# PROGRAMIRANJE U INTEGRISANIM TEHNOLOGIJAMA

Oznaka predmeta: PIT

Predavanje broj: 06

Nastavna jedinica: numpy,

Nastavne teme:

Biblioteka numpy. Korišćenje Python-a za numeričke metode. Greške zbog IEEE754. Boothov algoritam. Bisekcija. Monte Karlo.

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

Literatura:

Steven Lott: "Functional Python Programming", Packt Publishing, 2015.

# Bibilioteka numpy

- Instalacija modula: `pip install numpy`
  provera: `import numpy`
- Modula numpy namenjen je korišćenju python-a za numerička izračunavanja.
- Klasa `numpy.array` (`numpy.ndarray`) ima sledeće često korišćene atribute:
  - `ndarray.ndim` broj dimenzija polja (`rank`).
  - `ndarray.shape` lista veličine polja za svaku dimenziju
    npr. za matricu 3x4 `shape = (3,4).`
  - `ndarray.size` broj elemenata u polju.
  - `ndarray.dtype` tip elemenata polja (`numpy.int16`, `numpy.int32`, `numpy.float64` ...).
  - `ndarray.itemsize` veličina elementa polja u bajtovima (za `numpy.float64` je 8, za `numy.complex32` je 4),
    može i `ndarray.dtype.itemsize`.
  - `ndarray.data` bafer koji sadrži elemente polja, ono što se najčešće koristi je indeksni pristup elementu.

# Bibilioteka numpy

- Primer za atribute klase numpy.ndarray

```
import numpy as np

a = np.arange(15).reshape(3, 5)

print(a)          #  [[ 0,  1,  2,  3,  4],
                  #   [ 5,  6,  7,  8,  9],
                  #   [10, 11, 12, 13, 14]]

print(a.shape)   # (3, 5)
print(a.ndim)    # 2
print(a.dtype.name)      # int64 ili int32
print(a.itemsize)        # 8 ili 4
print(a.size)            # 15
print(type(a))           # <type 'numpy.ndarray'>

b = np.array([6, 7, 8])
print(b)                 # [6, 7, 8]
print(type(b))           # <type 'numpy.ndarray'>
```

# Bibilioteka numpy

```
import numpy as np
a = np.array([2,3,4])           # kreiranje 1D polja integer-a
print(a)                        # [2, 3, 4]
print(a.dtype)                  # int32 ili int64
b = np.array([1.2, 3.5, 5.1])             # kreiranje 1D polja float-a
print(b.dtype)                            # float64
a = np.array([(1.5,2,3), (4,5,6)])    # kreiranje 2D polja
print(a)        # [[ 1.5  2.   3. ]
                #  [ 4.   5.   6. ]]
a = np.array( [ [1,2], [3,4] ], dtype=complex ) # kreiranje complex
print(a)        # [[ 1.+0.j  2.+0.j]
                #  [ 3.+0.j  4.+0.j]]
a = np.zeros((3,4)) # kreiranje 2D nula matrice
print(a)        # [[ 0.  0.  0.  0.]
                #  [ 0.  0.  0.  0.]
                #  [ 0.  0.  0.  0.]]
a = np.ones( (2,3,4), dtype=np.int16 ) # kreiranje 3D int16 matrice
print(a)        # [[[1 1 1 1]
                #   [1 1 1 1]
                #   [1 1 1 1]]
                #  [[1 1 1 1]
                #   [1 1 1 1]
                #   [1 1 1 1]]]
```

# Bibilioteka numpy

```
a = np.empty( (2,3) )
print(a)       # u opstem slucaju ne zna se inicijalna vrednost

a = np.arange( 10, 30, 5 ) # kreiranje polja od [10,30) u koracima od 5
print(a)       # [10 15 20 25]

a = np.arange( 0, 2, 0.3 )
print(a)       # [ 0.   0.3  0.6  0.9  1.2  1.5  1.8]

a = np.linspace( 0, 2, 9 ) # kreiranje polja od 9 elemenata od 0 do 2
print(a)       # [ 0.    0.25  0.5   0.75  1.    1.25  1.5   1.75  2.  ]

x = np.linspace( 0, 2*math.pi, 100 ) # polje 100 radijana od o do 2PI
f = np.sin(x)                        # novo polje pripadnih sisnusa
for i in range(20):
    print('{:+6.4f}'.format(f[i]),' '*3,end='')
    if((i+1)%5==0):
        print()
# +0.0000    +0.0634    +0.1266    +0.1893    +0.2511
# +0.3120    +0.3717    +0.4298    +0.4862    +0.5406
# +0.5929    +0.6428    +0.6901    +0.7346    +0.7761
# +0.8146    +0.8497    +0.8815    +0.9096    +0.9341
```

