elm

Literals

```
True/False  : Bool
42          : number
3.14        : Float
'a'         : Char
"abc"       : String
"""multi-line
       string"""
```

Lists

A collection of items of the same type

```
1 :: [2,3] == [1,2,3]
List.map     List.indexedMap
List.foldl   List.concat
List.foldr   List.filter
```

Arrays

```
Array.empty
Array.fromList
Array.toList
Array.get
Array.set
```

Custom Types

Custom Types start with an upper case letter

```
type User
    = Regular String Int
    | Visitor String
```

Type Annotations

```
answer : Int
answer =
    42

factorial : Int → Int
factorial n =
    List.product
        (List.range 1 n)

distance :
    {x : Float, y : Float}
    → Float
distance { x, y } =
    sqrt (x ^ 2 + y ^ 2)
```

Destructuring

```
sum addends =
    let
        ( a, b ) = addends
    in
    a + b
sum (a, b) = a + b
f list = case list of
    []       → "Empty"
    [_]      → "One element"
    [a,b]    → "2 elements"
    a::b::_  → "More than 2"
myRecord = {x=1, y=2, z=3}
sum {x, y} = x + y
onlyX {x} = x
sum ({x, y} as whole) =
    x + whole.y + whole.z
type My = My String
toString (My string) = string
type My = My {foo:Int,bar:Int}
foo (My {foo}) = foo
```

Comments

```
-- a single
-- line comment
{- a multi-line comment
   {- can be nested -}
-}
```

Trick to comment blocks of code

```
{--}
add x y = x + y
--}
```

Tuples

Contains 2 or 3 items of different type.

```
(1,"2",True)
Tuple.first/second
```

The Elm Architecture

```
Browser.sandbox
Browser.element
Browser.document
Browser.application
-- headless
Platform.worker
```

Records

A collection of key/value pairs, similar to objects in JavaScript

```
point = { x = 0, y = 0 }
point.x == 0
List.map .x [ point, point2 ]
{ point | x = 6 }
{ point | x = point.x + 1
        , y = point.y + 1 }
```

Extensible Records have at least certain fields:

```
f : { b | key : a } → a
f = .key
```

Dictionaries

```
Dict.empty
Dict.fromList
Dict.toList
Dict.get
Dict.update
```

Sets

```
Set.empty
Set.fromList
Set.toList
Set.insert
Set.remove
```

Type Aliases

Type Aliases start with an upper case letter

```
type alias Name = String
type alias Age = Int

info : (Name, Age)
info = ("Steve", 28)

type alias Point =
    {x: Float, y: Float}

origin : Point
origin =
  {x = 0, y = 0}
```

Maybe / Result

```
type Maybe a
    = Just a
    | Nothing

type Result err a
    = Ok a
    | Err err
```

Common Functions

```
map : (a → b) → T a → T b
map2 : (a→b→c) → T a → T b → T c
indexedMap:(Int→a→b) → T a → T b
filter : (a → Bool) → T a → T a
fold : (a → b → b) → b → T a → b
andThen : (a → T b) → T a → T b
```

Functions

Functions start with a lower case letter. No parentheses or commas for arguments or code blocks.

```
square n = n^2

hypotenuse a b =
  sqrt (square a + square b)
```

Anonymous functions

Anonymous functions start with "\", that resemble lambda "λ"

```
square = \n → n^2
squares =
  List.map (\n → n^2)
    (List.range 1 100)
```

Optimizations

```
Html.lazy
Html.keyed
```

Debugging

```
Debug.toString
Debug.log
Debug.todo
```

Routing

```
import Url.Parser exposing (s,(</>),int,string,oneOf,map)

type Route = Blog Int | User String | Comment String Int
routeParser = oneOf
  [ map Blog     (s "blog"</>int)
  , map User     (s "user"</>string)
  , map Comment  (s "user"</>string</>s "comment"</>int)
  ]
```

Advanced Types

Opaque types don't expose constructors.

