Introducción a los Algoritmos
Tipos, Funciones y Patrones

Pedro Sánchez Terraf

CIEM-FaMAF — Universidad Nacional de Córdoba

FaMAF
UNC
16 de marzo de 2016

Antes de empezar...

1. Entramos en la *sesión de invitado.*
2. Aula Virtual:
   `http://www.famaf.proed.unc.edu.ar/course/view.php?id=271`.
3. Material viejo y extras (esta charla por ejemplo:
   `http://cs.famaf.unc.edu.ar/~pedro/introalg`.
   (forma fácil: buscar “Pedro Sanchez Terraf” en Google, y en mi página hay un link de la materia).
4. **Intro a Linux**: datos básicos para utilizar este sistema operativo.

Contenido

Demostraciones

$$
\begin{aligned}
& 5 * (x + 3) = 20 \\
\equiv \{ & \ \ldots \ \} \\
& 5 * (3 + x) = 20
\end{aligned}
$$

1. El “equivalente” ($\equiv$) es el “si y sólo si” ($\Leftrightarrow$).
2. Debo justificar usando propiedades **válidas** o **definiciones**.

Demostraciones

$$
\begin{aligned}
& 5 * (x + 3) = 20 \\
\equiv \{ \; & \text{Conmutativa +} \; \} \\
& 5 * (3 + x) = 20
\end{aligned}
$$

1. El “equivalente” ($\equiv$) es el “si y sólo si” ($\Leftrightarrow$).
2. Debo justificar usando propiedades **válidas** o **definiciones**.

Demostraciones

$$
\begin{aligned}
& 5 * (x + 3) = 20 \\
\equiv \{ \; & \text{Conmutativa +} \; \} \\
& 5 * (3 + x) = 20
\end{aligned}
$$

Expandimos la justificación:

Conmutativa de +:

$$a + b = b + a$$

Sustituyo a por $\_\_$
Sustituyo b por $\_\_$
para obtener

$$x + 3 = 3 + x.$$

1. El “equivalente” ($\equiv$) es el “si y sólo si” ($\Leftrightarrow$).
2. Debo justificar usando propiedades **válidas** o **definiciones**.

Demostraciones

$$
\begin{aligned}
& 5 * (x + 3) = 20 \\
\equiv \{ & \text{ Conmutativa + } \} \\
& 5 * (3 + x) = 20
\end{aligned}
$$

Expandimos la justificación:

Conmutativa de +:

$$a + b = b + a$$

Sustituyo a por __
Sustituyo b por __
para obtener

$$x + 3 = 3 + x.$$

1. El “equivalente” ($\equiv$) es el “si y sólo si” ($\Leftrightarrow$).
2. Debo justificar usando propiedades **válidas** o **definiciones**.

Demostraciones

$$
\begin{aligned}
& 5 * (x + 3) = 20 \\
\equiv \{ & \text{ Conmutativa + } \} \\
& 5 * (3 + x) = 20
\end{aligned}
$$

Expandimos la justificación:

Conmutativa de +:

$$a + b = b + a$$

Sustituyo a por x
Sustituyo b por 3
para obtener

$$x + 3 = 3 + x.$$

1. El "equivalente" ($\equiv$) es el "si y sólo si" ($\Leftrightarrow$).
2. Debo justificar usando propiedades **válidas** o **definiciones**.

Demostraciones

$$
\begin{aligned}
& 5 * (x + 3) = 20 \\
\equiv \{ & \text{ Conmutativa + } \} \\
& 5 * (3 + x) = 20
\end{aligned}
$$

Expandimos la justificación:

Conmutativa de +:

$$a + b = b + a$$

Sustituyo a por x
Sustituyo b por 3
para obtener

$$x + 3 = 3 + x.$$

1. El “equivalente” ($\equiv$) es el “si y sólo si” ($\Leftrightarrow$).
2. Debo justificar usando propiedades **válidas** o **definiciones**.

Demostraciones

$$
\begin{aligned}
& 5 * (x + 3) = 20 \\
\equiv \{ & \text{ Conmutativa + } \} \\
& 5 * (3 + x) = 20
\end{aligned}
$$

Expandimos la justificación:

Conmutativa de +:

$$a + b = b + a$$

Sustituyo a por x
Sustituyo b por 3
para obtener

$$x + 3 = 3 + x.$$

1. El “equivalente” ($\equiv$) es el “si y sólo si” ($\Leftrightarrow$).
2. Debo justificar usando propiedades **válidas** o **definiciones**.

