λ

ML Lab 7

Programmazione Funzionale

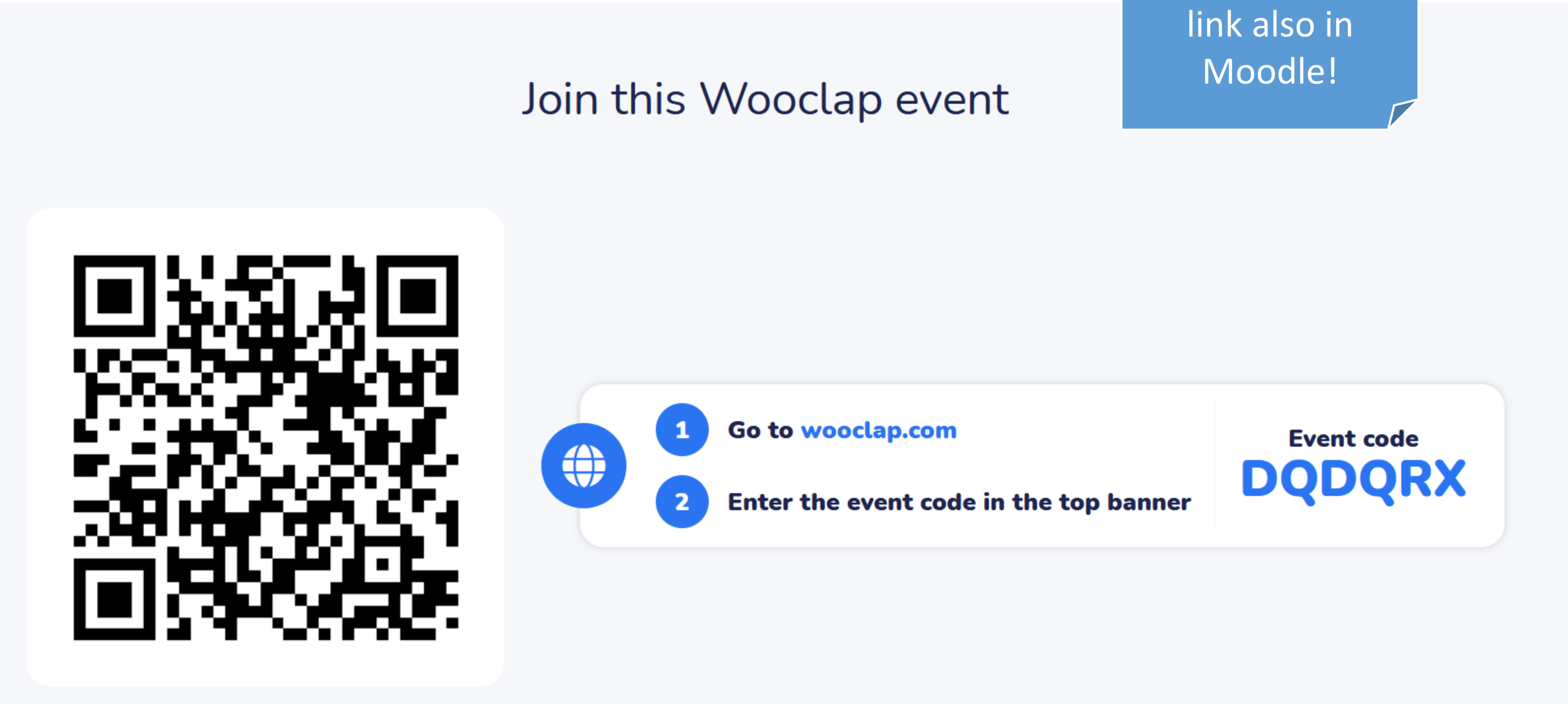
2024/2025

Università di Trento

Chiara Di Francescomarino

When you have time

You can find the link also in Moodle!



