F#: Name spaces and Modules

Prof. Michele Loreti

Programmazione Avanzata
Corso di Laurea in Informatica (L31)
Scuola di Scienze e Tecnologie

Namespace. . .

A namespace lets you organize code into areas of related functionality by enabling you to attach a name to a grouping of program elements.

Namespace. . .

A namespace lets you organize code into areas of related functionality by enabling you to attach a name to a grouping of program elements.

```
namespace [parent-namespaces.]identifier
```

Namespace...

A namespace lets you organize code into areas of related functionality by enabling you to attach a name to a grouping of program elements.

```
namespace [parent-namespaces.]identifier
```

If you want to put code in a namespace, the first declaration in the file must declare the namespace. The contents of the entire file then become part of the namespace.

Namespace...

A namespace lets you organize code into areas of related functionality by enabling you to attach a name to a grouping of program elements.

```
namespace [parent-namespaces.]identifier
```

If you want to put code in a namespace, the first declaration in the file must declare the namespace. The contents of the entire file then become part of the namespace.

Namespaces cannot directly contain values and functions. Instead, values and functions must be included in modules, and modules are included in namespaces. Namespaces can contain types, modules.

Modules

A module is a grouping of F# code, such as values, types, and function values.

Modules

A module is a grouping of F# code, such as values, types, and function values.

Grouping code in modules helps keep related code together and helps avoid name conflicts in your program.

Modules

A module is a grouping of F# code, such as values, types, and function values.

Grouping code in modules helps keep related code together and helps avoid name conflicts in your program.

```
module [accessibility-modifier] module-name =
    declarations
```

Modules

A module is a grouping of F# code, such as values, types, and function values.

Grouping code in modules helps keep related code together and helps avoid name conflicts in your program.

```
module [accessibility-modifier] module-name =
    declarations
```

We can build the module of Bstrees!

Modules: List. . .

Modules: List. . .

average: Returns the average of the elements in the list.

Modules: List. . .

average: Returns the average of the elements in the list.

```
// Signature:
List.average : ^T list -> ^T
  (requires ^T with static member (+)
    and ^T with static member DivideByInt
    and ^T with static member Zero)

// Usage:
List.average list

// Example
average([1.0 .. 10.0])
```

Modules: List. . .

averageBy: Returns the average of the elements generated by applying the function to each element of the list.

```
// Signature:
List.averageBy : ('T -> ^U) -> 'T list -> ^U
  (requires ^U with static member (+)
    and ^U with static member DivideByInt
    and ^U with static member Zero)

// Usage:
List.averageBy projection list


// Example
List.averageBy (fun x -> x**2.0) [ 1.0 .. 10.0 ];;
```

Modules: List...

filter : Returns a new collection containing only the elements of the collection for which the given predicate returns true.

```
// Signature:
List.filter : ('T -> bool) -> 'T list -> 'T list

// Usage:
List.filter predicate list

// Example
List.filter (fun x -> x%3=0) [ 1 .. 100 ];;
```

Modules: List...

map: Creates a new collection whose elements are the results of applying the given function to each of the elements of the collection.

```
// Signature:
List.map : ('T -> 'U) -> 'T list -> 'U list

// Usage:
List.map mapping list

// Example
List.map (fun x -> x*x) [ 1 .. 10 ];;
```

Modules: List. . .

reduce: Applies a function to each element of the collection, threading an accumulator argument through the computation.

Modules: List...

reduce: Applies a function to each element of the collection, threading an accumulator argument through the computation.

Given a function f and a list containing i0,i1,i2,...,ik computes:

```
f (... (f i0 i1) i2 ...) ik
```

Modules: List...

reduce: Applies a function to each element of the collection, threading an accumulator argument through the computation.

Given a function f and a list containing i0,i1,i2,...,ik computes:

```
        f (... (f i0 i1) i2 ...) ik
```

```
// Signature:
List.reduce : ('T -> 'T -> 'T) -> 'T list -> 'T

// Usage:
List.reduce reduction list

// Example:
List.reduce (fun x y -> x+y) [1..100]
```

Map-Reduce. . .

Map-Reduce is a programming pattern for processing and generating (possibly big) data sets.

Map-Reduce. . .

Map-Reduce is a programming pattern for processing and generating (possibly big) data sets.

Elements in the data set are not processed in isolation but as part of a gruop.

Map-Reduce. . .

Map-Reduce is a programming pattern for processing and generating (possibly big) data sets.

Elements in the data set are not processed in isolation but as part of a gruop.

The Map-Reduce patter relies on three main functions:

- a filter that restricts the dataset to the elements satisfying a predicate;

Map-Reduce. . .

Map-Reduce is a programming pattern for processing and generating (possibly big) data sets.

Elements in the data set are not processed in isolation but as part of a gruop.

The Map-Reduce patter relies on three main functions:

- a filter that restricts the dataset to the elements satisfying a predicate;
- a map function that processes elements dataset;

Map-Reduce. . .

Map-Reduce is a programming pattern for processing and generating (possibly big) data sets.

Elements in the data set are not processed in isolation but as part of a gruop.

The Map-Reduce patter relies on three main functions:

- a filter that restricts the dataset to the elements satisfying a predicate;
- a map function that processes elements dataset;
- a reduce function that combines result.

To be continued. . .

Functional programming at work

Prof. Michele Loreti

Programmazione Avanzata
Corso di Laurea in Informatica (L31)
Scuola di Scienze e Tecnologie

A simple calculator language. . .

Write an interpreter for a simple calculator language.

A simple calculator language. . .

Write an interpreter for a simple calculator language.

Instructions of the language corresponds to *symbols* you find over the calculator.

A simple calculator language. . .

Write an interpreter for a simple calculator language.

Instructions of the language corresponds to *symbols* you find over the calculator.

We start from a simple calculator:

7
8
9
÷
4
5
6
x
1
2
3
–
0
•
+
=

A simple calculator language. . .

A roadmap. . .

1. Define a datatype for statements;

A simple calculator language. . .

A roadmap. . .

1. Define a datatype for statements;
2. Define the datatype for programs;

A simple calculator language. . .

A roadmap. . .

1. Define a datatype for statements;
2. Define the datatype for programs;
3. Transform strings in programs;

A simple calculator language...

A roadmap...

1. Define a datatype for statements;
2. Define the datatype for programs;
3. Transform strings in programs;
4. Define the interpreter for single statements and for programs.