Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Algoritmos y Estructuras de Datos I
Segundo Cuatrimestre de 2025

Introducción a la Programación Funcional

NOTA Los ejercicios con * son para resolver en su casa.

Ejercicios

1. Resuelve las siguientes expresiones de cálculo lambda:
   (a) $(\lambda x.x + 2)\ 3$
   (b) $(\lambda x. - x)\ 7$
   (c) $(\lambda x.\lambda y.x * y)\ 5\ 2$
   (d) $(\lambda x.xx)(\lambda x.xx)$
2. Instalar el intérprete de Python y ejecutar las expresiones del ejercicio 12.
3. Definir el *or* y la implicación en Cálculo Lambda.
4. (*) Leer los capítulos 1 y 2 del libro Aprende Haskell por el bien de todos!. Y el capítulo 5 del apunte de la materia: *"Construcción de Programas Correctos"*.
5. (*) Instale una versión de `ghc` (*Glasgow Haskell Compiler*, el compilador e interprete para Haskell). Instale la herramienta *Stack*. Puede leer sobre Stack en `https://docs.haskellstack.org/en/stable/GUIDE/`
6. Utilize `ghci` para decidir si las expresiones $(2^{29})/(2^9)$ y 2^{20} son iguales. Recuerde que el operador de potenciación en `ghci` es infijo y se escribe "^".
7. ¿Que arrojará como resultado la evaluación de la siguiente expresión en `ghci`?

   ```
   (head.(drop 3)) "0123456"
   ```

   ¿Qué tipo tiene el valor resultante?
8. Podemos examinar los tipos de algunas expresiones a partir del comando : t, el cual, seguido de una expresión válida nos dice su tipo. Por ejemplo

   ```
   >:t tail
   tail :: [a] -> [a]
   ```

   Examina los tipos de las siguientes expresiones y de las funciones que intervienen en la misma:
   (a) `> True && False`
   (b) `> sqrt 16`
   (c) `> 5 == 5`
   (d) `> (2: [])`
   (e) `> (++)`
   (f) `> length [5,4,3,2,1]`
9. Escriba una función que tome dos tuplas representando coordenadas 2D y calcule su producto escalar. El producto escalar de dos vectores (x_1, y_1) y (x_2, y_2) , se calcula de la siguiente manera:

   $$x_1 * x_2,\ + y_1 * y_2$$

   Recuerde que Haskell cuenta con las siguientes proyecciones sobre tuplas:

```
fst :: (a, b) -> a
snd :: (a, b) -> b
```

¿Qué perfil tiene la función definida?

10. Defina una función `abs: Int -> Int` que calcula el valor absoluto de un número. Realice una definición por casos.

11. Defina una función que dada una lista de números, retorne la suma de sus elementos. Realice una definición por casos utilizando *pattern matching*.

12. Escriba una función currificada que dado dos números x e y , calcule x^y. Escriba el perfil de la función.

13. Defina una función que calcule la potencia de 2 usando la función currificada definida en 12. Es decir dado el número x, la función debe calcular 2^x.

14. Escriba, usando currificación, una función que sume tres números. Escriba el perfil de la función.

15. Defina una función que suma 4 a otros dos números, utilizando la función definida en 14 . Luego aplique la función a los siguientes valores y analice la salida.

```
> addFour 5 8
```

16. Analice el siguiente programa que dado un arreglo y la cantidad de elementos a explorar, retorna True si todos los elementos del mismo son 0, Falso en caso contrario.

```
zeros a 0 = True
zeros a n = zeros a (n-1) && (a!!(n-1) == 0)
```

¿Qué resultado arrojará dicho programa si lo ejecutamos con las siguientes entradas?:

(a) `> zeros [0,2,0] 3`

(b) `> zeros [0,0,3] 2`

(c) `> zeros [0,0,0,0] 4`

Construya para cada caso el árbol de ejecución del programa.

- El siguiente código implementa la misma funcionalidad, es decir retorna True si todos los elementos del mismo son 0.

```
zeroes []    = True
zeroes (a:as) = zeroes as && (a == 0)
```

¿ Qué diferencia tiene? ¿Cómo es el perfil de ambas funciones? . De un ejemplo de invocación de la misma.

17. (*) Dada la siguiente función que calcula el factorial de un número dado.

```
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

Analice cómo se realiza la recursión cuando se ejecuta la función con los siguientes valores:

(a) `> factorial 1`

(b) `> factorial  3`

(c) `> factorial 5`

Construya para cada caso el árbol de ejecución del programa.

18. Determine que realiza la siguiente función.

```
belongs e []  = False
belongs e (a:as) = belongs e as  ||  (a == e)
```

Escriba el perfil de la misma y de 2 ejemplos de ejecución de la misma.

19. (*) Escriba un programa que calcule el n-esimo valor en la sucesión de fibonacci. La sucesión de Fibonacci se trata de una serie infinita de números naturales que empieza con un 0 y un 1 y continúa añadiendo números que son la suma de los dos anteriores:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597\ldots$$

```
> fibonacci  0 = 0
> fibonacci  1 = 1
> fibonacci  2 = 1
> fibonacci  3 = 2
...
> fibonacci  11 = 89
```

20. Dado el siguiente programa, determine el perfil de la función.

```
length' [] = 0
length' (_:xs) = 1 + length' xs
```

Analice cómo se realiza la recursión cuando se ejecuta la función con los siguientes valores:

(a) `> length [2,5,7]`

(b) `> length  ['a','z']`

(c) `> length [[3,4],[]]`

Construya para cada caso el árbol de ejecución del programa.

Formas de Evaluación

NOTA: Recomendamos, antes de comenzar a resolver los ejercicios, repasar la teoría: evaluación de expresiones.
Recordar:

- **Orden Aplicativo:** se reduce siempre la expresión más adentro y más a la izquierda (siempre de izquierda a derecha).
- **Orden Normal:** se reduce siempre la expresión más afuera y más a la izquierda.

21. Muestra los pasos de reducción hasta llegar a la forma normal de la expresión:

```
2 * cuadrado (hd [2,4,5,6,7,8])
```

Considerando las siguientes definiciones para **cuadrado** y **head**:

```
cuadrado :: Int -> Int
cuadrado x = x * x

hd :: [a] -> a
hd (x:xs) = x
```

- a) utilizando el orden de reducción aplicativo.
- b) utilizando el orden de reducción normal.

22. Dada la definición: `linf = 1 : linf`. Resuelve los siguientes pasos para la expresión:

```
hd linf
```

- a) Muestre los pasos de reducción utilizando el orden aplicativo.
- b) Haga lo mismo pero siguiendo el orden de reducción normal.

Compara dichos resultados.

23. Dada la siguiente definición:

```
f :: Int -> Int -> Int
f x 0 = x
f x (n+1) = cuadrado (f x n)
```

Resuelve los siguientes pasos para la expresión:

```
f 2 3
```

- a) Muestra los pasos de reducción utilizando el orden aplicativo.
- b) Has lo mismo pero siguiendo el orden de reducción normal.

Compara dichos resultados.

24. Utilizando orden aplicativo y normal, evalua la siguiente expresión:

```
square inf
```

Considerando las siguientes definiciones para **square** e **inf**:

```
square :: Int -> Int
square x = x *x

inf :: Int
inf = inf + 1
```

25. (*) Resuelva el ejercicio 23 utilizando orden de reducción lazy.