# Bibilioteka numpy

```
a = np.arange(6); print(a)    # kreiranje 6 elem. skupa N0 [0 1 2 3 4 5]
a = np.arange(12).reshape(4,3) # kreiranje 4x3mat. elem. skupa N0
print(a)     # [[ 0  1  2]
             #  [ 3  4  5]
             #  [ 6  7  8]
             #  [ 9 10 11]]
a = np.arange(24).reshape(2,3,4) # kreiranje 2x4x3 mat. elem. skupa N0
print(a)     # [[[ 0  1  2  3]
             #   [ 4  5  6  7]
             #   [ 8  9 10 11]]
             #  [[12 13 14 15]
             #   [16 17 18 19]
             #   [20 21 22 23]]]
print(np.arange(10000))
# izlaz je doslovno
# [   0    1    2 ..., 9997 9998 9999]
print(np.arange(10000).reshape(100,100))
# izlaz je doslovno
# [[   0    1    2 ...,   97   98   99]
#  [ 100  101  102 ...,  197  198  199]
#  [ 200  201  202 ...,  297  298  299]
#  ...,
#  [9700 9701 9702 ..., 9797 9798 9799]
#  [9800 9801 9802 ..., 9897 9898 9899]
#  [9900 9901 9902 ..., 9997 9998 9999]]
```

# Bibilioteka numpy

```
np.set_printoptions(threshold=3000) # koliko se ispisuje elemenata
print(np.arange(2000)) # ispisace svih 2000 elemenata

# set_printoptions(
#   precision=8,
#    threshold=1000,
#     edgeitems=3,
#   linewidth=75,
#    suppress=False,
#     nanstr='nan',
#   infstr='inf',
#    formatter=None)

np.set_printoptions(precision=4)
print(np.array([1.123456789]))    # [ 1.1235]

np.set_printoptions(threshold=5)
print(np.arange(10))              # [0 1 2 ..., 7 8 9]

eps = np.finfo(float).eps
print(eps)                        # 2.22044604925e-16
```

# Bibilioteka numpy

```
eps = np.finfo(float).eps       # 2.22044604925e-16
x = np.arange(4.)
print(x)          # [ 0.  1.  2.  3.]
y = x**2 - (x + eps)**2
print(y)          #[ -4.9304e-32  -4.4409e-16   0.0000e+00   0.0000e+00]
np.set_printoptions(suppress=True)
print(y)          # [-0. -0.  0.  0.] prihvati da su nule

np.set_printoptions(formatter={'all':lambda x: 'int: '+str(-x)})
x = np.arange(3)
print(x)   # [int: 0 int: -1 int: -2] formatirani ispis
np.set_printoptions() # reset opcija stampe
print(x)   # [0 1 2]

Osnovne operacije
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
c = a-b
print(c)    # [20 29 38 47]
c = b**2
print(c)    # [0 1 4 9]
```

# Bibilioteka numpy

```
c= 10*np.sin(a)
print(c)    # [ 9.1295 -9.8803  7.4511 -2.6237]
print(a<35) # [ True  True False False]


A = np.array( [[1,1], [0,1]] )
B = np.array( [[2,0], [3,4]] )
print(A*B)   # prizvod isto-pozicioniranih elemenata
             # [[2 0]
             #  [0 4]]
C= A.dot(B)
print(C)   # proizvod matrica
           # [[5 4]
           #  [3 4]]
C = np.dot(A,B)
print(C)   # kao i prethodno, proizvod matrica

a = np.ones((2,3), dtype=int) # matrica 2x3 sa svim elementima = 1
a*=3
print(a)   # [[3 3 3]
           #  [3 3 3]]
b = np.random.random((2,3))
b+=a; print(b)   # [[ 3.6652  3.8975  3.1435]
                 #  [ 3.7595  3.7106  3.5617]]
```

