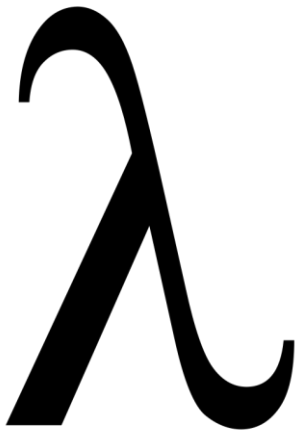


ML

Programmazione Funzionale

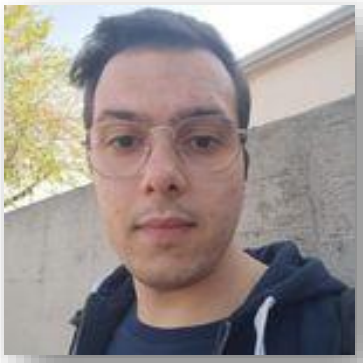
2023/2024

Università di Trento

Chiara Di Francescomarino

Tutoring

- Tue morning 11:30 – 12:30 (Aula PC A202)



Matteo Mariotti

Today

- Recap
- Variables
- Complex types
  - Tuples
  - Lists
- Functions

Recap

Example of expressions

```
> 1+2*3;
val it = 7: int
```

- `val`: value of the expression (7)
- `it`: name of the result
- `int`: type (inferred automatically) of the result

Types

- Basic types

  `unit, int, real, bool, char, string`

  - `unit`: Single value `()`, used for expressions that do not return a value
  - `int`: Integers, positive and negative. Note that `~3` is -3. This is actually an operator, that negates the integer.
- Complex types: constructed starting from other types

Operators

- Arithmetic operators: +,-,*,/, div
- String operators: ^
- Comparisons: =, <, >, <=, >=, <>
- Logical: `not, andalso, orelse`

If-then-else

- Syntax

```
if <p> then <exp1> else <exp2>;
```

- This (like everything in ML) is an expression. Therefore
  - `else` is required
  - Both parts must have values and the resulting expression a well-defined type
- Example

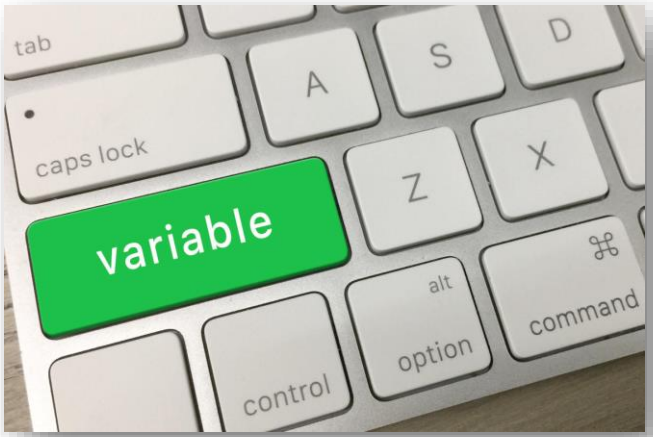
```
> if 1<2 then 3+4 else 5+6;
val it = 7: int
```

Type errors

```
> 1.0 + 2;
poly: : error: Type error in function application.

> #"a" ^ "bc";
poly: : error: Type error in function application.

> 1/2;
poly: : error: Type error in function application.

> if 1<2 then #"a" else "bc";
poly: : error: Type mismatch between then-part and else-
part.
```

Type conversions

- No automatic conversion of types
  - 5+7 and 5.0+7.0 are correct, resulting in `int` and `real`
  - 5+7.0 is wrong
- Conversion between types
  - `real`: integer to real
  - `ceil`, `floor`, `round` and `trunc`: real to integer
  - `ord`: character to integer
  - `chr`: reverse direction, i.e., integer to character
  - `str`: character to string



Variables

Variables

- Environment: Set of pairs of identifiers and values
- It is possible to add an identifier to the environment and bind it to a value
- Environment is modified by val-declarations (a sort of assignment)

```
val <name> = <value>;
val <name>:<type> = <value>;
val <name> = <expression>;
```

- Example

```
> val pi = 3.14159;
val pi = 3.14159: real
> val v = 10.0/2.0;
val v = 5.0: real
```

- Note that the response has the name of the variable, rather than it.

