



 elm	Literals	Lists	Arrays	Custom Types	Type Annotations	Destructuring
	True/False : Bool 42 : number (Int or Float) 3.14 : Float 'a' : Char "abc" : String "" "multi-line string""	A collection of items of the same type [1,2,3,4] 1 :: [2,3,4] 1 :: 2 :: 3 :: 4 :: []	Array.empty Array.fromList Array.toList Array.get Array.set	Custom Types start with an upper case letter type User = Regular String Int Visitor String	answer : Int answer = 42 factorial : Int -> Int factorial n = List.product (List.range 1 n)	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"
	Tuples	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 }	Dictionaries	Dict.empty Dict.fromList Dict.toList Dict.get Dict.update	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}
	Comments	Can contain 2 or 3 items of different type. (1,"2",True)	Sets	Set.empty Set.fromList Set.toList Set.insert Set.remove	Maybe / Result	type Maybe a = Just a Nothing type Result err a = Ok a Err err
	-- a single -- line comment {- a multi-line comment {- can be nested -} -} Trick to comment blocks of code {--} add x y = x + y --}	The Elm Architecture	Extensible Records have at least certain fields: f : { b key : a } -> a f = .key			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
Functions	Anonymous functions	Optimizations	Routing	Advanced Types	Constrained Type Variables	
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 start with "\", that resemble lambda "λ" square = \n -> n^2 squares = List.map (\n -> n^2) (List.range 1 100)	Html.lazy Html.keyed Debugging Debug.toString Debug.log Debug.todo	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)]	Opaque types don't expose constructors. Phantom type: type Currency a = Currency Int () Unit, Never	number (Int, Float) appendable (String, List a) comparable (Int, Float, Char, String, lists/tuples of comparable) compappend (String, List comparable)	
Conditionals	JavaScript Interop	Operators	Modules Imports	Side Effects Task/Cmd	Pattern Matching	
if k == 40 then n + 1 else if k == 38 then n - 1 else n	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: <div id='app'></div> <script src='elm.js'></script> <script> var app = Elm.Main.init({ node: document.getElementById('app'), flags: { key: 'value' } }); app.ports.prices.send(42); app.ports.time.subscribe(callback); </script>	+ - * / ^ // == /= < > <= >= max min not && xor ++ modBy remainderBy and or xor < > << >> :: Most can be used in "prefix notation" too: a + b == (+) a b	import List -- preferred import List as L import List exposing (..) import List exposing (map, foldl) import Maybe exposing (Maybe) import Maybe exposing (Maybe(..))	Task.perform Task.attempt Task.andThen Cmd.batch Tasks can be chained. Cmds only batched.	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)	
Commands	REPL	Tools	Counter			
elm repl elm init elm reactor elm make elm install elm bump elm diff elm publish	:exit :help :reset Backslash (\) for multi-line expressions	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-graphql	module Main exposing (main) import Browser import Html exposing (..) import Html.Events exposing (..) type alias Model = { count : Int } initialModel = { 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 = initialModel , view = view , update = update }			