OCaml

Getting Started Reference Sheet

Administrivia

- Command line interpreter, REPL, begun with `ocaml` and exited with `exit 0;;`
  - All expressions in REPL must be terminated with `;;`, not so in a script.

    ```
    # let rec x = 4 and woah (a, b) c = print_endline a ; print_endline b; x + c ;;
    val x : int = 4
    val woah : string * string -> int -> int = <fun>

    # woah ("currying", "or not") 12 ;
    currying
    or not
    - : int = 16
    ```

    Note the keywords for mutual recursion: `let rec ... and ...`.
  - To load a source file enter `#use "myfile.ml;;"`.
- Only multi-line, nestable, comments: `(* ... *)`.
- All declarations are preceded by `let` or `let rec` for recursive ones.
- Sequencing is via `;` and the result is the value of the final expression.
- Anonymous functions use the syntax: `fun x ... x -> ...`
  - `function` in-place of `fun` also works.
  - Infix functions can be used in prefix by enclosing them in parens; e.g., `(+) 1 2`.

Strings

- Double quotes for strings, single quote for characters, and a single quote may be used as part of an identifier.
  - String catenation with `(^)`.
  - Not arrays, or lists, of characters as in C or Haskell.
  - Expected `Print.printf string args`.
  - Also `print_string` and `read_line ()`.

Lists

- Syntax: `[x; ...; x]`
  - Tuples are optionally enclosed in parens; hence `[x, ..., x]` is a singleton list consisting of only one tuple!
- Expected functionals: `List.nth` for list lookup, `List.map`, `List.filter`, etc.
- Cons operation is denoted `::`.
- Arrays have syntax `[|x; ...; x|]` with 0-indexing lookup `arr.(n)`.

Type Construction

We can make an alias: `type myInt = int`
However we can also make a new ADT and pattern match on it.

```
type 'a roseTree = Leaf of 'a | MkRoseTree of 'a roseTree list

let rec treeFilter p = function
  | MkRoseTree ts   -> MkRoseTree (List.map (treeFilter p) ts)
  | Leaf a when p a -> Leaf a
  | _               -> MkRoseTree []
```

- As in Haskell, constructors must start with a capital letter **however** type names must begin with a lowercase letter.
- We may omit the `of ...` to obtain nullary constructors, as expected.
- Notice the optional guard `when` and the wildcard pattern `_`.

Example usage:

```
let test = MkRoseTree [ Leaf 1 ; MkRoseTree [ Leaf 2; Leaf 3] ; MkRoseTree [Leaf 4] ]

let rec even = function
 | 0 -> true
 | 1 -> false
 | n -> even (n-2)

(*
# treeFilter even test;;
- : int roseTree =
MkRoseTree
 [MkRoseTree []; MkRoseTree [Leaf 2; MkRoseTree []]; MkRoseTree [Leaf 4]]
*)
```