ITIS-LS "Francesco Giordani" Caserta

prof. Ennio Ranucci

a.s. 2023-2024

ELM

Lo scopo di Elm è quello di creare interfacce-utente grafiche basate su web browser. Viene generato un codice html/Javascript per permettere l'esecuzione dell'applicazione sul browser. Non genera eccezioni in fase di esecuzione, ma invece individua eventuali problemi durante la compilazione e suggerisce come risolverli. Questo linguaggio di programmazione si basa su 3 fattori: *performance usabilità robustezza*. Elm risulta molto utile per applicazioni per smartphone e tablet.

Elm è un linguaggio funzionale che si compila in html/JavaScript. Aiuta a creare siti Web e app Web.

Pone in primo piano la semplicità e sulla qualità degli strumenti.

Come creare app interattive con Elm? Con ELM si può scrivere codice JavaScript migliore.

Per installare eseguire: installer-for-windows.exe (contenuto nel file .zip)
Al termine dell'installazione eseguire: cmd.exe

Allinterno della finestra "Prompt dei comandi" cambiare cartella con il comando:
cd C:\Users\<username>
Eseguire il comando: elm init

La prima volta rispondere Y per creare un file e una directory in C:\Users\<username>:
- elm.json descrive il tuo progetto
- src/ contiene tutti i tuoi file Elm.

Ora con blocco note creare un file src/Hello.elm
Contenente il seguente codice:

```
import Html exposing (text)
text "Benvenuti in ELM"
```

Avviare un server in http://localhost:8000 con il comando: elm reactor
Aprire il browser e nella barra degli indirizzi digitare: http://localhost:8000/src/
Appare la lista dei file contenuti nella cartella SRC. Cliccare su Hello.elm per eseguirlo.

Se vogliamo la conversione del codice di Hello.elm in HTML/JavaScript digitiamo il comando:
elm make src/Hello.elm
troveremo la conversione nel file Index.html della cartella: C:\Users\<username>

Ecco un piccolo programma (classico esempio di codice ELM) che consente di incrementare e decrementare un numero:

```
module IncDec exposing (..)
import Browser
import Html exposing (Html, button, div, text)
import Html.Events exposing (onClick)
main =
  Browser.sandbox { init = init, update = update, view = view }
type alias Model = Int
init : Model
init =
  0
type Msg
  = Increment
  | Decrement
update : Msg -> Model -> Model
update msg model =
  case msg of
    Increment ->
      model + 1
    Decrement ->
      model - 1
view : Model -> Html Msg
view model =
  div []
    [ button [ onClick Decrement ] [ text "-" ]
    , div [] [ text (String.fromInt model) ]
    , button [ onClick Increment ] [ text "+" ]
    ]
```

Installare il modulo random: elm install elm/random

```
--   https://guide.elm-lang.org/architecture/text_fields.html

module Input exposing (..)
import Browser
import Html exposing (Html, Attribute, div, input, text)
import Html.Attributes exposing (..)
import Html.Events exposing (onInput)

main =
  Browser.sandbox { init = init, update = update, view = view }

type alias Model =
  { content : String
  }

init : Model
init =
  { content = "" }
```

```
type Msg
  = Change String

update : Msg -> Model -> Model
update msg model =
  case msg of
    Change newContent ->
      { model | content = newContent }

view : Model -> Html Msg
view model =
  div []
    [ input [ placeholder "Testo da capovolgere", value model.content, onInput Change ] []
    , div [] [ text (String.reverse model.content) ]
    ]
```

Salvare nella cartella SRC il file “IncDec.elm” contenente il codice sopra riportato.

Per eseguirlo ritornare alla finestra del browser collegata allindirizzo: http://localhost:8000/src/

e cliccare sul IncDec.elm

Altri esempi di codice ELM scaricati dal Web (vedere commento con il link):

Esercizio n# 3

```
--   https://guide.elm-lang.org/architecture/text_fields.html

-- Capovolge una stringa

module Capovolgi exposing (..)

import Browser

import Html exposing (Html, Attribute, div, input, text)

import Html.Attributes exposing (..)

import Html.Events exposing (onInput)

main =

  Browser.sandbox { init = init, update = update, view = view }

type alias Model =

  { content : String

  }

init : Model

init =
```

```
  { content = "" }

type Msg

  = Change String

update : Msg -> Model -> Model

update msg model =

  case msg of

    Change newContent ->

      { model | content = newContent }

view : Model -> Html Msg

view model =

  div []

    [ input [ placeholder "Testo da capovolgere", value model.content, onInput Change ] []

    , div [] [ text (String.reverse model.content) ]

]
```

Esercizio n# 4

E’ necessario installare il pacchetto random, nella finestra “promp dei comandi” dopo aver avviato ELM (elm init) eseguire il comando: elm install elm/random

```
-- Alla pressione di un pulsante viene  generato un numero casuale compreso tra 1 e 6.

--   https://guide.elm-lang.org/effects/random.html

module Casuali exposing (..)

import Browser

import Html exposing (..)

import Html.Events exposing (..)

import Random

main =

  Browser.element

    { init = init
```

```
    , update = update
    , subscriptions = subscriptions
    , view = view
    }
type alias Model =
  { dieFace : Int
  }
init : () -> (Model, Cmd Msg)
init _ =
  ( Model 1
  , Cmd.none
  )
type Msg
  = Roll
  | NewFace Int
update : Msg -> Model -> (Model, Cmd Msg)
update msg model =
  case msg of
    Roll ->
      ( model
      , Random.generate NewFace (Random.int 1 6)
      )
    NewFace newFace ->
      ( Model newFace
      , Cmd.none
      )
subscriptions : Model -> Sub Msg
subscriptions model =
```

```
  Sub.none
view : Model -> Html Msg
view model =
  div []
    [ h1 [] [ text (String.fromInt model.dieFace) ]
    , button [ onClick Roll ] [ text "Roll" ]
    ]
```

Esercizio n# 5

```
-- Confronta la stringa inserita con la password
--   https://guide.elm-lang.org/architecture/forms.html
module Form exposing (..)
import Browser
import Html exposing (..)
import Html.Attributes exposing (..)
import Html.Events exposing (onInput)
main =
  Browser.sandbox { init = init, update = update, view = view }
type alias Model =
  { name : String
  , password : String
  , passwordAgain : String
  }
init : Model
init =
  Model "" "" ""
type Msg
  = Name String
  | Password String
```

```
  | PasswordAgain String

update : Msg -> Model -> Model

update msg model =

  case msg of

    Name name ->

      { model | name = name }

    Password password ->

      { model | password = password }

    PasswordAgain password ->

      { model | passwordAgain = password }

view : Model -> Html Msg

view model =

  div []

    [ viewInput "text" "Name" model.name Name

    , viewInput "password" "Password" model.password Password

    , viewInput "password" "Re-enter Password" model.passwordAgain PasswordAgain

    , viewValidation model

    ]

viewInput : String -> String -> String -> (String -> msg) -> Html msg

viewInput t p v toMsg =

  input [ type_ t, placeholder p, value v, onInput toMsg ] []

viewValidation : Model -> Html msg

viewValidation model =

  if model.password == model.passwordAgain then

    div [ style "color" "green" ] [ text "OK" ]

  else

    div [ style "color" "red" ] [ text "Passwords do not match!" ]
```