Introducción a los Algoritmos 2C 2025

Práctico 3: Funciones y tuplas

Funciones

En matemática, se dice que una magnitud o cantidad es función de otra si el valor de la primera depende exclusivamente del valor de la segunda. Por ejemplo, supongamos que queremos viajar desde Córdoba a Buenos Aires en auto. La duración del viaje (t) dependerá (es función) de la distancia (d) entre Córdoba y Buenos Aires y de la velocidad (v) que lleve el auto. Del mismo modo, el área (a) de un círculo es función de su radio (r). Estas magnitudes a veces pueden vincularse de diferentes maneras. Así, el área (a) de un círculo es proporcional al cuadrado de su radio, lo que se expresa con la fórmula **`a = π * r^2`** y la duración de un viaje (t) es inversamente proporcional a la velocidad del vehículo (**`t = d/v`**). A la variable que se encuentra a la izquierda de estas fórmulas (el área, la duración) se la denomina variable dependiente, y a las variables de las que depende (el radio, la velocidad, la distancia) se las llama variables independientes.

De manera más abstracta, el concepto general de función se refiere a una regla que asigna a cada elemento de un primer conjunto (dominio) un único elemento de un segundo conjunto (codominio). Por ejemplo, cada número entero posee un único cuadrado, que resulta ser un número natural (incluyendo el cero):

$$\begin{array}{ccccccccc} \ldots & -2 & -1 & 0 & 1 & 2 & 3 & \ldots \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & +4 & +1 & 0 & +1 & +4 & +9 & \end{array}$$

Esta asignación constituye una función entre el conjunto de los números enteros y el conjunto de los naturales, si interpretamos los puntos suspensivos como una serie.

Diremos que una **función** relaciona los elementos de un conjunto, llamado dominio, elementos de otro conjunto, llamado codominio. La notación:

$$\mathbf{f : A \to B}$$

indica que **`f`** es una función con dominio **`A`** y codominio **`B`**.

La forma más tradicional que conocemos para definir una función (describir la relación entre elementos del dominio y el codominio) es una definición algebraica, que tiene la forma

`f.x ≐ <expresión que depende de x>`

donde `f` es el nombre de la función y `x` es la variable independiente. Por ejemplo, si consideramos la función duración de un viaje (duración) que depende de la velocidad (v) y la distancia (d), en un movimiento rectilíneo uniforme, podríamos definirla de la siguiente manera:

```
duracion.d.v ≐ d/v
```

En matemática acostumbramos a escribir esta misma expresión con paréntesis indicando las variables independientes: duracion (d,v) = d/v

Notar que para diferenciar la definición, del operador de relación "=" que comparaba dos expresiones y daba de resultado un booleano, utilizamos el símbolo "≐" para la definición.

Ejercicio 1: En las siguientes definiciones identificá las variables, las constantes y el nombre de la función

a) f.x ≐ 5 * x

b) duplica.a ≐ a + a

c) por2.y ≐ 2 * y

d) multiplicar.zz.tt ≐ zz * tt

Ejercicio 2: Escribí una función que dados dos valores, calcule su promedio. Luego definila en Haskell (https://repl.it/languages/haskell).

Ejercicio 3: Tomando las definiciones del punto 1 evaluá las siguientes expresiones. Justificá cada paso utilizando la notación aprendida. Luego, controlá los resultados en Haskell (https://repl.it/languages/haskell).

a) (multiplicar.(f.5).2) + 1

b) por2.(duplica.(3 + 5))

Ejercicio 4: Tomando las definiciones en el punto 2 demostrá que duplica y por2, si son aplicadas al mismo valor, dan siempre el mismo resultado. En otras palabras, la expresión duplica.x = por2.x es válida.

Tipado de funciones

Tomemos la función g

$$g.x.y \doteq 3*x - x * y > 0$$

Esta función toma dos argumentos de tipo Num y devuelve un valor de tipo Bool. Así, el tipo de g se declara de la siguiente forma:

$$g : Num \rightarrow Num \rightarrow Bool$$

Es decir, los tipos de los argumentos se listan primero, siguiendo el orden en el que serán llamados por la función, y en último lugar se coloca el tipo del resultado de evaluar la función.

