OCaml

sintaksa

```
end of line - ;;
print_string [val] - prints to console
comments - (* this is a comment *)

Types:
int - integers: 0, 2, -3 ...
float - real numbers 12.3, -0.33, 6.02e-23 ...
bool - boolean: true/false
unit - no value: ()
string -: "qwerty", "arkarbar" ...
char - characters: 'a', '2', '-' ...

Operators:
[+-*/] : integers: 2+3, -5-2 ...
[+-*/]. : floats: 2.3 +. 5 ...
not_ : (_ = space character) negation: not true = false
[<>] : less/more than: 3<5 = true, 4>6 = false ...
[val] ** [exponent] : powers (must be floats): 2. ** 10. = 1024.

Conditions:
if [condition] then [expression] else [expression];;
ex. if i<5 then i++ else i--;;

Conversions:
float_of_int [val] - transforms to float
int_of_float [val] - floors a float number
int_of_char [val] - ASCII value of character
char_of_int [val] - converts ASCII value to character

Cartesian product:
(3, 'a') -> int * char
2, 2., "2", '2' -> int * float * string * char
() - not mandatory
fst (a, b) -> returs first element: a
snd (a, b) -> returns second element: b

Lists:
[3;3;2] - int list
List.nth [1;2;3] 2 -> returns second element: 2
```

```
let rec dolzina sez = match sez with
| [] -> 0
| _ :: rep -> 1 + dolzina rep;;
```

strings

```
let str="Miha";;
String.get str 2;; (*izpiše znak na mestu 2*)
str.[2];;
Bytes.set str 2 '7';; (*zamenjamo znak na mestu 2 z drugim znakom*)
```

```
(* Napiši funkcijo, ki niz:string prepiše v seznam:char list *)

let rec explode2 i l =
  if (i < 0) then 1
  else explode2 (i-1) (str.[i]::l);;

let explode str = explode2 str (String.length str - 1)[]

epxlode "Miha je star 27 let."

(* urejen zapis *)
let explode str =
  let rec explode2 i l =
    if (i < 0) then 1
    else explode2 (i-1) (str.[i]::l);;
  in explode2 (String.length str - 1) [];;

epxlode "Miha je star 27 let."
```

```
(*napiši funkcijo, ki vhodni niz pretvori v seznam znakov*)

let rec aux i l s =
    if (i=0) then l
    else aux (i-1) (s.[i]::l) s;;

let explode s = aux (String.length s - 1) [] s;;

let str = "Miha";;
explode str;;

(*ali*)

let explode s =
    let rec aux i l s =
     if (i=0) then l else aux (i-1) (s.[i]::l) s in
 aux (String.length s - 1) [] s;;
```

arrays

```
element::array
```

```
let v = [| 3.14; 6.28; 9.42 |] ;;
val v : float array = [|3.14; 6.28; 9.42|]
let v = Array.create 3 3.14;;
val v : float array = [|3.14; 3.14; 3.14|]
```

```
array.(n);; - vrne n-ti nelement v arrayu

# v.(1) ;;
- : float = 3.14
# v.(0) <- 100.0 ;;
- : unit = ()
# v ;;
- : float array = [|100; 3.14; 3.14|]
```

```
#let n = 10;;
val n : int = 10;;
#let array = Array.create n 0;;
val v:int array = [|0;0;0;0;0;0;0;0;0;0|]
```

```
(*napiši funkcijo, ki vhodni seznam znakov pretvori v niz*)

let implode sez =
    let rec aux sez =
        if sez = [] then ""
        else (Char.escaped (List.hd sez))^(implode (List.tl sez));;

let sez = ['M'; 'i'; 'h'; 'a'];;
implode sez;;
```

```
(*funkcija, ki obrne dani seznam*)

let rec obrni sez = match sez with
| [] | [_] -> sez
| glava::rep -> (obrni rep)@[glava];;

let lst = ['M'; 'i'; 'h'; 'a'];;
obrni lst;;
```

```
(*funkcija, ki preveri, če je vhodni niz palindrom*)
let palindrom str = (str = (implode(obrni(explode str))));;
(*funkcija implode je od prej*)

palindrom "pericarezeracirep";;
```

```
let implode s =
  let izpis = String.create (List.length s) in
  let rec implode2 i s = match s with
  | [] -> izpis
  | g::r -> izpis.[i] <- g; implode2 (i+1) r
  in
  implode2 0 s;;
```

