ML 1 – Basic OCaml

CS 421 – Fall 2016

Revision 1.1

Assigned August 23, 2016
Due August 30, 2016 – September 2, 2016
Extension None past the allowed lab sign-up time

1 Change Log

1.1 Fixed the instructions for `double_minus_one`.

1.0 Initial Release.

2 Objectives and Background

The purpose of this ML is to test the student's ability to

- start up and interact with OCaml;
- define a function;
- write code that conforms to the type specified (this includes understanding simple Ocaml types, including functional ones);

Another purpose of MPs and MLs in general is to provide a framework to study for the exam. Several of the questions on the exam will appear similar to the MP and ML problems. By the time of the exam, your goal is to be able to solve any of the following problems with pen and paper in less than 2 minutes.

3 Done in Computer-Based Testing Facility

You are asked to sign up for a time next week to go to the CBTF, where you will be given a random portion (less than 25%) of this assignment to complete for your grade for the assignment. We recommend that you do the whole assignment beforehand to be comfortable with the problems, so you will be able to do the problems efficiently when you go in. To help yourself in that, please read the *Guide for Doing MPs* at

`http://courses.engr.illinois.edu/cs421/mps/index.html`

Be aware that if we have difficulties with the CBTF, this assignment will revert to an MP and will be turned in, in full, in the same manner as MPs.

4 Problems

Note: In the problems below, you do not have to begin your definitions in a manner identical to the sample code, which is present solely for guiding you better. However, you have to use the indicated name for your functions and values, and they will have to conform to any type information supplied, and have to yield the same results as any sample executions given, as well as satisfying the specification given in English. You need to leave the **`open Ml1common`** directive at the top of the file. If you remove it, your assignment may be graded incorrectly, costing you points.

1. (1 pt) Declare a variable `random` with the value `179`. It should have type `int`.

2. (1 pt) Declare a variable `title` with the value `"ML 1 -- Basic OCaml"`. It should have type `string`. It should **not** contain a "newline".

3. (1 pt) Declare a variable `greetings` with the value `"Hi there."`. It should have type `string`. It should **not** contain a "newline".

4. (1 pt) Declare a variable `salute` with a value of `"Greetings, my friend!"`. It should have the type of `string`.

5. (1 pt) Declare a variable `a` with the value `17`. It should have type `int`.

6. (1 pt) Declare a variable `pi` with a value of `3.14159`. It should have the type of `float`.

7. (1 pt) Declare a variable `e` with a value of `2.71828`. It should have the type of `float`.

8. (1 pt) Declare a variable `quarter` with a value of `0.25`. It should have the type of `float`.

9. (1 pt) Declare a variable `x` with the value `32.7`. It should have type `float`.

10. (1 pt) Declare a variable `s` with a value of `"Hi there"`. It should have the type of `string`.

11. (2 pts) Write a function `myFirstFun` that returns the result of multiplying the sum of a given integer and 3 by 4.

```
# let myFirstFun n = ... ;;
val myFirstFun : int -> int = <fun>
# myFirstFun 17;;
- : int = 80
```

12. (2 pts) Write a function `firstFun` that returns the result of multiplying a given integer by 2 and adding 5.

```
# let firstFun n = ... ;;
val firstFun : int -> int = <fun>
# firstFun 12;;
- : int = 29
```

13. (2 pts) Write a function `square` that returns the result of multiplying a given integer by itself.

```
# let square n = ... ;;
val square : int -> int = <fun>
# square 7;;
- : int = 49
```

14. (2 pts) Write a function `times_13` that returns the result of multiplying a given integer by 13.

```
# let times_13 n = ... ;;
val times_13 : int -> int = <fun>
# times_13 7;;
- : int = 91
```

15. (2 pts) Write a function `add_a` that adds the value of `a` from Problem 1 to its argument.

```
# let add_a n = ... ;;
val add_a : int -> int = <fun>
# add_a 13;;
- : int = 30
```

16. (2 pts) Write a function `circumference` that, when given a radius as a `float`, returns the circumference of a circle of that radius. You should use the value given in problem 2 for π.

```
# let circumference r = ...;;
val circumference : float -> float = <fun>
# circumference 1.0;;
- : float = 6.28318
```

(Your value may vary slightly from that printed here if you use a machine of different precision.)

17. (2 pts) Write a function `divide_e_by` that returns the result of dividing the value you gave in Problem 2 by the given `float`. You will not be tested on the value `0.0`.

```
# let divide_e_by x = ...;;
val divide_e_by : float -> float = <fun>
# divide_e_by e;;
- : float = 1.
```

(Your value may vary slightly from that printed here if you use a machine of different precision.)

18. (2 pts) Write a function `plus_quarter_times_3 y` that returns the result of multiplying by the float `3.0` the result of adding the value of `quarter` from Problem **??** to the float-valued input.

```
# let plus_quarter_times_3 y = ... ;;
val plus_quarter_times_3 : float -> float = <fun>
# plus_quarter_times_3 23.5;;
- : float = 71.25
```

(Your value may vary slightly from that printed here if you use a machine of different precision.)