Nota: Haskell indica el tipo de una función anteponiendo el comando :t al nombre de la función.

Ejercicio 5: Dar el tipo de las funciones del ejercicio 1 y el ejercicio 2.

Ejercicio 6: Dar el tipo de las siguientes funciones:

a) g.y ≐ 8 * y

b) h.z.w ≐ z + w

c) j.x ≐ x ≤ 0

Definiciones de Funciones por casos

Hay ocasiones en las que una sola fórmula no alcanza para definir una función. Así, existen funciones que para una parte del dominio requieren una definición y para otra parte del dominio necesitan de otra definición diferente. Es el caso, por ejemplo, de la función valor absoluto que dado un número devuelve su valor absoluto.

Una definición por casos de una función tendrá la siguiente forma general:

f.x ≐ ($B_0 \rightarrow f_0$
□ $B_1 \rightarrow f_1$
...
□ $B_n \rightarrow f_n$
)

donde las B_i son expresiones de tipo booleano, llamadas guardas, y las f_i son expresiones del mismo tipo que el resultado de f que nos describe algebraicamente la relación para cada caso. Para un argumento dado el valor de la función se corresponde con la expresión cuya guarda es verdadera para ese argumento.

Al trabajar con expresiones booleanas utilizaremos de nuestro formalismo los operadores booleanos, en particular $\wedge$, $\vee$, $\neg$ que pueden ser utilizados para combinar dos fórmulas para obtener una nueva fórmula. En haskell estos operadores se escriben && , || y not, respectivamente.

Ejercicio 7: Definí las funciones que describimos a continuación, luego implementarlas en haskell. Por ejemplo:

Enunciado: Definí la función `signo : Int → Int`, que dado un entero retorna su signo, de la siguiente forma: retorna 1 si x es positivo, -1 si es negativo y 0 en cualquier otro caso.

Solución:
signo.x ≐ (x > 0 → 1
□ x < 0 → -1
□ x = 0 → 0
)

Esta definición, en haskell se anota de la siguiente manera

```
signo :: Int -> Int
signo x   | x > 0 = 1
          | x < 0 = -1
          | x == 0 = 0
```

a) entre0y9 : Int → Bool, que dado un entero devuelve True si el entero se encuentra entre 0 y 9.

b) rangoPrecio : Int → String, que dado un número que representa el precio de una computadora, retorne “muy barato” si el precio es menor a 2000, “demasiado caro” si el precio es mayor que 5000, “hay que verlo bien” si el precio está entre 2000 y 5000, y “esto no puede ser!” si el precio es negativo.

c) absoluto : Int → Int, que dado un entero retorne su valor absoluto.

d) esMultiplo2 : Int → Bool, que dado un entero n devuelve True si n es múltiplo de 2.

Ayuda: usar mod, el operador que devuelve el resto de la división.

Combinando Funciones

En algunos ejercicios que siguen se van a utilizar algunas de las funciones que están en el Prelude (librería básica de Haskell). Por ejemplo:

mod 20 3 = 2 – el resto de la división entre 20 y 3 es 2.
div 14 3 = 4 – parte entera de la división entre 14 y 3 es 4.
max 8 10 = 10 – devuelve el max entre 2 números.
min 9 15 = 9 – devuelve el min entre 2 números.

Ejercicio 8: Definí la función esMultiploDe : Num→ Num→ Bool , que devuelve True si el segundo es múltiplo del primero. Ejemplo: esMultiploDe 3 12 = True.

Ejercicio 9: Definí la función esBisiesto: Num→ Bool , que indica si un año es bisiesto. Un año es bisiesto si es divisible por 400 o es divisible por 4 pero no es divisible por 100.

