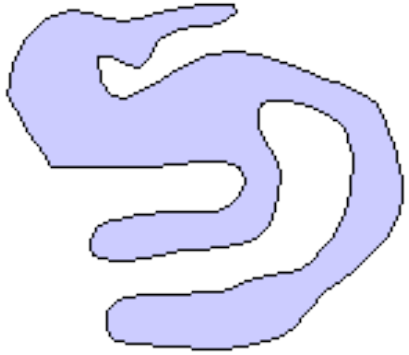
- Fallar (algo que se testea corriendo el mismo programa en distintas máquinas),
- y mucho más importante: estar mal hecho (tener *bugs*).
- Pero... ¿una persona, no puede equivocarse en una demostración escrita en papel?

En 2004, Georges Gonthier demostró formalmente el teorema de los cuatro colores. Lo hizo con una computadora (no los sólo los 10.000 casos, sino todo el teorema), escribiendo la demostración en un lenguaje que sirve para verificar demostraciones: Coq.

Observación: Coq no es el único lenguaje desarrollado para verificar demostraciones. En el paper `http://www.cs.ru.nl/~freek/comparison/comparison.pdf`
se muestran programas que prueban que $\sqrt{2}$ es irracional en 17 lenguajes distintos.

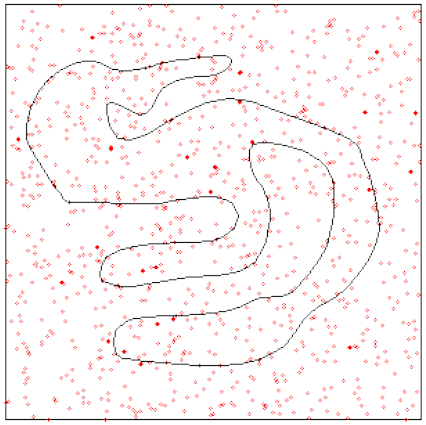
Estimaciones (Análisis)

Queremos calcular el área de la figura (o la integral de una curva)



Estimaciones (Análisis)

Tiramos 1000 puntos al azar en un cuadrado que contiene a la figura y contamos cuántos caen adentro de la figura.



Por ejemplo, si caen 280 y el tamaño del cuadrado es de $100cm^2$, el área de la figura es aproximadamente

$$\frac{280}{1000}100cm^2 = 28cm^2$$

Partes de un Conjunto (Álgebra)

(mucho más sobre esto más adelante)

- Queremos una fórmula para la cantidad de subconjuntos de un conjunto de n elementos.
- Una posibilidad (no es el enfoque de este taller): hacemos un programa que los cuente (o usamos Wolfram Alpha) y vemos los resultados:
  - 0 elementos $\rightarrow$ 1 subconjunto
  - 1 elemento $\rightarrow$ 2 subconjuntos
  - 2 elementos $\rightarrow$ 4 subconjuntos
  - 3 elementos $\rightarrow$ 8 subconjuntos
- Ah... ¡es 2^n! ¿Por qué? En principio, no sabemos. (pero es una estrategia muy buena para descubrir propiedades!)

Aplicaciones a la matemática

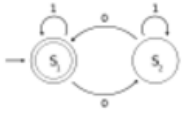
- Otra posibilidad (el enfoque de este taller): hacemos un programa que los cuente y analizamos qué estamos haciendo.
- Por ejemplo, subconjuntos del conjunto $\{1, 2, 3\}$:
  1. Calculamos primero todos los subconjuntos del conjunto $\{1, 2\}$:
     $\{\}, \{1\}, \{2\}, \{1, 2\}$
  2. A esa lista, le agregamos a cada subconjunto un 3:
     $\{3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$
  3. Juntamos todos los que ya teníamos y los nuevos:
     $\{\}, \{1\}, \{2\}, \{1, 2\}, \{3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$
- Ah... cada vez que agrego un elemento, ¡la cantidad de conjuntos se multiplica por 2! Eso me da 2^n.
- Enfoque de este taller: pensar los problemas en forma algorítmica para entenderlos mejor.

¿Y hoy en día para que sirve en matemática?

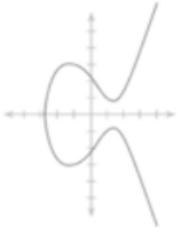
Otros ejemplos:

$P \to Q$

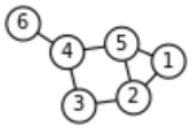
Mathematical logic



Automata theory



Number theory



Graph theory

Computability theory

P = NP ?