# Bibilioteka numpy

```
c = a+b
print(c)   # [[ 6.5508  6.8561  6.2206]
           #  [ 6.235   6.5875  6.6058]]
print(c.dtype.name)   # float64
d = np.exp(c*1j)
print(d)   # [[ 0.9039+0.4278j  0.9841-0.1774j  0.9830+0.1836j]
           #  [ 0.9999-0.0117j  0.9844+0.1761j  0.9977-0.0672j]]
print(d.dtype.name)   # complex128

# unarne operacije
a = np.random.random((2,3))
print(a)   # [[ 0.5059  0.0976  0.5169]
           #  [ 0.0018  0.7668  0.4717]]
print(a.sum())  # 2.36074163537  suma elemenata
print(a.min())  # 0.001836162017 minimalni element
print(a.max())  # 0.766837666526 maksimnalni element

b = np.arange(12).reshape(3,4)
print(b)                # [[ 0  1  2  3]  [ 4  5  6  7]  [ 8  9 10 11]]
print(b.sum(axis=0))    # [12 15 18 21] sabiranje po kolonama
print(b.min(axis=1))    # [0 4 8]       min element po redovima
print(b.cumsum(axis=1))# [[ 0  1  3  6]  [ 4  9 15 22]  [ 8 17 27 38]]
                        # kumulativno sumarni elemenati po redovima
```

# Bibilioteka numpy

```
# univerzalne funkcije
B = np.arange(3)
print(np.exp(B))   # [ 1.      2.7183  7.3891]
print(np.sqrt(B))  # [ 0.      1.      1.4142]
C = np.array([2., -1., 4.])
print(np.add(B,C))   # [ 2.  0.  6.]


# indeksiranje, odsecanje, iteracija
a = np.arange(10)**3
# [  0,   1,   8,  27,  64, 125, 216, 343, 512, 729]
print(a[2])   # 8
print(a[2:5]) # [ 8 27 64]
a[:6:2] = -1000 # ili a[0:6:2] = -1000
                # od 0-tog do 6-tog svaki drugi
                # element postaviti na -1000
np.set_printoptions(threshold=100)
print(a) # [-1000     1 -1000    27 -1000   125   216   343   512   729]
print(a[::-1]) # prikazi unazad
         # [  729   512   343   216   125 -1000    27 -1000     1 -1000]
for i in a:
    print(i**(1/3.)) # ako probate bice Runtime Warning (invalid value)
    # nan   1.0   nan   3.0   nan   5.0   6.0   7.0   8.0   9.0
```

# Bibilioteka numpy

```
# indeksiranje multidimenzionalnih polja
def f(x,y):
    return 10*x+y
b = np.fromfunction(f,(5,4),dtype=int) # popunjavanje preko funkcije
print(b)   # [[ 0  1  2  3]
           #  [10 11 12 13]
           #  [20 21 22 23]
           #  [30 31 32 33]
           #  [40 41 42 43]]
print(b[2,3])    # 23
print(b[0:5, 1]) # [ 1 11 21 31 41]
print(b[ : ,1])  # [ 1 11 21 31 41]
print(b[-1])     # kao b[-1,:]
                 # [40 41 42 43]
print(b[-1,...]) # [40 41 42 43]

c = np.array( [[[  0,  1,  2],
                [ 10, 12, 13]],
               [[100,101,102],
                [110,112,113]]])
print(c.shape) # (2, 2, 3)
```