funkcije višjega reda

funkcija, ki kot vhodni parameter sprejme drugo funkcijo

primer funkcije višjega reda: List.map (funkcija) [seznam]

```
List.map (fun x->x*x) [1;2;3;4;5];;
- : int list = [1; 4; 9; 16; 25]
```

```
(*fold right

List.fold_right  f [a1; a2;...; an] b

f a1 (...(f an-1 (f an b)))...*)
```

```
List.fold_right (fun x y -> x*y) [1;2;3] 5;;
List.fold_right (fun x y -> x*y) [1;2;3] 15;;
List.fold_right (^) ["M"; "i"; "h"; "a"] "abc";;

(* List.fold_left f a [b1;...;bn]
f (f (f a b1) b2) b3...)bn *)

List.fold_left (fun x y -> x*y) 5 [1;2;3];;
List.fold_left (fun x y -> x*y) 15 [1;2;3];;
List.fold_left (^) "abc" ["M"; "i"; "h"; "a"];;


- : int = 30
- : int = 90
- : string = "Mihaabc"

- : int = 30
- : int = 90
- : string = "abcMiha"
```

```
(*filter*)
List.filter (fun (_,x) -> x>5) [("Miha", 6); ("Ahim", 5); ("Luka", 1); ("Mojca", 10)];;
List.filter (fun x -> x > 10) [12; 5; 34; 6; 10; 6; 44];;


- : (string * int) list = [("Miha", 6); ("Mojca", 10)]
- : int list = [12; 34; 44]
```

```
(*for_all*)
List.for_all (fun x -> x>10) [12; 34; 11; 61; 44];;
List.for_all (fun x -> x>10) [5; 34; 11; 61; 44];;

- : bool = true
- : bool = false
```

```
let merge seznam1 seznam2 =
List.fold_right (fun x y -> x::y)seznam1 seznam2;;
```

🗒️ naloge

1. domača naloga

algoritmi za urejanje

selection sort

```
let rec find_min sez = match sez with
| g :: [] -> (g,[])
| g :: r -> let (m, s) = find_min r
in
if g<m then (g,r)
else (m,g::s);;
```

```
let rec selection_sort seznam =
if seznam = [] then []
else let (m,sez) = find_min seznam
in m :: (selection_sort sez);;
```

insertion sort

```
let rec insert seznam x = match seznam with
| [] -> [x]
| g::r -> if x<g then x::seznam
else g::(insert r x);;
```

```
let insertion_sort seznam =
let rec aux sored unsortred = match unsorted with
| [] -> sorted
| g::r -> aux (insert sorted g) r
in m :: (selection_sort seznam)
```

merge sort

```
let rec = split seznam = match sezna with
| [] | [_] -> (seznam,[])
| g1::g2::rep -> let (sez1,sez2) = split r in (g1::sez1,g2::sez2);;

let rec merge sez1 sez2 = match (sez1,sez2) with
|[],_ -> sez2
|_,[] -> sez1
g1::r1,g2::r2 -> if g1<g2 then g1::(merge r1 sez2) else g2::(merge sez1 r2);;
```

```
let rec merge_sort seznam = match eznam with
| [] | [_] -> seznam
| [] -> let (s2,s2) = split seznam in
  let (sez1,sez2) = (merge_sort s1, merge_sort s2) in
    merge sez1 sez2;;
```

quicksort

```
let pivot_split x sez  =
  (List.filter (fun y -> y<x) sez, List.filter (fun y -> x<=y) sez);;

let quick_sort seznam = match seznam with
| [] -> []
| g::r -> let (s1,s2) = pivot_split g r in
  (quick_sort s1)@[g]@(quick_sort s2);;
```

imperativno programiranje

array

```
let polje = [|1;2;5;6|];;
let polje1 = Array.make 5 "beseda";;

(* array - DIREKTNI DOSTOP! *)
polje.(2);;

(* array - sprememba elementa *)
polje.(2) <- 50;;

(* vnos elementa v matriko *)
let marika = Array.make_matrix 3 2 1;;
matrika.(2).(1) <- 10;;
```

for zanka

```
for i = 0 to 10 do (* to vedno pomeni navzgor *)
  print_int i; (* podpičje ignorira vrednost odgovora in gre na naslednji ukaz *)
  print_string "\n"
  done;;

for i = 10 downto 0 do (* downto vedno pomeni navzdol *)
  print_int i;
  print_string "\n"
  done;;
```

reference - kazalci

```
let n = ref 12;;

(* klicaj uporabimo za dostop do vrednosti, na katero kaže referenca *)
!n;;
if !n < 12 then true else false;;

(* := uporabljamo za spremembo vrednosti, na katero kaže kazalec *)
n := 13;; (* referenca zdaj keže na vrednost 13 *)
```

while zanka

```
(* while ... do ...done;; *)
```

```
let r = ref 1 in
  while !r < 11 do
    print_int !r;
    print_string " ";
    r := !r+1
  done;;
```

