USING FOLD TO CHECK PREDICATES (75/75 points)

1. Using `List.fold_left`, write a function `for_all : ('a -> bool) -> 'a list -> bool`. It takes as argument a list `l` of type `'a list`, and a predicate `p` of type `'a -> bool`. It must return `true` if and only if all elements of `l` satisfy the predicate `p`.
2. Using `List.fold_left`, write a function `exists : ('a -> bool) -> 'a list -> bool`. It takes as argument a list `l` of type `'a list`, and a predicate `p` of type `'a -> bool`. It must returns `true` if at least one element of `l` satisfies the predicate `p`.
3. Write a function `sorted : ('a -> 'a -> int) -> 'a list -> bool`, using `List.fold_left` that checks that a list of elements `l` of type `'a` is sorted, according to an ordering function `cmp` of type `'a -> 'a -> int`.
   The ordering function returns:
   - `1` (or any positive number) if the first element is greater than the second,
   - `-1` (or any negative number) if the first element is lesser than the second,
   - and `0` otherwise.

   For the `fold_left` part, you can use the type `'a option` as the accumulator: at each iteration of `fold_left`, if the list if sorted until now, the acccumulator is either `Some v`, where `v` is the previous element, or `None` otherwise.
   Remember, the empty list is sorted, so you can use the list with at least one element to check using `fold_left`.

YOUR OCAML ENVIRONMENT

```
let for_all p l =
  let liste = List.fold_left
      (fun liste element -> if p element then element::liste else liste)
      []
      l
  in
  if List.length liste == List.length l then true else false
;;

let exists p l =
  let liste = List.fold_left
      (fun liste element -> if p element then element::liste else liste)
      []
      l
  in
  if List.length liste >= 1 then true else false
;;

let sorted cmp l = match l with
  | [] -> true
  | _::lr ->
      let resultat = List.fold_left
          (fun (compteur, pos) element ->
             if (cmp (List.nth l pos) element <= 0) then (compteur + 1, pos + 1) else
               (compteur, pos + 1))
          (0,0)
          lr
      in
      match resultat with
      | (corrects,_) -> if corrects + 1 = List.length l then true else false
;;
```

Evaluate >

Switch >>

Typecheck

Reset Templ

Full-screen

Check & Sa

Exercise complete (click for details) 75 pts

v Exercise 1: `for_all` Completed, 25 pts

Found a toplevel definition for `for_all`.

Found List.fold_left 5 pts

Found `for_all` with compatible type.

Computing `for_all (fun x -> x mod 2 = 0) [-2; -3; 3; 4; 4; -5; 4; 3; -1; -2]`

Correct value `false` 1 pt

Computing `for_all ((=) 0) []`

Correct value `true` 1 pt

Computing `for_all ((<>) 3) [2; 3; 2; -1; 3; 2; 0; -2]`

Computing `for_all ((=) 0) [0]`
Correct value `true` 1 pt
Computing `for_all ((<>) 3) [0; -3; 3]`
Correct value `false` 1 pt
Computing `for_all ((<>) 3) [4; -1; 1; -4; -2]`
Correct value `true` 1 pt
Computing `for_all ((=) 0) [3; -1; -2; 2]`
Correct value `false` 1 pt
Computing `for_all (fun x -> x mod 2 = 0) [-1; -1; -5]`
Correct value `false` 1 pt
Computing `for_all (fun x -> x mod 2 = 0) [4; -4; -3; -4; 0; 0]`
Correct value `false` 1 pt
Found `for_all` with compatible type.
Computing
```
for_all
  (fun f -> sin f > 0.)
  [1.92260326480036792; 3.63607135234171608; -1.63879270570612157;
   1.47483390388598057; 3.83461440676621557; -2.08814437497695193]
```
Correct value `false` 1 pt
Computing `for_all ((>) (-.1.)) [-4.04596580905903291]`
Correct value `true` 1 pt
Computing
```
for_all
  (fun f -> (f /. 2.) < 2.)
  [-3.45183049617848958; 0.0815259815959024081; -1.93610269727139706;
   0.178016387435438794; 0.500698743843502214; 1.64241771017645632;
   -2.0589482037717084; 4.66510749409304637; -0.892860843638830559]
```
Correct value `false` 1 pt
Computing
```
for_all
  (fun f -> (f /. 2.) < 2.)
  [-2.6024999333670169; 1.08988410128031354; 4.47681626751950645;
   2.48301053375277725]
```
Correct value `false` 1 pt
Computing
```
for_all
  ((>) (-.1.))
  [3.59626873898866961; 2.89059014902365075; 3.21566787855563163;
   -2.21220133472654723; -2.57669778646516434; 0.278488857627206;
   1.30225918365442705]
```
Correct value `false` 1 pt
Computing
```
for_all
  (fun f -> sin f > 0.)
  [1.63089726692060122; 2.42346207885234932; 3.56205702210392161]
```
Correct value `false` 1 pt
Computing
```
for_all
  (fun f -> sin f > 0.)
  [-4.01855838705035495; 2.90341127157056889; -3.56817898289588475;
   -3.91470150037306386]
```
Correct value `true` 1 pt
Computing `for_all (fun f -> (f /. 2.) < 2.) []`
Correct value `true` 1 pt
Computing
```
for_all
  ((>) (-.1.))
  [3.37631002063020347; -2.34866456432590809; -3.40363502639416637;
   -2.3349568265867866; -3.97919987107452577]
```
Correct value `false` 1 pt
Computing `for_all (fun f -> (f /. 2.) < 2.) []`
Correct value `true` 1 pt

