MP 1 – Basic OCaml

CS 342 – Spring 2024
Revision 1.0

Assigned Tue, Jan 16, 2024
Due
Extension None past the allowed lab sign-up time

1 Change Log

1.0 Initial Release.

2 Objectives and Background

The purpose of this MP 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 in general is to provide a framework to study for exams. Several of the questions on each exam will appear similar to the MP 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 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 Common`** directive at the top of the file. If you remove it, your assignment may be graded incorrectly, costing you points.

3.1 Variable Declaration

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

2. (1 pt) Declare a variable `greetings` with the value `"Hi there."`. It should have type `string`. It should **not** contain a "newline".

3. (1 pt) Declare a variable `address` with a value of `"Greetings, my friend!"`. It should have the type of `string`. It should **not** contain a "newline".

4. (1 pt) Declare a variable `frozen` with a value of `"Do you want to build a snowman?"`. It should have the type of `string`. It should **not** contain a "newline".

5. (1 pt) Declare a variable `daffy` with a value of `"Th, th, that's all, Folks!"`. It should have the type of `string`. It should **not** contain a "newline".

6. (1 pt) Declare a variable `a` with the value `17.5`. It should have type `float`.

7. (1 pt) Declare a variable `pi` with a value of `3.14159`. It should have the type of `float`.

8. (1 pt) Declare a variable `e` with a value of `2.71828`. It should have the type of `float`.

9. (1 pt) Declare a variable `quarter` with a value of `0.25`. It should have the type of `float`.

10. (1 pt) Declare a variable `x` with the value `32.7`. It should have type `float`.

3.2 Simple Function Declaration

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 `cube` that returns the result of multiplying a given integer by itself twice (*i.e.* the cube of the input).

```
# let cube n = ... ;;
val cube : int -> int = <fun>
# cube 5;;
- : int = 125
```

3.3 Variable Use

16. (2 pts) Write a function `add_a` that adds the value of `a` from Problem 6 to its argument.

```
# let add_a n = ... ;;
val add_a : float -> float = <fun>
# add_a 13.0;;
- : float = 30.5
```

17. (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 7 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.)

18. (2 pts) Write a function `divide_e_by` that returns the result of dividing the value you gave in Problem 8 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.)

19. (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 9 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.)

20. (2 pts) Write a function `square_plus_x y` that returns the result of adding the value of `x` from Problem 10 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.)

3.4 Conditional Expressions

21. (3 pts) Write a function `salutations` 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 = ()
    ```

22. (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**). For any other string, it first prints out `"Dear, "`, followed by the given name, followed by `". I wish you the best in CS342."`, **followed by a "newline"**. Do not print the quotation marks. All spaces in the sample text above and below are one space long.

    ```
    # let hail name = ... ;
    val hail : string -> unit = <fun>
    # hail "Thomas";;
    Dear, Thomas. I wish you the best in CS342.
    - : unit = ()
    ```

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

    ```
    Can you come out to play?
    ```

    (**with a "newline" at the end**). For any other string, it first prints out `"Aw, come on, "`, followed by the given name, followed by `". We're going to have a wonderful time!"`, **also followed by a "newline"**. Do not print the quotation marks. All spaces in the sample text above and below are one space long.

    ```
    # let welcome name = ... ;;
    val welcome : string -> unit = <fun>
    # welcome "John";;
    Aw, come on, John. We're going to have a wonderful time!
    - : unit = ()
    ```

24. (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 CS342."` , **followed by a "newline"**. Do not print the quotation marks. All spaces in the sample text above and below are one space long.

```
# let greet name = ... ;;
val greet : string -> unit = <fun>
# greet "Angela";;
Hello, Angela. I hope you enjoy CS342.
- : unit = ()
```

25. (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

    `What's the low-down, man?`

    (**no "newline" at the end**). For any other string, it first prints out `"Hey, "`, followed by the given name, followed by `"! Give me five, man."`, **not followed by a "newline"**. Do not print the quotation marks. All spaces in the sample text above and below are one space long.

```
# let salute name = ... ;
val salute : string -> unit = <fun>
# salute "Ali";;
Hey, Ali! Give me five, man.- : unit = ()
```

26. (3 pts) Write a function `rectangle_area` that takes two arguments of type `float` representing the length and width of a rectangle, and returns the area of that rectangle, if both inputs are greater than or equal to zero. If either input is strictly less than zero, return `(-1.0)` instead. Pay careful attention to the type of this problem.

```
# let rectangle_area l w = ... ;;
val rectangle_area : float -> float -> float = <fun>
# rectangle_area 25.3 19.2;;
- : float = 485.76
```

27. (3 pts) Write a function `diff_square_9` that takes two floats, one and then another, and if the first float is strictly smaller than the second, then it returns the result of subtracting nine from the square of the second. If the first float is not strictly smaller than the second, and half the first is strictly larger than the second, return result of subtracting nine from the square of the first. If the first float is not strictly smaller than the second, and half the first is not strictly larger than the second, then return result of subtracting nine from the square of the difference of the first and the second.

```
# let diff_square_9 m n = ...;;
val diff_square_9 : float -> float -> float = <fun>
# diff_square_9 5.5 (-17.2);;
- : float = 21.25
```

28. (3 pts) Write a function `make_bigger` that takes a `float` and then another `float` and if the first input is strictly larger than 0.0, it adds the first to the second. If the first input is not strictly larger than 0.0 and the second is strictly less than 1.0, it returns the result of adding 1.0 to the second, and otherwise it returns the square of the second.

```
# let make_bigger x y= ...
val make_bigger : float -> float -> float = <fun>
# make_bigger (-5.2) 12.0;;
- : float = 144.
```

29. (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
```

30. (3 pts) Write a function `sign_times` that, when given a first and second integer, returns `1` if the product of the integers is positive, `0` if the product of the integers is zero and `-1` if the product of the integers is negative.

```
# let sign_times n m = ... ;;
val sign_times : int -> int -> int = <fun>
# sign_times 4 3;;
- : int = 1
```