FUNKCIONALNO PROGRAMIRANJE

Oznaka predmeta: FPR

Predavanje broj: 12

Nastavna jedinica: LISP,

Nastavne teme:

Posebni argumenti (optional, rest, key, aux). Podaci: svojstva. Štampanje i rad sa datotekama. Konstante, makroi, globalne i lokalne promenljive. Inkrementiranje , dekrementiranje, bitwise operacije. Iteracije, povratak iz funkcije, dispečer karakter. Brojevi i funkcije za brojeve. Poređenje karaktera, kreiranje niza. Poređenje karaktera, kreiranje niza. Stringovi.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

David S. Touretzky: "Common Lisp: A Gentle Introduction to Symbolic Computation", Dover Publications, 2013.

LISP, posebni argumenti

POSEBNI ARGUMENTI PROCEDURA (&optional, &rest, &key, &aux)

- Pri definisanju parametara procedure prvo se navode:
  - obavezni,
  - &OPTIONAL i niz neobaveznih,
  - &REST i jedan neobavezni,
  - &KEY parametri,
  - &AUX parametri.

- Deo argumenata procedura može biti neobavezan.
  - Ovim argumentima u definiciji procedure prethodi rezervisana reč &OPTIONAL.
  - Takvim argumentima se, ukoliko u pozivu nisu navedeni, a definicijom im nije definisana početna vrednost, pri pozivu dodeljuje vrednost NIL. Opšti oblik:

```
... &OPTIONAL <argument> ...)
ili
... &OPTIONAL (<argument> <pocetna vrednost)) ...)
```

LISP, posebni argumenti

- Pretpostavimo da listu shvatimo kao rečenicu kojoj treba dodati znak interpunkcije na kraj. Napisati proceduru INTERPUNK koja ce listi dodati na kraj tačku (simbol TACKA) ako je pozvana samo s tom listom kao argumentom, odnosno drugi argument kada on postoji u pozivu.

```
(defun interpunk (L &optional (znak 'tacka))
   (append L (list znak)) )
```

- Primer &optional:

```
(defun show-members (a b &optional c d) (write (list a b c d)))
(show-members 1 2 3)                                    (1 2 3 NIL)
(terpri)
(show-members 'a 'b 'c 'd)                              (A B C D)
(terpri)
(show-members 'a 'b)                                    (A B NIL NIL)
(terpri)
(show-members 1 2 3 4)                                  (1 2 3 4)
```

- Napisati repnu rekurzivnu proceduru EXPT1, koja ce računati x**y.
  - Koristiti neobavezni parametar kako bi akumulirali parcijalni rezultat.

```
(defun expt1 (x y &optional (R 1))
   (if (zerop y) R
                 (expt1 x (- y 1) (* R x))))
```

LISP, posebni argumenti

- Neobaveznim argumentima može prethoditi i rezervisana reč &REST.
  - U definiciji procedure joj sledi samo jedan formalni argument.
  - Pri pozivu se tom argumentu pridružuje lista koju čine svi neobavezni stvarni argumenti.

  Opšti oblik:

```
... &REST <naziv liste neobaveznih argumenata>)
```

```
(defun show-members (a b &rest values) (write (list a b values)))
(show-members 1 2 3)                              (1 2 (3))
```

- &KEY i &AUX parametri:

```
(defun podaci (&key (ime 'Elvis) (prezime 'Presley))
   (list "Podaci:" ime prezime))
                    (podaci)                  ("Podaci:" ELVIS PRESLEY)
                    (podaci :ime 'Arron)      ("Podaci:" ARRON PRESLEY)
```

```
(defun prva-trecina-liste ( L &aux (start 0) (end (length L)) )
    (reverse (nthcdr (/ (- end start) 3/2) (reverse L))))
```

LISP, posebni argumenti

- Primer za &rest:

```
(defun show-members (a b &rest values) (write (list a b values)))
(show-members 1 2 3)                                (1 2 (3))
(terpri)
(show-members 'a 'b 'c 'd)                          (A B (C D))
(terpri)
(show-members 'a 'b)                                (A B NIL)
(terpri)
(show-members 1 2 3 4)                              (1 2 (3 4))
(terpri)
(show-members 1 2 3 4 5 6 7 8 9)                    (1 2 (3 4 5 6 7 8 9))
```