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo $Bool$ puede ser:

1. **válida** si es $True$ para todos los valores de sus variables
2. **satisfactible** si hay al menos un valor de las variables que las hace $True$
3. **no válida** si es $False$ para algún valor de sus variables;
4. **no satisfactible** si es $False$ para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo *Bool* puede ser:

1. **válida** si es *True* para todos los valores de sus variables
2. **satisfactible** si hay al menos un valor de las variables que las hace *True*
3. **no válida** si es *False* para algún valor de sus variables;
4. **no satisfactible** si es *False* para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo *Bool* puede ser:

1. **válida** si es *True* para todos los valores de sus variables (puedo demostrar que es equivalente a *True*);
2. **satisfactible** si hay al menos un valor de las variables que las hace *True*
3. **no válida** si es *False* para algún valor de sus variables;
4. **no satisfactible** si es *False* para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo $Bool$ puede ser:

1. **válida** si es $True$ para todos los valores de sus variables (puedo demostrar que es equivalente a $True$);
2. **satisfactible** si hay al menos un valor de las variables que las hace $True$
3. **no válida** si es $False$ para algún valor de sus variables;
4. **no satisfactible** si es $False$ para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo $Bool$ puede ser:

1. **válida** si es $True$ para todos los valores de sus variables (puedo demostrar que es equivalente a $True$);
2. **satisfactible** si hay al menos un valor de las variables que las hace $True$ (hay un ejemplo);
3. **no válida** si es $False$ para algún valor de sus variables;
4. **no satisfactible** si es $False$ para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo $Bool$ puede ser:

1. **válida** si es $True$ para todos los valores de sus variables (puedo demostrar que es equivalente a $True$);
2. **satisfactible** si hay al menos un valor de las variables que las hace $True$ (hay un ejemplo);
3. **no válida** si es $False$ para algún valor de sus variables;

4. **no satisfactible** si es $False$ para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo $Bool$ puede ser:

1. **válida** si es $True$ para todos los valores de sus variables (puedo demostrar que es equivalente a $True$);
2. **satisfactible** si hay al menos un valor de las variables que las hace $True$ (hay un ejemplo);
3. **no válida** si es $False$ para algún valor de sus variables; (hay un contraejemplo);
4. **no satisfactible** si es $False$ para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo $Bool$ puede ser:

1. **válida** si es $True$ para todos los valores de sus variables (puedo demostrar que es equivalente a $True$);
2. **satisfactible** si hay al menos un valor de las variables que las hace $True$ (hay un ejemplo);
3. **no válida** si es $False$ para algún valor de sus variables; (hay un contraejemplo);
4. **no satisfactible** si es $False$ para todos los valores de sus variables

Expressiones Booleanas Válidas y Satisfactibles

Una expresión de tipo $Bool$ puede ser:

1. **válida** si es $True$ para todos los valores de sus variables (puedo demostrar que es equivalente a $True$);
2. **satisfactible** si hay al menos un valor de las variables que las hace $True$ (hay un ejemplo);
3. **no válida** si es $False$ para algún valor de sus variables; (hay un contraejemplo);
4. **no satisfactible** si es $False$ para todos los valores de sus variables (puedo demostrar que es equivalente a $False$);

Tipos Básicos y Derivados

Trabajamos en la Compu

1. Abrir Terminal.
2. Abrir el intérprete de Haskell `ghci`.

Tipos Básicos

Introdujimos los siguientes tipos básicos y otros (derivados) que podemos fabricar con ellos. Una expresión tiene tipo

1. **Num**: si su valor es un número (real).
   Ejemplos:
   - $10 * 5 + 7$,
   - x^2,
   - `length` $[x, y, z]$.
2. **Bool**: si su valor es "verdadero" (*True*) o "falso" (*False*).
   Ejemplos:
   - $10 * 5 + 7 = x^2$,
   - $3 * 7 < 1$,
   - $p \equiv q$,
3. **Char**: si su valor es una caracter (letras, etc.).
   Ejemplos:
   - `'a'`,
   - `'1'`,
   - `' '` (espacio).

