PROGRAMACIÓN LÓGICA Y FUNCIONAL

SESIÓN 03

CICLO: AGOSTO 2020

Universidad Tecnológica del Perú

Contenido

- Bienvenida
- Pautas de trabajo
- Elabora programas con funciones de control
- Laboratorio

Universidad Tecnológica del Perú

Pautas de trabajo

- Los días que tengamos clases debemos conectarnos a través de Zoom.
- La participación de los estudiantes se dará través del **chat de Zoom.**
- En Canvas encontrarán la clase de hoy, el ppt de la sesión 03, Laboratorio 03

Universidad Tecnológica del Perú

Recordando la sesión anterior

¿ Qué se entiende por tipo de datos?

Levantemos la mano para participar

Logro de la sesión

Al finalizar la presente sesiónel el estudiante conoce funciones y funciones de control

Utilidad

¿“Porque es importante conocer funciones y las funciones de control?

Utiliza el chat para participar

Booleanos

- Se representan por el tipo "Bool" y contienen dos valores: "True" y "False". El standar prelude incluye varias funciones para manipular valores booleanos: (&&), (||) y not.

Ejemplo:

True, False

Enteros

- Representados por el tipo "Int", se incluyen los enteros positivos y negativos tales como el -273, el 0 ó el 383. Como en muchos sistemas, el rango de los enteros utilizables está restringido. También se puede utilizar el tipo Integer que denota enteros sin límites superior ni inferior.

  Ejemplo:

  1,2,3,…

Flotantes

- Representados por el tipo "Float", los elementos de este tipo pueden ser utilizados para representar fraccionarios así como cantidades muy largas o muy pequeñas. Sin embargo, tales valores son sólo aproximaciones a un número fijo de dígitos y pueden aparecer errores de redondeo en algunos cálculos que empleen operaciones en punto flotante. Un valor numérico se toma como un flotante cuando incluye un punto en su representación o cuando es demasiado grande para ser representado por un entero. También se puede utilizar notación científica; por ejemplo 1.0e3 equivale a 1000.0, mientras que 5.0e-2 equivale a 0.05.

Ejemplo:

1.2 , 1.4 , 55.3 , ….

Caracteres

- Representados por el tipo "Char", los elementos de este tipo representan caracteres individuales como los que se pueden introducir por teclado. Los valores de tipo caracter se escriben encerrando el valor entre comillas simples, por ejemplo 'a', '0', '.' y 'Z'. Algunos caracteres especiales deben ser introducidos utilizando un código de escape; cada uno de éstos comienza con el caracter de barra invertida (\) , seguido de uno o más caracteres que seleccionan el caracter requerido.

Ejemplo:

‘a’, ’b’, ‘c’,…

Listas

- Si a es un tipo cualquiera, entonces [a] representa el tipo de listas cuyos elementos son valores de tipo a.
- Hay varias formas de escribir expresiones de listas:
  - La forma más simple es la lista vacía, representada mediante [].
  - Las listas no vacías pueden ser construidas enunciando explícitamente sus elementos (por ejemplo, [1,3,10]) o añadiendo un elemento al principio de otra lista utilizando el operador de construcción (:). Estas notaciones son equivalentes:

    [1,3,10] = 1:[3,10] = 1:(3:[10]) = 1:(3:(10:[]))
  - El operador (:) es asociativo a la derecha, de forma que 1:3:10:[] equivale a (1:(3:(10:[]))), una lista cuyo primer elemento es 1, el segundo 3 y el último 10.

Las listas se representan mediante la notación de corchetes.
La lista

$$[1, 2, 3, 4, 5]$$

es de tipo [Int]
La lista

$$[1, 2, 'a', 'b']$$

es inválida porque no existe un tipo que se le pueda asignar.

Cadenas

- Una cadena es tratada como una lista de caracteres y el tipo "String" se toma como una abreviación de "[Char]". Las cadenas pueden ser escritas como secuencias de caracteres encerradas entre comillas dobles. Todos los códigos de escape utilizados para los caracteres, pueden utilizarse para las cadenas.