Phantom type:

```
type Currency a
  = Currency Int

() Unit, Never
```

Constrained Type Variables

```
number                     (Int, Float)
appendable             (String, List a)
comparable        (Float,Char,String,
 Int,lists/tuples of comparable)
```

Conditionals

```
if k == 40 then
    n + 1
else if k == 38 then
    n - 1
else
    n
```

The Elm Architecture II

```
init : ( Model, Cmd Msg )
update : Msg -> Model -> (Model,Cmd Msg)
subscriptions : Model -> Sub Msg
view : Model -> Html Msg
```

Operators

```
+  -  *  /  ^              math
//                         int division
==  /=                     equality
<  >  <=  >=  max  min     comparison
not  &&  ||  xor           booleans
++                         append
modBy  remainderBy         fancy math
and  or  xor               bitwise
◁  ▷  <<  >>               functions
::                         cons
```

Most can be used in "prefix notation" too:

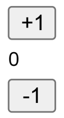
```
a + b == (+) a b
```

Hello World

```
module Main exposing (main)
import Html exposing (..)
main =
  div [] [text "Hello World!"]
```

Counter

Available at ellie-app.com



```
module Main exposing (main)

import Browser
import Html exposing (..)
import Html.Events exposing (..)

type alias Model = { count : Int }
init             = { count = 0   }
type Msg = Increment | Decrement

update msg model =
   case msg of
      Increment →
         { model | count = model.count + 1 }
      Decrement →
         { model | count = model.count - 1 }

view model = div []
  [ button [onClick Increment] [text "+1"]
  , div [] [text◁String.fromInt model.count]
  , button [onClick Decrement] [text "-1"]
  ]

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

Commands

```
elm repl
elm init
elm reactor
elm make
elm install
elm bump
elm diff
elm publish
```

REPL

```
:exit
:help
:reset
```

Backslash (\) for multi-line expressions

Tools

ellie-app.com, shortcut to save: [⌘][shift][return]

```
elm-format      elm-test
elm-doc elm-doc-preview
elm-spa elm-live/elm-go
elm-json      elm-review
elm-xref     elm-graphql
elm-optimize-level-2
```

JavaScript Interop

Ports, incoming and outgoing values:

```
port prices : (Float → msg) → Sub msg
port time : Float → Cmd msg
```

From JS, start Elm with flags and talk to these ports:

```
var app = Elm.Main.init ( {
  node: document.getElementById('app'),
  flags: { key: 'value' } } );
app.ports.prices.send(42);
app.ports.time.subscribe(callback);
```

Pipe Operator

```
viewNames1 names =
    String.join ", " (List.sort names)
viewNames2 names =
    names
        ▷ List.sort
        ▷ String.join ", "
viewNames3 names =
    String.join ", " ◁ List.sort names
```

Modules Imports

```
import List                    -- preferred
import List as L
import List exposing (..)
import List exposing ( map, foldl )
import Maybe exposing ( Maybe )
import Maybe exposing ( Maybe(..) )
```

Side Effects Task / Cmd

```
Task.perform      Task.attempt
Task.andThen      Cmd.batch
```

Tasks can be chained. Cmds only batched.

Hello World with Elm-UI

```
module Main exposing (main)
import Element exposing (..)
main =
 layout [] ◁
   el [] [text "Hello World!"]
```

Pattern Matching

```
case maybeList of
    Just xs → xs
    Nothing → []

case xs of
    [] →
        Nothing
    first :: rest →
        Just (first, rest)

case n of
    0 → 1
    1 → 1
    _ → fib (n-1) + fib (n-2)
```

Source: https://github.com/lucamug/elm-cheat-sheet/   Elm official guide: https://guide.elm-lang.org/   Elm syntax: https://elm-lang.org/docs/syntax   Note: ◁ = <|, ▷ = |>, → = ->   Version 1.4