

# Introducción a la Programación

## Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2025

Departamento de Computación - FCEyN - UBA

Práctica 3: Introducción a Haskell





# Archivos y Carpetas

- ▶ Un archivo tendrá un nombre y una extensión.
- ▶ Por ejemplo, `miPrograma.py`, `notas_de_reunion.txt`
- ▶ La extensión nos indica a los usuarios y al sistema operativo qué tipo de archivo es, por ejemplo: `doc` es un documento de MS Word, `txt` es un archivo de texto plano sin formato, `hs` es un archivo haskell y `py` un archivo de python.
- ▶ Las carpetas pueden almacenar más carpetas y archivos.
- ▶ Las carpetas en Linux se crean con dos estructuras ocultas que indican cómo ir un nivel arriba y cómo marcar la ruta actual.
- ▶ `..` se usa para poder ir a la carpeta madre de la carpeta actual
- ▶ `.` se usa para indicar el path de la carpeta actual

# Uso de una terminal

La terminal es una forma de acceder al sistema de archivos, y ejecutar programas, sin un sistema de ventanas.

- ▶ Para abrir una Terminal vamos a: Terminal > New Terminal; o atajo de teclado Ctrl + Alt + T
- ▶ En la terminal nos mostrará la ubicación actual y el usuario que estamos actualmente en sesión:  
estudiante@pc-1010:~/Documents/

# Uso de una terminal

Algunos comandos útiles para movernos en la terminal (las mayúsculas y minúsculas son importantes!):

**pwd** path of working directory, la ruta completa de la ubicación actual

**ls** lista el directorio actual

**cd** cambiar de directorio, por ejemplo, `cd ..` van al directorio .. (un nivel arriba), `cd Documents` ingresa al directorio Documents (siempre y cuando la carpeta exista en donde estoy actualmente)

**mkdir** crea un directorio (carpeta). Por ejemplo, `mkdir ProgramasHaskell` (Sugerencias: no usen espacios en los nombres)

**rm** para borrar directorios y archivos, por ejemplo `rm -r ProgramasHaskell` (para borrar una carpeta y todo lo que tiene adentro hay que usar `-r` de recursivo)

**touch** para crear un archivo, por ejemplo `touch mi_programa.hs`

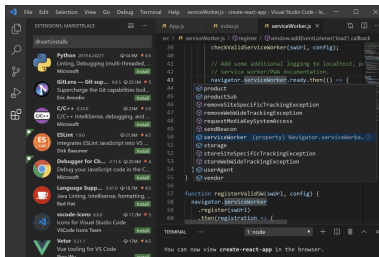
# Ejemplo de comandos

```
petitPrince@PC-B612:~$ cd ~/Documents$ mkdir Programacion_1
petitPrince@PC-B612:~$ cd ~/Documents$ cd Programacion_1
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ ls -l
total 0
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ ls -a
. ..
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ pwd
/home/petitPrince/Documents/Programacion_1
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ touch notas_reunion.txt
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ ls
notas_reunion.txt
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ ls -a
. .. notas_reunion.txt
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ ls -l
total 0
-rw-rw-r-- 1 petitPrince petitPrince 0 abr 3 16:59 notas_reunion.txt
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ ls -la
total 8
drwxrwxr-x 2 petitPrince petitPrince 4096 abr 3 16:59 .
drwxr-xr-x 24 petitPrince petitPrince 4096 abr 3 16:58 ..
-rw-rw-r-- 1 petitPrince petitPrince 0 abr 3 16:59 notas_reunion.txt
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ rm notas_reunion.txt
petitPrince@PC-B612:~$ cd ~/Documents/Programacion_1$ cd ..
petitPrince@PC-B612:~$ rm -r Programacion_1
```

# Configurando Vscode

VS code es una IDE (Integrated Development Environment), existen MUCHAS:

- ▶ Visual Studio (<https://visualstudio.microsoft.com/es/>)
- ▶ Eclipse (<https://www.eclipse.org/>)
- ▶ IntelliJ IDEA (<https://www.jetbrains.com/es-es/idea/>)
- ▶ **Visual Code o Visual Studio Code** (<https://code.visualstudio.com/>)
  - ▶ Es un editor de textos que se “convierte” en IDE mediante *extensions*.
  - ▶ Lo utilizaremos para programar en Haskell y Python.