Variables

- We do not need to specify the type
- Variables cannot be modified!
- `val` creates an association between a name and a variable
- A new declaration creates a new variable, and does not change the value of the existing one

```
> val pi = 3;
val pi = 3: int
> val pi = 3.14159;
val pi = 3.14159: real
```

pi	3.14159
pi	3
…	…

create two variables with name `pi`, where the second hides the first

- Note that even the type can change
- Old definition is still there, but (at least at the top-level environment) it is no longer accessible

Variable identifiers

- Any “reasonable" sequence of alphanumeric characters
- We won't use things like this. . .

```
> val $$$ = "ab";
val $$$ = "ab": string
```

Examples

```
> val pi = 3.14159;
val pi = 3.14159: real
> val radius = 4.0;
val radius = 4.0: real
> pi * radius * radius;
val it = 50.26544: real
> val area = pi * radius * radius;
val area = 50.26544: real
```

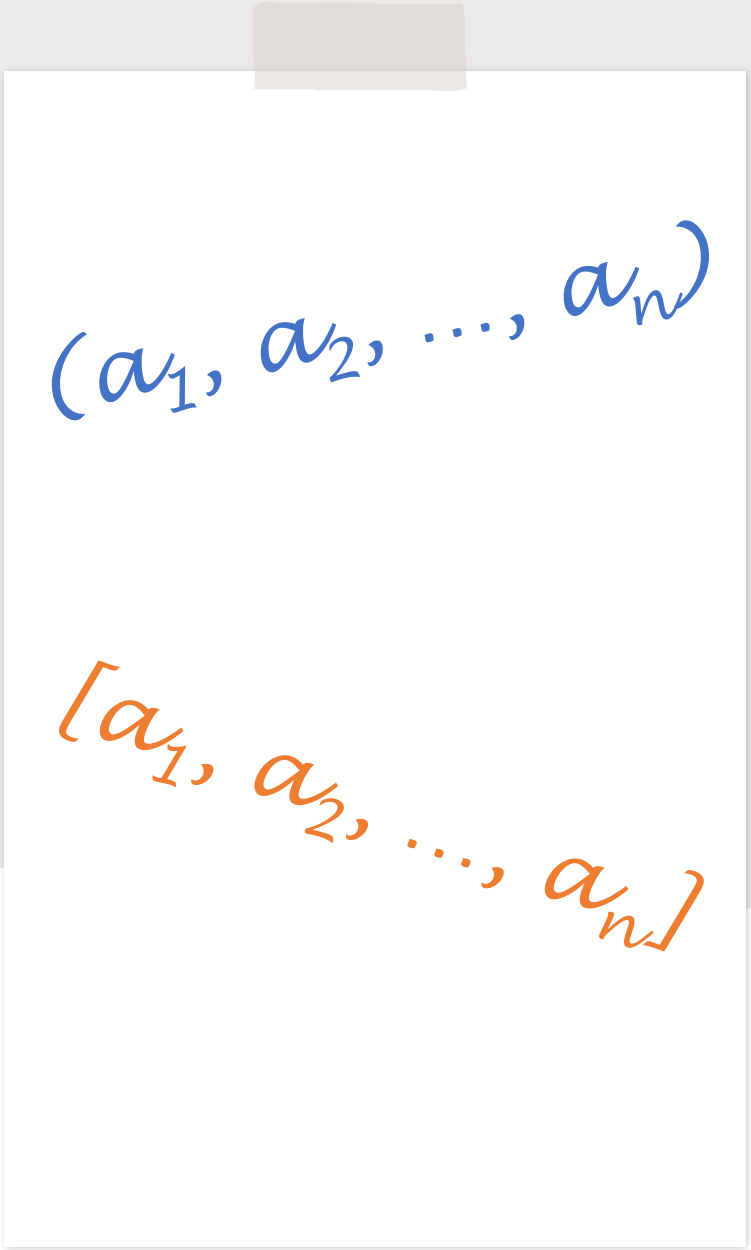
Exercise L2.1

- What does the environment contain after these command?
- `val a = 3;`
- `val b = 98.6;`
- `val a = "three";`
- `val c = a ^ str(chr(floor(b)));`

Solution L2.1

- What does the environment contain after these command?

```
> val a = 3;
val a = 3: int
> val b = 98.6;
val b = 98.6: real
> val a = "three";
val a = "three": string
> val c = a ^ str(chr(floor(b)));
val c = "threeb": string
```



Complex types

$(a_1, a_2, \ldots, a_n)$

Tuples

Tuples

- We have already seen something when looking at the operators
- Example

```
> val t = (1, 2, 3);
val t = (1, 2, 3): int * int * int
```

