TODAY’S AGENDA

HW1 Reminders/Tips
Type Synonyms
Type Generality
Equality Types
More Syntactic Sugar
HW1 REMINDERS

• Don’t use pattern-matching! (just for the hw)

• If you want to introduce a new variable in your function, use a let expression. (Good for avoided repeated calculations).

• Test your code & include ‘tests’ in submission

• Tests can just be calling functions on different inputs, and manually inspecting the output (put ‘tests’) in hw1_test.sml

• What to test? “Some, one, none”
What does `int * int * int` represent?

In HW1 we’re calling it a date

Wouldn’t it be nice to reflect this representation in the source code itself?

```
type date = int * int * int
```
**TYPE VS. DATATYPE**

**datatype** introduces a new type name, distinct from all existing types

```java
datatype suit = Club | Diamond | Heart | Spade
datatype rank = Jack | Queen | King | Ace | Num of int
```

type is just another name

```java
type card = suit * rank
```
TYPE SYNONYMS

Why?

• For now, just for convenience
• It doesn’t let us do anything new
• Easier to read code

Later in the course we will see another use related to modularity.
Let's write a function that appends two string lists...
TYPE GENERALITY

We would expect

\[
\text{string list} \times \text{string list} \rightarrow \text{string list}
\]

• But the type checker found

\[
\text{'a list} \times \text{'a list} \rightarrow \text{'a list}
\]

• Why is this OK?
MORE GENERAL TYPES

The type

`'a list * 'a list -> 'a list`

is more general than the type

`string list * string list -> string list`

and “can be used” as any less general type, such as

`int list * int list -> int list`
MORE GENERAL TYPES

The type

`'a list * 'a list -> 'a list`

is not more general than the type

`int list * string list -> int list`

Takeaway: More general types “can be used” as any less general type.
THE TYPE GENERICITY RULE

The “more general” rule

A type $t_1$ is more general than the type $t_2$ if you can take $t_1$, replace its type variables consistently, and get $t_2$
EQUALITY TYPES

Let's write a list containment function…
EQUALITY TYPES

The double quoted variable arises from use of the \( = \) operator

- We can use \( = \) on most types like \texttt{int}, \texttt{bool}, \texttt{string}, tuples (that contain only “equality types”)
- Functions and \texttt{real} are not ”equality types”

Generality rules work the same, except substitution must be some type which can be compared with \( = \)
If-then-else is implemented as syntactic sugar for a case statement.
Let's write a function using pattern-matching that acts like an if-expression returning something of type int.