Řešení domácích úkolů - maxis

```
type 'a tree = Node of 'a * 'a tree list

let maxis tree =
  let rec maxis' acc (Node (a, ss)) =
    let update (acc, metoo) son = let acc', top' = maxis' acc son
                                  acc', metoo && not top'
    let acc, metoo = ss |> List.fold_left update (acc, true)
    (if metoo then a :: acc else acc), metoo
  maxis' [] tree |> fst

let maxis_seq tree =
  let rec maxis' = function
    | Node (a, []) -> seq { yield a }, true
    | Node (a, ss) -> let maxs, flags = ss |> List.map maxis' |> List.unzip
                      let metoo = flags |> List.for_all ((<>) true)
                      seq { if metoo then yield a
                            yield! Seq.concat maxs }, metoo
  maxis' tree |> fst |> List.of_seq
```

Řešení domácích úkolů - rost

```
let rost = function
  | [] -> []
  | xs -> let n = List.length xs
          let a, p = Array.zero_create n, Array.create n []
          let add i x = a.[i] <- x; p.[i] <- x :: if i>0 then p.[i-1] else []
          let find_and_add len x =
            let rec bsearch l r =
              if l = r then l
              else let m = (l + r) / 2
                   if a.[m] < x then bsearch (m+1) r else bsearch l m
            let i = if x > a.[len] then len+1 else bsearch 0 len
            add i x
            max i len

          add 0 xs.Head
          let longest = xs.Tail |> List.fold_left find_and_add 0
          p.[longest] |> List.rev
```

Řešení domácích úkolů - ssrt

```
let ssrt : string [] -> string list = function
  | ss when ss.Length = 0 -> []
  | ss ->
   let slen (s : string) = s.Length
   let radix_by by (buck : 'a list []) bucklist what =
     bucklist |> List.iter (fun i -> buck.[i] <- [])
     what |> Seq.iter (fun w -> let i = by w in buck.[i] <- w::buck.[i])
     (bucklist |> List.fold_right (fun i acc -> List.rev buck.[i] @ acc)) []
   let n, l = Array.length ss, ss |> Seq.map slen |> Seq.max
   let charbucks, chars = Array.create 256 [], Array.create (l+1) []

   let wds = ss |> radix_by (fun s -> l-slen s) (Array.zero_create (l+1)) [0..l]
   let letters = wds |> Seq.map_concat (Seq.mapi (fun i c -> i, int c))
   letters |> radix_by (fun s -> 255-snd s) (Array.zero_create 256) [0..255]
           |> List.iter (fun (i,c) -> if chars.[i].Length=0 || chars.[i].Head<>c
                                      then chars.[i] <- c::chars.[i])

   let missing, sorted = ref wds, ref []
   for i = l downto 1 do
     while (!missing).Length>0 && (!missing).Head.Length = i do
       sorted := (!missing).Head :: !sorted
       missing := (!missing).Tail
     sorted := radix_by (fun s->int s.[i-1]) charbucks chars.[i-1] !sorted
   !missing @ !sorted
```

Řešení domácích úkolů - rsam

```
let rsamodules num =
  let primes, mods = ResizeArray.of_list [], ResizeArray.of_list []
  let rec step len =
    let s = Array.zero_create len
    let delprime p = for j in {p-(len-1)%p-1..p..len-1} do s.[j] <- s.[j]+1
    let addnew i si = if si = 0 then primes.Add(len + i)
                      if si = 2 && mods.Count < num then mods.Add(len + i)
    primes |> ResizeArray.iter delprime
    s |> Seq.iteri addnew
    if mods.Count < num then step (len + len)
  step 2
  mods |> ResizeArray.to_list
```

F# - priority operátorů

```
as                            %right
when                          %right
|                             %left
;                             %right
let                           %nonassoc
function, fun, match, try     %nonassoc
if                            %nonassoc
->                            %right
:=                            %right
,                             %nonassoc
or ||                         %left
& &&                          %left
<OP >OP $OP = |OP &OP         %left
^OP                           %right
::                            %right
:?> :?                        %nonassoc
-OP +OP                       %left
*OP /OP %OP                   %left
**OP                          %right
"f x" "lazy x" "assert x"     %left
"| rule"                      %right  -- pattern match rules
!OP ?OP ~OP -OP +OP           %left
.                             %left
f(x)                          %left  - high precedence application
f<types>                      %left  - type application
```

Počáteční . a $ se ignorují, takže `.*` a `$*` mají stejnou prioritu jako `*`.

Výjimky

```
exception Fail of string

let rec mem x = function
  | y::xs -> if x=y then true else mem x xs
  | _     -> raise (Fail "oh no")

try mem 4 [5] with
  | Fail str -> printfn "%s" str; rethrow ()
  | :? FailException as e -> printfn "%s" (e.ToString()); rethrow()

try mem 4 [5] finally printfn "Ended"
```

Vyvolání základních výjimek

assert, failwith, invalid_arg, invalid_op, not_found

Užitečné funkce

```
set : 'a seq -> 'a Set
  funkcionální množiny, Set.{add,count,diff,mem,next_elt,subset,union,...}
  Set<'a>.{Add,Remove,Contains,Count,Is{Sub,Super}SetOf,IsEmpty,(+),(-)}
dict : 'a * 'b seq -> IDictionary<'a, 'b>
  má .[] : 'a -> 'b, enumeruje 'a * 'b
  má ContainsKey, TryGetValue, Keys, Values
```

```
                      Vstupy a výstupy
                      ----------------
any_to_string : 'a -> string, print_any, prerr_any : 'a -> unit
printf, printfn, eprintf, eprintfn, sprintf : format -> params -> unit
fprintf, fprintfn : TextWriter -> format -> params -> unit
"%b %s %[id] %u %[xXo] %[efg] %M %O(ToString) %A(any_to_string)"

System.IO.TextWriter má Write, WriteLine
"{0}", "{0,10}", "{0,-10:00.00}"
System.IO.TextReader má Peek(), Read() : integer, ReadLine(), ReadToEnd()

System.Console.{In,Out,Error}, Write, WriteLine, Read, ReadLine
System.IO.StreamReader(file,enc,usebom,bufsz), StreamWriter(file,app,enc,bufsz)

System.IO.File.{CreateText,AppendText,OpenText,ReadAll{Bytes,Lines,Text}(fn,enc)
}
System.IO.File.{WriteAll{Bytes,Lines,Text}(fn,what,enc), AppendAllText}

use ident = expr1 in expr2
let ident = expr1 in
  try expr2
  finally (match (ident:>obj) with | null -> () |
                                   | _ -> (ident:>System.IDisposable).Dispose())
```