FUNKCIONALNO PROGRAMIRANJE

Oznaka predmeta: FPR

Predavanje broj: 10

Nastavna jedinica: PYTHON, LISP,

Nastavne teme:

PYTHON: Mape, filteri, redukcija, lambda.
LISP: Osnovni tipovi. Izraz. Atom. Lista. Osnovne primitive. Procedure. setf, let, let*.

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

Literatura:

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

FUNKCIONALNO PROGRAMIRANJE

- Korišćenje mapiranje je oblika:

```
map( funkcija, sekvenca )
```

primeri:

posmatrajmo primer bez korišćenja map-a

```
>>> items = [1, 2, 3, 4, 5]
>>> squared = []
>>> for x in items:
     squared.append(x ** 2)
>>> squared
                                        [1, 4, 9, 16, 25]
```

rešenje prethodnog primera uz upotrebu map-a

```
>>> items = [1, 2, 3, 4, 5]
>>> def sqr(x):
       return x ** 2
>>> list(map(sqr, items))
                                        [1, 4, 9, 16, 25]
```

ili sa malo kompaktnijim kodom

```
>>> items = [1, 2, 3, 4, 5]
>>> list(map((lambda x: x**2), items))
                                        [1, 4, 9, 16, 25]
```

FUNKCIONALNO PROGRAMIRANJE

- Korišćenje liste funkcija kao sekvence (upotrebom lambde):

```
def square(x):  return (x**2)
def cube(x):    return (x**3)
funcs = [square, cube]
for r in range(5):
    value = list(map(lambda x: x(r), funcs))
    print (value)
```

```
[0, 0]
[1, 1]
[4, 8]
[9, 27]
[16, 64]
```

- Korišćenje map-a je kao i petlje:

```
>>> def mymap(aFunc, aSeq):
        result = []
        for x in aSeq: result.append(aFunc(x))
        return result
>>> list(map(sqr, [1, 2, 3]))          #[1, 4, 9], sqr od ranije
>>> mymap(sqr, [1, 2, 3])              #[1, 4, 9]
```

FUNKCIONALNO PROGRAMIRANJE

- Map je ugrađen tako da je brži od ručno kodovanog (my)map-a.
- Moguće je koristiti funkciju i liste za argumente mapa ako funkcija zahteva dva argumenta.

```
>>> pow(2,10)
1024
>>> pow(3,11)
177147
>>> pow(4,12)
16777216
>>>
>>> list(map(pow,[2, 3, 4], [10, 11, 12]))
[1024, 177147, 16777216]
```

- Pravljenje liste n-torki u Python-u 3x:

```
>>> m = [1,2,3]
>>> n = [1,4,9]
>>> def f(*x): return x;
>>> new_tuple = list(map(f, m, n))
>>> new_tuple
[(1, 1), (2, 4), (3, 9)]
```

FUNKCIONALNO PROGRAMIRANJE

- Filteri
  ekstrahuju svaki element sekvence za koga funkcija ima vrednost True.

```
>>> list(range(-5,5))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
>>>
>>> list( filter((lambda x: x < 0), range(-5,5)))
                                        [-5, -4, -3, -2, -1]
```

  isto kao:

```
>>> result = []
>>> for x in range(-5, 5):
        if x < 0:
                result.append(x)
>>> result
                                        [-5, -4, -3, -2, -1]
```

- Redukcija
  redukuje listu na jednu vrednost tako što primenjuje funkciju na elemente liste po paru argumenata i tako redom do kraja liste.

```
>>> from functools import reduce
>>> reduce( (lambda x, y: x * y), [1, 2, 3, 4] )
24
```

FUNKCIONALNO PROGRAMIRANJE

- Ekvivalentan kod redukcije:

```
>>> L = [1, 2, 3, 4]
>>> result = L[0]
>>> for x in L[1:]:
        result = result * x
>>> result
24
```

- Ekvivalentan kod redukcije kao funkcije:

```
>>> def myreduce(fnc, seq):
        tally = seq[0]
        for next in seq[1:]:
                tally = fnc(tally, next)
        return tally
>>> myreduce( (lambda x, y: x * y), [1, 2, 3, 4])
24
>>> myreduce( (lambda x, y: x / y), [1, 2, 3, 4])
0.041666666666666664
```