🗒️ naloge

tipi (neke vrste OOP)

```
type koordinatni_sistem = {x:int; y:int; ime;char};;

let tocka = {12,42,'C'};;
```

```
tocka.x
tocka.y
tocka.ime
```

```
type oseba = {ime:string; priimek:string; starost:int; spol:char; student:bool};;

let o1 = {
  ime="Andrej";
  primek="Erjavec";
  mutable starost="20";
  spol='M';
  mutable student=true;
};;

o1.priimek;; (* izpiše priimek *)
o1.starost <- 21;; (* spremenimo starost *)
```

```
(* primer funkcije, ki dela s tipom oseba *)
let rojstni_dan oseba =
  print_string "Vse najboljse ";
  print_string oseba.ime;
  print_string "!\n";
  oseba.starost ← (oseba-starost+1);;
```

```
type koordinate = float*float

let ljubljana:koordinate = (46.1,14.5)
```

```
type logicna_vrednost = Res | NiRes
(* Res | NiRes - konstruktor *)
```

```
type barva = Modra | Rdeca | Rumena;;
type material = Usnje | Jeans | Bombaz;;
type velikost = S | M | L;;
type obleka = barva*material*velikost;;

let hlace:obleka = (Modra,Bombaz,M);;
```

```
type barva =
  | RBG of int*int*int;; (* RGB je konstruktor tipa barva *)
  | CB of int
  | CMYK of int*int*int*int;;


RGB (255,0,0);; (* - : barva = RGB (255, 0, 0) *)

let b1 = RGB (255,0,0);;
let b2 = RGB (14,234,0);;
let b3 = CB 24;;

(* napišimo funkcijo, ki pove ali je barva crno-bela (CB) *)
let ni_CB barva = match barva with
  | CB _ -> false
  | _ -> true;;
```

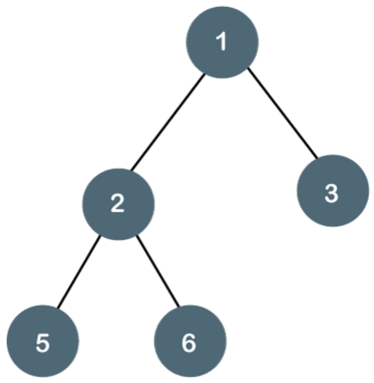
🗒️ naloge

rekurzivne podatkovne strukture

binarna drevesa

```
type binarno_drevo =
  | List of int
  | Stars of binarno_drevo * int
  | Velik_stars of binarno_drevo * binarno_drevo * int;;

List 3;;
Velik_stars (Velik_stars (List 5, List 6, 2), List 3, 1);;
```



```
type 'a binarno_drevo =
| List of 'a
| Tree of 'a binarno_drevo * 'a binarno_drevo;;
```

objekti in razredi

```
class hisa color okna vrata sobe etaze =
object
    (* spremenljivke oz. atributi *)
    val mutable barva = (color:string)
    val st_oken = (okna:int)
    val st_vrat = (vrata:int)
    val st_sob  = (sobe:int)
    val st_etaz = (etaze:int)
    val visina_etaze = 2.5
    val povrsina_sobe = 5

    (* funkcije oz- metode *)
    method get_barva = barva
    method get_st_sob = st_sob
    method set_barva novabarva = barva <- novabarva
    method kvadratura = povrsina_sobe * st_sob
    method visina = visina_etaze *. (float_of_int st_etaz)
end;;

let primerjaj_visine h1 h2 =
    if (h1#visina > h2#visina)
    then "Prva hisa je vecja."
    else "Druga hisa ni manjsa.";;
```

```
let h1 = new hisa "bela" 7 3 5 1;;
let h2 = new hisa "turkizna" 10 5 8 2;;
```

```
h1#get_barva;;
h1#set_barva "roza";;

primerjaj_visine h1 h2;;
```

```
class prevozno_sredstvo ime povprecna_hitrost =
object
  val ime_vozila = (ime:string)
  val povprecna_hitrost = (povprecna_hitrost:int)

  method potreben_cas razdalja =
    (razdalja / povprecna_hitrost)*60
end;;
```

```
let avtomobil = new prevozno_sredstvo "avto" 90;;
let tovornjak = new prevozno_sredstvo "MAN" 60;;
```

```
avtomobil#potreben_cas 250;;
tovornjak#potreben_cas 250;;
```

```
class izpitna_naloga s_t d_t tez =
object
  val mutable skupne_tocke = (s_t:int)
  val mutable dosezene_tocke = (d_t:int)
  val mutable tezavnost = (tez:int)

  method get_skupne_tocke = skupne_tocke
  method get_dosezene_tocke = dosezene_tocke
  method get_tezavnost = tezavnost

  method set_skupne_tocke x = skupne_tocke <- x
  method set_dosezene_tocke x = dosezene_tocke <- x
  method set_tezavnost x = tezavnost <- x

  method to_string =
    print_string "Izpitnma naloga tezavnosti ";
    print_string tezavnost;
    print_string ".\\nDosezene tocke: ";
    print_string dosezene_tocke;
    print_string ".\\nTezavnost: ";
    print_string tezavnost;
end;;
```

