```
                    Objective Categorical Abstract Machine Language
                    -----------------------------------------------
(* Komentář (* Vnořený komentář *) *)

(* Typy *)
♣ type unit = ()

♣ type int = ... | -2 | -1 | 0 | 1 | 2 | ...        Má o bit méně, tj. 31 nebo 63.
Konstanty jsou 17 = 0x11 = 0o21 = 0b10001           Existuje int32, int64, nativeint
   i + j,  i - j,  i * j,  i / j,  i mod j
   -i (~-i)      negace
   lnot i        bitová inverze
   i lsl j       logical shift left
   i lsr j       logical shift right
   i asr j       arithmetic shift right
   i land j      bitový AND
   i lor j       bitový OR
   i lxor j      bitový XOR

♣ type float = ... (**
Konstanty jsou 1., 0.2, 3141.5926E–3
   x +. y,  x -. y,  x *. y,  x /. y
   -.x (~-.x)                      negace
   x ** y                          mocnění
   infinity, neg_infinity, nan, epsilon_float     speciální hodnoty
   int_of_float x                  konverze float na int
   float_of_int i                  konverze int na float

♣ type char = ... | 'a' | 'b' | ...      Pouze 8–bitové znaky
Konstanty jsou 'a', 'b', '\\', '\n', '\'', '\x41', '\061'
   Char.code 'x'         hodnota znaku jako int
   Char.chr 33           char z danou intovou hodnotou
   Char.uppercase 'y
   Char.lowercase 'z'*)

♣ type string = ...      Není to seznam charů, je to primitivní typ
Konstanty jsou "ahoj", "\""
   s ^ t                 konkatenace
   s.[0]                 první znak
   string_of_int,      int_of_string
   string_of_bool,    bool_of_string
   string_of_float,  float_of_string
   String.length s
   String.copy s
   String.lowercase
   String.uppercase
   ...

♣ type bool = false | true
   x < y, x <= y, x > y, x >= y
   x = y      strukturální porovnání
   x <> y     negace předchozího
   x == y     porovnání identity, "physical equality"
   x != y     negace předchozího
   not b      negace
   min i j
   max i j
   b && c     AND se zkráceným vyhodnocováním
   b || c     OR se zkráceným vyhodnocováním

|| &&                          doleva         Priority a asociativity operátorů.
= == != <> < <= > >=           doleva         Pořadí je dle rostoucí priority.
+ - +. -.                      doleva
* / *. /. mod land lor lxor    doleva
lsl lsr asr **                 doprava
lnot                           doleva
~- - ~-. -.                    doprava
aplikace funkce, konstruktoru  doleva
. .[ .(                        žádná
```

Variable alias proměnné

```
let x = 1;;
let twopi = let pi = 4. *. atan 1.0 in 2. *. pi;;
```

Funkci můžeme vytvořit jako `fun i -> i + 1`

```
let inc = fun i -> i + 1;;
```

Pro zjednodušení je `let id v1 ... vn = e` zkratka za `let id = fun v1 ... vn -> e`

Můžeme stanovovat explicitní typy

```
let cmp (i:int) (f:float) : bool = f == float_of_int i;;
```

Podmínky pomocí `if then else`

```
let is_even n = if n mod 2 = 0 then true else false;;
```

Rekurze

`let factorial n = if n = 1 then 1 else n * factorial (n-1)` Nefunguje!

```
let rec factorial n = if n = 1 then 1 else n * factorial (n-1)

let factorial n =
    let rec fac acc n = if n = 1 then acc else fac (acc*n) (n-1)
    in fac 1 n
```

Funkce jako argumenty

```
let derivace f x =
    let dx = 1e-5
    in (f (x +. dx) -. f (x -. dx)) /. (2. *. dx);;
-> val derivace : (float -> float) -> float -> float = <fun>

derivace (derivace (fun x -> x *. x)) 10.;;
-> - : float = 1.99996463834395377
```

Operátory

```
5 + 5;;
(+) 5 5;;
let inc = (+) 1;;
let (+) = (+.);;
```

infix operátor: `[=<>@^|&+-*/$%][=<>@^|&+-*/$%.:!?~]*`
prefix operátor: `[!?~][=<>@^|&+-*/$%.:!?~]*`

Pojmenované a nepovinné argumenty

```
let f ~x ~y = x + y;;
let g ?(x = 1) y = x + y;;

f ~y:1 ~x:2;;
let y = 1 and x = 2 in f ~y ~x;;
g 10;;
g ~x:10;;
```