OCaml: Tuples

Programming Languages

William Killian

Millersville University

Tuples

- Tuples are a ***product type***
- Used for when we want to group entities together
- Elements are access by location

```
type student = string * int * float
```

- We created a new type called student
- It is an *alias* (or another name for a tuple)
- This tuple contains a string, an int, and a float

Tuple Syntax

- How could we store a point?
- What is its datatype (as a tuple)?
- How can we create a new point?

Tuple Syntax

- How could we store a point?
  - We should be able to store a point as a pair of coordinates
  - We can access its data by "location"
- What is its datatype (as a tuple)?

  ```
  type point = float * float
  ```

  This means that a point is modeled as two floats
- How can we create a new point?

  ```
  let my_point = (1.2, 0.0)
  let my_point : point = (1.2, 0.0)
  let my_point : float * float = (1.2, 0.0)
  (* all three of these are the same! *)
  ```

Tuple Syntax

Expression / Value:

```
("Will Killian", 327291, 3.38)
```

Type:

```
string * int * float
```

Always enclosed in parentheses

Datatypes can be deduced for each element

Immutable – you cannot change a tuple

- You can read from a tuple
- You can create a new tuple

Tuple Bindings

Binding refresher: providing a name to a value

```
let point = (2.0, 3.14)
```

Extracting the “x” value of the point:

```
  let (x, _) = point
```

Extracting the “y” value of the point:

```
  let (_, y) = point
```

Note: The _ means to ignore

Tuple Bindings

```
let big = (1, 3.14, "hello", true, 5)
```

1. What is the type of big?

2. How can we extract the 2nd, 4th, and 5th elements with identifiers "pi", "passing", and "courses" ?

3. How can we compare the 1st and 5th element for equality? (hint: two steps)

Tuple Bindings

```
let big = (1, 3.14, "hello", true, 5)
```

1. What is the type of big?

   ```
   int * float * string * bool * int
   ```

2. How can we extract the 2nd, 4th, and 5th elements with identifiers "pi", "passing", and "courses" ?

   ```
   let (_, pi, _, passing, courses) = big
   ```

3. How can we compare the 1st and 5th element for equality? (hint: two steps)

   ```
   let eq = let (first, _, _, _, last) in
             first = last
   ```

Pattern Matching

- Tuples can lend to clean, expressive code when combined with pattern matching
- Can be combined with other patterns (e.g. for lists)

Problem: Compute the centroid (geometric average) of three points which form a triangle.

```
let points = [(0.0, 1.0),
              (6.0, 2.0),
              (3.0, 5.0)]
```

What is the type of points?

Pattern Matching Examples

Normal List:

```
match l with
| [] -> (* empty list *)
| h::t -> (* have more *)
```

Normal Tuple:

```
match p with
| (0,0) -> (* origin *)
| (x,y) -> (* general point *)
```

Centroid

```
let centroid lst =
  let rec average sum n lst =
    match lst with
    | [] ->
        let (x, y) = sum in (* pull out each coordinate *)
          (x /. n, y /. n) (* compute average *)
    | (x,y)::lst’ ->
        (* pull out each coordinate *)
        let (xs, ys) = sum in (* evolve arguments *)
          average (x +. xs, y +. ys) (n +. 1.0) lst'
  in
  average (0.0, 0.0) 0.0 lst (* sum=(0.0, 0.0) n=0.0 *)
```

Pattern Matching Problem

- Count the number of origin points in a list

```
let rec count_origin lst =
```

Pattern Matching Problem

- Count the number of origin points in a list

```
let rec count_origin lst =
  match lst with
  | []            -> 0
  | (0,0)::lst' -> count_origin lst' + 1
  | _::lst'       -> count_origin lst'
```

Pattern Matching Problem

- Count the number of origin points in a list

```
let rec count_origin lst =
  match lst with
  | []       -> 0
  | p::lst' -> count_origin lst' +
                (if p = (0,0) then 1 else 0)
```