  Ejemplo:

  “Hola”, “Adios”

TIPOS DE DATOS

Haskell es fuertemente tipificado y podemos utilizar los siguientes tipos básicos:

Tipo	Nombre	Ejemplo
Int	entero	5
Fractional	fracciones	3/4
Float	flotante	3.141592
(a,b)	tuplas	(1,2)
Char	caracter	'a'
[Char]	cadena	"Hola"
		(que es una lista de caracteres 'H' 'o' 'l' 'a')

Para saber el tipo de una expresión: `:type [1,2]` ⇒ Tipo de la expresión

Tuplas

- Si t1, t2, ..., tn son tipos y n>=2, entonces hay un tipo de n-tuplas escrito (t1, t2, ..., tn) cuyos elementos pueden ser escritos también como (x1, x2, ..., xn) donde cada x1, x2, ..., xn tiene tipos t1,t2, ..., tn respectivamente.
- Ejemplo: (1, [2], 3) :: (Int, [Int], Int) ('a', False) :: (Char, Bool) ((1,2),(3,4)) :: ((Int, Int), (Int, Int))
- Obsérvese que, a diferencia de las listas, los elementos de una tupla pueden tener tipos diferentes. Sin embargo, el tamaño de una tupla es fijo.
- En determinadas aplicaciones es útil trabajar con una tupla especial con 0 elementos denominada tipo unidad. El tipo unidad se escribe como () y tiene un único elemento que es también ().

Ejemplo:
[(“a”,1),(”b”,2)]