v Exercise 2: `exists` Completed, 25 pts

Found a toplevel definition for `exists` .
Found List.fold_left 5 pts
Found `exists` with compatible type.
Computing `exists ((<>) 3) [-5; -4; 0; 3; 2]`
Correct value `true` 1 pt
Computing `exists ((=) 0) [-3; -5; 0; 1; 2; -2; -1]`
Correct value `true` 1 pt
Computing `exists ((<>) 3) [-2; 1; -4; 4; -3; 3; 1]`
Correct value `true` 1 pt

Correct value `true` 1 pt
Computing `exists ((=) 0) [4; 1; -1; 1; -4; 0; 4; 3; -3; -5]`
Correct value `true` 1 pt
Computing `exists (fun x -> x mod 2 = 0) [4; -4]`
Correct value `true` 1 pt
Computing `exists ((<>) 3) [4; 2; -4; -3; -4; -3]`
Correct value `true` 1 pt
Computing `exists ((=) 0) [-5; -3; 4; 4; -5; 4; 3; -3; 0]`
Correct value `true` 1 pt
Computing `exists (fun x -> x mod 2 = 0) [-1; 3; 0]`
Correct value `true` 1 pt
Found `exists` with compatible type.
Computing `exists (fun f -> sin f > 0.) [2.97196737635266217; 1.69305144470402791]`
Correct value `true` 1 pt
Computing `exists ((>) (-.1.)) []`
Correct value `false` 1 pt
Computing
```
exists
  ((>) (-.1.))
  [3.4037884492175543; 2.24180762995262661; 2.24262290789090191;
   -0.246747981429872532; -4.78105432794478613]
```
Correct value `true` 1 pt
Computing
```
exists
  (fun f -> sin f > 0.)
  [0.105920269024982971; -0.294656505416947212; -1.71955884015427518;
   4.47203417023808569]
```
Correct value `true` 1 pt
Computing
```
exists
  (fun f -> (f /. 2.) < 2.)
  [0.747721460737552412; 0.769093539680217653; 3.61909185000399347;
   -1.09570090746299931; -0.947625573218246586; -3.42515912316868576]
```
Correct value `true` 1 pt
Computing
```
exists
  (fun f -> sin f > 0.)
  [-1.95026083935689787; 0.186581725384085573; 0.0102876455197975503;
   -1.39138717651869648; -3.02560528227742598; -3.74759154762064517;
   1.16508542867759779; 4.67975327323112644; 1.91956869223674076;
   -2.78018978930318283]
```
Correct value `true` 1 pt
Computing
```
exists
  ((>) (-.1.))
  [-3.8538931657516633; -3.70500277708605319; 1.32809294196757044]
```
Correct value `true` 1 pt
Computing
```
exists
  (fun f -> (f /. 2.) < 2.)
  [-3.93228326786018245; -0.820925558157917123; 1.8443180150071905]
```
Correct value `true` 1 pt
Computing
```
exists
  ((>) (-.1.))
  [-4.40031450793599443; 1.17367971476364463; 2.55092705322569469;
   -0.46861509346945418; 2.43916734455530726; -4.83396032734593106;
   -4.19044299629083739; -1.99759379590968011]
```
Correct value `true` 1 pt
Computing
```
exists
  (fun f -> (f /. 2.) < 2.)
  [-4.0004756391345051; -0.757270363655809575; 1.07800422864800272]
```
Correct value `true` 1 pt

