# Assignment 3:  OCaml Solitaire[[1]](#footnote-1)

This problem involves a solitaire card game invented just for this question. You will write a program that tracks the progress of a game; writing a game player is a challenge problem. You can do parts (a)–(e) before understanding the game if you wish.

A game is played with a card-list and a goal. The player has a list of held-cards, initially empty. The player makes a move by either drawing, which means removing the first card in the card-list from the card-list and adding it to the held-cards, or discarding, which means choosing one of the held-cards to remove. The game ends either when the player chooses to make no more moves or when the sum of the values of the held-cards is greater than the goal.

The objective is to end the game with a low score (0 is best). Scoring works as follows: Let sum be the sum of the values of the held-cards. If sum is greater than goal, the preliminary score is five times (sum − goal), else the preliminary score is (goal − sum). The score is the preliminary score unless all the held-cards are the same color, in which case the score is the preliminary score divided by 2 (and rounded down as usual with integer division; use ML’s div operator).

1. Write a function card\_color, which takes a card and returns its color (spades and clubs are black, diamonds and hearts are red). Note: One case-expression is enough.
2. Write a function card\_value, which takes a card and returns its value (numbered cards have their number as the value, aces are 11, everything else is 10). Note: One case-expression is enough.
3. Write a function remove\_card, which takes a list of cards cs, a card c, and an exception e. It returns a list that has all the elements of cs except c. If c is in the list more than once, remove only the first one. If c is not in the list, raise the exception e. You can compare cards with =.
4. Write a function all\_same\_color, which takes a list of cards and returns true if all the cards in the list are the same color. Hint: An elegant solution uses nested pattern-matching.
5. Write a function sum\_cards, which takes a list of cards and returns the sum of their values.
	* For bonus points, write a function sum\_cards\_tr, which takes a list of cards and returns the sume of their values; however, it should make use of a locally defined helper function that is tail recursive.
6. Write a function score, which takes a card list (the held-cards) and an int (the goal) and computes the score as described above.
7. Write a function officiate, which “runs a game.” It takes a card list (the card-list) a move list (what the player “does” at each point), and an int (the goal) and returns the score at the end of the game after processing (some or all of) the moves in the move list in order. Use a locally defined recursive helper function that takes several arguments that together represent the current state of the game. As described above:
	* The game starts with the held-cards being the empty list.
	* The game ends if there are no more moves. (The player chose to stop since the move list is empty.)
	* If the player discards some card c, play continues (i.e., make a recursive call) with the held-cards not having c and the card-list unchanged. If c is not in the held-cards, raise the IllegalMove exception.
	* If the player draws and the card-list is (already) empty, the game is over. Else if drawing causes the sum of the held-cards to exceed the goal, the game is over (after drawing). Else play continues with a larger held-cards and a smaller card-list.

Sample solution for (g) is under 25 lines.

## Suggestions

* A source file is provided for you to begin with. It defines a variety of datatypes that your response should utilize. The solution to problem A is also provided.
* I suggest that you download and install OCaml on your own computer so that you can more rapidly test as you go. Instructions for installing OCaml on a variety of platforms are available at: <http://ocaml.org/docs/install.html>. You can then use the OCaml REPL (Read-Eval-Print Loop) to experiment with OCaml and run your program as you are developing.
* I recommend using an editor such as Notepad++ or Sublime Edit to write your source code in and then running the REPL from the command-line
* A good OCaml reference is available at <https://realworldocaml.org/>.
* When you write your code in a file, you can load that file into the REPL using #use. For example to load the functions in assignment3.ml, you would type

“#use “assignment3.ml”;;

into the REPL.

**Division of Labor**

As in most assignments this semester, you may work alone or in teams of two.  Note that only one turn-in is required per team, but each student must submit the trivia and the README separately.

## Logistics & Submission

This assignment is Due on Thursday, October 6th by 5:00 PM. You should submit all of your source files and your README file (see below) via Blackboard.

## README files

Your README file should contain whatever information you think is necessary for a typical user to run your program and for a software maintainer (or grader!) to understand (and evaluate) your code.  Explain both the negative aspects of your program (limitations, known bugs) and the positive aspects (extensions, special features).  If you don’t include the former, we will assume you didn’t realize the limitations were there.  If you don’t include the latter, you may not get extra credit.  For the person who has to understand the insides of your code, you may need to describe your choice of modularization (abstractions), data structures, and algorithms.  Be sure to explain anything you did that is likely to be different from what other students may have done, and justify any design decisions for which the rationale isn’t immediately clear.  Your write-up should be clearly written, using full sentences and paragraphs.  This write-up need not necessarily be long, but it is important.  All write-ups must be in plain text (README.txt) or Word (README.doc) format.  Other formats will *not* be accepted.

**Trivia Assignment**

Before **5pm on Tuesday, September 27th**, submit a Word document containing answers to the following questions:

1. Are you working alone or in a team?  If a team, who is your partner?
2. Enter the following OCaml code into the OCaml REPL and determine what the output is. Experiment with a few different calls to mystery using different values then describe in general terms what the function mystery is calculating.

**let rec** mystery x = if x <= 1 then 1 else mystery (x - 1) + mystery (x - 2);;

**mystery(5);;**

1. Write the OCaml code to create a variable called hand that contains a list of 5 cards. The cards can be of any value that you would like.
2. Write the OCaml code to call the function card\_color with the first item from the hand list, as defined in the previous problem.

**Grading Scale**

For each programming project, points will be divided roughly 80/20 between the code itself and the external documentation (README file).  A more detailed breakdown follows:

* Code
	+ ~60% correctness, completeness, and efficiency:
	Your code should implement everything that was required in the assignment.  It should produce the right output given normal, expected inputs, and some sort of reasonable response to unexpected inputs.  Unless otherwise instructed, and as long as it does not severely compromise programming style, you are expected to use the most efficient data structures and algorithms.
	+ ~20% programming style (including internal documentation and program organization):
	Your code should have appropriate abstractions, data structures, algorithms, and variable names; declarations for all constants; and a judicious number of helpful comments.  To paraphrase Don Knuth, instead of thinking of programming as explaining to the computer what you want it to do, think of programming as explaining to the TA what you want the computer to do.
* README file
	+ ~15% completeness:
	Your write-up should include all the items discussed in the README portion of the submission instructions above.
	+ ~5% readability:
	Your write-up should be well organized, clear, and concisely presented.
1. This assignment is slightly modified from an [↑](#footnote-ref-1)