CSE216
Foundations of Computer Science

Instructor: Zhoulai Fu

State University of New York, Korea

Some slides taken from Cornell's CS3110. Thanks!
https://www.cs.cornell.edu/courses/cs3110/2014fa/lecture_notes.php

Today

- Data types in Ocaml

Data types

Data type

- Sum Type
- Record type
- Tuple type

Sum type

```
type day = Mon | Tue | Wed | Thu | Fri |
Sat | Sun
```

- Superficially similar to an enum in Java or C
- Created a one-of type named day
- Created seven constructors from Mon to Sun
- That's effectively how Booleans are defined in OCaml:

```
type bool = false | true
```

Example

```
# type suit = Heart | Diamond| Club | Spade;;
type suit = Heart | Diamond | Club | Spade
```

```
# Club ;;
- : suit = Club
```

Example: day_to_int

```
let day_to_int (d : day) =
  if d=Mon then 1
  else if d=Tue then 2
  else if d=Wed then 3
  else if d=Thu then 4
  else if d=Fri then 5
  else if d=Sat then 6
  else (* d=Sun *) 7
```

Note: Ocaml compiler/interpreter does not see new lines

An alternative (and better) way: Pattern matching

```
let day_to_int (d : day) = match d with
  Mon -> 1 (* you can put an “|” in front)
| Tue -> 2
| Wed -> 3
| Thu -> 4
| Fri -> 5
| Sat -> 6
| Sun -> 7
```

Record type

```
type time = {hour: int; min: int; ampm: string}
```

Record type

```
type time = {hour: int; min: int; ampm: string}
```

– Creates a *each-of type* named **time**

Record type

```
type time = {hour: int; min: int; ampm: string}
```

– Creates a *each-of type* named **time**

- To *build* a record:

```
{hour=10; min=10; ampm="am"}
```

– order of *fields* doesn't matter; could write

```
{min=10; ampm="am"; hour=10}
```

Record type

```
type time = {hour: int; min: int; ampm: string}
```

– Creates a *each-of type* named **time**

- To *build* a record:

```
{hour=10; min=10; ampm="am"}
```

– order of *fields* doesn't matter; could write

```
{min=10; ampm="am"; hour=10}
```

- To *access* fields of record variable **t**:

```
t.min
```

Exercise

- Which kind of data type, sum type or record type would be better represented with record types rather than sum types?

    - Coins, which can be pennies, nickels, dimes, or quarters
    - Students, who have names and NetIDs
    - A dessert, which has a sauce, a creamy component, and a crunchy component
    - MBTI types, ISTJ, ESTJ etc

Exercise

- Define a data type **student**, which has a name (string) and a NetID (int)

Pair type

- type t = int * string

- type point = float * float

Pairs

A **pair** of data: two pieces of data glued together

e.g.,

- `(1,2)`
- `(true, "Hello")`
- `("cs", 3110)`

- Syntax: `(e1,e2)`

- Evaluation:
  - If `e1-->v1` and `e2-->v2`
  - Then `(e1,e2)--> (v1,v2)`
  - A pair of values is itself a value

- Type-checking:
  - If `e1:t1` and `e2:t2`,
  - then `(e1,e2):t1*t2`

Accessing Pairs

- **Syntax**: **`fst e`** and **`snd e`**
  - *Projection functions*

- **Evaluation**:
  - If **`e-->(v1,v2)`**
  - then **`fst e --> v1`**
  - and **`snd e --> v2`**

- **Type-checking**:
  - If **`e: ta*tb`**,
  - then **`fst e`** has type **`ta`**
  - and **`snd e`** has type **`tb`**

Tuples

- `(e1,e2,…,en)`
- `t1 * t2 * … * tn`
- `fst e, snd e, ???`

Pattern matching tuples

```
let sum_triple (triple:int*int*int)  =
    let (x, y, z) = triple
    in x + y + z
```

- `(x, y, z)` is a *pattern*
  - because it's on the LHS of equals in `let`

Exercise: Rewrite sum_triple using match … with … syntax

Pattern matching records

The same syntax works for records:

```
type stooges = {larry:int; moe:int; curly:int}

let sum_stooges (s:stooges)  =
    let {larry=x; moe=y; curly=z} = s
    in x + y + z
```

Exercise 1

1 Define an algebraic datatype `grade` that represents a grade in a course. The datatype should include constructors for an A, B, C, D, and F.

Exercise 2

2 Then, write a function `to_num` that takes a grade as input and returns a numerical equivalent, where A is 4.0, B is 3.0, C is 2.0, D is 1.0, and F is 0.0. Your function should have the following type.

```
val to_num: grade -> float
```

Exercise 3

Define a type, `time`, which holds the hour, minute, and second as separate values.

Exercise 4

Write a function `seconds_since_midnight2` with the following type:

```
val seconds_since_midnight2 : time -> int
```