

GIST A0

BY ANDREW SIKOWITZ

PURPOSE OF GIST

- Briefly go over the assignment
- Mention important OCaml tips, tricks, and syntax for the assignment
- Go over the hardest parts of the assignment in more detail
 - Example problems
 - Strategies
- This one is a bit longer than usual, due to the lack of an in-person session and the starting and final tips sections

STARTING TIPS: THE MAKEFILE

- The release ships with a Makefile!
- This makefile defines commands that can be run in the terminal by typing ``make`` or ``make <cmd>``
 - ``make``: Start utop and execute “warmup.ml”
 - ``make test``: Compile and execute “warmup.ml”
 - ``make check``: Check your OCaml dev environment is set up correctly
 - ``make finalcheck``: ``make check`` with additional checks (see writeup)
 - ``make docs``: Generate documentation files in the “doc” folder
 - ``make clean``: Clean up build and doc files

STARTING TIPS: CODING STYLE

- You will be graded on coding style
- Grading based on four categories:
 - Documentation: Are your top-level functions documented? Well?
 - Testing: Have you tested your functions? Well?
 - Comprehensibility: Is your code well-organized and easy to read?
 - Formatting: Is your code well spaced? Are lines under 80 characters?
- **Don't use imperative features!**
- See the “coding standards” page for more details
 - https://www.cs.cornell.edu/courses/cs3110/2018fa/coding_standards.html

STARTING TIPS: CODING STYLE

- You should read the style guide, especially for formatting:
 - multi-line functions
 - single-line and many-line if expressions
 - match statements
- There are different, correct ways to write the same code
 - Still, stay consistent. Especially with your spacing.
- <https://ocaml.org/learn/tutorials/guidelines.html>
- <http://www.cs.cornell.edu/courses/cs3110/2017fa/handouts/style.html>
 - Shorter, but less comprehensive and no longer the official style guide

OVERVIEW FOR A0

- Introductory assignment
- 3 Functions:
 - Valid Date: use those boolean operators and conditionals!
 - Syracuse: recursion!
 - Nacci: recursion, with lists and pattern matching!
- Fill in implementations in the file “warmup.ml”
- They should get progressively harder (not necessarily more code)

IF EXPRESSIONS “RETURNING” BOOLS

- Replace them with `&&` and `||`
- `if b then true else false => b`
- `if b then false else true => not b`
- `if b1 then true else if b2 then true else false => b1 || b2`
- `if b1 then false else if b2 then true else false => not b1 && b2`
- `if b1 then true else if b2 then false else true => b1 || not b2`
- `if b1 then true else if b2 then false else if b3 then true else false => b1 || (not b2 && b3)`

::VS @

- ::
 - “cons”
 - Add an element onto the head of a list
 - Very fast; $O(1)$
- @
 - “append”
 - Combine two lists
 - Can be slow; for $l1 @ l2$, $O(\text{List.length } l1)$

HELPER FUNCTIONS (PART I)

- Abstract out functionality into helper functions!

```
let sum lst =  
  let rec sum_acc acc lst =  
    match lst with  
    | [] -> acc  
    | h::t -> sum_acc (h+acc) t in  
  sum_acc 0 lst
```

HELPER FUNCTIONS (PART 2)

- Sometimes one recursive function can't do everything you want
 - A single for or while loop can't always accomplish what you want either
- Make a helper function!
- Ex: Write a function `[map_sum]` that sums each list in a list of lists
 - `map_sum[[1;2;3]; [4;5;6]; [7]; []; [8;9]] = [6; 15; 7; 0; 17]`

HELPER FUNCTIONS (PART 2)

```
let rec map_sum lst =  
  let rec sum = function  
    | [] -> 0  
    | h::t -> h + sum t in  
  match lst with  
  | [] -> []  
  | h::t -> (sum h) :: (map_sum t)
```