# Configurando Vscode

**Vamos a instalar la extensión de Haskell:**

- ▶ Abrir Visual Studio Code en sus computadoras

# Configurando Vscode

## Vamos a instalar la extensión de Haskell:

- ▶ Abrir Visual Studio Code en sus computadoras
- ▶ Abrir el buscador apretando ctrl+P (se abre una barra arriba)

# Configurando Vscode

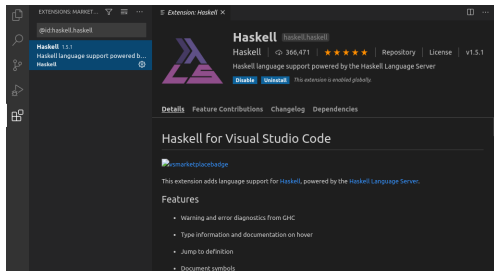
## Vamos a instalar la extensión de Haskell:

- ▶ Abrir Visual Studio Code en sus computadoras
- ▶ Abrir el buscador apretando ctrl+P (se abre una barra arriba)
- ▶ Buscar `ext install haskell.haskell`

# Configurando Vscode

## Vamos a instalar la extensión de Haskell:

- ▶ Abrir Visual Studio Code en sus computadoras
- ▶ Abrir el buscador apretando ctrl+P (se abre una barra arriba)
- ▶ Buscar `ext install haskell.haskell`
- ▶ En la barra de la izquierda se abre el buscador de extensiones con una sola opción encontrada. Hacemos click y la instalamos (si no lo está).



# Configurando Vscode

**Ahora la extensión de Syntax Highlighting:** (si no funciona no es tan grave)

- ▶ Abrir el buscador apretando ctrl+P (se abre una barra arriba)

# Configurando Vscode

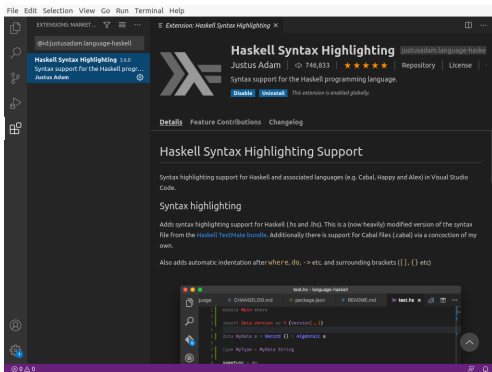
**Ahora la extensión de Syntax Highlighting:** (si no funciona no es tan grave)

- ▶ Abrir el buscador apretando ctrl+P (se abre una barra arriba)
- ▶ Buscar `ext install justusadam.language-haskell`

# Configurando Vscode

**Ahora la extensión de Syntax Highlighting:** (si no funciona no es tan grave)

- ▶ Abrir el buscador apretando `ctrl+P` (se abre una barra arriba)
- ▶ Buscar `ext install justusadam.language-haskell`
- ▶ En la barra de la izquierda se abre el buscador de extensiones con una sola opción encontrada. Hacemos click y la instalamos (si no lo está).



# Primeros pasos en Haskell

**Hagamos nuestro primer programa:**

- ▶ Abrir un archivo nuevo `File > New File`



# Primeros pasos en Haskell

## Hagamos nuestro primer programa:

- ▶ Abrir un archivo nuevo File > New File
- ▶ Definir nuestra primera función: `doubleMe x = x + x`
  - ▶ De qué tipo son los parámetros de entrada y salida de esta función? EN LA MATERIA ES OBLIGATORIO DEFINIRLO!

# Primeros pasos en Haskell

## Hagamos nuestro primer programa:

- ▶ Abrir un archivo nuevo File > New File
- ▶ Definir nuestra primera función: `doubleMe x = x + x`
  - ▶ De qué tipo son los parámetros de entrada y salida de esta función? EN LA MATERIA ES OBLIGATORIO DEFINIRLO!
- ▶ Guardar el archivo como `test.hs`
  - ▶ Es importante recordar dónde lo guardamos
  - ▶ Vamos a guardarlo en Escritorio/guia3/

# Primeros pasos en Haskell

## Hagamos nuestro primer programa:

- ▶ Abrir un archivo nuevo File > New File
- ▶ Definir nuestra primera función: `doubleMe x = x + x`
  - ▶ De qué tipo son los parámetros de entrada y salida de esta función? EN LA MATERIA ES OBLIGATORIO DEFINIRLO!
- ▶ Guardar el archivo como `test.hs`
  - ▶ Es importante recordar dónde lo guardamos
  - ▶ Vamos a guardarlo en Escritorio/guia3/
- ▶ Abrir una Terminal Terminal > New Terminal (o atajo de teclado Ctrl + Alt + T)