FUNKCIONALNO PROGRAMIRANJE

- Npr. konkatenacija string literala u rečenicu:

```
import functools
>>> L = ['Testing ', 'shows ', 'the ', 'presence', ', ','not ',
'the ', 'absence ', 'of ', 'bugs']
>>> functools.reduce( (lambda x,y:x+y), L)
'Testing shows the presence, not the absence of bugs'
```

- Naravno, ovo može jednostavnije što smo koristili ranije:

```
>>> ''.join(L)
'Testing shows the presence, not the absence of bugs'
```

Ili korišćenjem operatora:

```
>>> import functools, operator
>>> functools.reduce(operator.add, L)
'Testing shows the presence, not the absence of bugs'
```

- Mala caka sa default argumentom ako je lista za redukciju prazna:

```
>>> import functools, operator
>>> functools.reduce(operator.add,['hm'])
>>> hm
```

FUNKCIONALNO PROGRAMIRANJE

- Lambda, primeri:

```
>>> def f (x): return x**2
...
>>> print (f(8))
64
>>>
>>> g = lambda x: x**2
>>>
>>> print (g(8))
64
>>> def make_incrementor (n): return lambda x: x + n
>>>
>>> f = make_incrementor(2)
>>> g = make_incrementor(6)
>>>
>>> print (f(42), g(42))
44 48
>>>
>>> print (make_incrementor(22)(33))
55
```

FUNKCIONALNO PROGRAMIRANJE

```
>>> import functools
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print (list(filter(lambda x: x % 3 == 0, foo)))
                                                    [18, 9, 24, 12, 27]
>>> print (list(map(lambda x: x * 2 + 10, foo)))
                                          [14, 46, 28, 54, 44, 58, 26, 34, 64]
>>> print (functools.reduce(lambda x, y: x + y, foo))
                                                                        139
>>> nums = list(range(2, 50))
>>> for i in range(2, 8):
...     nums = list(filter(lambda x: x == i or x % i, nums))
...
>>> print (nums)
                [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> sentence = 'It is raining cats and dogs'
>>> words = sentence.split(' ')
>>> print (words)
                              ['It', 'is', 'raining', 'cats', 'and', 'dogs']
>>> lengths = list(map(lambda word: len(word), words))
>>> print (lengths)
                                                    [2, 2, 7, 4, 3, 4]
```

FUNKCIONALNO PROGRAMIRANJE

- Pravljenje proizvoda 2 liste te sortiranje.

```
>>> list_1 = ['a','b','c']
>>> list_2 = [1,2]

>>> paired = list(itertools.product(list_1, list_2))
>>> paired
[('a', 1), ('a', 2), ('b', 1), ('b', 2), ('c', 1), ('c', 2)]

>>> wanted = sorted(paired, key=lambda x: x[1])
>>> wanted
[('a', 1), ('b', 1), ('c', 1), ('a', 2), ('b', 2), ('c', 2)]

>>> wanted = sorted(paired, key=lambda x: x[1], reverse=True)
>>> wanted
[('a', 2), ('b', 2), ('c', 2), ('a', 1), ('b', 1), ('c', 1)]
>>>
```

LISP, primitive

LISt Processing

PODACI - Osnovni tipovi

(izraz, atom, broj, simbol, lista, a-lista)

- IZRAZ
  - ATOM
    - BROJ
      - CELI npr. 123
      - REALNI npr. 123,456
      - KOMPLEKSNI npr. #C(5,6)
    - SIMBOL npr, popis-imena
  - LISTA
    - PRAZNA LISTA npr. ()* NIL
    - CONS LISTA npr, (a b)
    - A-LISTA npr, ((a 1) (b 2))
- Osnovne primitive

  (first, last, rest, reverse, length, append, list, cons, nthcdr, ...)