- We can mix types in tuple definitions

```
> val t = (4, 5.0, "six");
val t = (4, 5.0, "six"): int * real * string
```

- We can also use a complex type instead of a simple one

```
> val t = (1, (2, 3, 4));
val t = (1, (2, 3, 4)): int * (int * int * int)
```

Accessing tuple components:

- Let us consider

```
> val t = (4, 5.0, "six");
```

- Then

```
> #1 (t);
val it = 4: int
> #3 (t);
val it = "six": string
> #4 (t);
poly: : error: Type error in function application.
Function: #4 : 'a * 'b * 'c * 'd -> 'd
Argument: (t) : int * real * string
Reason: Can't unify {4: 'a} to int * real * string (Field 4
missing)
```

$[a_1, a_2, \ldots, a_n]$

Lists

Lists

- Represented with square brackets
- Example

```
> [1,2,3];
val it = [1, 2, 3]: int list
```

- Note that all elements of a list (unlike tuples) must be of the same type

More examples

```
> [1.0,2.0];
val it = [1.0, 2.0]: real list

> [1,2.0];
poly: : error: Elements in a list have different types.
Item 1: 1 : int
Item 2: 2.0 : real
Reason:
Can't unify int (*In Basis*) with real (*In Basis*)
(Different type constructors)

> [];
val it = []: 'a list
```

Note that for an empty list, ML cannot determine the type of the elements

Head of a list: `hd`

- Head: first element of a list

```
> val L = [2,3,4];
val L = [2, 3, 4]: int list
> val M = [5];
val M = [5]: int list

> hd(L);
val it = 2: int
> hd(M);
val it = 5: int
```

Tail of a list: `tl`

- Tail: all the rest

```
> L;
val it = [2, 3, 4]: int list
> M;
val it = [5]: int list

> tl (L);
val it = [3, 4]: int list
> tl (M);
val it = []: int list
```

Note that ML can determine the type of this empty list

Concatenation of lists: @

- Example

```
> [1,2] @ [3,4];
val it = [1, 2, 3, 4]: int list
```

- Note that both lists must be of the same type

```
> [1,2] @ ["a","b"];
poly: : error: Type error in function application.
Function: @ : int list * int list -> int list
Argument: ([1, 2], ["a", "b"]) : int list * string list
Reason:
Can't unify int (*In Basis*) with string (*In Basis*)
(Different type constructors)
```

- Two different types of concatenation
  - ^: Strings
  - @: List

Cons: ::

- An operator that takes an element of type 'a and a list of type 'a list and combines them

```
> 2 :: [3,4];
val it = [2, 3, 4]: int list
> 2 :: nil;
val it = [2]: int list
> 2 :: [];
val it = [2]: int list
```

- :: is right associative

```
> 1 :: 2 :: 3 :: nil;
val it = [1, 2, 3]: int list
> (1 :: (2:: (3::nil)));
val it = [1, 2, 3]: int list
```

- The other way would make no sense

Strings to lists: `explode`

```
> explode ("abcd");
val it = [#"a", #"b", #"c", #"d"]: char list
> explode ("");
val it = []: char list
```

Lists to strings: `implode`

```
> implode ([ #"a", #"b", #"c", #"d"]);
val it = "abcd": string
> implode (nil);
val it = "": string
> implode (explode ("xyz"));
val it = "xyz": string
```

The ML type system

- Basic types int, real, bool, char, string
- Complex types: For now, 2 constructors:
  - `T1 * T2 * ... * Tn (tuples)`
  - `T list`

Examples

```
> [1, 2, 3];
val it = [1, 2, 3]: int list

> ("ab", [1,2,3], 4);
val it = ("ab", [1, 2, 3], 4): string * int
list * int

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

A list of a int*int list

Exercise L2.2

- What are the values of the following expressions?

```
#2 (3,4,5);

hd [3,4,5];

tl [3,4,5];
```

Solutions L2.2

```
> #2 (3,4,5);
val it = 4: int

> hd [3,4,5];
val it = 3: int

> tl [3,4,5];
val it = [4, 5]: int list
```

Exercise L2.3

- What are the values of the following expressions?

