
# Map, Filter y Tuplas

Hernan Quintana (hquintan@ulima.edu.pe)

# Map

- La función map es una función de alto orden (*high order function*) muy importante dentro del paradigma funcional, ya que nos permite **transformar** una lista de elementos.
- En cuanto a forma, es similar a *list comprehension*, pero de sintaxis diferente.

# Sintaxis

```
map <FUNCION_A_MAPEAR> <LISTA_A_TRANSFORMAR>
```

- Tiene 2 formas de sintaxis:
  a) Pasando una función como argumento de entrada. Esta función será la que se va a aplicar para cada elemento de la lista.

  b) Pasando una función anónima.

# a. Función como argumento de entrada

```haskell
cuadrado :: Int -> Int
cuadrado num = num * num

cuadradoLista :: [Int] -> [Int]
cuadradoLista lista = map cuadrado lista
```

**Nota:** Tomar en cuenta tipo de argumento de entrada debe ser igual al tipo de los elementos de la lista.

# b. Función anónima

```
cuadradoLista :: [Int] -> [Int]
cuadradoLista lista = map (\x -> x * x) lista
```

**Nota:** Tomar en cuenta tipo de argumento de entrada de la función anónima debe ser igual al tipo de los elementos de la lista.

# Ejercicios

1. Se tiene una lista de números en forma de Strings. Implementar una función que nos permita convertir estos números a enteros.
2. Calcular el promedio de las notas del salón (EC1 y EC2) de una lista de alumnos. La lista de alumnos tiene la siguiente forma:

```
listaAlumnos = [ [11, 14], [10, 11], [18, 19], [4, 9] ]
```

# Filter

- Es una función de alto orden (*high order function*) que permite filtrar los elementos de una lista dada la aplicación de cierta **función predicado**.
- Una **función predicado** es una función que retorna un valor Booleano.

# Sintaxis

```
filter <FUNCION_PREDICADO> <LISTA_A_FILTRAR>
```

- Igual que la función **map** tiene dos formas de sintaxis:
  a) Pasando la función predicado como argumento de entrada.
  b) Pasando una función anónima como argumento de entrada.

# a) Argumento de entrada

```haskell
mayorCero :: Int -> Bool
mayorCero num = num > 0

filtrarMayoresCero :: [Int] -> [Int]
filtrarMayoresCero lista = filter mayorCero lista
```

# b) Función anónima

```haskell
filtrarMayoresCero :: [Int] -> [Int]
filtrarMayoresCero lista = filter (\x -> x > 0) lista
```

# Ejercicios

- De la lista de promedios, requerimos implementar una función que me indique el porcentaje de alumnos desaprobados del salón.

# Tuplas

- Qué pasa si quiero representar una lista de valores pero de distinto tipo de dato? Utilizamos **TUPLAS**.

```
tupla = ("Leo", 20, 1.8, True)
```

- Para acceder a los valores, se hará uso del pattern matching:

```
(nombre, edad, estatura, egresado) = tupla
```

  - **Nota:** Puede utilizar *wildcards* o comodines.

```
(nombre, _, _, _) = tupla
```

# Ejercicio

```
listaAlumnos = [
    ("Juan", 11, 14),
    ("Pedro", 10, 10),
    ("Leo", 18, 19),
    ("Hernan", 4, 9)
]
```

- Implementar una función que me devuelva el alumno que tuvo la mayor nota promedio de todo el salón.
- Implementar una función que me devuelva la mejor nota de los desprobados.