LISP, primitive

- Naziv primitive : FIRST, CAR
  Tip rezultata argumenta : lista
  Tip rezultata : atom
  Rezultat : prvi element liste
  Primer : (first '(a b c d))
  Rezultat : a
- Naziv primitive : SECOND, (THIRD, FOURTH, FIFTH, SIXTH, SEVENTH, EIGHTH, NINETH, TENTH)
  Tip rezultata argumenta : lista
  Tip rezultata : atom
  Rezultat : drugi (treći, četvrti, peti, šesti, sedmi, osmi, deveti, deseti) element liste
  Primer : (third '(a b c d))
  Rezultat : c
- Naziv primitive : LAST
  Tip rezultata argumenta : lista
  Tip rezultata : lista
  Rezultat : lista koju čini samo poslednji element
  Primer : (last '(a b c d))
  Rezultat : (d)

LISP, primitive

- Naziv primitive : REST, CDR
  Tip rezultata argumenta : lista
  Tip rezultata : lista
  Rezultat : lista bez prvog elementa
  Primer : (rest '(a b c d))
  Rezultat : (b c d)
- Naziv primitive : REVERSE
  Tip rezultata argumenta : lista
  Tip rezultata : lista
  Rezultat : lista s elementima u obrnutom redosledu
  Primer : (reverse '(a b c d))
  Rezultat : (d c b a)
- Naziv primitive : LENGTH
  Tip rezultata argumenta : lista
  Tip rezultata : broj
  Rezultat : broj elemenata u listi
  Primer : (length '(a b c d))
  Rezultat : 4

LISP, primitive

- Naziv primitive : APPEND
  Tip rezultata argumenta : liste
  Tip rezultata : lista
  Rezultat : jedinstvena lista s elementima iz svih ravedenih lista
  Primer : (append '(a b c d) '(x) '(y z))
  Rezultat : (a b c d x y z)
- Naziv primitive : LIST
  Tip rezultata argumenta : izraz
  Tip rezultata : lista
  Rezultat : lista svih navedenih izraza
  Primer : (list '(a b c d) 'x '(y z))
  Rezultat : ((a b c d) x (y z))
- Naziv primitive : CONS
  Tip rezultata argumenta : izraz lista
  Tip rezultata : lista
  Rezultat : lista s rezultatom izraza ispred ostalih elemenata liste
  Primer : (cons 'a '(b c d))
  Rezultat : (a b c d)

LISP, primitive

- Naziv primitive : NTHCDR
  Tip rezultata argumenta : *broj* lista
  Tip rezultata : lista
  Rezultat : lista bez prvih *broj* elemenata
  Primer : (nthcdr 2 '(a b c d))
  Rezultat : (c d)
- Naziv primitive : SETF
  Tip rezultata argumenta : {simbol izraz} . . .
  Tip rezultata : izraz
  Rezultat : SETF NE IZRAČUNAVA simbol, VEC SAMO izraz. Kao sporedni efekt se svakom simbolu trajno pridružuje pripadni izraz.
  Primer : (setf listaabcd '(a b c d))
  Rezultat : (a b c d)
- Naziv primitive : ASSOC
  Tip rezultata argumenta : ključ a-lista
  Tip rezultata : prvi element a-liste koji odgovara ključu
  Rezultat : a-lista ciji je prvi element jednak ključu
  Primer : (assoc 'b '((a 1) (b 2)))
  Rezultat : (b 2)

LISP, primitive

- Naziv primitive : MAX, MIN, ABS, +, -, *, /, REM, SQRT

(max no1 no2 . . .)	daje maksimalnu vrednost od navedenih
(min no1 no2 . . .)	daje minimalnu vrednost od navedenih
(abs no1)	daje apsolutnu vrednost navedenog broja
(+ no1 . . .)	daje sumu svih navedenih brojeva
(- no1 no2 . . .)	daje razliku prvog i sume ostalih brojeva
(* no1 no2 . . .)	daje proizvod navedenih brojeva
(/ no1 no2 . . .)	daje količnik prvog sa proizvodom ostalih brojeva
(rem no1 no2)	daje ostatak deljenja prvog i drugog broja
(sqrt no1)	daje vrednost drugog korena datog broja

Primeri:

```
(rest '( c ))                    NIL
(first ())                       NIL
(rest ())                        NIL
(first '((a b) '(c d)))          (a b)
(first '(rest (a b c)))          rest
(first (rest '(a b c)))          b
(first (rest (a b c)))           ERROR
```