# Bibilioteka numpy

```
A = np.arange(4).reshape(2,2)
for row in A:
    print(row)
# [0 1]
# [2 3]
for el in A.flat: # kretanje po matrici kao po 1d
    print(el)
# 0
# 1
# 2
# 3

a = np.floor(10*np.random.random((3,4))) # mat. 3x4 slucajnih [0,9]
print(a)  # [[ 6.  0.  0.  0.]
          #  [ 5.  9.  7.  4.]
          #  [ 8.  4.  2.  7.]]
print(a.shape) # (3, 4) prikazuje velicinu a
print(a.ravel())          # ne menja a vec vraca flat vector od mat. a
print(a.reshape(6,2))     # ne menja a vec vraca matricu 6x2
print(a.T)                # ne menja a vec vraca transponovanu matricu a
                          # a.T.shape = (4, 3) a.shape = (3 4)
a.resize(2,6)           # menja a koje postaje matrica 2x6
print(a.reshape(6,-1))# izracunace dimenziju
```

# Bibilioteka numpy

```
# stek npr. dva polja
a = np.arange(4)
b = np.arange(4)
c = np.vstack(  (a,b)  ) # vertikalni stek, kreira 2x4
print(c)   # [[0 1 2 3]    od a
           #  [0 1 2 3]]   od b
c = np.hstack(  (a,b)  ) # kreira 1x8
print(c)   # [0 1 2 3 0 1 2 3] od a b

# drugi primer
a = np.arange(4).reshape(2,2) # kreiraj 1x4 vrati kao 2x2
b = np.arange(4).reshape(2,2) # kreiraj 1x4 vrati kao 2x2
c = np.vstack(  (a,b)  ) # kreira 4x2
print(c)   # [[0 1]  a
           #  [2 3]  a
           #  [0 1]  b
           #  [2 3]] b
c = np.hstack(  (a,b)  ) # kreira 2x4
print(c)   # [[0 1 0 1]
           #  [2 3 2 3]]
#probajte za 3x3
```

# Bibilioteka numpy

```
from numpy import newaxis
a = np.array([4.,2.])
b = np.array([2.,8.])
print(a[:,newaxis]) # [[ 4.]
                    #  [ 2.]]
print(np.column_stack((a[:,newaxis],b[:,newaxis])))
# [[ 4.  2.]
#  [ 2.  8.]]
print(np.vstack((a[:,newaxis],b[:,newaxis])))
# [[ 4.]
   [ 2.]
   [ 2.]
   [ 8.]]
#split
a = np.arange(18).reshape(2,9)
print(a) # [[ 0  1  2  3  4  5  6  7  8]
         #  [ 9 10 11 12 13 14 15 16 17]]
print(np.hsplit(a,3))  # podeli u 3 polja
# [array([[ 0,  1,  2],
#         [ 9, 10, 11]]), array([[ 3,  4,  5],
#         [12, 13, 14]]), array([[ 6,  7,  8],
#         [15, 16, 17]])]
# print(np.vsplit(a,2))
```

# Bibilioteka numpy

```
#kopiranja
a = np.arange(4)
b = a                    # isti objekat
print(id(a)," ",id(b)) # isti objekat


# view
a = np.arange(12).reshape(3,4)   # mat 3x4 elementi od 0..11
c = a.view()                     # drugi pogled na podatke
print(c is a)                 # False

print(c.flags.owndata) # False
c.shape = 2,6
c[0,4]= 100
print(a[1,0])  # 100, promenjeno preko c
#dobijanje view-a preko slice-a
s = a[:,1:3] # prva i druga kolona a
s[:] = 200
print(a)
# [[  0 200 200   3]
#  [100 200 200   7]
#  [  8 200 200  11]]
```

# Bibilioteka numpy

```
# potpuna kopija
a = np.arange(3)
b = a.copy()
b[0]=100
print(a[0])  # 0
# kombinovanje izracunavanja i postavljanja
a = np.arange(12)**2
i = np.array( [ 1,1,3,8,5 ] ) # za indekse
print(a[i])                   # [ 1  1  9 64 25]
j = np.array([[3,4], [9,7]])  # za indekse
print(a[j])                   # prema indeksnom polju 2x2
# [[ 9 16]
#  [81 49]]
a = np.arange(12).reshape(3,4)
print(a) # [[ 0  1  2  3]
         #  [ 4  5  6  7]
         #  [ 8  9 10 11]]
i = np.array([ [0,1], [1,2] ])
j = np.array([ [2,1], [3,3] ]) # i i j moraju biti istih dimenzija
print(a[i,j]) # lista indeksa i i j: a[0,2] a[1,1]...
# [[ 2  5]
#  [ 7 11]]
print(a[i,2])  # [[ 2  6]    analogno prethodnom
               #  [ 6 10]]
```