```
explode ("foo");

implode ([#"f", #"o", #"o"]);

"c" :: ["a", "t"];
```

Solutions L2.3

```
> explode ("foo");
val it = [#"f", #"o", #"o"]: char list

> implode ([#"f", #"o", #"o"]);
val it = "foo": string

> "c" :: ["a", "t"];
val it = ["c", "a", "t"]: string list
```

Exercise L2.4

- What is wrong with the following expressions? When possible, correct them

```
#4 (3,4,5);

hd([]);

#1 (1);
```

Solutions L2.4

```
> #4 (3,4,5);
poly: : error: Type error in function application.
Function: #4 : 'a * 'b * 'c * 'd -> 'd
> #3 (3,4,5);
val it = 5: int

> hd([]);
poly: : warning: The type of (it) contains a free type
variable. Setting it to a unique
monotype.
Exception- Empty raised

> #1 (1);
poly: : error: Type error in function application.
Function: #1 : {1: 'a, ...} -> 'a
Argument: (1) : int
```

(1) Is not a tuple

Exercise L2.5

- What is wrong with the following expressions? When possible, correct them

```
explode ["bar"];

implode ( #"a", #"b") ;

["r"]::["a","t"];
```

Solutions L2.5

```
> explode ["bar"];
poly: : error: Type error in function application.
Function: explode : string -> char list
Argument: ["bar"] : string list
> explode "bar";
val it = [#"b", #"a", #"r"]: char list

> implode ( #"a", #"b") ;
poly: : error: Type error in function application.
Function: implode : char list -> string
implode [ #"a", #"b"] ;
> implode [ #"a", #"b"] ;
val it = "ab": string

> ["r"]::["a","t"];
poly: : error: Type error in function application.
Function: :: : string list * string list list -> string list list
Argument: (["r"], ["a", "t"]) : string list * string list
> "r"::["a","t"];
val it = ["r", "a", "t"]: string list
> ["r"]@["a", "t"];
val it = ["r", "a", "t"]: string list
```

Exercise L2.6

- What is wrong with the following expressions? When possible, correct them

```
tl [];

1 @ 2;
```

Solutions L2.6

```
> tl [];
poly: : warning: The type of (it) contains a free type
variable. Setting it to a unique
monotype.
Exception- Empty raised

> 1 @ 2;
poly: : error: Type error in function application.
Function: @ : 'a list * 'a list -> 'a list
Argument: (1, 2) : int * int
```

Exercise L2.7

- What are the types of the following expressions?

```
(1.5, ("3",[4,5]));

[[1,2],nil,[3]];
```

Solutions L2.7

```
> (1.5, ("3",[4,5]));
val it = (1.5, ("3", [4, 5])): real * (string *
int list)

>[[1,2],nil,[3]];
val it = [[1, 2], [], [3]]: int list list
```

Exercise L2.8

- What are the types of the following expressions?

```
[ (2,3.5), (4,5.5), (6,7.5)];

([#"a", #"b"], [nil,[1,2,3]]);
```

Solutions L2.8

```
> [ (2,3.5), (4,5.5), (6,7.5)];
val it = [(2, 3.5), (4, 5.5), (6, 7.5)]: (int *
real) list

> ([#"a", #"b"], [nil,[1,2,3]]);
val it = ([#"a", #"b"], [[], [1, 2, 3]]): char
list * int list list
```

Exercise L2.9

- Give examples of objects of the following types, without using empty lists

```
int list list list

(int * char) list

string list * ( int * (real * string))
* int
```

Solutions L2.9

```
> [[[1,2]],[[3,4]]];
val it = [[[1, 2]], [[3, 4]]]: int list list
list

> [(1,#"a"), (2,#"b")];
val it = [(1, #"a"), (2, #"b")]: (int * char)
list

> ( ["ab","cd"], (4,(2.5,"ef")), 7);
val it = (["ab", "cd"], (4, (2.5, "ef")), 7):
string list * (int * (real * string)) * int
```

Exercise L2.10

- Give examples of objects of the following types, without using empty lists

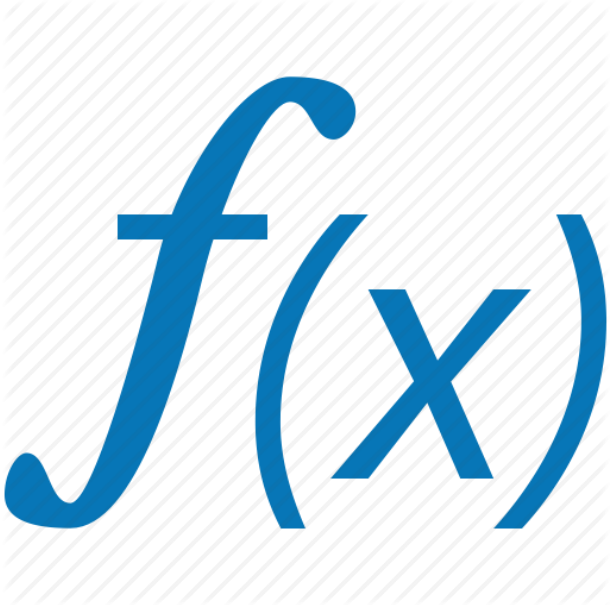
```
((int * int) * (bool list) * real) *
(real * string)
```

