

# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO



## FACULTAD DE CIENCIAS

ESTRUCTURAS DISCRETAS  
2025-1

### PRÁCTICA 1. FUNCIONES Y CONDICIONALES EN HASKELL

**Profesor:**

Ulises Rodríguez Domínguez

**Ayudantes de la materia:**

Irvin Javier Cruz González

Raúl Eduardo Martínez Dámaso

## Contexto

Como ya vimos en clase, crear una función en haskell es similar a como escribimos una función en la vida real, es decir, nosotros hacemos lo siguiente

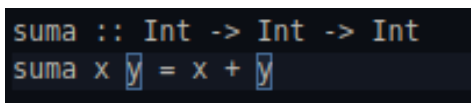
$$f(p_1, p_2, p_3, \dots, p_4) = x$$

Donde primero lo que se tiene que hacer es crear la firma de la función, empezando por el **nombre** que tendrá la función, seguido de `::` y después añadiendo los `n` parámetros de algún tipo existente en Haskell separados por `→` y el valor que devolverá la función de algún tipo existente en Haskell precedido de un `→`.

Supongamos que queremos programar una función que realice la suma de dos números enteros, entonces lo que haríamos sería lo siguiente

$$\text{suma}(x, y) = x + y$$

Donde notamos que el nombre de la función es `suma` (igual pueden poner el que ustedes les parezca mejor según sea el caso), tenemos dos parámetros y nuestra función devuelve la suma de estos dos parámetros. A continuación se anexa un ejemplo



```
suma :: Int -> Int -> Int
suma x y = x + y
```

Figure 1: Función que suma dos números enteros.

**NOTA:** Recuerden que deben mantener la coherencia de lo que devuelva su función, en el anterior ejemplo se calcula la suma de dos números enteros, por lo que la función debe devolver un número entero.

Ahora que ya sabemos los pasos para crear funciones en Haskell, veamos que como en todo lenguaje de programación existe la estructura condicional **if-then-else** que nos servirá a lo largo del curso.

La sintaxis de dicha estructura es la siguiente:

- Se comienza por declarar la palabra **if** seguido de alguna condición que devuelva algo de tipo **Bool**.
- Seguido de la palabra **then** que lo que quiere decir es que si se cumple la condición del **if**, se ejecuta el código que esté antes del **else**
- Y por último, si la condición del **if** no se llega a cumplir se ejecuta el código que esté después del **else**

A continuación un ejemplo del uso de la estructura

```

diasDeLaSemana :: Int -> String
diasDeLaSemana x = if x == 1
                    then "Lunes"
                    else if x == 2
                        then "Martes"
                        else if x == 3
                            then "Miercoles"
                            else ""

```

Figure 2: Función que dependiendo del número nos devuelve el día de la semana correspondiente a dicho número.

## Objetivos

El objetivo de esta práctica es que los alumnos desarrollen la habilidad de saber interpretar los datos de una función para poder crear esa misma función pero en el lenguaje de programación de Haskell y además saber utilizar la estructura condicional if-then-else que será utilizada a lo largo del curso.

## Ejercicios

### 1. Distancia entre dos puntos en el plano cartesiano:

La función **distanciaPuntos** debe recibir dos parámetros que serán tuplas de dos elementos de tipo flotante respectivamente, es decir,  $(x_1, y_1)$  y  $(x_2, y_2)$ . La función debe devolver un valor de tipo flotante que represente la distancia entre los puntos  $(x_1, y_1)$  y  $(x_2, y_2)$ .

**NOTA 1:** La distancia entre dos puntos está definida como

$$distancia = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

**NOTA 2:** Una tupla en haskell se declara como:

$$(Tipo1, Tipo2)$$

donde *Tipo1* y *Tipo2* son tipos existentes en haskell, más en específico los tipos mencionados en los ejercicios y así como se declara, con la misma sintaxis se devuelve.

```

tuplaPorDos :: (Int, Int) -> (Int, Int)
tuplaPorDos (x, y) = (x*2, x*2)

```

Figure 3: Función que recibe una tupla con dos parámetros enteros y devuelve la misma tupla pero con los elementos multiplicados por dos.