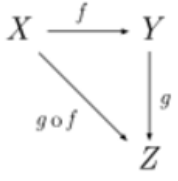
Computational complexity theory

GNITIRW-TERCES

Cryptography

$\Gamma \vdash x : \mathrm{Int}$

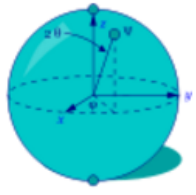
Type theory



Category theory

Computational geometry

Combinatorial optimization



Quantum computing theory

Manos a la obra

Recursos para el Taller

Usuarios para los laboratorios

Quienes no tienen cuenta en los laboratorios:

- Después de clase, pedir a los conservadores abrir una cuenta. Llevar LU.
- HOY: Ingresar con usuario y contraseña genérico: `clinux01`

Intérprete de Haskell

GHCI (The Glasgow Haskell Compiler Interactive environment)

- ¿Ubuntu? `sudo apt-get install ghc`
- ¿Mac? `brew install haskell-platform`
- ¿Windows? `https://www.haskell.org/platform/windows.html`

Editores de texto

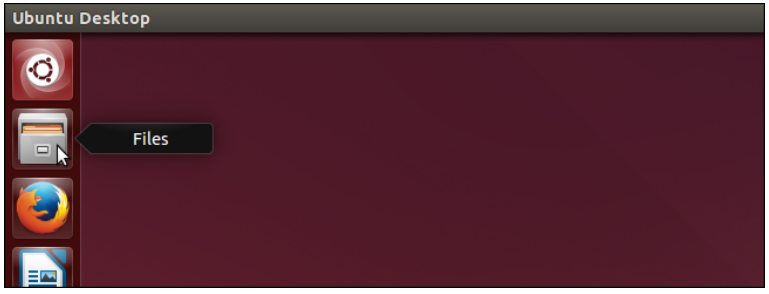
- ¿Ubuntu? gedit, Sublime, vim, Atom, etc.
- ¿Mac? Atom, Textmate, TextEdit, Sublime, vim, etc.
- ¿Windows? Atom, Notepad++, Sublime, etc.

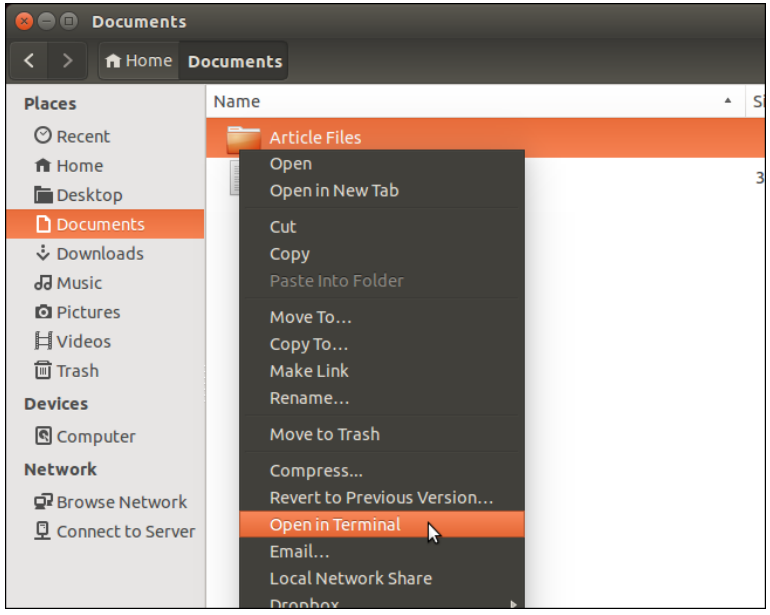
¡¡Trabajen **cómodos**!!

Linux: Terminal (o Consola)

¿Qué es una consola? Es una interfaz desde la cual podemos escribir y ejecutar comandos.

¿Cómo abrimos una consola? Mediante la interfaz gráfica de Ubuntu, o mediante el atajo `Ctrl + Alt + T`.





Comandos útiles en la consola

- `pwd` (*print working directory*). Imprime el nombre del directorio actual, donde estamos parados.
- `ls` (*list*). Lista los contenidos de un directorio.
- `cd` (*change directory*). Nos permite navegar por los directorios.
- `mkdir` (make directory). Permite crear directorios.
- `cp, mv, rm`. (copy, move, remove). Permite copiar/mover/borrar archivos respectivamente. `mv` también se usa para renombrar archivos.

Programación funcional

- Un programa en un lenguage funcional es un conjunto de ecuaciones orientadas que definen una o más funciones.

  Por ejemplo:

```
doble x = 2 * x
triple x = 3 * x
```

- La ejecución de un programa en este caso corresponde a la evaluación de una expresión, habitualmente solicitada desde la consola del entorno de programación.

```
Prelude> doble 10
20
```

- La expresión se evalúa usando las ecuaciones definidas en el programa, hasta llegar a un resultado.
- Las ecuaciones orientadas junto con el mecanismo de reducción describen algoritmos (definición de los pasos para resolver un problema).