- Primer za &key:

```
(defun show-members (&key a b c d ) (write (list a b c d)))
(show-members :a 1 :c 2 :d 3)                       (1 NIL 2 3)
(terpri)
(show-members :a 'p :b 'q :c 'r :d 's)              (P Q R S)
(terpri)
(show-members :a 'p :d 'q)                          (P NIL NIL Q)
(terpri)
(show-members :a 1 :b 2)                            (1 2 NIL NIL)
```

LISP, podaci: svojstva

- Napisati repnu rekurzivnu proceduru REVERSE1, koja će kao rezultat vraćati listu koju čine elementi ulazne liste, ali u obrnutom redosledu. Pretpostaviti da primitiva REVERSE ne postoji.

```
(defun reverse1 (L &optional (LL nil))
   (if (null L) LL
                (reverse1 (rest L) (cons (first L) LL)) ) )
```

PODACI - Svojstva (get, setf, remprop)

- Simbol može istovremeno imati više pridruženih vrednosti.
  - Njegova osnovna vrednost se dohvata njegovim "izračunavanjem".
  - Ostale vrednosti se nazivaju svojstva i dohvataju, odnosno, pridružuju se na poseban način.
- Primitiva GET dohvata svojstvo čiji je naziv jednak rezultatu <svojstva> od simbola čije je ime jednako rezultatu <simbola>.
  Opšti oblik:

```
(GET <simbol> <svojstvo>)
```

- Specijalna forma SETF, osim za pridruživanje osnovne vrednosti simbolu, služi i za pridruživanje svojstava simbolu. Oba <simbol> i <svojstvo> imaju isto značenje kao i kod primitive GET.

LISP, podaci: svojstva

Tada je opšti oblik:

```
(SETF (GET <simbol> <svojstvo>) <iznos svojstva>)
(SETF (GET 'Perica   'otac    ) 'Svetozar       )
(SETF (GET 'Svetozar 'otac    ) 'Stevan         )
```

- Svojstvo se može ukloniti primitivom REMPROP. Oba: <simbol> i <svojstvo> imaju isto značenje kao i kod primitive GET.
  Opšti oblik:

```
(REMPROP <simbol> <svojstvo>)
```

- Pretpostavite da, ako je poznat otac osobe x (ime oca je definisano kao vrednost svojstva OTAC), napisati proceduru DEDA koja kao rezultat vraća ime očevog oca osobe x, ako je poznato, a inače NIL.

```
(defun deda (x)
   (when (get x 'otac )
         (get (get x 'otac) 'otac)) )

(defun ded1 (x)
   (let ((otac (get x 'otac)))
   (when otac (get otac 'otac))) )
```

LISP, podaci: svojstva

- Napisati proceduru ADAM, koja vraća ime najudaljenijeg pretka po muškoj liniji, koristeći svojstvo OTAC opisano u prethodnom zadatku. Zbog jednostavnosti pretpostavite da se niti jedno ime ne pojavljuje na više mesta.

```
(defun Adam(ime)
   (if (equal (get ime 'otac) nil) ime
                                    (Adam (get ime 'otac))))
```

- Pretpostavite da gradovi imaju svojstvo POLOZAJ, čija je vrednost lista s dve koordinate. Pretpostavljajući da vredi Euklidova geometrija, napisati proceduru UDALJENOST, koja ce izračunati najmanju (vazdušnu) udaljenost između zadanih gradova, koristeći svojstvo POLOZAJ.

```
(setf (get 'g1 'polozaj) '(100 100))
(setf (get 'g2 'polozaj) '(200 200))
(defun udaljenost (g1 g2)
 (sqrt (+ (* (- (first  (get g1 'polozaj)) (first  (get g2 'polozaj)))
             (- (first  (get g1 'polozaj)) (first  (get g2 'polozaj))))
          (* (- (second (get g1 'polozaj)) (second (get g2 'polozaj)))
             (- (second (get g1 'polozaj)) (second (get g2 'polozaj))))
 ))))
```

LISP, štampanje i rad sa datotekama

- Primitivom LOAD izvršava se unapred pripremljena <datoteka> s naredbama LISP-a. Rezultat primitive je T.
  Opšti oblik: `(LOAD <datoteka>)`

  ```
  (load "C:/Program Files (x86)/LispWorks Personal/pozdrav.lsp")
  ```