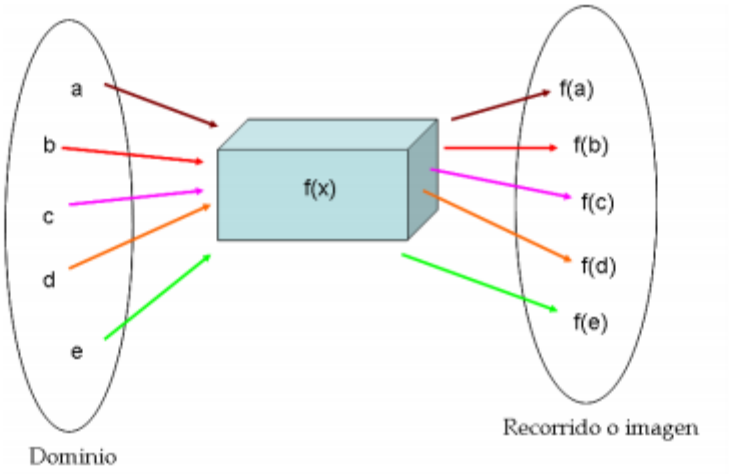
Funciones

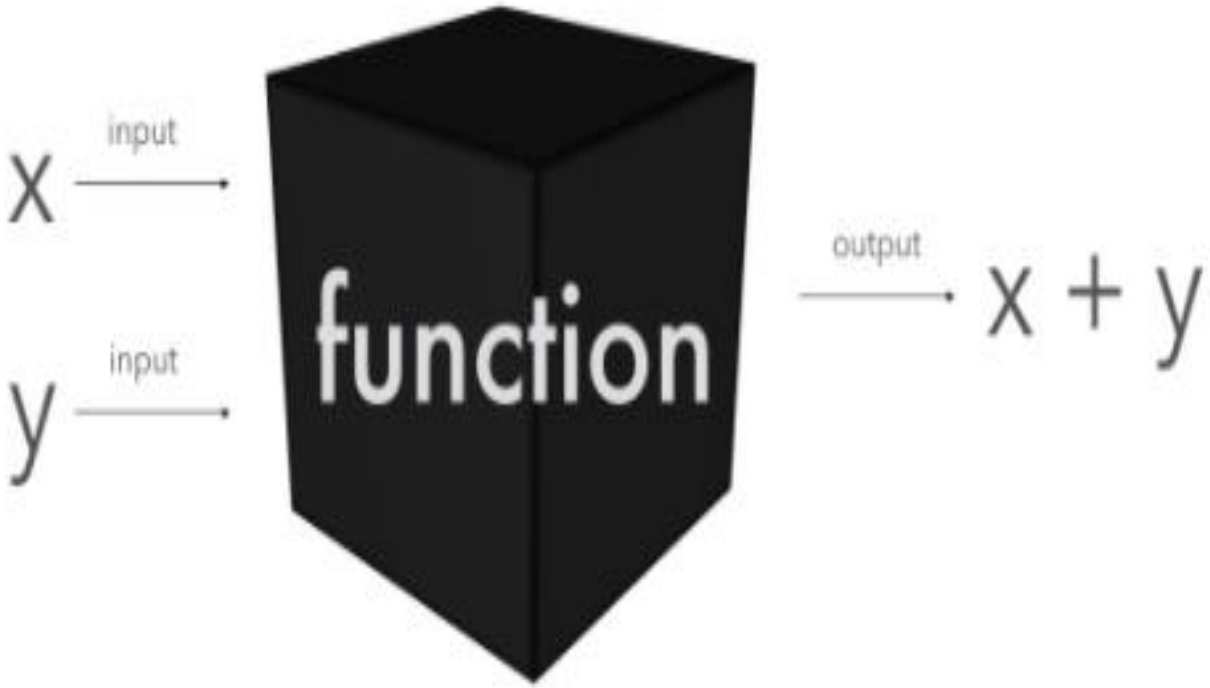
- Si a y b son dos tipos, entonces a->b es el tipo de una función que toma como argumento un elemento de tipo a y devuelve un valor de tipo b. Las funciones en Haskell son objetos de primera clase. Pueden ser argumentos o resultados de otras funciones o ser componentes de estructuras de datos. Esto permite simular mediante funciones de un único argumento, funciones con múltiples argumentos.

Ejemplo:

even(7) = false , show(44)=“44”

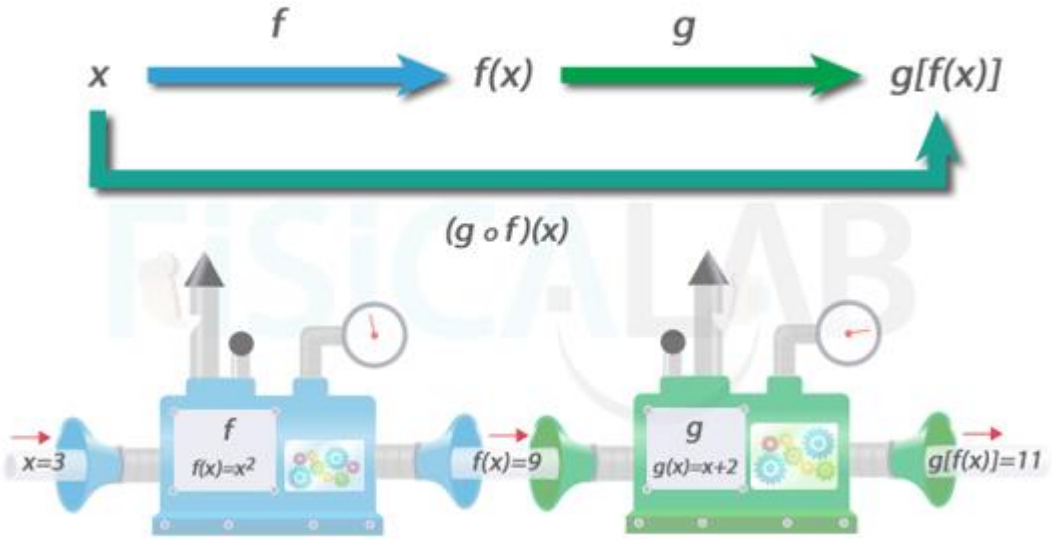
FUNCIÓN




x
input
y
input
function
output
x + y

COMPOSICIÓN DE FUNCIONES

Sean dos funciones f(x) y g(x). La composición de f con g se define como
$(f \circ g)(x) = f[g(x)]$



f g x = g(f(x))