LISP, primitive

Primeri:

```
(first '(p h w))                                        p
(rest '(b k p h))                                       (k p h)
(first '((a b) (c d)))                                  (a b)
(rest '((a b) (c d)))                                   ((c d))
(first(rest '((a b) (c d))))                            (c d)
(rest(first '((a b) (c d))))                            (b)
(rest(first(rest '((a b) (c d)))))                      (d)
(first(rest(first '((a b) (c d)))))                     b

(first (rest (first (rest '((a b) (c d) (e f))))))      d
(first (first (rest (rest '((a b) (c d) (e f))))))      e
(first (first (rest '(rest ((a b) (c d) (e f))))))      (a b)
(first (first '(rest (rest '((a b) (c d) (e f))))))     ERROR
(first '(first (rest (rest '((a b) (c d) (e f))))))     first
'(first (first (rest (rest '((a b) (c d) (e f))))))     sve
```

LISP, primitive

Primeri:

Koristeći isključivo FIRST i REST dopisati šta je potrebno da rezultat bude simbol kruska:

```
'(jabuka narandja kruška grozdje)
'((jabuka narandja) (kruška grozdje))
'(((jabuka) (narandja) (kruška) (grozdje)))
'(jabuka (narandja) ((kruška)) (((grozdje))))
'((((jabuka))) ((narandja)) (kruška) grozdje)
'((((jabuka) narandja) kruška) grozdje)
```

Napisani su rezultati izračunavanja zadanih nizova formi:

`(setf ab-list '(a b))`	`(a b)`
`ab-list`	`(a b)`
`'ab-list`	`ab-list`
`(first 'ab-list )`	`ERROR`
`(first ab-list)`	`a`
`(rest 'ab-list)`	`ERROR`
`(rest ab-list)`	`(b)`

LISP, primitive

```
(setf ab-list '(a b)
      xy-list '(x y))                    (x y)
ab-list                                  (a b)
xy-list                                  (x y)
t                                        T
nil                                      NIL
()                                       NIL
```

Prikazani su rezultati izračunavanja nizova formi:

```
(setf novi-pocetak 'a stara-lista '(b c))    (b c)
(cons novi-pocetak stara-lista)              (a b c)
(first (cons novi-pocetak stara-lista))      a
(rest (cons novi-pocetak stara-lista))       (b c)

(setf ab-list '(a b)
      xy-list '(x y))                        (x y)
(append ab-list xy-list)                     (a b x y)
(append ab-list xy-list ab-list)             (a b x y a b)
(append ab-list '() xy-list '())             (a b x y)
(append 'ab-list xy-list)                    ERROR
(append '((a) (b)) '((c) (d)))               ((a) (b) (c) (d))
```

LISP, primitive

Dati su rezultati izračunavanja datih nizova formi:

```
(setf pocetak 'a sredina 'b kraj 'c)                    c
(list pocetak sredina kraj)                             (a b c)
(pocetak sredina kraj)                                  ERROR

'(početak sredina kraj)                                 sve se prepise
(setf ab-list '(a b))                                   (a b)
(list ab-list ab-list)                                  ((a b) (a b))
(list ab-list ab-list ab-list)                          ((a b) (a b) (a b))
(list 'ab-list ab-list)                                 (ab-list (a b))

(setf ab-list '(a b) cd-list '(c d))                    (c d)
(append ab-list cd-list)                                (a b c d)
(list ab-list cd-list)                                  ((a b) (c d))
(cons ab-list cd-list)                                  ((a b) c d)
(append ab-list ab-list)                                (a b a b)
(list ab-list ab-list)                                  ((a b) (a b))
(cons ab-list ab-list)                                  ((a b) a b)
(append 'ab-list ab-list)                               ERROR
(list 'ab-list ab-list)                                 (ab-list (a b))
(cons 'ab-list ab-list)                                 (ab-list a b)
```

LISP, primitive

```
(append '(a b c) '())                                   (a b c)
(list '(a b c) '())                                     ((a b c) NIL)
(cons '(a b c) '())                                     ((a b c))
```