Tipos Básicos

Introdujimos los siguientes tipos básicos y otros (derivados) que podemos fabricar con ellos. Una expresión tiene tipo

1. **Num**: si su valor es un número (real).
   Ejemplos:
   - $10 * 5 + 7$,
   - x^2,
   - `length` $[x, y, z]$.
2. **Bool**: si su valor es "verdadero" ($True$) o "falso" ($False$).
   Ejemplos:
   - $10 * 5 + 7 = x^2$,
   - $3 * 7 < 1$,
   - $p \equiv q$,
3. **Char**: si su valor es una caracter (letras, etc.).
   Ejemplos:
   - `'a'`,
   - `'1'`,
   - `' '` (espacio).

Tipos Básicos

Introdujimos los siguientes tipos básicos y otros (derivados) que podemos fabricar con ellos. Una expresión tiene tipo

1. **Num**: si su valor es un número (real).
   Ejemplos:
   - $10 * 5 + 7$,
   - x^2,
   - `length` $[x, y, z]$.
2. **Bool**: si su valor es "verdadero" (*True*) o "falso" (*False*).
   Ejemplos:
   - $10 * 5 + 7 = x^2$,
   - $3 * 7 < 1$,
   - $p \equiv q$,
3. **Char**: si su valor es una caracter (letras, etc.).
   Ejemplos:
   - `'a'`,
   - `'1'`,
   - `' '` (espacio).

Tipos en Haskell

Podemos averiguar tipos con `ghci` usando `:t`.

La compu no entiende *Num*

No se pueden representar todos los números reales en la compu, pero podemos usar enteros (tipo *Int*), números con una cantidad fija de decimales (tipo *Float*) y hay más.

Todos los demás tipos sí están definidos en Haskell.

Tipos en Haskell

Podemos averiguar tipos con `ghci` usando `:t`.

La compu no entiende Num

No se pueden representar todos los números reales en la compu, pero podemos usar enteros (tipo Int), números con una cantidad fija de decimales (tipo $Float$) y hay más.

Todos los demás tipos sí están definidos en Haskell.

Tipos en Haskell

Podemos averiguar tipos con `ghci` usando `:t`.

La compu no entiende *Num*

No se pueden representar todos los números reales en la compu, pero podemos usar enteros (tipo *Int*), números con una cantidad fija de decimales (tipo *Float*) y hay más.

Todos los demás tipos sí están definidos en Haskell.

Tipos Derivados

Con los tipos básicos podemos hacer listas y tuplas. Las “tuplas” (pares, ternas, etc.) se escriben entre paréntesis y tienen tamaño fijo.

1. **Listas.**
   Ejemplos:
   1. $[x, y+z]$ (de tipo $[Num]$),
   2. $[True, p]$ (de tipo $[Bool]$),
   3. $[\texttt{"hola"}, \texttt{"chau"}]$ (de tipo $[String]$)
2. **Tuplas.**
   Ejemplos:
   1. $(3, -10, x*2)$ (de tipo (Num, Num, Num)),
   2. $(x*5, True)$ (de tipo $(Num, Bool)$),
   3. $(\texttt{"Juan"}, 1.75)$ (de tipo $(String, Num)$),

Comparar los tipos que obtenemos para ambos ejemplos (3) en Haskell.

Se pueden combinar, por ejemplo, listas de listas $[[1,2],[5],[8,9,10]]$ (de tipo $[[Num]]$), listas de pares $[(\texttt{"Juan"}, 1.75), (\texttt{"Jose"}, 1.83)]$, pares de listas $([1,2,4], [\texttt{'b'}, \texttt{'1'}])$ etcétera.

Tipos Derivados

Con los tipos básicos podemos hacer listas y tuplas. Las “tuplas” (pares, ternas, etc.) se escriben entre paréntesis y tienen tamaño fijo.

1. **Listas**.
   Ejemplos:
   1. $[x, y+z]$ (de tipo $[Num]$),
   2. $[True, p]$ (de tipo $[Bool]$),
   3. ["hola", "chau"] (de tipo $[String]$)
2. **Tuplas**.
   Ejemplos:
   1. $(3, -10, x*2)$ (de tipo (Num, Num, Num)),
   2. $(x*5, True)$ (de tipo $(Num, Bool)$).
   3. ("Juan", 1.75) (de tipo $(String, Num)$),

Comparar los tipos que obtenemos para ambos ejemplos (3) en Haskell.