dodatek k poglavju objekti in razredi

```
class hisa b okna vrata sobe etaze =
object
    (*spremenljivke oz. atributi*)
    val mutable barva = (b:string)
    val st_oken = (okna:int)
    val st_vrat = (vrata:int)
    val st_sob = (sobe:int)
    val st_etaz = (etaze:int)
    val visina_etaze = 2.5
    val povrsina_sobe = 5
    (* funkcije oz. metode *)
    method get_barva = barva
    method get_st_sob = st_sob
    method set_barva novabarva = barva <- novabarva
    method kvadratura = povrsina_sobe * st_sob
    method visina = visina_etaze *. (float_of_int st_etaz)
end;;

class mesto (ime:string) =
object
    val ime=ime
    val mutable st_prebivalcev = 2
    val mutable st_his = 0
    val mutable hise = ([]:hisa list)

    method get_ime = ime
    method get_prebivalci = st_prebivalcev
    method get_st_his = st_his

    method dodajHiso h =
        hise <- h::hise;
        st_prebivalcev <- st_prebivalcev + (h#get_prebivalci);
        st_his <- (st_his + 1);
end;;
```

```
let h1 = new hisa "bela" 7 3 5 1 4;;
let h2 = new hisa "turkizna" 18 3 12 3 6;;
let m1 = new mesto "Idrija";;

m1#dodajHiso h1;;
```

dedovanje

```
(* dedovanje *)

class hisa color okna vrata sobe etaze prebivalci=
object (self)
    (* spremenljivke oz. atributi *)
    val mutable barva = (color:string)
    val st_oken = (okna:int)
    val st_vrat = (vrata:int)
    val st_sob  = (sobe:int)
    val st_etaz = (etaze:int)
    val prebivalci = (prebivalci:int)
    val visina_etaze = 2.5
    val povrsina_sobe = 5

    (* funkcije oz- metode *)
    method get_barva = barva
    method get_st_sob = st_sob
    method set_barva novabarva = barva <- novabarva
    method kvadratura = povrsina_sobe * st_sob
    method visina = visina_etaze *. (float_of_int st_etaz)
    method to_string = "Hisa s kvadraturo "^string_of_int (self#kvadratura)^" m2."
end;;

class vikend barva okna vrata sobe pogled =
object (self)
    inherit hisa barva okna vrata sobe 1 0 as super
    val pogled_na_morje = (pogled:bool)
    method to_string = match pogled with
    | false -> super#to_string
    | true -> super#to_string^ "ima tudi polged na morje."
    initializer print_string (self#to_string^"\n") (* to se izvede takoj ko ustvarimo
 objekt *)
end;;
```

```
let v1 = new vikend "zelena" 4 1 4 true;;
let h1 = new hisa "bela" 2 3 4 5 6;;
```

```
v1#kvadratura;;
v1#to_string;;
```