Join this Wooclap event

1 Go to wooclap.com

2 Enter the event code in the top banner

Event code
DQDQRX

Running ML

- Three ways:
  - Command line poly/interface poly
    - In order to enter through command line type rlwrap `poly`
    - We see **>** once we are inside the Poly environment
  - If `foo` is a file name, there are two options

```
poly < foo
use "foo";
```
(inside the Poly environment)

Old exercises

Exercise 5.7

- Write a function `sumList` that takes a list of integers and returns a pair of the sum of the odd positions and the sum of the even positions using the `let val` declaration.
- For instance
  - `sumList`([1,2,3,4]) = (4,6)
  - `sumList([]) = (0,0)`

Solution 5.7

```
> fun sumList (nil) = (0,0)
    | sumList ([x]) = (x,0)
    | sumList (x::y::zs) =
    let
        val (sumOdd, sumEven) = sumList (zs)
    in
        (x+sumOdd, y+sumEven)
    end;

val sumList = fn: int list -> int * int
> sumList [1,2,3,4,5];
val it = (9, 6): int * int
```

Exercise 5.9

- Write a function `comb` to compute $\binom{n}{m}$, while printing n, m and the result.
- You can use the factorial (write an auxiliary function `factorial`) to compute the function $\binom{n}{m} = \frac{n!}{m!(n-m)!}$
- For instance
  - `comb 5 2;`

```
n is 5
m is 2
Result is 10
```

Solution 5.9

```
fun factorial 0 = 1
| factorial n = n * factorial(n-1);
fun comb n m =
    (
    print ("n is ");
    print(Int.toString(n));
    print ("\n");
    print ("m is ");
    print(Int.toString(m));
    print ("\n");
    print ("Result is ");
    print (Int.toString (factorial(n) div (factorial(m) * factorial(n-
    m))));
    print ("\n")
    );
comb 5 2;
n is 5
m is 2
Result is 10
val it = (): unit
```

Exercise 5.10

- Write a function `printXs` that given an integer n, print 2^n `Xs`

- For instance
  - `printXs 3;`

```
XXXXXXXX
```

Solution 5.10

```
> fun makelist 0 = "X"
    | makelist n = makelist (n-1) ^ makelist (n-1);
val makelist = fn: int -> string
> fun printXs n = print(makelist n);
val printXs = fn: int -> unit

> printXs 3;
XXXXXXXXval it = (): unit
```

- Or, alternatively

```
> fun print_2n (0) = print("X")
        |print_2n (n) = (
               print_2n (n-1);
               print_2n (n-1)
               );
```

Exercise 5.11

- Write expressions to
    1. Open a file “zap” for reading
    2. Read 5 characters from the instream in1
    3. Read a line of text from the instream in1
    4. Find the first character waiting on the in1, without consuming it
    5. Read the entire file from instream in1
    6. Close the file whose instream is in1

Solution 5.11

- Write expressions to
  1. Open a file “zap” for reading
     `val in1 = TextIO.openIn(“zap”);`
  2. Read 5 characters from the instream in1
     `TextIO.inputN(in1,5);`
  3. Read a line of text from the instream in1
     `TextIO.inputLine(in1)`
  4. Find the first character waiting on the in1, without consuming it
     `TextIO.lookaehead(in1);`
  5. Read the entire file from instream in1
     `TextIO.input(in1)`
  6. Close the file whose instream is in1
     `TextIO.closeIn (in1)`

Exercise 6.3

- Write a function `tabulate` that takes an initial value a, an increment δ, a number of points n, and a function F from reals to reals and print a table with columns corresponding to x and $F(x)$, where $x = a, a + \delta, a + 2\delta, \dots, a + (n-1)\delta$
- For instance
  - `tabulate (1.0,0.1,9,fn x => x*x);`

```
1.0  1.0
1.1  1.21
1.2  1.44
1.3  1.69
1.4  1.96
1.5  2.25
1.6  2.56
1.7  2.89
1.8  3.24
```