Se pueden combinar, por ejemplo, listas de listas $[[1,2],[5],[8,9,10]]$ (de tipo $[[Num]]$), listas de pares $[(\texttt{"Juan"}, 1.75), (\texttt{"Jose"}, 1.83)]$, pares de listas $([1,2,4], [\texttt{'b'}, \texttt{'1'}])$ etcétera.

Tipos Derivados

Con los tipos básicos podemos hacer listas y tuplas. Las “tuplas” (pares, ternas, etc.) se escriben entre paréntesis y tienen tamaño fijo.

1. **Listas**.
   Ejemplos:
   1. $[x, y+z]$ (de tipo $[Num]$),
   2. $[True, p]$ (de tipo $[Bool]$),
   3. ["hola", "chau"] (de tipo $[String]$)
2. **Tuplas**.
   Ejemplos:
   1. $(3, -10, x*2)$ (de tipo (Num, Num, Num)),
   2. $(x*5, True)$ (de tipo $(Num, Bool)$).
   3. $(\texttt{"Juan"}, 1.75)$ (de tipo $(String, Num)$),

Comparar los tipos que obtenemos para ambos ejemplos (3) en Haskell.

Se pueden combinar, por ejemplo, listas de listas $[[1,2],[5],[8,9,10]]$ (de tipo $[[Num]]$), listas de pares $[(\texttt{"Juan"}, 1.75), (\texttt{"Jose"}, 1.83)]$, pares de listas $([1,2,4], [\texttt{'b'}, \texttt{'1'}])$ etcétera.

Tipos Derivados

Con los tipos básicos podemos hacer listas y tuplas. Las “tuplas” (pares, ternas, etc.) se escriben entre paréntesis y tienen tamaño fijo.

1. **Listas**.
   Ejemplos:
   1. $[x, y+z]$ (de tipo $[Num]$),
   2. $[True, p]$ (de tipo $[Bool]$),
   3. $[$ `"hola"`, `"chau"` $]$ (de tipo $[String]$)
2. **Tuplas**.
   Ejemplos:
   1. $(3, -10, x*2)$ (de tipo (Num, Num, Num)),
   2. $(x*5, True)$ (de tipo $(Num, Bool)$).
   3. $($`"Juan"`$, 1.75)$ (de tipo $(String, Num)$),

Comparar los tipos que obtenemos para ambos ejemplos (3) en Haskell.

Se pueden combinar, por ejemplo, listas de listas $[[1,2],[5],[8,9,10]]$ (de tipo $[[Num]]$), listas de pares $[($`"Juan"`$, 1.75), ($`"Jose"`$, 1.83)]$, pares de listas $([1,2,4], [$`'b'`$,$`'1'`$])$ etcétera.

Tipos Derivados

Con los tipos básicos podemos hacer listas y tuplas. Las “tuplas” (pares, ternas, etc.) se escriben entre paréntesis y tienen tamaño fijo.

1. **Listas**.
   Ejemplos:
   1. $[x, y+z]$ (de tipo $[Num]$),
   2. $[True, p]$ (de tipo $[Bool]$),
   3. `[ "hola", "chau"]` (de tipo $[String]$)
2. **Tuplas**.
   Ejemplos:
   1. $(3, -10, x*2)$ (de tipo (Num, Num, Num)),
   2. $(x*5, True)$ (de tipo $(Num, Bool)$).
   3. $(\texttt{"Juan"}, 1.75)$ (de tipo $(String, Num)$),

Comparar los tipos que obtenemos para ambos ejemplos (3) en Haskell.

Se pueden combinar, por ejemplo, listas de listas $[[1,2],[5],[8,9,10]]$ (de tipo $[[Num]]$), listas de pares $[(\texttt{"Juan"}, 1.75), (\texttt{"Jose"}, 1.83)]$, pares de listas $([1,2,4], [\texttt{'b'}, \texttt{'1'}])$ etcétera.