# Bibilioteka numpy

```
print(a[:,j]) # analogno prethodnom, prvi indeks iz a tj. 0,1,2
# [[[ 2  1]   # drugi iz j tj. [2,1],[3,3]
#   [ 3  3]]
#  [[ 6  5]
#   [ 7  7]]
#  [[10  9]
#   [11 11]]]

# linspace
time = np.linspace(20, 145, 5) # linearna podela [20,145] na 5 clanova
print(time)
# [  20.     51.25   82.5   113.75  145.  ]

data = np.sin(np.arange(20)).reshape(5,4)
print(data)
# matrica 5x4 sin-usa za 0..19
# [[ 0.      0.8415  0.9093  0.1411]
#  [-0.7568 -0.9589 -0.2794  0.657 ]
#  [ 0.9894  0.4121 -0.544  -1.    ]
#  [-0.5366  0.4202  0.9906  0.6503]
#  [-0.2879 -0.9614 -0.751   0.1499]]
ind = data.argmax(axis=0) # indeks maksimalnog elementa u datoj koloni
print(ind) # [2 0 3 1]
```

# Bibilioteka numpy

```
# indeksiranje
a = np.arange(5)    # [0, 1, 2, 3, 4]
a[[1,3,4]] = 0      # [0, 0, 2, 0, 0]
a[[0,0,2]]=[1,2,3] # [2, 0, 3, 0, 0], upisuje redom po indeksima
                    # na nulti stavi 1, na nulti stavi 2, na drugi 3
a[[0,0,2]]+=1       # opet za dati indeks racuna samo poslednje
print(a)            # [3, 0, 4, 0, 0]

# indeksiranje Bulovim poljima
a = np.arange(12).reshape(3,4)
b = a>4
print(b)
# [[False False False False]
#  [False  True  True  True]
#  [ True  True  True  True]]
print(a[b]) # samo selektovani elementi
# [ 5  6  7  8  9 10 11]
a[b]=0 # dodela selektovanim elemenatima
print(a)
# [[0 1 2 3]
#  [4 0 0 0]
#  [0 0 0 0]]
```

# Bibilioteka numpy

```
# ix_() funkcija za kombinovanje vektora
a = np.array([2,3,4,5])
b = np.array([8,5,4])
c = np.array([5,4,6,8,3])

ax,bx,cx = np.ix_(a,b,c)

print(ax)
# [[[2]],
#  [[3]],
#  [[4]],
#  [[5]]])

print(bx)
# [[[8],
#   [5],
#   [4]]])

print(cx)
# [[[5, 4, 6, 8, 3]]]

print(ax.shape, bx.shape, cx.shape)
# ((4, 1, 1), (1, 3, 1), (1, 1, 5))
```

# Bibilioteka numpy

```
result = ax+bx*cx
print(result)
# [[[42, 34, 50, 66, 26],
#   [27, 22, 32, 42, 17],
#   [22, 18, 26, 34, 14]],
#  [[43, 35, 51, 67, 27],
#   [28, 23, 33, 43, 18],
#   [23, 19, 27, 35, 15]],
#  [[44, 36, 52, 68, 28],
#   [29, 24, 34, 44, 19],
#   [24, 20, 28, 36, 16]],
#  [[45, 37, 53, 69, 29],
#   [30, 25, 35, 45, 20],
#   [25, 21, 29, 37, 17]]])
print(result[3,2,4])  # 17
print(a[3]+b[2]*c[4]) # 17

# linearna algebra
a = np.array([[1.0, 2.0],
              [3.0, 4.0]])
print(a)
# [[ 1.  2.]
#  [ 3.  4.]]
```