- Primitivom PRINT ispisuje se rezultat <forme>. Kao rezultat primitive javlja se isto što se i ispisuje.
  Opšti oblik:

  ```
  (PRINT <forma>)
  ```

- Primitivom READ učitava se izraz koji se ne izvršava, već doslovno postaje rezultat primitive.
  Opšti oblik:

  ```
  (READ)  npr.    (print (read))     ;ispisi uneto
                  (setf ime (read))  ;definisi uneto kao ime
  ```

- Primitiva READ-CHAR učitava samo jedan znak. Opšti oblik:

  ```
  (READ-CHAR )
  ```

- Primitivom WITH-OPEN-FILE sve obuhvaćene primitive PRINT (READ, READ-CHAR) upućuju se na pisanje (čitanje) na (sa) specifične datoteke, čije je ime definisano rezultatom <datoteke>, a <smer> podataka se treba izračunati u

LISP, štampanje i rad sa datotekama

INPUT ili OUTPUT. Ovde <telo> može sadržavati proizvoljan broj izraza, pri čemu će navedene U/I primitive koristiti definisanu datoteku.
Opšti oblik:

```
(WIDTH-OPEN-FILE (<datoteka> <smer>)
    <telo>)
```

- Primitiva EVAL nije vezana za čitanje i pisanje podataka, ali se može podesno upotrebiti u kombinaciji s primitivom READ. Funkcija joj je da izvrši rezultat <izraza>.
  Opšti oblik:

```
(EVAL <izraz> )
npr.    (eval (read))
        (+ 1 1)         ; uneseno
        2
```

- Primer formatiranog ispisa:

```
(format t "~A ~D ~X ~5,2F" "tekst" 20 255 3.14)
                                                tekst 20 FF  3.14
(format t "~A~& ~D~& ~X~& ~5,2F~&" "tekst" 20 255 3.14)
                                                tekst
                                                20
                                                FF
                                                3.14
```

LISP, štampanje i rad sa datotekama

```
(with-open-file (stream "/Py/lisp.txt" :direction :output)
   (format stream "Welcome to LISP")
   (terpri stream)
   (format stream "OK")
   (terpri stream)
   (format stream "Submit your Tutorials, White Papers and
Articles into our Tutorials  Directory.")
)


(let ( (in (open "/Py/lisp.txt" :if-does-not-exist nil)) )
   (when in
      (loop
            for line = (read-line in nil)
            while line do (format t "~a~%" line)
      )
      (close in)
   )
)
```

LISP, konstante, makroi, globalne i lokalne promenljive

- Primer definisanja "konstante" i formatiranog ispisa

```
(write-line "Hello World") ; komentar
(defconstant MYPI 3.14)
(defun area-circle(rad)
  (terpri)
  (format t "Radius: ~5f" rad)
  (format t "~%Area: ~10f" (* MYPI rad rad))
)
(area-circle 10)
```

- Ispis tipa:

```
(setq x 10)
(print (type-of x))     ispisuje FIXNUM
```

- Korišćenje makroa:

```
(defmacro setTo10(num)
  (setq num 10)(print num))
(setq x 25)
(print x)
(setTo10 x)
```

- Globalna promenljiva: `(defvar x 234)`
- Postavljanje vrednosti simbola korišćenjem setq (i globalno i lokalno):

```
(setq x 10)      (setq y 20)
(format t "x = ~2d y = ~2d ~%" x y)
```

LISP, inkrementiranje , dekrementiranje, bitwise operacije

- Primer lokalnih promenljivih korišćenjem let:

```
(let ( (x 'a)
       (y 'b)
       (z 'c)
     )
 (format t "x = ~a y = ~a z = ~a" x y z)
)
```

- Uvećavanje i umanjenje vrednosti varijable:

```
(setf A 10)                    ; A=10
(incf A 100)          ; A=10+100=110
(decf A 50)           ; A=110-50=60
```

- Poređenja (=,/=,>,<,>=,<=,max,min):

```
(/= 10 20)                                  T
```

- Bitwise operacije:

```
logand, logior, logxor, lognor, logeqv
(npr. A=60, B=13) (logand A B)   rezultat je 12
```

- Korišćenje formata:

```
(setq day 4)
(case day
(1 (format t "~% Monday"   ))  (2 (format t "~% Tuesday" ))
(3 (format t "~% Wednesday"))  (4 (format t "~% Thursday"))
(5 (format t "~% Friday"   ))  (6 (format t "~% Saturday"))
(7 (format t "~% Sunday"   )))
```