Para hacer ahora...

1. Abrir una terminal y, quienes usen la cuenta común: crear un directorio propio (`mkdir APELLIDO`) y trabajar ahí adentro (`cd APELLIDO`).
2. Crear un archivo de texto usando *alguno* de los siguientes comandos:
   - `gedit clase1.hs`
   - `atom clase1.hs`
   - `subl clase1.hs`
   - `vim clase1.hs` (si ya saben manejarlo)
3. Escribir dentro del archivo `f x y = x * x + y * y` y guardarlo en la misma carpeta.
4. Abrir una nueva terminal, ir al mismo dir y ejecutar `ghci` (abre el intérprete de Haskell).
5. Ejecutar alguna operación simple, por ejemplo `8 * 7` .
6. Cargar el archivo: `:l clase1.hs` (si hubieran abierto el GHCI en otra carpeta pueden abrir el archivo con `:l <directorio del archivo>`).
7. Dentro de GHCI, ejecutar lo siguiente: `f 2 3`
8. Agregar al código la función `g x y z = x + y + z * z` y volver a guardar.
9. En `ghci` , recargar el programa: `:r`
10. Ejecutar `g 2 3 4`
11. Si quieren, pueden cerrar el intérprete ejecutando: `:q`

Construcción del conocimiento

A lo largo de este curso van a ir desarrollando y escribiendo código que resuelve problemas.

Con esto en mente, es importante que hagan código **ordenado**, **declarativo** y **explicado**.

Además de guardar el trabajo que van haciendo para poder **reutilizarlo**.

Primer ejercicio: Programar las siguientes funciones

- $doble(x) = 2x$
- $suma(x, y) = x + y$
- $||(x_1, x_2)|| = \sqrt{x_1^2 + x_2^2}$
- $f(x) = 8$
- $respuestaATodo = 42$

- `doble x = ??`
- `suma x y = ??`
- `normaVectorial x1 x2 = ??`
- `funcionConstante8 x = ??`
- `respuestaATodo = ??`

Ejecutar las siguientes expresiones en el intérprete

```
*Main> doble 10
*Main> doble (-1)
*Main> suma (-1) 4
*Main> normaVectorial 3 5
*Main> funcionConstante8 0
*Main> respuestaATodo
*Main> doble 10 20
```

Para ayuda, ver el machete

Machete

- Números
  - `1` (números)
  - `1.3, 1e-10, 6.022140857e23` (números con decimales)
  - `(-1)` (negativos, conviene utilizar paréntesis en general)
- Funciones básicas
  - `+, *, /, -`
  - `div, mod`
  - `sqrt, **, ^`
- Uso de funciones (suponiendo f, g, h, fn funciones)
  - Para aplicar una función, utilizamos el nombre de la función seguido de parametros con espacios entre medio:
    - `f x1 x2 x3 x4 x5 x6`
    - Equivalente a $f(x_1, x_2, x_3, x_4, x_5, x_6)$ en matemática
    - Ejemplo: `sqrt 4`
    - Ejemplo: `div 2 3`
  - Para indicar que un parámetro es resultado de otra operación, usamos paréntesis:
    - `fn (g x1) (h x2 x3) x4 x5`
    - Equivalente a $fn(g(x_1), h(x_2, x_3), x_4, x_5)$ en matemática
    - Ejemplo: `sqrt ((sqrt 10) - 3)`
    - Ejemplo: `div (mod 3 5) (mod 4 3)`

Definiciones de funciones por casos

Podemos usar guardas para definir funciones por casos:

$$f(n) = \begin{cases} 1 & \text{si } n = 0 \\ 0 & \text{si no} \end{cases}$$

```
f n | n == 0 = 1
    | n /= 0 = 0
```

Palabra clave "si no".

```
f n | n == 0 = 1
    | otherwise = 0
```

¿Qué pasa si invertimos las guardas? **¿Por qué?**

Presten atención al orden de las guardas. ¡Cuando las condiciones se solapan, el orden de las guardas cambia el comportamiento de la función!

Definiciones de funciones por casos

La función Signo:

$$signo(n) = \begin{cases} 1 & \text{si } n > 0 \\ 0 & \text{si } n = 0 \\ -1 & \text{si } n < 0 \end{cases}$$

Ejercicios

- Implementar la función `signo`.
- Implementar la función `absoluto` que calcula el valor absoluto de un número. ¿Está bueno repetir? ¿Conviene reutilizar?
- Implementar la función `maximo` que devuelve el máximo entre 2 números.
- Implementar la función `maximo3` que devuelve el máximo entre 3 números.