Solution 6.3

```
> fun tabulate(x,delta,0,F) = ()
    | tabulate(x,delta,n,F) = (
        print(Real.toString(x));
        print("\t");
        print(Real.toString(F(x)));
        print("\n");
        tabulate(x+delta,delta,n-1,F)
    );
val tabulate = fn: real * real * int * (real -> real) -> unit
```

Solution 6.3

```
> tabulate (1.0,0.1,9,fn x => x*x);
1.01.0
1.11.21
1.21.44
1.31.69
1.41.96
1.52.25
1.62.56
1.72.89
1.83.24
val it = (): unit
```

Exercise 6.6

- Use `filter` to find the elements of a list of reals that are greater than 0.
- For instance, applied to a list of reals such as `L = [1.1,~1.2,~1.3,1.4]`, it should return `[1.1,1.4]`

Solution 6.6

```
> val L = [1.1,~1.2,~1.3,1.4];
val L = [1.1, ~1.2, ~1.3, 1.4]: real list

> filter(fn(x)=>x>0.0, L);
val it = [1.1, 1.4]: real list
```

Exercise 6.7

- Write a function `readAndSum` that, given a text file containing a number per line, transforms each number into an integer, sums them and returns the sum.
- You do not have to handle exceptions.
- In ML there exists a function `Int.fromString (fn: string -> int option)`
- You can use support functions, if needed.
- For instance , given a file “numbers” containing

```
5
8
10
```

```
readAndSum (“numbers”) = 23
```

Solution 6.7

```
> fun readAndSum1 file =
       if TextIO.endOfStream(file) then 0
       else
   valOf(Int.fromString(valOf(TextIO.inputLine(file)))
   ) + readAndSum1(file);
> fun readAndSum filename =
       readAndSum1(TextIO.openIn(filename));
```

New exercises

Exercise 7.1

- Write, in curried form, a function `applyList` that takes a list of functions and a value and applies each function to the value, producing a list of the results. If the list is empty it returns the empty list.
- For instance
  - `applyList [fn x=>x*2, fn x => x*x*x] 4 = [8,64]`

Solution 7.1

```
> fun applyList nil _ = nil
    | applyList (F::Fs) a = F(a)::(applyList Fs a);
val applyList = fn: ('a -> 'b) list -> 'a -> 'b list

> applyList [fn x=>x*2, fn x => x*x*x] 4;
val it = [8, 64]: int list
```

Exercise 7.2

- Given a function F that takes a parameter of product type with n components and the n components, define a function `curry` that applied to F produces a function `G` such that

$$G\ x_1\ \ldots\ x_n\ =\ F\ (x_1, \ldots, x_n)$$

- n should be fixed (e.g., n=3)
- For instance
  - `curry (fn (x,y,z)=>x*y*z) 1 2 3 = 6`

Solution 7.2

```
> fun curry F x1 x2 x3 = F(x1,x2,x3);
val curry = fn: ('a * 'b * 'c -> 'd) -> 'a -> 'b -> 'c -> ‘d

> curry (fn (x,y,z)=>x*y*z);
val it = fn: int -> int -> int -> int
> curry (fn (x,y,z)=>x*y*z) 1 2 3;
val it = 6: int
```

- We can also name G

```
> val G = curry (fn (x,y,z)=>x*y*z);
val G = fn: int -> int -> int -> int
> G 1 2 3;
val it = 6: int
```

Exercise 7.3

- Use `map, foldr` and `foldl` for turning a list of integers into a list of reals with the same values
- For instance
  - `toReal ([1,2,3])= [1.0, 2.0, 3.0]`

Solution 7.3

```
> val f = map real;
val f = fn: int list -> real list
> f [1,2,3];
val it = [1.0, 2.0, 3.0]: real list
```

Exercise 7.4

- Use `map`, `foldr` and `foldl` for computing the logical AND of a list of Booleans
- For instance
  - `andb [true, false, true] = false`

Solution 7.4

```
> val andb = foldr (fn (x,y) => x andalso y)
true;
val andb = fn: bool list -> bool
> andb [true, false, false];
val it = false: bool
```