Más sobre Listas

Las listas se construyen a partir de $[\,]$ (lista vacía) y de $\triangleright$ (agregar elementos)

En el *Teórico*

$$[2,3] = 2 \triangleright \underline{[3]} = 2 \triangleright (3 \triangleright [\,])$$

En *Haskell*

```
[2,3] = 2 : [3] = 2 : (3 : [])
```

Funciones (“operadores”) de Listas

Pongo una lista de funciones que usaremos.

En el *Teórico*	En *Haskell*
$head$, $tail$, $+\!\!+$, !, $\uparrow$, $\downarrow$, #	`head tail ++ !! take drop length`

Ejemplo

- `head [1,2,4]`
- `tail [1,2,4]`
- `[1] ++ [1,2]`
- `[3,4,5,6] !! 2`
- `take 2 [3,4,5]`
- `drop 2 [3,4,5]`

Ejercicio

Probar más ejemplos y descubrir que hacen estas funciones.
Describir su comportamiento y tipo por escrito

Funciones (“operadores”) de Listas

Pongo una lista de funciones que usaremos.

En el *Teórico*	En *Haskell*
$head$, $tail$, $+\!\!+$, !, $\uparrow$, $\downarrow$, #	`head tail ++ !! take drop length`

Ejemplo

- `head [1,2,4]`
- `tail [1,2,4]`
- `[1] ++ [1,2]`
- `[3,4,5,6] !! 2`
- `take 2 [3,4,5]`
- `drop 2 [3,4,5]`

Ejercicio

Probar más ejemplos y descubrir que hacen estas funciones.
Describir su comportamiento y tipo por escrito

Funciones (“operadores”) de Listas

Pongo una lista de funciones que usaremos.

En el *Teórico*	En *Haskell*
$head$, $tail$, $++$, $!$, $\uparrow$, $\downarrow$, $\#$	`head tail ++ !! take drop length`

Ejemplo

- `head [1,2,4]`
- `tail [1,2,4]`
- `[1] ++ [1,2]`
- `[3,4,5,6] !! 2`
- `take 2 [3,4,5]`
- `drop 2 [3,4,5]`

Ejercicio

Probar más ejemplos y descubrir que hacen estas funciones.

Describir su comportamiento y tipo por escrito

Definiciones de Funciones (I)

Para **definir** una función en el *Teórico* usamos el signo $\doteq$.

Definición de S

$$S.x \doteq x + 1$$

Decidamos si

$$S.(1+1) = S.1 + S.1$$

Definiciones de Funciones (I)

Para **definir** una función en el *Teórico* usamos el signo $\doteq$.

Definición de S

$S.x \doteq x + 1$

$$
\begin{array}{ll}
 & \underline{S.(1+1)} = S.1 + S.1 \\
\equiv \{ & \text{Definición de } S \ \} \\
 & 1+1+1 = \underline{S.1} + \underline{S.1} \\
\equiv \{ & \text{Definición de } S \text{ x2} \ \} \\
 & 1+1+1 = 1+1+1+1 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & 3 = 4 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & \mathit{False}
\end{array}
$$

Definiciones de Funciones (I)

Para **definir** una función en el *Teórico* usamos el signo $\doteq$.

Definición de S

$$S.x \doteq x + 1$$

$$
\begin{array}{ll}
 & \underline{S.(1+1)} = S.1 + S.1 \\
\equiv \{ & \text{Definición de } S \ \} \\
 & 1 + 1 + 1 = \underline{S.1} + \underline{S.1} \\
\equiv \{ & \text{Definición de } S \text{ x2} \ \} \\
 & 1 + 1 + 1 = 1 + 1 + 1 + 1 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & 3 = 4 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & \mathit{False}
\end{array}
$$

Definiciones de Funciones (I)

Para **definir** una función en el *Teórico* usamos el signo $\doteq$.

Definición de S

$$S.x \doteq x + 1$$

$$
\begin{array}{ll}
 & \underline{S.(1+1)} = S.1 + S.1 \\
\equiv \{ & \text{Definición de } S \ \} \\
 & 1+1+1 = \underline{S.1} + \underline{S.1} \\
\equiv \{ & \text{Definición de } S \text{ x2} \ \} \\
 & 1+1+1 = 1+1+1+1 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & 3 = 4 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & \mathit{False}
\end{array}
$$

Definiciones de Funciones (I)

Para **definir** una función en el *Teórico* usamos el signo $\doteq$.

