- val pi = 3.14159;
  > val pi = 3.14159 : real
- val a = pi* 2.0 *2.0;
  > val a = 12.56636 : real
- val a = 2 * pi;

...  
! Type clash: expression of type
!   real
! cannot have type
!   int

- val area = fn r => pi*r*r;
  > val area = fn : real -> real
- val sqr = fn r => r*r;
  > val sqr = fn : int -> int
- val sqr = fn r:real => r*r;
  > val sqr = fn : real -> real
- val sqr = fn r => r*r:real;
  > val sqr = fn : real -> real
- fun area (r) = pi*r*r;
> val area = fn : real -> real
- val pi = "yabedabadoo";
> val pi = "yabedabadoo" : string
- area(2.0);
> val it = 12.56636 : real
- area;
> val it = fn : real -> real
- it(2.0);
> val it = 12.56636 : real
- it(2);

...
! Type clash: expression of type
!   int
! cannot have type
!   real
Numeric Types

**int**: the integers

- constants 0 1 ~1 2 ~2 0032...
- infix operators + - * div mod

**real**: the floating point numbers

- constants 0.0 ~1.414 2 3.94e~7 ...
- infix operators + - * /
Overloading

Functions defined for both int and real:

- operators ~ + - *
- relations < <= > >=

You must tell the type checker what type is intended, if there is any ambiguity.
Useful library of functions, collected together into structures.

\begin{center}
\begin{tabular}{ccc}
Int & Real & Math \\
\end{tabular}
\end{center}

The basis library is automatically loaded when using SML/NJ.

May need to be explicitly loaded in Moscow ML.

\begin{verbatim}
> load "Math";
> val it = () : unit
- fun f u = Math.sin(u)/u;
> val f = fn : real -> real
\end{verbatim}

To load your own file of definitions:

- use "myfile";
**Strings**

**Type** string

- **constants** "" "A" "yaba!!dabadoo$\n"

- **size**: string -> int
determines the number of characters in a string.

- **s1 ~ s2**
  the concatenation of strings s1 and s2

- **relations** < <= > >=

**Structure** String
**Characters**

**Type** char

- **constants** "A" "y" " "

- **ord**: char → int integer value of a character.

- **chr**: int → char

- **relations** < <= > >=

**Structure** Char
Truth Values

Type bool

- constants true false
- not: bool -> bool
- if p then x else y

$p \text{ andalso } q$

\[
\text{if } p \text{ then } q \text{ else false}
\]

$p \text{ orelse } q$

\[
\text{if } p \text{ then true else } q
\]

Structure Bool
Pairs and Tuples

- (2,3);
> val it = (2, 3) : int * int

- (2.0,2,3,"aa");

> val it = (2.0, 2, 3, "aa") : real * int * int * string

Tuples are useful for representing vectors, presenting functions with multiple arguments, obtaining multiple results from a function, etc.

- fun addtwice (m,n) = m + 2*n;
> val addtwice = fn : int * int -> int
- fun negvec(x,y):real*real = (\sim x, \sim y);

> val negvec = 
fn : real * real -> real * real

- negvec(1.0,1.0);

> val it = (\sim 1.0, \sim 1.0) : real * real

- fun addvec((x1,y1),(x2,y2)):real*real = (x1+x2,y1+y2);

> val addvec = fn : (real * real) *  
   (real * real) -> real * real

- fun subvec(v1,v2) = addvec(v1,negvec v2);

> val subvec = fn : (real * real) *  
   (real * real) -> real * real