EDUCACIÓN CON RESPONSABILIDAD SOCIAL

UNIVERSIDAD DE COLIMA

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

PROGRAMACION FUNCIONAL

MATA LOPEZ WALTER ALEXANDER

MORFIN GALLARDO EDSON PAUL

3er SEMESTRE GRUPO D

ACTIVIDAD 10 – REPORTE TECNICO

ENTREGA 6-DIC-2020

En el siguiente programa se hace uso de los condicionales if,case,cond y pattern matching de funciones para validar entradas, en distintos módulos se uno de los condicionales anteriormente mencionados para realizar operaciones matemáticas básicas las cuales se pueden encontrar en una calculadora básica tal como suma, resta, multiplicación y División.

En caso de que las entradas no sean correctas se le indica al usuario un mensaje de error y se indica el problema, esto se hace para evitar errores a la hora de ejecutar el Código y ayudarle al usuario a saber cuál fue el error

En la siguiente sección se muestra el código fuente con una breve explicación de lo que hace cada módulo y función

La última función al ser únicamente para probar el código no se documentó ya que el programa funciona perfectamente sin esa parte del código

Código Fuente:

```
defmodule Multi do
  @moduledoc "Por medio de pattern matching este modulo verifica las entradas y c
omprueba que sean correctas para funcione correctamente el modulo"

  @doc """
  La funcion comprueba que ambas entradas sean numeros sin importar su tipo, en c
aso de ser verdadero imprime el resultado de la multiplicacion
  """
  def ccond(n1,n2) when is_number(n1) and is_number(n2), do: {:ok,"el resultado e
s#{n1*n2}"}

  @doc """
  En caso de que solo mande una entrada que sea numero, se imprime que hace falta
 un numero
  """
  def ccond(n) when is_number(n), do: {:error,"te falta un numero"}

  @doc """
  Esta funcion se activa cuando se mandan 2 entradas pero una o mas no son numero
s
  """
  def ccond(_,_),do: {:error,"algun dato no es un numero"}

  @doc """
  Senala al usuario que falta una entrada
  """
  def ccond(_), do: {:error,"falta un dato"}
end
```

```elixir
defmodule Resta do
  @moduledoc "La utilidad del modulo resta es restar 2 entradas, para funcionar c
orrectamente se validan las entradas con case dentro de las funciones "


  @doc """
  Funcion de dos entradas, con case y and se comparan las entradas y en caso de s
er verdaderas se imprime el resultado de la resta, en caso de ser falsa se imprim
e error
  """
  def res(n1,n2) do

  case is_number(n1) and is_number(n2) do
    true -> {:ok,"el resultado es #{n1-n2}"}
    false -> {:error,"uno o mas datos son erroneos"}
  end
end

@doc """
  Esta Funcion imprime que hace falta un dato independientemente del tipo del mis
mo
  """
  def res(_),do: {:error,"falta un dato"}

end

defmodule Div do
  @moduledoc "Este modulo verifica las entradas con IF"

  @doc "La funcion recibe 2 entradas y verifica que ambas sean numeros y que la s
egunda entrada sea distinta a 0 "
  def cif(n1,n2) do
    if is_number(n1)==true and (is_number(n2)==true and n2>0 or n2<0) do

      {:ok,"el resultado de la division de #{n1}/#{n2} es #{n1/n2}"}

    else
      {:error,"no es numero o intentaste dividir entre 0"}
    end
  end
  @doc "En caso de que solo se mande una entrada, marcara error "
  def cif(_), do: {:error,"falta un dato"}
end
```

```elixir
defmodule Suma do
  @moduledoc "Este modulo por medio de cond verifica y suma los datos"

  @doc "Por medio de cond verifica que ambas entradas sean numero, en caso de ser
 verdadero suma las entradas"
  def suma(n1,n2) do
  cond do
    is_number(n1) and is_number(n2) ==true -
>{:ok,"el resultado de la suma es #{n1+n2}"}
    true ->{:ok,"datos erroneos"}
    end
  end

  @doc "funcion que marca error en caso de mandar una sola entrada "
  def suma(_),do: {:error,"falta un dato"}
end

defmodule Test do

  def test do
    IO.inspect("Modulo multiplicacion")
    IO.inspect(Multi.ccond(10,20))
    IO.inspect(Multi.ccond(10,"cadena"))
    IO.inspect(Multi.ccond(:a,20))
    IO.inspect(Multi.ccond("aaaa",2))
    IO.inspect(Multi.ccond(10))
    IO.inspect(Multi.ccond("45"))
    IO.inspect(" ")

    IO.inspect("Modulo resta")
    IO.inspect(Resta.res(10,20))
    IO.inspect(Resta.res(10,"cadena"))
    IO.inspect(Resta.res(:a,20))
    IO.inspect(Resta.res("aaaa",2))
    IO.inspect(Resta.res(10))
    IO.inspect(Resta.res("45"))
    IO.inspect(" ")

    IO.inspect("Modulo division")
    IO.inspect(Div.cif(10,20))
    IO.inspect(Div.cif(10,"cadena"))
    IO.inspect(Div.cif(:a,20))
    IO.inspect(Div.cif("aaaa",2))
    IO.inspect(Div.cif(10))
    IO.inspect(Div.cif("45"))
```

```
    IO.inspect(Div.cif("45",0))
    IO.inspect(Div.cif(10,0))
    IO.inspect(Div.cif(0,100))
    IO.inspect(" ")

    IO.inspect("Modulo suma")
    IO.inspect(Suma.suma(10,20))
    IO.inspect(Suma.suma(10,"cadena"))
    IO.inspect(Suma.suma(:a,20))
    IO.inspect(Suma.suma("aaaa",2))
    IO.inspect(Suma.suma(10))
    IO.inspect(Suma.suma("45"))
    IO.inspect(Suma.suma("45",0))
    IO.inspect(Suma.suma(10,0))
    IO.inspect(Suma.suma(0,100))
  end
end
```

Codigo siendo ejecutado

```
iex(38)> Test.test()
"Modulo multiplicacion"
{:ok, "el resultado es200"}
{:error, "algun dato no es un numero"}
{:error, "algun dato no es un numero"}
{:error, "algun dato no es un numero"}
{:error, "te falta un numero"}
{:error, "falta un dato"}
" "
"Modulo resta"
{:ok, "el resultado es -10"}
{:error, "uno o mas datos son erroneos"}
{:error, "uno o mas datos son erroneos"}
{:error, "uno o mas datos son erroneos"}
{:error, "falta un dato"}
{:error, "falta un dato"}
" "
"Modulo division"
{:ok, "el resultado de la division de 10/20 es 0.5"}
{:error, "no es numero o intentaste dividir entre 0"}
{:error, "no es numero o intentaste dividir entre 0"}
{:error, "no es numero o intentaste dividir entre 0"}
{:error, "falta un dato"}
{:error, "falta un dato"}
{:error, "no es numero o intentaste dividir entre 0"}
{:error, "no es numero o intentaste dividir entre 0"}
{:ok, "el resultado de la division de 0/100 es 0.0"}
" "
"Modulo suma"
{:ok, "el resultado de la suma es 30"}
{:ok, "datos erroneos"}
{:ok, "datos erroneos"}
{:ok, "datos erroneos"}
{:error, "falta un dato"}
{:error, "falta un dato"}
{:ok, "datos erroneos"}
{:ok, "el resultado de la suma es 10"}
{:ok, "el resultado de la suma es 100"}
{:ok, "el resultado de la suma es 100"}
iex(39)>
```