LISP, iteracije

- Obična petlja loop:

```
(setq a 10)
(loop
    (setq a (+ a 1))
    (write a)
    (terpri)
    (when (> a 17) (return a))
)
(loop for x in '(Elvis Aaron Presley)
   do (format t " ~s" x)
)
(loop for a from 10 to 20
   do (print a)
)
```

- Primer petlje do:

```
(do (
     (x  0 (+ 2 x))
     (y 20 (- y 2))
    )
    ((= x y)(- x y))                          ;ili ((= x y) 'POZDRAV)
        (format t "~% x = ~d  y = ~d" x y)
)
```

LISP, iteracije, povratak iz funkcije, dispečer karakter

- Primer petlje dotimes i dolist

```
(dotimes (n 11)
   (print n) (print (* n n))
)
(dolist (n '(0 1 2 3 4 5 6 7 8 9 10))
   (format t "~% Number: ~d Square: ~d" n (* n n))
)
```

- Primer povratka iz funkcije korišćenjem return-from:

```
(defun myfunc (num)
   (return-from myfunc 10)
   num
)
(write (myfunc 20))
```

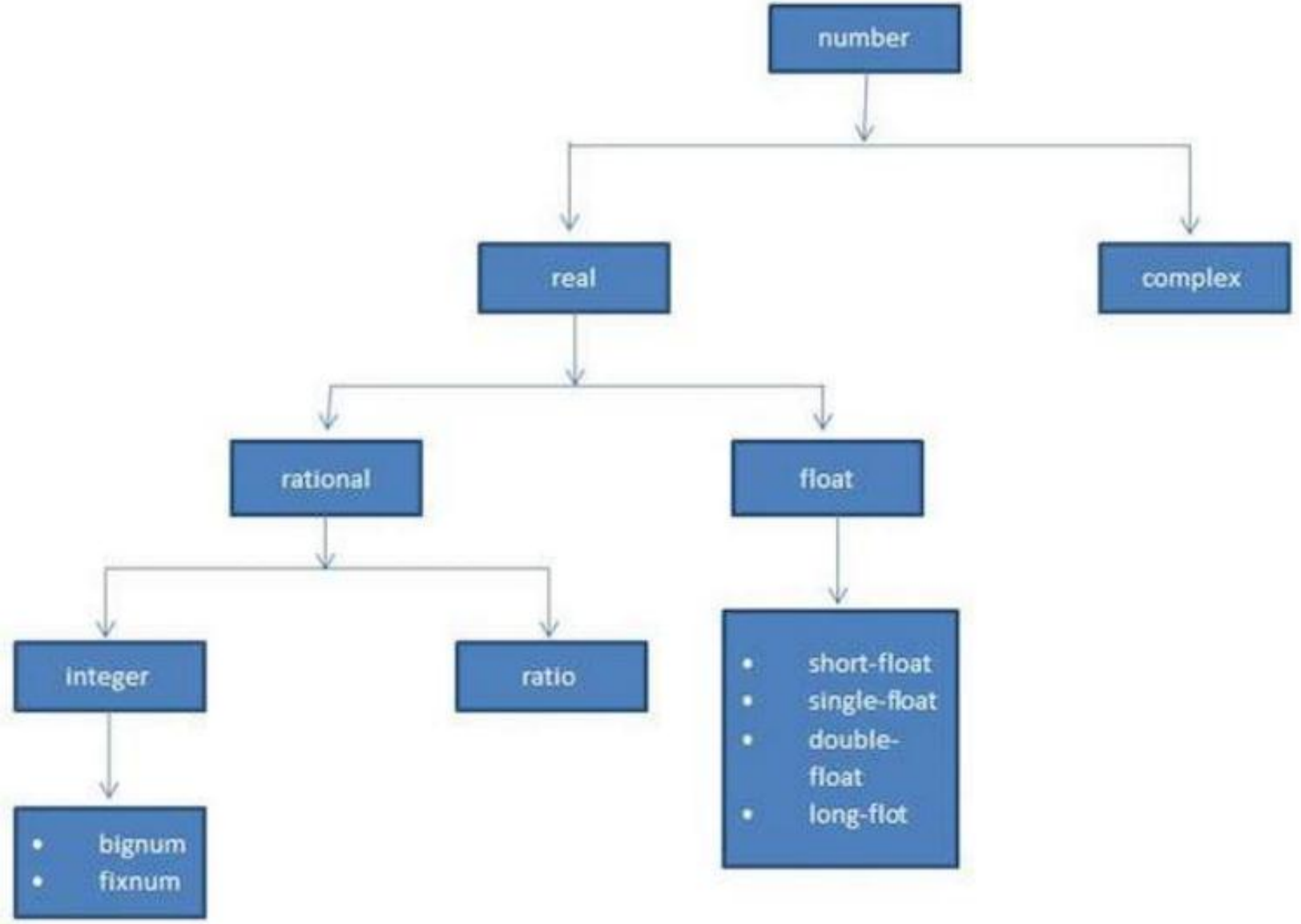
- Primer za funkciju lambda:

```
(write (   (lambda (a b c x)   (+ (* a (* x x)) (* b x) c))
                    4 2 9 3 ))
```