Dati su rezultati izračunavanja niza formi:

```
(setf alat (list 'cekic 'odvijac))                      (cekic odvijac)
(cons 'klesta alat)                                     (klesta cekic odvijac)
alat                                                    (cekic odvijac)
(setf alat (cons 'klesta alat))                         (klesta cekic odvijac)
alat                                                    (klesta cekic odvijac)
(append '(testera turpija) alat)                        (testera trupija klesta cekic odvijac)
alat                                                    (klesta cekic odvijac)
(setf alat (append '(testera turpija) alat))            (testera trupija klesta cekic odvijac)
alat                                                    (testera trupija klesta cekic odvijac)

(cons (first nil) (rest nil))                           (NIL)
(setf novi-pocetak 'a lista-koja-se-menja '(b c))       (b c)
(setf lista-koja-se-menja (cons novi-pocetak lista-koja-se-menja))
                                                        (a b c)
novi-pocetak                                            a
lista-koja-se-menja                                     (a b c)
```

LISP, primitive

Dati su rezultati izračunavanja datih nizova formi:

```
(setf abc-list '(a b c))                                  (a b c)
(rest abc-list)                                           (b c)
(nthcdr 1 abc-list)                                       (b c)
(nthcdr 2 abc-list)                                       (c)
(nthcdr 4 abc-list)                                       NIL
(nthcdr 2 'abc-list)                                      ERROR

(setf abc-list '(a b c) ab-cd-list '((a b) (c d)))        ((a b) (c d))
(nthcdr 2 abc-list)                                       (c)
(last abc-list)                                           (c)
(first (last abc-list))                                   c
(last ab-cd-list)                                         ((c d))
(last 'abc-list)                                          ERROR
```

Dati su rezultati izračunavanja datih nizova formi:

```
(setf ab-list '(a b) ab-cd-list '((a b) (c d)))           ((a b) (c d))
(length ab-list)                                          2
(length ab-cd-list)                                       2
(length (append ab-list ab-list))                         4
(reverse ab-list)                                         (b a)
```

LISP, primitive

```
(reverse ab-cd-list)                          ((c d) (a b))
(reverse (append ab-list ab-list))            (b a b a)

(setf l '(a b c) m '(x y z))                  (x y z)
(list l m)                                    ((a b c) (x y z))
(list 'l 'm)                                  (l m)
(list 'l m)                                   (l (x y z))
(list l 'm)                                   ((a b c) m)
(append l m)                                  (a b c x y z)
(list (list l m)(list l m))    (((a b c) (x y z)) ((a b c) (x y z)))
```

Dati su rezultati izračunavanja zadanih formi:

```
(length '(Platon Sokrat Aristotel))           3
(length '((Platon) (Sokrat) (Aristotel)))     3
(length '((Platon Sokrat Aristotel)))         1
(reverse '(Platon Sokrat Aristotel))          (Aristotel Sokrat Platon)
(reverse '((Platon) (Sokrat) (Aristotel)))    ((Aristotel) (Sokrat) (Platon))
(reverse '((Platon Sokrat Aristotel)))        ((Platon Sokrat Aristotel))
```

Dati su rezultati izračunavanja zadanih formi:

```
(length '((auto sevrolet) (pice cocacola) (hrana hamburger)))   3
```

LISP, procedure

```
(reverse '((auto sevrolet) (pice cocacola) (hrana hamburger)))
                ((hrana hamburger) (pice cocacola) (auto sevrolet))
(append '((auto sevrolet) (pice cocacola)))
                ((auto sevrolet) (pice cocacola))
```

- Dati su rezultati izračunavanja datih nizova formi:

```
(setf Mirko '((visina 170) (tezina 70)))        ((visina 170) (tezina 70))
(assoc 'tezina Mirko)                           (tezina 70)
```

DEFINISANJE PROCEDURA (defun)

- Opšti oblik specijalne forme DEFUN za definisanje procedura

```
(DEFUN <ime procedure) (<parametar1> ...)
  <formal>
  ...
)
```

- Dva su pristupa:
  - Progressive envelopment: odrediti potrebne primitive kroz niz eksperimenata za računarom
  - Comment translation : definisati šta treba napraviti koristeći se prirodnim jezikom, a zatim to prevesti u LISP

LISP, procedure

Primeri:

Napisati proceduru koja ce vraćati listu koju čine samo prvi i poslednji element ulazne liste:

```
(defun krajevi (lista)
   (cons (first lista) (last lista)) )
```

Napisati proceduru ZAMENI, koja ulaznu dvo-eIementnu listu vraća u obrnutom redosledu. Pri tome ne koristiti REVERSE.

```
(defun zameni (lista)
   (list (first (last lista)) (first lista)))
```

Napisati proceduru CONSTRUCT koja radi isto što i primitiva CONS.

```
(defun construct (izraz lista)
   (append (list izraz) lista) )
```

Napisati proceduru NA-LEVO, koja vraća ulaznu listu, ali tako da je prvi element postao poslednji.

```
(defun na-levo (lista)
   (append (rest lista) (list (first lista)) ))
```

LISP, procedure

Napisati proceduru NOLAST, koja vraća ulaznu listu od koje je odbačen poslednji eIemenat.

```
(defun nolast (lista)
   (reverse (rest (reverse lista))) )
```

Napisati proceduru NOLASTN, koja vraća ulaznu listu od koje je odbačeno posljednjih n elemenata, pri cemu je n drugi ulazni parametar.

```
(defun nolastn (lista n)
   (reverse (nthcdr n (reverse lista))) )
```

Napisati proceduru NA-DESNO, koja okreće ulaznu listu u obrnutom smeru od procedure NA-LEVO, dakle, tako da poslednji element postane prvi.

```
(defun na-desno (lista)
   (append (last lista) (reverse (rest (reverse lista))) ) )
```

Napisati proceduru koja vraća listu koja je dvostruko veća od ulazne.

```
(defun dvostruka (lista)
   (append lista lista) )
```

LISP, procedure, setf, let, let*

Napisati proceduru SREDINA koja uklanja prvi i poslednji element iz liste.

```
(defun sredina (lista)
   (reverse (rest (reverse (rest lista)))) )
```

RAZLIKE IZMEDJU SETF, LET i LET*

- Opšti oblik specijalne forme SETF:

```
(SETF simbol1 izraz1 ... )
```

Opis: Rezultat izraz1 pridružuje se simbolu simbol1, itd . . .
Primer:

```
(setf ab '(a b) cd '(c d) )
```

- Opšti oblik specijalne forme LET:

```
(LET ( (simbol1 izraz1) • • • (simbolM izrazM))
 forma1
 • • •
)
```

Opis: izraz1 ... izrazM se izračunavaju "paralelno" tj, svi se izračunavaju pre nego se njihovi rezultati pridruže simbolima simbol1 ... simbolM.

LISP, setf, let, let*

Rezultat LETa je rezultat poslednje forme.

Primer :

```
(let (   (prvi      (first lista))
         (poslednji (last  lista)) )
     (cons prvi poslednji)
)
```

- Opšti oblik specijalne forme LET*:

```
(LET* ( (simbol1 izrazl) • • • )
  forma1
  • • •
)
```

Opis: izraz1 ... izrazM se izračunavaju "sekvencijalno", tj. kako se pojedini izraz izračuna, njegov se rezultat pridruži datom simbolu.

Rezultat LETa je rezultat posljednje forme.

- Analizirati sledeći kod i pripadni izlaz:

```
(setf x 'izvan)                          izvan
(let ( (x 'unutar )
       (y       x ) )
  (list x y )
)                                        (unutar izvan)
```

LISP, setf, let, let*

```
(setf x 'izvan )                                    izvan
(let* ((x 'unutar)
       (y x ) )
  (list x y ) )                                     (unutar unutar)
```

- Razlog za ovakve rezultate:
  - LET pridružuje sve rezultate izraza istovremeno.
    - Pri pridruživanju y<-x simbol x ima vanjsku vrednost ' izvan'.
  - LET* pridružuje rezultate izraza sekvencijalno
    - Pri pridruživanju y<-x simbol x već ima unutrašnju vrednost 'unutar'.

•Napisati proceduru koja ce vraćati listu koju čine samo prvi i poslednji element ulazne liste. Koristiti primitivu LET.

```
(defun prvpos (lista)
   (let (
          ( x  (first lista) )
          ( y  (last  lista) )
        )
        (cons x y )
   )
)
```