STANDARD ML

MODULES

a structure is a collection of bindings

```
structure MyModule = struct
    (*bindings*)
end
```

a module can contain any kind of binding

```
structure MyModule = struct
    val answer = 42
    exception Failure of int
    type key = int
    fun foo x = x
end;
```

outside a module refer to a binding from another module by:

```
MyModule.answer;
```

```
open MyModule;
answer;
```

- used to get direct access to the bindings of a module
- considered bad style

SIGNATURES

a signature is a type for a module

```
signature SIGNAME = sig
     (*types for bindings*)
end

structure ModuleName :> SIGNAME = struct
     (*bindings*)
end
```

```
signature MATHLIB = sig
    val pi: real
    val deg2rad: real -> real
end;

structure MathLib :> MATHLIB = struct
    val pi = 3.14
    fun deg2rad x = x / 180.0 * pi
end;
```

a module will not type-check unless it matches the signature

```
signature CONSTANTS = sig
    val pi: real
    val e: real
end;
```

```
structure MathConstants :> CONSTANTS = struct
    val pi = 3.14
end; (*ERROR*)
```

```
structure MathConstants :> CONSTANTS = struct
    val pi = 3
    val e = 2.71
end; (*ERROR*)
```

SIGNATURE MATCHING

```
structure Foo :> BAR
```

- every type in `BAR` is provided in `Foo` as specified
- every val binding in `BAR` is provided in `Foo`
- every exception in `BAR` is provided in `Foo`
- `Foo` can have more bindings than specified by `BAR`

FUNCTORS

a functor is a parameterized module

```
functor Functor (Module: SIG) =
  struct
    (*bindings*)
  end;
```

applying a functor

```
structure FModule = Functor(Module);
```

```
signature ORDERED_TYPE = sig
  type t
  val compare: t * t -> order
end;

structure Int' = struct
  type t = int
  val compare = Int.compare
end;
```

```
functor SortedList (Elt: ORDERED_TYPE) = struct
  fun add x [] = [x]
    | add x (hd :: tl) = case Elt.compare (x, hd) of
        EQUAL => hd::tl
      | LESS => x :: hd :: tl
      | GREATER => hd :: (add x tl)
  (*more functions*)
end;
```

```
structure SortedIntList = SortedList(Int');

open SortedIntList;
add 5 (add 6 (add 2 (add 4 (add 3 (add 1 []))))); 
```

```
structure SortedStringList = SortedList(struct
  type t = string
  val compare = String.compare
end);
open SortedStringList;
add "abc" (add "hij" (add "efg" (add "nop" (add "klm" [])))); 
```