- Primer za mapcar:

```
(write (mapcar '1+ '(23 34 45 56 67 78 89)))     (24 35 46 57 68 79 90)
(write (mapcar '+ '(1 3 5 7 9 11 13) '(2 4 6 8)))          (3 7 11 15)
(defun cubeMylist(lst)
   (mapcar #'(lambda(x) (* x x x)) lst))
(write (cubeMylist '(1 3 5)))                               (1 27 125)
```

LISP, brojevi



LISP, brojevi i funkcije za brojeve

- Primer ispisa brojeva u lisp-u:

```
razlomak    (write (/ 1 2))
razlomak    (write ( + (/ 1 2) (/ 3 4)))
kompleksan  (write ( + #c( 1 2) #c( 3 -4)))
```

- Primeri nekih funkcija za brojeve:

```
(write (/ 45 78))                    15/26
(write (floor 45 78))                0       45
(write (/ 3456 75))                  1152/25
(write (floor 3456 75))              46       6
(write (ceiling 3456 75))            47     -69
(write (truncate -7 2))              -3      -1
(write (round -7 2))                 -4       1
(write (ffloor 3456 75))             46.0     6
(write (fceiling 3456 75))           47.0   -69
(write (ftruncate 11 3.5))           3.0    0.5
(write (fround 3456 75))             46.0     6
(write (mod 3456 75))                6
(setq c (complex 6 7))               #C(6 7)
(write c)                            #C(6 7)
(write (complex 5 -9))               #C(5 -9)
(write (realpart c))                 6
(write (imagpart c))                 7
```

LISP, poređenje karaktera, kreiranje niza

- Poređenja karaktera:

```
; case-sensitive comparison
  (write (char= #\a #\b))                          NIL
  (write (char= #\a #\a))                          T
  (write (char= #\a #\A))                          NIL
;case-insensitive comparision
  (write (char-equal #\a #\A))                     T
  (write (char-equal #\a #\b))                     NIL
  (write (char-lessp #\a #\b #\c))                 T
  (write (char-greaterp #\a #\b #\c))              NIL
```

- Kreiranje niza:

```
(write (setf my-array (make-array '(10)))) #(NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL)
(setf (aref my-array 0) 25)
(setf (aref my-array 1) 23)
(setf (aref my-array 2) 45)
(setf (aref my-array 3) 10)
(setf (aref my-array 4) 20)
(setf (aref my-array 5) 17)
(setf (aref my-array 6) 25)
(setf (aref my-array 7) 19)
(setf (aref my-array 8) 67)
(setf (aref my-array 9) 30)
(write my-array)                    #(25 23 45 10 20 17 25 19 67 30)
```

LISP, nizovi

- Kreiranje 2d niza:

```
(setf x (make-array '(3 3)
   :initial-contents '((0 1 2 ) (3 4 5) (6 7 8)) )
)
(write x)                                  #2A((0 1 2) (3 4 5) (6 7 8))
```

- Kreiranje niza u dotimes petlji:

```
(setq a (make-array '(4 3)))
(dotimes (i 4)
   (
      dotimes (j 3)
        (setf (aref a i j) (list i 'x j '= (* i j)))
   )
)
(dotimes (i 4)
   (
     dotimes (j 3)
       (print (aref a i j))
   )
)
```

```
(0 X 0 = 0)
(0 X 1 = 0)
(0 X 2 = 0)
(1 X 0 = 0)
(1 X 1 = 1)
(1 X 2 = 2)
(2 X 0 = 0)
(2 X 1 = 2)
(2 X 2 = 4)
(3 X 0 = 0)
(3 X 1 = 3)
(3 X 2 = 6)
```