19. (2 pts) Write a function `square_plus_x y` that returns the result of adding the value of `x` from Problem 1 to the square of the float-valued input.

```
# let square_plus_x y = ... ;;
val square_plus_x : float -> float = <fun>
# square_plus_x 23.17;;
- : float = 569.548900000000117
```

(Your value may vary slightly from that printed here if you use a machine of different precision.)

20. (2 pts) Write a function `double_minus_one` that takes an `int`, doubles its value, and then subtracts one.

```
# let double_minus_one  n = ...;;
val double_minus_one  : int -> int = <fun>
# double_minus_one 1;;
- : int = 1
```

21. (3 pts) Write a function `diff_square_9` that takes an integer and if the integer is between 3 and -3, squares it and substracts 9, and otherwise substracts its square from 9.

```
# let diff_square_9 m = ...;;
val diff_square_9 : int -> int = <fun>
# diff_square_9 5;;
- : int = -16
```

22. (3 pts) Write a function `salute` that takes a string, which is assumed to be a person's name, and prints out a greeting as follows: If the name is `"Elsa"`, it prints out the string

```
Halt! Who goes there!
```

followed by a "newline" at the end (that is, there should be only one "newline" produced). For any other string, it first prints out `"Hail, "`, followed by the given name, followed by `". We warmly welcome you!"`, followed by a "newline". Do not print the quotations; they were included to help make blank spaces visible. All spaces in the sample text above and below are one space long.

```
# let salutations name = ... ;;
val salutations : string -> unit = <fun>
salutations "Malisa";;
Hail, Malisa. We warmly welcome you!
- : unit = ()
```

23. (3 pts) Write a function `hail` that takes a string, which is assumed to be a person's name, and prints out a greeting as follows: If the name is `"Elsa"`, it prints out the string

```
"Wayell, hah theya, Ayelsa!"
```

(no "newline" at the end), and it does not print the quotation marks). For any other string, it first prints out `"Dear, "`, followed by the given name, followed by `". I wish you the best in CS421."`, **followed by a "newline"**.

```
# let hail name = ... ;
val hail : string -> unit = <fun>
# hail "Thomas";;
Dear, Thomas. I wish you the best in CS421.
- : unit = ()
```

24. (3 pts) Write a function `abs_diff` that takes two arguments of type `float` and returns the absolute value of the difference of the two. Pay careful attention to the type of this problem.

```
# let abs_diff x y = ... ;;
val abs_diff : float -> float -> float = <fun>
# abs_diff 15.0 11.5;;
- : float = 3.5
```

25. (3 pts) Write a function `greet` that takes a string, which is assumed to be a person's name, and prints out a greeting as follows: If the name is `"Elsa"`, it prints out the string

```
"Hey Elsa, cool man!"
```

(no "newline" at the end), and for any other string, it first prints out `"Hello, "`, followed by the given name, followed by: `". I hope you enjoy CS421."` , **followed by a "newline"**.

```
# let greet name = ... ;;
val greet : string -> unit = <fun>
# greet "Angela";;
Hello, Angela. I hope you enjoy CS421.
- : unit = ()
```

26. (3 pts) Write a function `make_bigger` that takes a `float` and if it is less than 0.0, multiplies it by (-1.0) (use the parentheses), if it is at least 0.0, but less than 1.0, adds 0.5 to it, and if it is at least 1.0, squares it.

```
# let make_bigger x = ...
val make_bigger : float -> float = <fun>
# make_bigger 12.0;;
- : float = 144.
```

27. (3 pts) Write a function `dist_double` that, when given a string and an integer, first prints the string, a comma and a space, then "`I guess it's double or nothing!`" followed by a newline, and then returns double the integer it was given.

```
# let dist_double s n = ...;;
val dist_double : string -> int -> int = <fun>
# dist_double "Well, Sam" 8;;
Well, Sam, I guess it's double or nothing!
- : int = 16
```

28. (3 pts) Write a function `has_smallest_square` that, when given one integer and then another, returns the one that has the smallest squared value. If they have the same squared value, but are not the same value, it should return the smallest value given.

```
# let has_smallest_square m n = ... ;;
val has_smallest_square : int -> int -> int = <fun>
# has_smallest_square 4 6;;
- : int = 4
```

29. (3 pts) Write a function `has_smallest_abs` that, when given one integer and then another, returns the one that has the smallest absolute value. If they have the same absolute value, it should return the second value given.

```
# let has_smallest_abs m n = ... ;;
val has_smallest_abs : int -> int -> int = <fun>
# has_smallest_abs 6 4;;
- : int = 4
```

30. (3 pts) Write a function `sign` that, when given an integer, returns `1` if the integer is positive, `0` if the integer is zero and `-1` if the integer is negative.

```
# let sign n = ... ;;
val sign : int -> int = <fun>
# sign 4;;
- : int = 1
```