EDUCACIÓN CON RESPONSABILIDAD SOCIAL

UNIVERSIDAD DE COLIMA

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

EDSON PAUL MORFIN GALLARDO

MATA LOPEZ WALTER ALEXANDER

PROGRAMACION FUNCIONAL

INGENIERIA EN COMPUTACION INTELIGENTE

3ER SEMESTRE

GRUPO D.

INDICE

MATA LOPEZ WALTER ALEXANDER

Uso de listas

Las listas pueden contener estructura de datos con información heterogénea, esto quiere decir que la información puede ser de distinto tipo, entera, flotante, cadenas, booleanas,etc.

Pueden ser escritas de forma directa

```
PS C:\Users\edson> erl
Eshell V11.1  (abort with ^G)
1> [1,2,3,4,"A","B","CDE",1.23456,true,false].
[1,2,3,4,"A","B","CDE",1.23456,true,false]
```

También se pueden agregar elementos a listas existentes con "++"

```
2> [1,2,3,4,"A","B","CDE",1.23456,true,false]++["Nuevo elemento agregado 1",5,6,7,8,9,"diez",true].
[1,2,3,4,"A","B","CDE",1.23456,true,false,
 "Nuevo elemento agregado 1",5,6,7,8,9,"diez",true]
```

o también se pueden quitar elementos con "--".

```
3> [1,2,3,4,"A","B","CDE",1.23456,true,false]--[1,2,3,4,"CDE","A","B",true,false].
[1.23456]
```

La información se borra de izquierda a derecha y solo elimina un dato por dato marcado, esto quiere decir que si hay 5 "A", en una lista y escribes --["A"], solo borrara un elemento.

```
4> ["A",1,"A",2,"B",1,"B",2,"C",1,"C",2]--["A","B","C"].
[1,"A",2,1,"B",2,1,"C",2]
```

Acceder a los elementos por medio de

Head Tail

Head (cabeza) Tail (Cola) funciona para recolectar datos de una lista, dependiendo de cuantos head y tail tengamos será el numero de datos que recogerá cada uno de ellos.

Si solo tenemos un H y un T, H (head) solo recolectará el primer dato de la lista y el T (tail) recolectara el resto

```
5> [H|T]=["1er dato","2do dato","3er dato","resto de datos",2,3,4,5,6,7,8,9,0].
["1er dato","2do dato","3er dato","resto de datos",2,3,4,5,
 6,7,8,9,0]
6> H.
"1er dato"
7> T.
["2do dato","3er dato","resto de datos",2,3,4,5,6,7,8,9,0]
```

Para hacer uso de distintos Heads y Tails tenemos que darles un carácter adicional a la "H / T", en caso de que se quiera usar los mismos H / T se tendría que borrar los datos que guardaron previamente, esto pasa porque son variables inmutables.

```
8> [H|T]=["Nuevo dato 1","ND 2","ND3"].
** exception error: no match of right hand side value ["Nuevo dato 1","ND 2",
                                                       "ND3"]
9> f(H).
ok
10> f(T).
ok
11> [H|T]=["Nuevo dato 1","ND 2","ND3"].
["Nuevo dato 1","ND 2","ND3"]
12> H.
"Nuevo dato 1"
13> T.
["ND 2","ND3"]
```

```
17> [H1,Ha,H2,Hb|T]=[1,2,3,4,5,6,7,8,9,"a","b","c","d","e","f","g"].
[1,2,3,4,5,6,7,8,9,"a","b","c","d","e","f","g"]
18> H1.
1
19> Ha.
2
20> H2.
3
21> Hb.
4
22> T.
[5,6,7,8,9,"a","b","c","d","e","f","g"]
```

Algoritmo Push y Pop

Este tipo de algoritmo extrae uno o mas datos de una o mas listas y lo agrega a una variable nueva

```
1> Lvacio=["Vacio"].
["Vacio"]
2> L1=["A"|Lvacio].
["A","Vacio"]
3> L2=["B"|L1].
["B","A","Vacio"]
4> L3=["C"|L2].
["C","B","A","Vacio"]
5> L1.
["A","Vacio"]
6> L2.
["B","A","Vacio"]
7> L3.
["C","B","A","Vacio"]
```

Primero se crea una serie de listas enlazadas. Por ejemplo creamos la Lista Vacia, le ingresamos un dato para que sea mas sencillo de ver como funciona el enlace.

En L1, agregamos A y agregamos los datos de Lvacio.

En L2, agregamos B y los datos de L1 (como L1 se enlazó con Lvacio, entonces se agregaran los datos de L1 y Lvacio)

En L3 agregamos C y los datos de L2(A su vez, se agregan los datos de L1 y Lvacio).

```
11> [Extraer|L2]=L3.
["C","B","A","Vacio"]
12> Extraer
.
"C"
13> L2.
["B","A","Vacio"]
```

Aquí lo que se hizo fue extraer los datos de L2 (B,A,Vacio) de L3 (C,B,A,Vacio), por lo tanto solo quedó C.