```
(bool * int) * char
```

```
real * int list list list list
```

Solutions L2.10

```
> (((5,6), [true, false], 5.6), (6.7, "abc"));
val it = (((5, 6), [true, false], 5.6), (6.7,
"abc")):
((int * int) * bool list * real) * (real *
string)


> ((true, 7), #"a");
val it = ((true, 7), #"a"): (bool * int) * char


> (7.8, [[[[1,2],[3,4]]]]);
val it = (7.8, [[[[1, 2], [3, 4]]]]): real * int
list list list list
```



Functions

Functions

- In ML, just another type of value
- Represented by parametrized expressions
- Calculate a value based on parameters
  - No collateral effects
- Syntax `fn` (corresponds with λ in the λ-calculus, that we will see later)

```
fn <param> => <expression>;
```

- Example

```
fn n => n+1;
```

- We can directly apply the function to the parameter

```
(fn n => n+1) 5;
```

value 5 is associated to formal parameter n, and then the function is evaluated

Functions and names

- We can associate the functions to a name, just like values

```
> val increment = fn n => n+1;
val increment = fn: int -> int
```

- We also have a syntactic sugar notation for functions with names

```
> fun increment n = n+1;
val increment = fn: int -> int
```

- And then write

```
> increment 5;
val it = 6: int
```

Function types

- Example: function that converts character from lower to upper case

```
> fun upper(c) = chr (ord(c) - 32);
val upper = fn: char -> char
```

- Poly gives the type of the function. This is
  - The keyword `fn`
  - The type of the argument
  - The symbol `->`
  - The type of the result
- When using the function:

```
> upper (#"b");
val it = #"B": char
```

or

```
> upper #"b";
val it = #"B": char
```

The ML type system

- If `T1` and `T2` are types, so is

  ```
  T1 -> T2
  ```

- This is the type of functions that take an object of type `T1` and produce one of type `T2`
- Note that `T1` and `T2` can be any type, including function types

Specifying types

- ML deduces the types of functions automatically, as in our first example
- Another example

```
> fun square (x) = x * x;
val square = fn: int -> int
```

- But multiplication is also defined for reals. If we want to square real numbers, we can write

```
> fun square (x:real) = x * x;
val square = fn: real -> real
```

Examples

- Use of a function

```
> radius;
val it = 4.0: real
> pi;
val it = 3.14159: real
> pi * square (radius);
val it = 50.26544: real
```

- or

```
> pi * square radius;
val it = 50.26544: real
```

Multiple arguments

- All functions in ML have exactly one parameter but this parameter can be a tuple
- Example: Largest of three reals

```
> fun max3(a:real,b,c) = (* maximum of reals *)
if a>b then
if a>c then a else c
else
if b>c then b else c;
val max3 = fn: real * real * real -> real

> max3(5.0,4.0,7.0);
val it = 7.0: real
```

Note the syntax for comments

What would happen if we didn't specify that a was a real?

The input is a tuple of 3 int

```
>fun max3(a,b,c) =
  if a>b then
    if a>c then a else c
    else
      if b>c then b else c;
val max3 = fn: int * int * int -> int

> max3 (4,5,7);
val it = 7: int
```

Tuples as parameters

- Since the function takes a single parameter, of type tuple, we can also write

```
> val t = (4,5,6);
val t = (4, 5, 6): int * int * int
> max3 t;
val it = 6: int
```

Exercise L2.11

- Write a function to compute the cube of a real number

Solutions L2.11

```
> fun cube (x:real) = x * x * x;
val cube = fn: real -> real

> cube (2.9);
val it = 24.389: real

> cube 2;
poly: : error: Type error in function
application.
Function: cube : real -> real
Argument: 2 : int
```

Exercise L2.12

- Write a function to compute the smallest component of a tuple of type int*int*int

Solution L2.12