20. 4. 2021

```
(* ponovimo *)
class pravokotnik a b =
    object
        val a = a
        val b = b
        val mutable barva = ""

        method vrni_a = a
        method vrni_b = b
        method vrni_barva = barva

        method ploscina = a*b
        method obseg = 2*a + 2*b
        method pobarvaj novabarva = barva <- novabarva
    end;;

class kvader a b c =
    object (self)
        inherit pravokotnik a b as super
        initializer print_string ("Izdelujemo nov kvader z volumnom "^(string_of_int s
elf#prostornina)^".\n")
        val c = c

        method povrsina = 2 * (super#ploscina + a*c + b*c)
        method prostornina = super#ploscina * c
    end;;

(* initializer *)
```

```
let p1 = new pravokotnik 5 7;;
p1#ploscina;;
```

```
let k1 = new kvader 4 5 6;;
k1#povrsina;;
k1#prostornina;;
```

privatne metode

```
(* privatne metode *)
class stavek besedilo =
    object (self)
        val besedilo = (besedilo:string)

        method vrni_besedilo = besedilo
        method private explode besedilo =
            let rec aux i l =
                if (i<0) then l else aux (i-1) (besedilo.[i]::l)
            in aux (String.length besedilo - 1) []

        method private implode seznam =
            let izpis = Bytes.create (List.length seznam) in
            let rec aux i s = match s with
                | [] -> izpis
                | g::r -> Bytes.set izpis i g; aux (i+1) r
            in aux 0 seznam

        method obrni_besedilo =
            self#implode (List.rev(self#explode besedilo))
end;;
```

abstraktni razredi

```
(* abstraktni razredi *)

class virtual pravilni_ngon barva stranice =
  object
    val barva = (barva:string)
    val st_stranic = (stranice:int)
    method virtual ploscina : int
    method virtual obseg : int
  end;;

class kvadrat barva a =
  object
    inherit pravilni_ngon barva 4
    method ploscina = a*a
    method obseg = 4*a
  end;;
```

```
class sortirano_polje polje =
    object (self)
        val mutable polje = (polje:int array)
        method vrni_polje = polje
        method private n = Array.length polje
        method zdruzi polje2 =
            let n2 = Array.length polje2 in
            let novoPolje = Array.make (self#n + n2) 0 in
            let p1 = ref 0 in
            let p2 = ref 0 in (* kazalca na aktivno mesto v obeh poljih *)
            while (!p1 + !p2) < (self#n + n2) do
                let prviManjsi = ( if !p1=self#n then false else
                                       if !p2 < n2 && polje.(!p1) > polje2.(!p2) then
 false else true)
                in if prviManjsi then
                    (novoPolje.(!p1 + !p2) <- polje.(!p1); p1:= !p1+1)
                else
                    (novoPolje.(!p1 + !p2) <- polje2.(!p2); p2:= !p2+1)
            done;
            polje <- novoPolje
    end;;
```

```
let p1 = new sortirano_polje [|1;5;7;12;15;17|];;
let stevila = [|1;5;7;8;9;11;13;14;24;27|];;
p1#zdruzi stevila;;
p1#vrni_polje;;
```

upravljanje s težavami - error handling

```
exception Foo;;
exception Omara;;
exception Posebna_tezava of int*char;;

raise Foo;
raise Omara;;
raise (Posebna_tezava (12*'C'));;
```

```
try raise (Boo(-15)) with
| Boo 12 -> "dobra napaka"
| Bii i when i<0 -> "slaba napaka";;
```

```
let f a b = a*a-2*b+b+b*b*b
```

> če se funkcija začne z: = @ ^ | & + - * / $ % in nadaljuje s poljubno mnogo ! $ % & * + - . / : ? @ ^| — potem je funkcija infiksna

stack

```
(*  podatkovna struktura sklad *)

module Stack =
struct
  exception Napaka of string
  type 'a t = {mutable c: 'a list; mutable len: int}

  let create = {c=[]; len=0}
  let clear s = s.c <- []; s.len=0
  let push x s = s.c <- x::s.c; s.len <- s.len+1
  let pop s = match s.c with
    | hd::tl -> s.c <- tl; s.len <- s.len-1; hd
    | [] -> raise (Napaka "Sklad je prazen!")
  let is_empty s = (s.len = 0)
  let length = s.len
end;;
```

queue

```
(*  podatkovna struktura vrsta *)

module Queue =
struct
  exception Napaka of string
  type 'a t = {mutable c: 'a list; mutable len: int}

  let create = {c=[]; len=0}
  let clear s = s.c <- []; s.len=0
  let push x s = s.c <- s.c@[x]; s.len <- s.len+1
  let dequeue s = match s.c with
    | hd::tl -> s.c <- tl; s.len <- s.len-1; hd
    | [] -> raise (Napaka ("Sklad je prazen!"))
  let is_empty s = (s.len = 0)
  let length s = s.len
end;;


let qu = Queue.create;;

Queue.clear qu;;
Queue.push 12 qu;;
Queue.push 13 qu;;
Queue.push 14 qu;;

Queue.dequeue qu;;
Queue.dequeue qu;;

Queue.length qu;;
```