Listas Binarias

Las listas binarias permiten almacenar cadenas de caracteres (Byte)

La sintaxis de estas son: escribir las listas dentro de flechas dobles << “Lista”>>, los números los toma como si fueran código ASCII

```
14> <<"Esta es una lista Binaria">>.
<<"Esta es una lista Binaria">>
```

```
16> <<69,115,116,97,32,101,115,32,117,110,97,32,99,97,100,101,110,97,32,100,101,32,116,101,120,116,111>>.
<<"Esta es una cadena de texto">>
```

Al igual que las listas normales, esta tiene una simulación de las operaciones de extracción de Head y Tail y Concatenacion.

```
17> <<H1:1/binary,H2:2/binary,T/binary>> = <<"Esta es una lista binaria">>.
<<"Esta es una lista binaria">>
18> H1.
<<"E">>
19> H2.
<<"st">>
20> T.
<<"a es una lista binaria">>
```

Al igual que H|T de las listas normales, se pueden declarar varias Heads, en este caso fueron 2.

```
25> A = <<"Hola ">>.
<<"Hola ">>
26> B = <<"Como estas?">>.
<<"Como estas?">>
27> C = <<" Mucho gusto">>.
<<" Mucho gusto">>
28> D = <<A/binary,B/binary,C/binary>>.
<<"Hola Como estas? Mucho gusto">>
```

En esta foto se puede observar como concatenar 2 o mas Listas Binarias

```
29> byte_size(A).
5
30> byte_size(B).
11
31> byte_size(C).
12
32> byte_size(D).
```

Byte_size sirve para ver el tamaño de una lista binaria, muestra el numero de caracteres que contiene, incluyendo los vacíos.

Tuplas

Las tuplas permiten organizar los datos, son usadas en casos donde es mas sencillo acceder al elemento por identificador que por un índice, que algunas veces este índice no es conocido. Esto evita la "problemática" que surgió al borrar o agregar datos a las listas simples, ya que borra de izquierda a derecha y en caso de tener mas de un dato en esa lista, no seria posible borrar un elemento especifico

Se pueden crear tuplas para integrar conjuntos de datos homogéneos de elementos individuales heterogéneos

```
33> {{"Pistolas","Jose","Jose"},{9,11,2001},{"Colima"}}.
{{"Pistolas","Jose","Jose"},{9,11,2001},{"Colima"}}
```

Esta tupla contiene 3 tuplas mas, 2 de cadenas y una de datos enteros, que hace referencia a nombre, fecha y estado.

```
39> {date(),time()}.
{{2020,10,25},{21,18,53}}
```

Date, time sirve para hacer una tupla de la fecha del sistema y una tupla del tiempo del sistema, incluyendo segundos

Modificar los elementos dentro de una tupla

Cambiar un elemento dentro de una tupla sin cambiar el resto de elementos

```
43> Tupla_1 = {"Esta","Es","Una","Tupla","Bien chida"}.
{"Esta","Es","Una","Tupla","Bien chida"}
44> erlang:setelement(3,Tupla_1,"Es una Modificacion de ").
{"Esta","Es","Es una Modificacion de ","Tupla","Bien chida"}
```

erlang:setelement(posición,elemento,modificación).

Con esta instrucción se pueden cambiar elementos específicos de una tupla, puede ser de una variable o declararse directamente

```
45> erlang:setelement(4,{"Juan","Ve","Por","Las Tortillas"},"El chesco").
{"Juan","Ve","Por","El chesco"}
```

Agregar un elemento al final de la tupla

```
46> erlang:append_element(Tupla_1,"que se le agregó un elemento al ultimo").
{"Esta","Es","Una","Tupla","Bien chida",
 "que se le agregó un elemento al ultimo"}
```

Igual que antes, esta instrucción puede ser usada con una variable o delcararse la tupla directamente.

```
47> erlang:append_element({"Juan","Ve","Por","Las Tortillas"},"y un chesco").
{"Juan","Ve","Por","Las Tortillas","y un chesco"}
```

Obtener un elemento de la tupla dado el índice

Erlang:element(posición de elemento ,tupla)

```
48> erlang:element(4,{"Juan","Ve","Por","Las Tortillas"}).
"Las Tortillas"
```

```
52> erlang:element(5,Tupla_1).
"Bien chida"
```

Eliminar un elemento de la tupla

Erlang:delete_element(3,tupla).

```
53> erlang:delete_element(1,{"Juan","Ve","Por","Las Tortillas"}).
{"Ve","Por","Las Tortillas"}
```

```
54> erlang:delete_element(5,Tupla_1).
{"Esta","Es","Una","Tupla"}
```

Lista de propiedades

Es una lista de tuplas (clave, valor)

Se utiliza la librería proplists

```
74> Refresco = [{presentacion,"Con Mucha Azucar"},{precio,26},{color,"Negro"}].
[{presentacion,"Con Mucha Azucar"},
 {precio,26},
 {color,"Negro"}]
75> proplists:get_value(color,Refresco).
"Negro"
```

Este es un uso muy útil de las tuplas, ya que se puede acceder a un dato por medio de una clave sin necesidad de saber la posición del mismo.