```
> fun min3 (a,b,c) =
  if a<b then if a<c then a else c
  else
  if b<c then b else c;
val min3 = fn: int * int * int -> int

> min3 (2,3,4);
val it = 2: int
> min3 (3,2,4);
val it = 2: int
> min3 (4,3,2);
val it = 2: int
```

Exercise L2.13

- Find the third element of a list (it doesn't have to work properly on shorter lists)

Solution L2.13

```
> fun third (l) = hd (tl(tl(l)));
val third = fn: 'a list -> 'a

> third [2,3,4];
val it = 4: int
> third [2,3,4,5];
val it = 4: int
> third [2,3];
Exception- Empty raised
```

Exercise L2.14

- Write a function to reverse a tuple of length 3

Solution L2.14

```
> fun reverse(a,b,c) = (c,b,a);
val reverse = fn: 'a * 'b * 'c -> 'c * 'b * 'a

> reverse (1,2,3);
val it = (3, 2, 1): int * int * int
> reverse (1.0,2,"a");
val it = ("a", 2, 1.0): string * int * real
```

Exercise L2.15

- Find the third character of a string (it doesn't have to work properly on shorter strings)

Solution L2.15

```
> fun thirdchar(s) = third(explode s);
val thirdchar = fn: string -> char

> fun thirdchar(s) = hd(tl(tl(explode
s)));
val thirdchar = fn: string -> char


> thirdchar "abcd";
val it = #"c": char
```

Exercise L2.16

- Cycle a list once, i.e., convert `[a1,...,an]` to `[a2,...,an,a1]`. It doesn't have to work on the empty list

Solution L2.16

```
> fun cycle (l) = tl(l) @ [hd(l)];
val cycle = fn: 'a list -> 'a list

> cycle [1,2,3,4];
val it = [2, 3, 4, 1]: int list
> cycle [1,2];
val it = [2, 1]: int list
> cycle [1];
val it = [1]: int list
```

Exercise L2.17

- Given 3 integers, produce a pair consisting of the smallest and the largest

Solution L2.17

```
> fun min3 (a,b,c) = if a<b then if a<c then a else c
                     else
                     if b<c then b else c;
val min3 = fn: int * int * int -> int
> fun max3 (a,b,c) = if a>b then if a>c then a else c
                     else
                     if b>c then b else c;
val max3 = fn: int * int * int -> int
> fun min_max_pair (a,b,c) = (min3(a,b,c),max3(a,b,c));
val query = fn: int * int * int -> int * int

> query (1,2,3);
val it = (1, 3): int * int
> query(3,4,2);
val it = (2, 4): int * int
```

Exercise L2.18

- Given three integers (a tuple), produce a list of them in sorted order

Solution L2.18

```
> fun medium (a,b,c) = if a<b then if b<c then b else if a<c then c
else a else if b>c then b else if a<c then a else c;
val medium = fn: int * int * int -> int
> fun sort (a,b,c) = min(a,b,c)::medium(a,b,c)::[max(a,b,c)];
val sort = fn: int * int * int -> int list

> sort3 (3,2,1);
> fun sort (a,b,c) = [min3(a,b,c)] @ [if a<b then if b<c then b else if
a<c then c else a else if c<b then b else if a<c then a
else c] @ [max3(a,b,c)];
val sort = fn: int * int * int -> int list

> sort (1,2,3);
val it = [1, 2, 3]: int list
> sort (3,2,1);
val it = [1, 2, 3]: int list
> sort (1,3,2);
val it = [1, 2, 3]: int list
```

Exercise L2.19

- Round a real number to the nearest decimal (10^{th}) (e.g., 2.56 → 2.6)

Solution L2.19

```
> fun rnd (r:real) = real (round(r *10.0)) / 10.0;
val rnd = fn: real -> real

> rnd (5.678);
val it = 5.7: real
> rnd 5.628;
val it = 5.6: real
```

Exercise L2.20

- Given a list, remove the second element. It doesn't need to work on lists shorter than 2.

Solution L2.20

```
>fun rem l = hd(l) :: tl(tl(l));
val rem = fn: 'a list -> 'a list

> rem [1,2,3,4];
val it = [1, 3, 4]: int list
> rem [1,2];
val it = [1]: int list
```

Summary

- Variables
- Complex types
  - Tuples
  - Lists
- Functions

Next time

- Recursion
- Patterns