# Primeros pasos en Haskell

## Hagamos nuestro primer programa:

- ▶ Abrir un archivo nuevo File > New File
- ▶ Definir nuestra primera función: `doubleMe x = x + x`
  - ▶ De qué tipo son los parámetros de entrada y salida de esta función? EN LA MATERIA ES OBLIGATORIO DEFINIRLO!
- ▶ Guardar el archivo como `test.hs`
  - ▶ Es importante recordar dónde lo guardamos
  - ▶ Vamos a guardarlo en `Escritorio/guia3/`
- ▶ Abrir una Terminal Terminal > New Terminal (o atajo de teclado `Ctrl + Alt + T`)
- ▶ En la terminal asegurarse que estemos en el directorio donde guardamos el archivo
  - ▶ `cd ~/Escritorio/guia3/`

# Primeros pasos en Haskell

## Hagamos nuestro primer programa:

- ▶ Abrir un archivo nuevo File > New File
- ▶ Definir nuestra primera función: `doubleMe x = x + x`
  - ▶ De qué tipo son los parámetros de entrada y salida de esta función? EN LA MATERIA ES OBLIGATORIO DEFINIRLO!
- ▶ Guardar el archivo como `test.hs`
  - ▶ Es importante recordar dónde lo guardamos
  - ▶ Vamos a guardarlo en `Escritorio/guia3/`
- ▶ Abrir una Terminal Terminal > New Terminal (o atajo de teclado `Ctrl + Alt + T`)
- ▶ En la terminal asegurarse que estemos en el directorio donde guardamos el archivo
  - ▶ `cd ~/Escritorio/guia3/`
- ▶ Ahora vamos a abrir el intérprete interactivo de Haskell: `ghci`

# Primeros pasos en Haskell

## Hagamos nuestro primer programa:

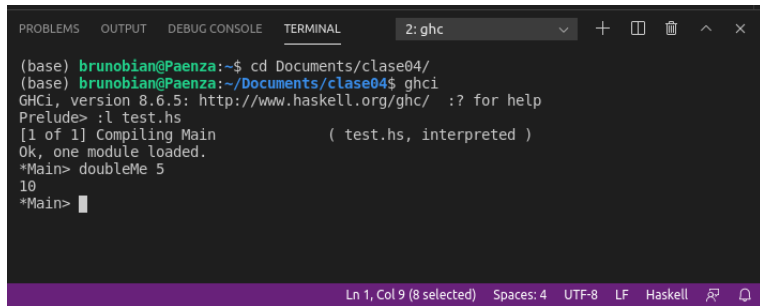
- ▶ Abrir un archivo nuevo File > New File
- ▶ Definir nuestra primera función: `doubleMe x = x + x`
  - ▶ De qué tipo son los parámetros de entrada y salida de esta función? EN LA MATERIA ES OBLIGATORIO DEFINIRLO!
- ▶ Guardar el archivo como `test.hs`
  - ▶ Es importante recordar dónde lo guardamos
  - ▶ Vamos a guardarlo en `Escritorio/guia3/`
- ▶ Abrir una Terminal Terminal > New Terminal (o atajo de teclado `Ctrl + Alt + T`)
- ▶ En la terminal asegurarse que estemos en el directorio donde guardamos el archivo
  - ▶ `cd ~/Escritorio/guia3/`
- ▶ Ahora vamos a abrir el intérprete interactivo de Haskell: `ghci`
- ▶ Dentro del intérprete tenemos que pedirle que cargue nuestro archivo: `:l test.hs`

# Primeros pasos en Haskell

## Hagamos nuestro primer programa:

- ▶ Abrir un archivo nuevo File > New File
- ▶ Definir nuestra primera función: `doubleMe x = x + x`
  - ▶ De qué tipo son los parámetros de entrada y salida de esta función? EN LA MATERIA ES OBLIGATORIO DEFINIRLO!
- ▶ Guardar el archivo como `test.hs`
  - ▶ Es importante recordar dónde lo guardamos
  - ▶ Vamos a guardarlo en Escritorio/guia3/
- ▶ Abrir una Terminal Terminal > New Terminal (o atajo de teclado Ctrl + Alt + T)
- ▶ En la terminal asegurarse que estemos en el directorio donde guardamos el archivo
  - ▶ `cd ~/Escritorio/guia3/`
- ▶ Ahora vamos a abrir el intérprete interactivo de Haskell: `ghci`
- ▶ Dentro del intérprete tenemos que pedirle que cargue nuestro archivo: `:l test.hs`
- ▶ Ahora nuestra función ya existe y podemos usarla: `doubleMe 5`

# Primeros pasos en Haskell



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  2: ghc  +  [icon]  [icon]  ^  x