v Exercise 3: `sorted` Completed, 25 pts

Found a toplevel definition for `sorted` .
Found function List.fold_left 5 pts
Found `sorted` with compatible type.
Computing `sorted (fun x y -> compare y x) [4; 4; 0; -1; -1; -3; -4]`
Correct value `true` 1 pt
Computing
```
sorted
  (fun x y -> let x = abs x and y = abs y in compare x y)
  [0; -1; -1; -1; 1; -1; -3]
```

Computing `sorted (fun x y -> let x = abs x and y = abs y in compare x y) [1; -4; 4]`
Correct value `true` 1 pt
Computing `sorted (fun x y -> let x = abs x and y = abs y in compare x y) []`
Correct value `true` 1 pt
Computing `sorted compare []`
Correct value `true` 1 pt
Computing `sorted (fun x y -> compare y x) [4; 3; 0; -1; -1; -2; -2; -4; -5; -5]`
Correct value `true` 1 pt
Computing `sorted (fun x y -> compare y x) [-5]`
Correct value `true` 1 pt
Computing `sorted (fun x y -> compare y x) [4; 3; 1; -4]`
Correct value `true` 1 pt
Computing `sorted compare [-2; -4; -5; -4; -2; -4]`
Correct value `false` 1 pt
Found `sorted` with compatible type.
Computing `sorted (fun x y -> compare y x) [-4.96042565801813]`
Correct value `true` 1 pt
Computing
```
  sorted
    (fun x y -> compare y x)
    [3.85710253994050767; 2.76742900903412625; 0.0274380943898382412]
```
Correct value `true` 1 pt
Computing
```
  sorted
    compare
    [-1.17055579380313191; -4.68048734909683528; 2.54713047172048324;
     3.30954634285957283; 0.399083177482666; 4.83134570534725505;
     4.17356177468029266; -0.709474461033099]
```
Correct value `false` 1 pt
Computing
```
  sorted
    (fun x y -> compare y x)
    [3.03066435187370331; 2.58670480438912875; -0.364238098905207863;
     -0.633693952647358394; -2.89071008654286343]
```
Correct value `true` 1 pt
Computing `sorted (fun x y -> compare y x) [4.96277166986232565]`
Correct value `true` 1 pt
Computing
```
  sorted
    (fun x y -> compare y x)
    [4.89458807665028; 4.49510000684069411; 2.19349259271139818;
     -1.56648517438258938; -3.52255139166816633; -3.95680642193261534]
```
Correct value `true` 1 pt
Computing
```
  sorted
    (fun x y -> compare y x)
    [4.73630843253803491; 3.24982548824338124; 0.444326170303456;
     -0.611400466142406174]
```
Correct value `true` 1 pt
Computing
```
  sorted
    (fun x y -> compare y x)
    [-1.89339148598357365; 2.12840601421895137; -2.04017934851030525;
     -1.94689541971573377; 0.564353105856728376; 0.411527967854320664;
     1.75195680712416468; -0.0178802097651633574; -2.95233283274888558]
```
Correct value `false` 1 pt
Computing
```
  sorted
    (fun x y -> let x = abs x and y = abs y in compare x y)
    [-2.07131651796599758; 0.353383298888339858; 1.75827186782273159;
     0.138311546476317382; -2.89736003719107771; 2.54228140070838649;
     -0.784375715049862698; 3.35020092768150768; -1.61652063553599845;
     0.0788537327720364445]
```
Correct value `false` 1 pt
Computing
```
  sorted
    (fun x y -> let x = abs x and y = abs y in compare x y)
    [3.19141621057961444; 2.66699290102758724; -2.04619078976213;
     -4.03241903831085846]
```
Correct value `false` 1 pt