Introducción a la programación funcional

Taller de Álgebra I

Segundo cuatrimestre de 2016

Taller de Álgebra I

- ¿De qué se trata el taller de Álgebra I?
  1. Dar una introducción a la computación y a la programación, apta tanto para estudiantes de computación como para estudiantes de matemática.
  2. Programar los algoritmos que se ven en las teóricas y prácticas...
  3. ...usando programación funcional.
     - ¿qué? ¿por qué?
- 3 horas semanales de clase teórico-práctica (modalidad taller).
- En los laboratorios de Computación.

Régimen de cursada y evaluación

- Régimen de evaluación:
  - Un parcial a mediados del cuatrimestre que contendrá algún ejercicio de los vistos en clase y alguno nuevo con dificultad similar.
  - Un trabajo práctico en grupos de exactamente tres personas a fines del cuatrimestre.
  - En caso de no aprobar en alguna de las instancias:
    - Recuperatorio del parcial al final del cuatrimestre.
    - Recuperatorio del TP al final del cuatrimestre.
  - **Importante:** En las instancias de evaluación solo podrán utilizarse las técnicas y herramientas vistas en clase.
- No es necesario aprobar el taller para anotarse en las materias correlativas.
- Sí es necesario aprobar el taller para rendir el final de Álgebra I.

Listas de correo del taller

- Si te inscribiste por el SIU, deberías haber recibido el mail de bienvenida.
- Si no te llegan los mails o necesitás usar otra dirección, comunicate con nuestro listmaster: `gino.scarpino (arroba) gmail.com`

- Anuncios para alumnos $\longrightarrow$ `algebra1-alu (arroba) dc.uba.ar`
  - Anuncios importantes sobre la cursada (¡leé tu mail seguido!).
  - Entre ustedes: para buscar grupo, 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.

Contenidos del taller

- Introducción a la programación funcional
- Tipos de datos
- Reducción y recursión
- Funciones auxiliares
- Recursión sobre listas
- Nuevos tipos y Pattern matching
- Tipos paramétricos
- Tipos recursivos
- Funciones de alto orden

Recursos para el Taller

Intérprete de Haskell

GHCI (The Glasgow Haskell Compiler Interactive environment)

- ¿Ubuntu? `sudo apt-get install ghc`
- ¿Mac? `brew install haskell-platform`
- ¿Windows? http://www.haskell.org/ghc/download, o https://www.haskell.org/hugs/pages/downloading.htm

Editores de texto

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

¡¡Trabajen **cómodos**!!

Más recursos

¿Cómo trabajar en grupo?

Formas de compartir código

1. Email
2. Dropbox
3. Google Docs
4. svn
5. **git**

La **ComCom** (Comisión de Alumnos de Computación) ofrece cursos de Git. ¡Los invitamos a que participen!

Los materiales de los talleres pasados los pueden encontrar en:

`https://talleres-comcom.gitlab.io/taller-git/`

Terminal (o Consola)

¿Qué es una consola?

Comandos útiles para consolas

- `Ctrl + Alt + T`
- `pwd`
- `mkdir`
- `ls` (`dir` en Windows)
- `cd`
- `cp`, `mv`, `rm` (`copy`, `move`, `del` en Windows)

Explicación y ejemplos de lo anterior

- `pwd`: imprime el nombre del directorio actual. Donde estamos parados.

```
usuario@computadora1:~$ pwd
/home/usuario/
usuario@computadora1:~$
```

- `ls`: lista los contenidos de un directorio

```
usuario@computadora1:~$ ls
Documentos Escritorio Descargas Imagenes
usuario@computadora1:~$
```

- `cd`: nos permite navegar por los directorios

```
usuario@computadora1:~$ cd Escritorio
usuario@computadora1/Escritorio:~$ ls
usuario@computadora1/Escritorio:~$ cd ..
usuario@computadora1:~$
```

- `mkdir`: permite crear carpetas.

```
usuario@computadora1:~$ cd Escritorio
usuario@computadora1/Escritorio:~$ mkdir taller_computadora1
usuario@computadora1/Escritorio:~$ ls
taller_computadora1
usuario@computadora1/Escritorio:~$ cd taller_computadora1
usuario@computadora1/Escritorio/taller_computadora1:~$
```

Explicación y ejemplos de lo anterior

- `gedit`: Un editor de texto básico.

```
../taller_computadora1:~$ gedit ejemplo1.hs
```

- `cp/mv/rm`: copia/mueve/borra archivos

```
../taller_computadora1:~$ ls
ejemplo1.hs
../taller_computadora1:~$ cp ejemplo1.hs clase1.hs
../taller_computadora1:~$ ls
ejemplo1.hs clase1.hs
../taller_computadora1:~$ rm ejemplo1.hs
../taller_computadora1:~$ ls
clase1.hs
../taller_computadora1:~$ mv clase1.hs ../clase1.hs
../taller_computadora1:~$ ls
../taller_computadora1:~$ cd ..
../Escritorio:~$ ls
clase1.hs
../taller_computadora1:~$
```

(`mv` también se usa para renombrar archivos)

Ejercicios

1. Crear un archivo de texto con el siguiente contenido: `suma x y = x - y`
   (sí, sabemos que está mal).
2. Guardarlo con el nombre **clase1.hs** en alguna carpeta (no se olviden la ubicación).
3. Abrir una consola e ir al carpeta donde guardaron el archivo.
4. Ejecutar `ghci` (abre el interprete de Haskell).
5. Ejecutar alguna operación simple, por ejemplo `8 * 7`.
6. Cargar el archivo: `:l <nombre archivo>` (si hubieran abierto el GHCI en otra carpeta pueden abrir el archivo con `:l <directorio del archivo>`).
7. Dentro de GHCI, ejecutar lo siguiente: `suma 2 3`
8. Corregir y guardar la función en el archivo para que efectivamente sea la suma (sin cerrar la consola de GHCI).
9. Recargar el programa: `:r`
10. Ejecutar `suma 2 3` y ver que dé `5` :)
11. Si quieren, pueden cerrar el intérprete ejecutando: `:q`

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).

Primer ejercicio: Programar las siguientes funciones

$doble(x) = 2x$

$suma(x, y) = x + y$

$||(v_1, v_2)|| = \sqrt{v_1^2 + v_2^2}$

$f(x) = 8$

$respuesta = 42$

- `doble x = ??`
- `suma x y = ??`
- `normaVectorial v1 v2 = ??`
- `funcionConstante8 x = ??`
- `respuestaATodo = ??`

Ejecutar las siguientes expresiones en el intérprete

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

Definiciones de funciones por casos

Podemos usar guardas para definir funciones por casos:

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

```
delta n | n == 0 = 1
        | n /= 0 = 0
```

Palabra clave "si no".

```
delta 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

Ejercicio: 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 `abs` 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.

Ejercicios

1. Programar la siguiente función:

$$f(n_1, n_2, n_3) = \begin{cases} n_1 & \text{si } n_2 < 10 \\ n_1 + n_3 & \text{si } n_2 \geq 10 \end{cases}$$

2. Programar una función que tome tres parámetros $a, b, c \in \mathbb{R}$ y que calcule alguna de las raíces de la función cuadrática $f(x) = ax^2 + bx + c$.