(base) brunobian@Paenza:~$ cd Documents/clase04/
(base) brunobian@Paenza:~/Documents/clase04$ ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> :l test.hs
[1 of 1] Compiling Main                ( test.hs, interpreted )
Ok, one module loaded.
*Main> doubleMe 5
10
*Main> 
```

Ln 1, Col 9 (8 selected) Spaces: 4 UTF-8 LF Haskell [icon] [icon]



# Algunos comando útiles

- ▶ `ghci` → abrir el intérprete de haskell
- ▶ `:l (:load) file.hs` → cargar el archivo `file.hs`
- ▶ `:r (:reload)` → recargar el último archivo (siempre hacerlo después de modificar el código!)
- ▶ `:q (:quit)` → salir del interprete `ghci`
- ▶ `:t (:type) E` → conocer el tipo de una expresión `E`. Ejemplo `:t (+)`

## **I M P O R T A N T E !**

Sólo se pueden utilizar las funciones que figuran en el campus teo.  
Resolver un ejercicio con una función no habilitada, se dará por no  
realizado dicho ejercicio!

## **I M P O R T A N T E !**

Sólo se pueden utilizar las funciones que figuran en el campus teo.  
Resolver un ejercicio con una función no habilitada, se dará por no realizado dicho ejercicio!

Ya tenemos todo lo necesario para hacer la Guía 3  
**Ahora a programar!!**

# Ejercicio 1

- a) Implementar la función parcial  $f :: \text{Integer} \rightarrow \text{Integer}$  definida por extensión de la siguiente manera:

$$f(1) = 8, f(4) = 131, f(16) = 16$$

cuya especificación es la siguiente:

```
problema f (n:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere:  $\{n = 1 \vee n = 4 \vee n = 16\}$   
  asegura:  $\{(n = 1 \rightarrow result = 8) \wedge (n = 4 \rightarrow result = 131) \wedge (n =$   
     $16 \rightarrow result = 16)\}$   
}
```

- b) Análogamente, especificar e implementar la función parcial  $g :: \text{Integer} \rightarrow \text{Integer}$

$$g(8) = 16, g(16) = 4, g(131) = 1$$

- c) A partir de las funciones definidas en los ítems 1 y 2, implementar las funciones parciales  $h = f \circ g$  y  $k = g \circ f$

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- c) **maximo3**: devuelve el máximo entre tres números enteros.

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

c) **maximo3**: devuelve el máximo entre tres números enteros.

Una posible especificación

```
problema maximo3 ( $x,y,z: \mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: { res es igual a  $x$ , o a  $y$  o a  $z$  }  
  asegura: { res es mayor o igual a  $x$ , y a  $y$ , y a  $z$  }  
}
```

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

c) **maximo3**: devuelve el máximo entre tres números enteros.

Una posible especificación

```
problema maximo3 ( $x,y,z: \mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: { res es igual a  $x$ , o a  $y$  o a  $z$  }  
  asegura: { res es mayor o igual a  $x$ , y a  $y$ , y a  $z$  }  
}
```

Otra forma de especificar, usando lógica

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

c) **maximo3**: devuelve el máximo entre tres números enteros.

Una posible especificación

```
problema maximo3 (x,y,z:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: { res es igual a  $x$ , o a  $y$  o a  $z$  }  
  asegura: { res es mayor o igual a  $x$ , y a  $y$ , y a  $z$  }  
}
```

Otra forma de especificar, usando lógica

```
problema maximo3 (x,y,z:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: {(res = x)  $\vee$  (res = y)  $\vee$  (res = z)}  
  asegura: {(res  $\geq$  x)  $\wedge$  (res  $\geq$  y)  $\wedge$  (res  $\geq$  z)}  
}
```



## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- g) **sumaDistintos:** que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- g) **sumaDistintos:** que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

Esto tiene (al menos) dos interpretaciones posibles:

- ▶ Cuando hay algún número repetido no lo sumo
  - ▶  $\text{sumaDistintos}(1,1,2) = 2$
- ▶ Cuando hay algún número repetido lo sumo una sola vez
  - ▶  $\text{sumaDistintos}(1,1,2) = 3$

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- g) **sumaDistintos**: que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

Esto tiene (al menos) dos interpretaciones posibles:

- ▶ Cuando hay algún número repetido no lo sumo
  - ▶  $\text{sumaDistintos}(1,1,2) = 2$
- ▶ Cuando hay algún número repetido lo sumo una sola vez
  - ▶  $\text{sumaDistintos}(1,1,2) = 3$

Una posible especificación de la primera opción