Ejercicio 10: Definí la función dispersion : Num→ Num→ Num→ Num, que toma los tres valores y devuelve la diferencia entre el más alto y el más bajo. Ayuda: extender max y min a tres argumentos, usando las versiones de dos elementos. De esa forma se puede definir dispersión sin escribir ninguna guarda (las guardas están en max y min, que estamos usando).

Ejercicio 11: Definí la función celsiusToFahr : Num→ Num, pasa una temperatura en grados Celsius a grados Fahrenheit. Para realizar la conversión hay que multiplicar por 1.8 y sumar 32.

Ejercicio 12: Definí la función fahrToCelsius : Num→ Num, la inversa de la anterior. Para realizar la conversión hay que primero restar 32 y después dividir por 1.8.

Ejercicio 13: Definí la función haceFrioF : Num→ Bool , indica si una temperatura expresada en grados Fahrenheit es fría. Decimos que hace frío si la temperatura es menor a 8 grados Celsius.

Tuplas

Una manera de formar un nuevo tipo es combinando los otros ya existentes en tuplas (i.e., haciendo su producto cartesiano). El ejemplo más conocido es, quizás, el de un par de números como (3,2). (3,2) es un elemento de tipo (Num, Num). Por ejemplo, podemos definir una función que toma dos pares y los suma coordenada a coordenada de la siguiente forma

sumaPares : (Num,Num) → (Num,Num) → (Num,Num)
sumaPares.(a, b).(c, d) ≐ (a+c, b+d)

Ejercicio 14: Definí las funciones que describimos a continuación, luego implementalas en Haskell.

a) segundo3 : (Num, Num, Num) → Num, que dada una terna de enteros devuelve su segundo elemento.

b) ordena : (Num, Num) → (Num, Num), que dados dos enteros los ordena de menor a mayor.

c) rangoPrecioParametrizado : Num → (Num, Num) → String que dado un número x, que representa el precio de un producto, y un par (menor, mayor) que represente el rango de precios que uno espera encontrar, retorne “muy barato” si x está por debajo del rango, “demasiado caro” si está por arriba del rango, “hay que verlo bien” si el precio está en el rango, y “esto no puede ser!” si x es negativo.

d) mayor3 : (Num, Num, Num) → (Bool , Bool , Bool), que dada una una terna de enteros devuelve una terna de valores booleanos que indica si cada uno de los enteros es mayor que 3.

Por ejemplo: mayor3.(1, 4, 3) = (False, True, False) y mayor3.(5, 1984, 6) = (True, True, True)

e) todosIguales : (Num, Num, Num) → Bool que dada una terna de enteros devuelva True si todos sus elementos son iguales y False en caso contrario.

Por ejemplo: todosIguales.(1, 4, 3) = False y todosIguales.(1, 1, 1) = True

f) notaPromedio : (String, Num, Num, Num) → Num que dada una tupla que representa el nombre de un/a estudiante y las notas que sacó en cada parcial, calcule la nota promedio.

Por ejemplo: notaPromedio.(“Joaquín”, 10, 5, 7) = 7,33

g) condicionFinal : (String, Num, Num, Num) → String que dada una tupla que representa el nombre de un/a estudiante y las notas que sacó en cada parcial, devuelva “promoción” si las 3 notas son mayores a 7, “regular” si no promociona pero las 3 notas son mayores a 4, y “libre” en caso contrario.

Por ejemplo: condicionFinal.(“Juan”, 10, 5, 7) = “regular”

condicionFinal.(“María”, 10, 8, 7) = “promoción”

h) condicionFinalEstudiante : (String, Num, Num, Num) → (String, String) que dada una tupla que representa el nombre de un/a estudiante y las notas que sacó en cada parcial, devuelva un par con el nombre del/la estudiante y la condición final como la calculada en el ejercicio anterior.

Por ejemplo: condicionFinal.(“Juan”, 10, 5, 7) = (“Juan”, “regular”)

condicionFinal.(“María”, 10, 8, 7) = (“María”, “promoción”)