LISP, nizovi

- Kreiranje 1d niza korišćenjem 3d niza:

```
(setq myarray (make-array '(3 2 3)
      :initial-contents
          '( ((a b c) (1 2 3))
             ((d e f) (4 5 6))
             ((g h i) (7 8 9)) )  )
)
(setq array1 (make-array 4 :displaced-to myarray
                           :displaced-index-offset 2))
(write myarray)
(terpri)
(write array1)
          #3A(((A B C) (1 2 3)) ((D E F) (4 5 6)) ((G H I) (7 8 9)))
          #(C 1 2 3)
```

- Kreiranje 2d niza korišćenjem 3d niza datog iznad:

```
(setq array2 (make-array '(3 2) :displaced-to myarray
                                :displaced-index-offset 2))
           #3A(((A B C) (1 2 3)) ((D E F) (4 5 6)) ((G H I) (7 8 9)))
           #2A((C 1) (2 3) (D E))
```

LISP, nizovi

```
(write (make-array 5 :initial-element 5))
(terpri) #(5 5 5 5 5)
(write (make-array '(2 3) :initial-element 'a))
(terpri) #2A((A A A) (A A A))
; pointer popunjenosti na 5
(write(length (make-array 14 :fill-pointer 5))) ;#(NIL NIL NIL NIL NIL)
(terpri) 5
; postavljanje pointera popunjenosti ne utice na dimenziju niza
(write (array-dimensions (make-array 14 :fill-pointer 5)))
(terpri) (14)
; bit array sa inicijalno svim 1.
(write(make-array 10 :element-type 'bit :initial-element 1))
(terpri) #*1111111111
; character array, svi karakteri su a
(write(make-array 10 :element-type 'character :initial-element #\a))
(terpri) "aaaaaaaaaa"
  ; 2d array sa svim elementima 'a i koji se moze podesavati
  (setq myarray (make-array '(2 2) :initial-element 'a :adjustable t))
  (write myarray) #2A((A A) (A A))
  (terpri)
  ;podesavanje niza
  (adjust-array myarray '(1 3) :initial-element 'b)
  (write myarray) #2A((A A B))
```

LISP, stringovi

- Stringovi i operacije poređenja (slično kao i karakteri):

```
;case-sensitive comparison
(write (string= "this is test" "This is test"))          (terpri)   NIL
(write (string> "this is test" "This is test"))          (terpri)   0
(write (string< "this is test" "This is test"))          (terpri)   NIL
;case-insensitive comparision
(write (string-equal    "this is test" "This is test"))  (terpri)   T
(write (string-greaterp "this is test" "This is test"))  (terpri)   NIL
(write (string-lessp    "this is test" "This is test"))  (terpri)   NIL
;checking non-equal
(write (string/= "this is test" "this is Test"))         (terpri)   8
(write (string-not-equal "this is test" "This is test")) (terpri)   NIL
(write (string/= "lisp" "lisping"))                      (terpri)   4
(write (string/= "decent" "decency"))                               5
```

LISP, stringovi

- Neke operacije sa nizovima:

```
(write-line (string-upcase "a big hello"))                    A BIG HELLO
(write-line (string-capitalize "a big hello"))                A Big Hello

(write-line (string-trim       "."  "...a big hello..."))     a big hello
(write-line (string-left-trim  "."  "...a big hello..."))     a big hello...
(write-line (string-right-trim "."  "...a big hello..."))     ...a big hello
(write-line (string-trim       ".a" "...a big hello..."))     big hello

(write (length "Hello World"))                                11
(write-line (subseq "Hello World" 6))                         World
(write (char "Hello World" 6))                                #\W

(write (sort (vector "Amal" "Akbar" "Anthony") #'string<))
                                          #("Akbar" "Amal" "Anthony")

(write (merge 'vector (vector "Rishi" "Zara" "Priyanka")
                      (vector "Anju" "Anuj" "Avni") #'string<))
                         #("Anju" "Anuj" "Avni" "Rishi" "Zara" "Priyanka")

(write-line (reverse "Abcd Efgh"))                                hgfE dcbA
(write-line (concatenate 'string "Danas " "je lep dan."))
                                                    Danas je lep dan.
```