```
problema sumaDistintos (x,y,z:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: { - }  
  asegura: {si los 3 parámetros son distintos entonces  $res = x + y + z$ }  
  asegura: {si 2 parámetros son iguales, res es igual al no repetido}  
  asegura: {si los 3 parámetros son iguales,  $res = 0$ }  
}
```

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- g) **sumaDistintos**: que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

Esto tiene (al menos) dos interpretaciones posibles:

- ▶ Cuando hay algún número repetido no lo sumo
  - ▶  $\text{sumaDistintos}(1,1,2) = 2$
- ▶ Cuando hay algún número repetido lo sumo una sola vez
  - ▶  $\text{sumaDistintos}(1,1,2) = 3$

Otra especificación de la primera opción

```
problema sumaDistintos (x,y,z:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: {(  $x \neq y$ )  $\wedge$  ( $x \neq z$ )  $\wedge$  ( $y \neq z$ ) }  $\rightarrow$  res =  $x + y + z$ }  
  asegura: {(  $x = y$ )  $\wedge$  ( $x \neq z$ )  $\wedge$  ( $y \neq z$ ) }  $\rightarrow$  res =  $z$ }  
  asegura: {(  $x \neq y$ )  $\wedge$  ( $x = z$ )  $\wedge$  ( $y \neq z$ ) }  $\rightarrow$  res =  $y$ }  
  asegura: {(  $x \neq y$ )  $\wedge$  ( $x \neq z$ )  $\wedge$  ( $y = z$ ) }  $\rightarrow$  res =  $x$ }  
  asegura: {(  $x = y$ )  $\wedge$  ( $x = z$ )  $\wedge$  ( $y = z$ ) }  $\rightarrow$  res =  $0$ }  
}
```

# Ej 7

Algunas operaciones útiles para manipular enteros:

► mod:

►  $\text{mod } 8123 \ 10 = ?$

►  $\text{mod } 2142 \ 10 = ?$

►  $\text{mod } 4 \ 10 = ?$

► div:

►  $\text{div } 8123 \ 10 = ?$

►  $\text{div } 2142 \ 10 = ?$

►  $\text{div } 4 \ 10 = ?$

## Ej 7

Algunas operaciones útiles para manipular enteros:

► mod:

►  $\text{mod } 8123 \ 10 = 3$

►  $\text{mod } 2142 \ 10 = 2$

►  $\text{mod } 4 \ 10 = 4$

► div:

►  $\text{div } 8123 \ 10 = 812$

►  $\text{div } 2142 \ 10 = 214$

►  $\text{div } 4 \ 10 = 0$

## Ej 7

Algunas operaciones útiles para manipular enteros:

► mod:

►  $\text{mod } 8123 \ 10 = 3$

►  $\text{mod } 2142 \ 10 = 2$

►  $\text{mod } 4 \ 10 = 4$

► div:

►  $\text{div } 8123 \ 10 = 812$

►  $\text{div } 2142 \ 10 = 214$

►  $\text{div } 4 \ 10 = 0$

**mod n 10** → me da el ultimo digito de  $n$ .

**div n 10** → le *saca* el ultimo digito a  $n$ .

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- i) **digitoUnidades:** dado un número entero, extrae su dígito de las unidades.



## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- i) **digitoUnidades:** dado un número entero, extrae su dígito de las unidades.

```
problema digitoUnidades (x:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: { result es el último dígito de x }  
}
```

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- j) **digitoDecenas:** dado un número entero, extrae su dígito de las decenas.

## Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

- j) **digitoDecenas**: dado un número entero, extrae su dígito de las decenas.

```
problema digitoDecenas (x:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: { result es el dígito de x correspondiente a las decenas }  
}
```

Ejercicio 4: Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números.

- b) **todoMenor:** dadas dos tuplas  $\mathbb{R} \times \mathbb{R}$ , decide si es cierto que cada coordenada de la primera tupla es menor a la coordenada correspondiente de la segunda tupla.

## Ejercicio 4: Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números.

- b) **todoMenor**: dadas dos tuplas  $\mathbb{R} \times \mathbb{R}$ , decide si es cierto que cada coordenada de la primera tupla es menor a la coordenada correspondiente de la segunda tupla.

```
problema todoMenor (t1, t2:  $\mathbb{R} \times \mathbb{R}$ ) : Bool {  
  requiere: {True}  
  asegura: { result = true  $\leftrightarrow$  la primera componente de t1 es menor  
             que la primera componente de t2, y la segunda componente de  
             t1 es menor que la segunda componente de t2 }  
}
```