## Lecture 1: Introduction to OCaml

**CS 6371: Advanced Programming Languages**  
January 18, 2022

| OCaml has a built-in type “int” that supports the usual binary operators. | #1+1;;  
- : int = 2  
#1+2*3;;  
- : int = 7 |
| --- | --- |
| Use “let” to define a function. OCaml responds by telling you the “type” of the new function you’ve created. This one is a function from two integers to an integer. | #let add x y = x+y;;  
add : int -> int -> int = <fun>  
#add 3 4;;  
- : int = 7 |
| Instead of applying a function with syntax “f(x,y)”, OCaml uses syntax “(f x y)”. | #let hypotenuse x y =  
let xsquared = x*x in  
let ysquared = y*y in  
(xsquared + ysquared);;  
hypotenuse : int -> int -> int = <fun> |
| “let … in …” can be used within a function definition to declare variables and assign them values. Note that a variable’s definition never changes! It is assigned exactly once. | #if 3<4 then (add 1 2) else (add 5 6) ;;  
- : int = 3 |
| In OCaml, “if…then…else…” is an expression not a command. It’s like “… ? … : …” in C or Java. | #let test x = if x<4 then “yes” else 0;;  
Toplevel input:  
>let test x = if x<4 then “yes” else 0;;  
> This expression has type int, but is used with type string. |
| The two branches of the “if” must return values of the same type. The example produces an error because one branch returns a string while the other returns an int. | #true;;  
- : bool = true  
#false;;  
- : bool = false |
| In addition to integers and strings, OCaml also has booleans. Conjunction is “&&” and disjunction is “||” just like in C or Java. Unlike C, booleans and integers are not interchangeable! | "foo" ^ "bar";;  
- : string = "foobar" |
| The “^” operator performs string concatenation. | #let rec factorial n =  
  if n<=1 then 1 else n*(factorial (n-1));;  
factorial : int -> int = <fun>  
#type color = Red | Blue | Dark of color | Light of color;;  
Type color defined.  
#Red;;  
- : color = Red  
#Dark Blue;;  
- : color = Dark Blue  
#Light (Dark Blue);;  
- : color = Light (Dark Blue) |
| A “recursive function” calls itself. To define a recursive function, put “rec” after the “let”. | #Light Dark Blue;;  
Toplevel input:  
>Light Dark Blue;;  
> This expression has type color -> color, but is used with type color. |
| In OCaml you can define your own types with the “type” directive. In this type, “Red”, “Blue”, “Dark”, and “Light” are the “type constructors” for type “color”. | Notice that I used parentheses in the last example. If I hadn’t, an error would have resulted. This is because type constructors associate left by default. |
The “match … with …” operator allows you to test whether a value matches a type constructor. The left side of each -> is called a “pattern”. Patterns can contain variables. If the pattern matches, the variables become bound to the respective parts of the value being tested and may be used with the right-hand side of the ->.

Anywhere you would normally put a variable in a pattern you can instead put an underscore. Underscore matches to anything (just like a variable) except that it doesn’t bind any variable to the matching sub-expression.

OCaml has a list type. Lists are enclosed in brackets and elements are separated by semicolons. The :: operator (called “cons”) inserts an element onto the head of a list.

All elements of a list must have the same type.

You can use “match” to match lists. The pattern “[]” matches the empty list. Pattern “a::b” matches a list with at least one element. Pattern “a::b::c” matches a list with at least two elements, etc.

A “tuple” is a fixed-length collection of values. The members of the collection need not have the same type. This is an example of a string-int pair.

Tuples are useful when you want to return more than one value from a function.

The tuple with zero elements is called “unit”. It is useful when you don’t want to pass anything to a function.