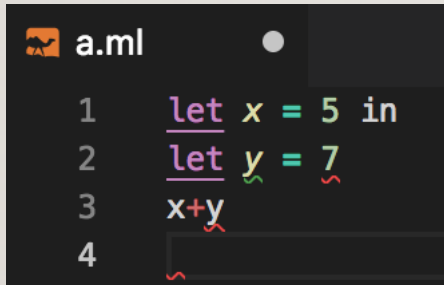

FINAL TIPS: GRADING SCOPES

- Make sure to pass make check!
 - We can't grade your assignment if you don't...
- Please read the grading scopes section
 - It tells you what you have to do to get what grade
 - This may not be immediately obvious
- We will put emphasis on the core of the assignments when grading
 - Make sure you have a rock solid implementation for earlier scopes, before you rush onto the harder parts

FINAL TIPS: EDITOR / COMPILER ERRORS

- Small errors can cause large problems
- Check over the area of code where you're getting errors

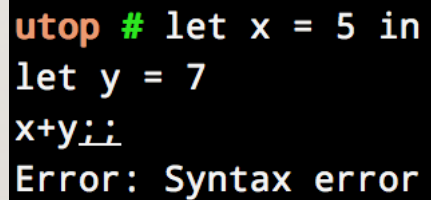
VS Code

A screenshot of the Visual Studio Code editor interface. The file name 'a.m1' is shown in the top left. The code is as follows:

```
1  let x = 5 in
2  let y = 7
3  x+y
4
```

Red squiggly lines indicate syntax errors: one under the closing brace of the first 'let' expression, one under the second 'let' expression, and one under the 'x+y' expression.

utop

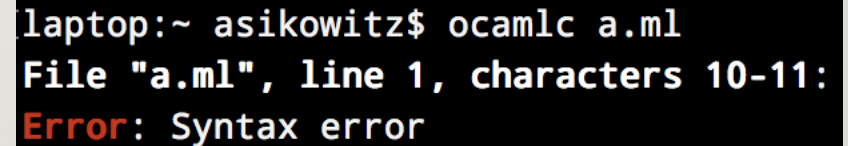
A screenshot of the utop REPL. The code entered is:

```
utop # let x = 5 in
let y = 7
x+y;;
```

The output shows:

```
Error: Syntax error
```

ocamlc

A screenshot of a terminal window. The command executed is 'ocamlc a.m1'. The output is:

```
laptop:~ asikowitz$ ocamlc a.m1
File "a.m1", line 1, characters 10-11:
Error: Syntax error
```

FINAL TIPS: TYPE ERRORS

- A lot of mistakes in OCaml result in type errors

```
utop # not 5;;  
Error: This expression has type int but an expression was expected of type  
      bool
```

```
utop # let f a b = a b;;  
val f : ('a -> 'b) -> 'a -> 'b = <fun>  
-( 17:37:11 )-< command 1 >  
utop # f 1 2;;  
Error: This expression has type int but an expression was expected of type  
      'a -> 'b
```


FINAL TIPS: TYPE ERRORS

- Try to be considerate of:
 - What functions you are using
 - What arguments you are passing into those functions
 - The types of those functions and arguments

```
utop # let f a b =  
      let sum = a + b in  
      not sum;;  
Error: This expression has type int but an expression was expected of type bool
```

- OCaml tries to infer the types of variables
 - It assumes [sum] is an integer, as it is the result of the (+) function
 - It then gets confused when [sum] is treated as a bool
 - It is passed into the [not] function, which expects a bool

FINAL TIPS: TESTING

- As per the writeup, you can write tests using assertions:
 - `let () = assert (actual = expected)`
- Ex:
 - `let () = assert (sum [1; 2; 3; 4; 5] = 15)`
 - `let () = assert (sum [] = 0)`
 - `let () = assert (sum [-5; 5] = 0)`
- Make sure to test edge cases!
- Put these at the bottom of “warmup.ml”
 - In the future, tests will go in other files