Exercise 7.5

- Use `map`, `foldr` and `foldl` for defining the function `implode`
- For instance
  - `implode[#”b”,#”c”] = “bc”`

Solution 7.5

```
> val implode = (foldr (op ^) "") o (map str);
val implode = fn: char list -> string
> implode [#”a”,#”b”];
val it = "ab": string
```

Exercise 7.6

- Given a binary tree datatype `’a btree`

```
datatype ‘a btree =
    Empty |
    Node of ‘a * ‘a btree * ‘a btree;
```

- Write a function `postOrder` that returns a list of the nodes of a binary tree in postorder, where the label at the root follows the postorder traversal of the left and right subtrees (first the labels of the tree on the left, then the ones of the tree on the right and finally the root).
- For instance
  - `postOrder (Node ("ML", Node ("as", Node ("a", Empty, Empty), Node ("in", Empty, Empty)), Node ("types", Empty, Empty))) = ["a", "in", "as", "types", "ML"]`

Solution 7.6

```
fun postOrder (Empty) = nil
     | postOrder(Node(a,left,right)) =
     postOrder (left) @ postOrder (right) @
[a];
val postOrder = fn: 'a btree -> 'a list

> postOrder (Node ("ML", Node ("as", Node ("a",
Empty, Empty), Node ("in", Empty, Empty)), Node
("types", Empty, Empty)));
val it = ["a", "in", "as", "types", "ML"]:
string list
```

Exercise 7.7

- Given a binary tree datatype `’a btree`

```
datatype ‘a btree =
    Empty |
    Node of ‘a * ‘a btree * ‘a btree;
```

- Write a function `inOrder` that returns the list of the nodes of a binary tree in inorder, where the label at the root is between the inorder traversal of the left and right subtrees, i.e., first the labels in the left tree, then the root and finally the labels in the right tree.
- For instance
  - `inOrder (Node ("ML", Node ("as", Node ("a", Empty, Empty), Node ("in", Empty, Empty)), Node ("types", Empty, Empty))) = ["a", "as", "in", "ML", "types"]`

Solution 7.7

```
fun inOrder (Empty) = nil
      | inOrder(Node(a,left,right)) =
      inOrder (left) @ [a] @ inOrder (right);
val inOrder = fn: 'a btree -> 'a list

> inOrder (Node ("ML", Node ("as", Node ("a",
Empty, Empty), Node ("in", Empty, Empty)), Node
("types", Empty, Empty)));
val it = ["a", "as", "in", "ML", "types"]:
string list
```

Exercise 7.8

- Define a type `mapTree` that is a specialization of `btree` so that it has a label type that is a set of domain-range pairs
- Define a tree `t1` that has a single node with the pair ("a",1) at the root

Solution 7.8

```
> type ('d, 'r) mapTree = ('d * 'r) btree;
type ('a, 'b) mapTree = ('a * 'b) btree
> val t1 = Node(("a",1), Empty, Empty): (string, int) mapTree;
val t1 = Node (("a", 1), Empty, Empty): (string, int) mapTree
```

Exercise 7.9

- Write a function `sumTree` for a mapTree T of type `(‘a,’b) mapTree.` The function visits the tree and returns the sum of the second component of the label of all nodes.
- For instance
  - `sumTree (Node(("a",1), Node(("c",2), Empty, Node(("d",3), Empty, Empty)), Empty) = 6`

Solution 7.9

```
> fun sumTree Empty = 0
    | sumTree (Node((a,b),left,right)) = b + sumTree (left) +
sumTree (right);
val sumTree = fn: ('a * int) btree -> int

> val t2 = Node(("a",1), Node(("c",2), Empty, Node(("d",3),
Empty, Empty)), Empty): (string, int) mapTree;
val t2 = Node    (("a", 1), Node (("c", 2), Empty, Node (("d",
3), Empty, Empty)), Empty): (string, int) mapTree

> sumTree t1;
val it = 1: int

> sumTree t2;
val it = 6: int
```