Funciones

$$f : \mathcal{A} \to \mathcal{B}$$
$$f(x) \mapsto \ldots$$

Ejemplos: *sucesor*, *sumaCuadrados* (2 argumentos) y *pi* (constante)

$$sucesor : \mathbb{Z} \to \mathbb{Z} \quad | \quad sumaCuadrados : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \quad | \quad \pi : \mathbb{R}$$
$$sucesor(x) \mapsto x+1 \quad | \quad sumaCuadrados(x,y) \mapsto x^2+y^2 \quad | \quad \pi \mapsto 3.1415927\ldots$$

✓ Evaluación de una función para ciertos valores de la variable:

$$sucesor(1) \Longrightarrow 2 \quad sumaCuadrados(2,3) \Longrightarrow 13$$

No importa el orden en que se evalúe una función, siempre se obtiene el mismo resultado.
Es decir que podemos evaluar una expresión en cualquier orden y siempre obtendremos el mismo resultado. Dadas

$$f(x) = x * x + 2$$
$$g(y) = y + y$$

$$\begin{aligned} f(g(5)) &= g(5) * g(5) + 2 = 10 * 10 + 2 = 102 \\ &= f(5+5) = f(10) = 10 * 10 + 2 = 102 \end{aligned}$$

EJEMPLO USO DE FUNCIÓN

La definición de las siguientes funciones

$$f\ x = x + 2$$

$$fact(n) = n \times fact(n-1)$$
$$fact(0) = 1$$

se hace en Haskell de la siguiente manera

```
f x = x + 2

fact (n) = n * fact (n-1)
fact (0) = 1
```

EJEMPLO USO DE FUNCIÓN

Funciones

Area de un círculo

```
circleArea r = pi * r^ 0
```

Secuencia de Fibonacci

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

ESTRUCTURA DE CONTROL: if

No hay *estructuras de control*, pero si existe la función especial **if - then - else** .

Ejemplo

Utilización de la función **if-then-else** *en Haskell.*

```
f x = if x > 2 then 0 else 1

g y = if y > 0 then 3
```

Nota: La construcción del **if-then** no tiene sentido, ya que siempre debe haber algo con lo cual se pueda reescribir

$$g\ (-1)$$

no tiene con que reescribirse.

Comparación de Patrones

- Un patrón es una expresión como argumento en una ecuación
- Es posible definir una función dando más de una ecuación para ésta.

✓ Al aplicar la función a un parámetro concreto la *comparación de patrones* determina la ecuación a utilizar.

Patrones para listas

Es posible utilizar patrones al definir funciones que trabajen con listas.

- $[]$ sólo unifica con un argumento que sea una lista vacía.
- $[x]$, $[x, y]$, etc. sólo unifican con listas de uno, dos, etc. argumentos.
- $(x : xs)$ unifica con listas con al menos un elemento

```
suma          :: [Integer] → Integer
suma []       = 0
suma (x : xs) = x + suma xs
```

```
primero (x:xs) = x
cola (x:xs) = xs
```

Recursividad en Haskell?

- La recursión es muy importante en Haskell ya que, al contrario que en los lenguajes imperativos, realizamos cálculos declarando como es algo, en lugar de declarar como obtener algo. Por este motivo no hay bucles while o bucles for en Haskell y en su lugar tenemos que usar la recursión para declarar como es algo.

Qué hemos aprendido el día de hoy?

Utiliza el chat para participar

Actividad grupal

Universidad Tecnológica del Perú

LABORATORIO 03

Universidad Tecnológica del Perú

Universidad Tecnológica del Perú