### 2. Hipotenusa de un triángulo rectángulo:

La función **hipotenusa** debe recibir dos parámetros de tipo flotante **b** y **h** donde b representa la base y h la altura. La función debe devolver un valor de tipo flotante que represente el valor de la hipotenusa que se calcula respecto a la base y altura del triángulo rectángulo.

**NOTA:** La hipotenusa de un triángulo rectángulo está definida como

$$hipotenusa = \sqrt{b^2 + h^2}$$

### 3. Pendiente de la recta que pasa por dos puntos:

La función **pendiente** debe recibir dos parámetros que serán tuplas de dos elementos de tipo flotante respectivamente, es decir,  $(x_1, y_1)$  y  $(x_2, y_2)$ . La función debe devolver un valor de tipo flotante que represente la pendiente de la recta que pasa por dos puntos.

**NOTA:** La pendiente de una recta que pasa por dos puntos está definida como

$$pendiente = \frac{y_2 - y_1}{x_2 - x_1}$$

### 4. Raíces de una ecuación cuadrática:

La función **raices** debe recibir tres parámetros flotantes **a**, **b** y **c** que representan la ecuación

$$ax^2 + bx + c = 0$$

. La función debe devolver una tupla de dos elementos de tipo flotante que representan las raíces de una ecuación cuadrática.

**NOTA:** Para calcular las raíces cuadráticas se puede utilizar la fórmula general de segundo grado que está definida como

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

### 5. Área de un triángulo por medio de la fórmula de Herón:

La función **areaTriangulo** recibe tres parámetros de tipo flotante **x**, **y** y **z** que representan las longitudes de los tres lados de un triángulo. La función debe devolver un valor de tipo flotante que representa el área de un triángulo calculado por medio de la fórmula de Herón.

**NOTA:** El área de un triángulo por medio de la fórmula de Herón está definida de la siguiente manera:

$$Area = \sqrt{S(S-a)(S-b)(S-c)}$$

donde S es el semiperímetro del triángulo definido como

$$S = \frac{a + b + c}{2}$$

### 6. Función comparador:

La función **comparador** recibe dos parámetros de tipo entero **x** y **y**. Y la función devuelve un valor específico de tipo entero de acuerdo a los siguientes casos:

6.1 **0**, si x es igual a y.

6.2 **-1**, si x es menor a y.

6.3 **1**, si x es mayor a y.

### 7. Maximo entre tres números:

La función **maximo** debe recibir los parámetros **x**, **y** y **z**. La función debe devolver el entero máximo entre x, y y z.

### 8. Números ordenados de forma descendente:

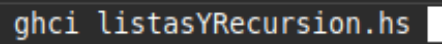
La función **esDescendente** recibe cuatro parámetros de tipo entero **x**, **y**, **z** y **w**. La función debe devolver un valor de tipo booleano de acuerdo a los siguientes casos:

8.1. True, si los números fueron ingresados de manera descendente.

8.2. False, si los números no fueron ingresados de manera descendente.

## Compilación de código

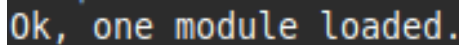
1. Para compilar su archivo `.hs`, lo primero que tienen que hacer es posicionarse dentro de la terminal en la que se encuentran en la carpeta donde se encuentra su archivos `.hs`
2. Ejecutan el comando **ghci** seguido del nombre del archivo como se muestra a continuación:



```
ghci listasYRecursion.hs
```

Figure 4: Función que dependiendo del número nos devuelve el día de la semana correspondiente a dicho número.

3. Si ghci compiló correctamente su código les tiene que mostrar el siguiente mensaje:



```
Ok, one module loaded.
```

Figure 5: Función que dependiendo del número nos devuelve el día de la semana correspondiente a dicho número.

## Entrega

La fecha de entrega será el día **30 de Agosto a las 23:59** horas por medio de classroom y se adjuntará el link del repositorio de GitHub (el repositorio tendrá que ser público).

Además dentro de un comentario privado tendrán que escribir los integrantes del equipo (Equipos de 2).