Definición de S

$S.x \doteq x + 1$

$$
\begin{array}{ll}
 & \underline{S.(1+1)} = S.1 + S.1 \\
\equiv \{ & \text{Definición de } S \ \} \\
 & 1+1+1 = \underline{S.1} + \underline{S.1} \\
\equiv \{ & \text{Definición de } S \text{ x2} \ \} \\
 & 1+1+1 = 1+1+1+1 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & 3 = 4 \\
\equiv \{ & \text{Aritmética} \ \} \\
 & \mathit{False}
\end{array}
$$

Definiciones de Funciones (III)

Un ejemplo de función que tiene una lista como argumento.

En el *Teórico*

$$head.(x \triangleright xs) \doteq x$$

En *Haskell*

```
head (x : xs) = x
```

Ejercicio

Aplicarla a $[1,2,3]$, a `[ "hola", "chau"]` y a $[[],[1,2]]$.

Definiciones de Funciones (III)

Un ejemplo de función que tiene una lista como argumento.

En el *Teórico*

$$head.(x \triangleright xs) \doteq x$$

En *Haskell*

```
head (x : xs) = x
```

Ejercicio

Aplicarla a $[1,2,3]$, a `[ "hola", "chau"]` y a $[[],[1,2]]$.

Tarea

1. Escribir las funciones de los ejercicios 4, 5, 6 (temperaturas) y 7(c) (*rangoPrecioParametrizado*) de la Guía Adicional 1.
2. Guardar el archivo con el nombre `apellido_nombre.hs`, todo minúsculas, sin números (ejemplo: `sanchezterraf_pedro.hs`), uno por alumno.
3. Enviarlo a mi email (pedrost arroba gmail punto com).
4. Entregar por escrito la descripción de las funciones, sus tipos y los ejemplos de *head* (evaluar $head.[1,2,3]$, $head.[$ `"hola"`, `"chau"`$]$ y $head.[[],[1,2]]$).

Una vez creado el archivo, se puede abrir con `ghci`.

1. En `ghci`, cargamos nuestro archivo con :l (dos puntos ele). `:l apellido_nombre.hs` Si no anda corregirlo antes de enviar.
2. Podemos ahora probar ejemplos.

Tarea

1. Escribir las funciones de los ejercicios 4, 5, 6 (temperaturas) y 7(c) (*rangoPrecioParametrizado*) de la Guía Adicional 1.
2. Guardar el archivo con el nombre `apellido_nombre.hs`, todo minúsculas, sin números (ejemplo: `sanchezterraf_pedro.hs`), uno por alumno.
3. Enviarlo a mi email (pedrost arroba gmail punto com).
4. Entregar por escrito la descripción de las funciones, sus tipos y los ejemplos de *head* (evaluar $head.[1,2,3]$, $head.[$ `"hola"`, `"chau"`$]$ y $head.[[],[1,2]]$).

Una vez creado el archivo, se puede abrir con `ghci`.

1. En `ghci`, cargamos nuestro archivo con :l (dos puntos ele).
   `:l apellido_nombre.hs` Si no anda corregirlo antes de enviar.
2. Podemos ahora probar ejemplos.

Tarea

1. Escribir las funciones de los ejercicios 4, 5, 6 (temperaturas) y 7(c) (*rangoPrecioParametrizado*) de la Guía Adicional 1.
2. Guardar el archivo con el nombre `apellido_nombre.hs`, todo minúsculas, sin números (ejemplo: `sanchezterraf_pedro.hs`), uno por alumno.
3. Enviarlo a mi email (pedrost arroba gmail punto com).
4. Entregar por escrito la descripción de las funciones, sus tipos y los ejemplos de *head* (evaluar $head.[1,2,3]$, $head.[$ `"hola"`, `"chau"`$]$ y $head.[[],[1,2]]$).

Una vez creado el archivo, se puede abrir con `ghci`.

1. En `ghci`, cargamos nuestro archivo con :l (dos puntos ele).
   `:l apellido_nombre.hs` Si no anda corregirlo antes de enviar.
2. Podemos ahora probar ejemplos.