Mapas

Los mapas son una estructura de datos, sus elementos se almacenan de manera similar que las tuplas, con una clave y un valor, la clave puede ser de cualquier tipo al igual que el contenido

La sintaxis para crear un mapa es poner un #al inicio y => después de ingresar la clave, después de => se ingresa el contenido.

```
78> Refresco= #{presentacion => "Con Mucha Azucar"}.
#{presentacion => "Con Mucha Azucar"}
```

Para **agregar otro índice a un mapa** se hace lo siguiente:

```
80> Refresco1= Refresco#{precio=>26}.
#{precio => 26,presentacion => "Con Mucha Azucar"}
```

Para cambiar un índice existente se usa dos puntos ‘:= ‘ en lugar de ‘ => ‘

```
88> Refresco2= Refresco1#{presentacion := "Reducido en azucar"}.
#{precio => 26,presentacion => "Reducido en azucar"}
```

Para **Extraer un valor de un mapa** se hace lo que se conoce como pattern matching o corcondancia, se ingresa el índice que buscara y una variable y el mapa donde buscara el índice, cuando el índice

ingresado hace match con un índice del mapa, este extrae el valor y lo ingresa a la variable previamente declarada.

```
89> #{presentacion := PresentacionNuevaCocacola} = Refresco2.
#{precio => 26,presentacion => "Reducido en azucar"}
90> PresentacionNuevaCocacola.
"Reducido en azucar"
```

Eliminar una clave de un mapa

Maps:remove(clave,mapa)

```
93> Refresco2.
#{precio => 26,presentacion => "Reducido en azucar"}
94> Refresco3 = maps:remove(precio,Refresco2).
#{presentacion => "Reducido en azucar"}
95> Refresco3.
#{presentacion => "Reducido en azucar"}
```

Buscar una clave

```
96> maps:get(precio,Refresco2).
26
```

Determinar si es Mapa

```
97> is_map(Refresco3).
true
```

Obtener la cantidad de claves

```
98> map_size(Refresco2).
2
99> map_size(Refresco3).
1
```

Refresco 3 es menor que refresco 2 porque eliminamos una clave anteriormente

Obtener las claves.

```
100> maps:keys(Refresco3).
[presentacion]
101> maps:keys(Refresco2).
[precio,presentacion]
```

Conversion de datos

```
131> Lista = [1,2,4,5,6,7,8,9].
[1,2,4,5,6,7,8,9]
132> list_to_tuple(Lista).
{1,2,4,5,6,7,8,9}
```

Convertimos una lista en una tupla

```
135> ListaAtom=atom_to_list(atom).
"atom"
136> ListaAtom.
"atom"
```

Convertimos un atom en una lista

```
138> BinaryList=list_to_binary(Lista).
<<1,2,4,5,6,7,8,9>>
```

Convertimos una lista a una lista binaria

```
139> BinaryListAtom=list_to_binary(ListaAtom).
<<"atom">>
```

Lo mismo con la lista atom previamente declarada

```
140> AtomBinaryList=binary_to_atom(BinaryListAtom).
atom
```

Y lo volvemos a convertir en un Atom

```
144> Integer= (10).
10
145> BinaryInt=integer_to_binary(Integer).
<<"10">>
```

Entero a binario.

```
147> IntBinaryList=binary_to_list(BinaryInt).
"10"
```

Y ahora lo hacemos lista

```
148> IntInt=list_to_integer(IntBinaryList).
10
```

Y lo regresamos a que sea un entero nuevamente

```
137> io:format("Esto es un ~n Salto de linea ~n wow ~n").
Esto es un
 Salto de linea
 wow
ok
```

```
158> TuplaA= list_to_tuple([5,4,3,2,1]).
{5,4,3,2,1}
159> TuplaB= list_to_tuple([10,9,8,7,6]).
{10,9,8,7,6}
```

Convertimos 2 listas en tuplas y guardamos las variables

```
174> TuplaF =tuple_to_list(TuplaA)++ tuple_to_list(TuplaB).
[5,4,3,2,1,10,9,8,7,6]
```

Después en otra variable convertimos a listas y agregamos las variables previamente convertidas en tuplas