# Bibilioteka numpy

```
b = a.transpose() # transponuje a
print(a)          # ne menja a
# [[ 1.  2.]
#  [ 3.  4.]]
print(b)
# [[ 1.  3.]
#  [ 2.  4.]]
b= np.linalg.inv(a) # inverzna od a
print(a)            # ne menja a
# # [[ 1.  2.]
# #  [ 3.  4.]]
print(b)
# # [[-2.   1. ]
# #  [ 1.5 -0.5]]
print(np.eye(2)) # jedinicna matrica
# # [[ 1.  0.]
# #  [ 0.  1.]]
c=np.array([[10, 20],
            [30, 40]])
rind,cind=np.where(c) # matrice indeksa [0,0,1,1]  [0,1,0,1]
print(np.dot(a,c))    # proizvod matrica
# [[  70.  100.]
#  [ 150.  220.]]
```

# Bibilioteka numpy

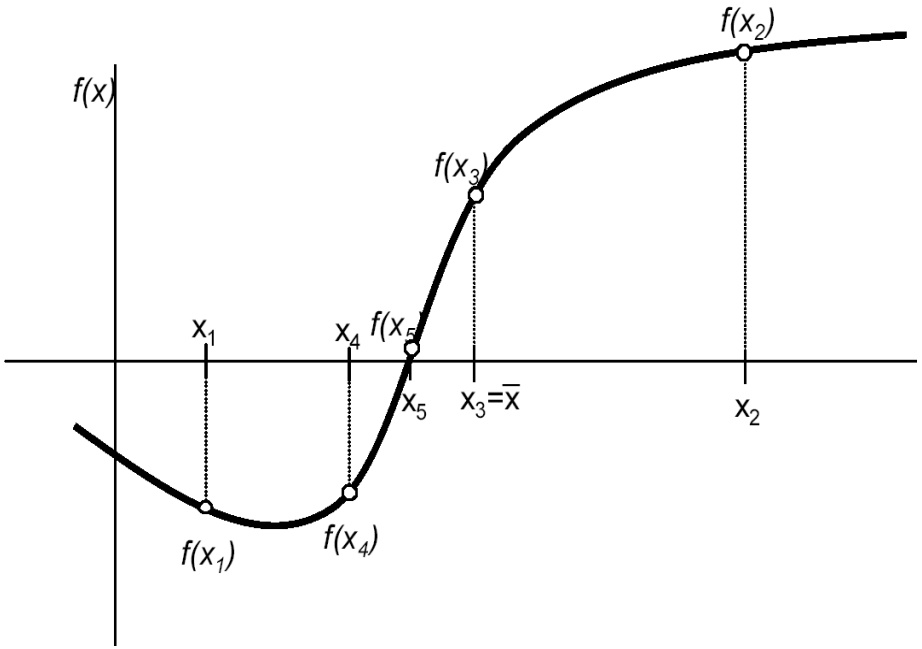
```
a = np.arange(4)+1
a.resize(2,2)
b = np.linalg.det(a) # determinanta b=-2


# neka je system linearnih jednacina
# 1x+2y=5
# 3x+4y=7
# trazi se resenje navedenog sistema linearnih jednacina
a = np.array([[1.0, 2.0],
              [3.0, 4.0]])
y = np.array([[5.],
              [7.]])
print(np.linalg.solve(a, y)) # resenje sistema
                             # 1x+2y=5
                             # 3x+4y=7
# [[-3.]
#  [ 4.]]
```

# Numeričke metode

- Osnovna prednost numeričkih postupaka je u tome da se rešenje može dobiti čak i u slučajevima kada nije moguće pronaći analitičko rešenje.
- Rešenje koje daje numerički postupak je uvek **aproksimacija**, ali uglavnom po volji tačna.
- Kategorije problema koji se rešavaju:
  - Traženje rešenja nelinearnih jednačina
  - Različite vrste interpolacija i ekstrapolacija
  - Derivacija (izvod) bilo kog reda čak i kada je funkcija zadana tabelarno
  - Integracija bilo koje funkcije čak i kada je ona zadana tabelarno
  - Rešavanje diferencijalnih jednačina gde su dani početni uslovi za varijable
  - Aproksimacije tabelarnih podataka funkcijama ...

# Primena bisekcije

Traži se koren jednačine $f(x) = 0$ u intervalu (x1, x2) gde je $f(x1) \cdot f(x2) < 0$ .

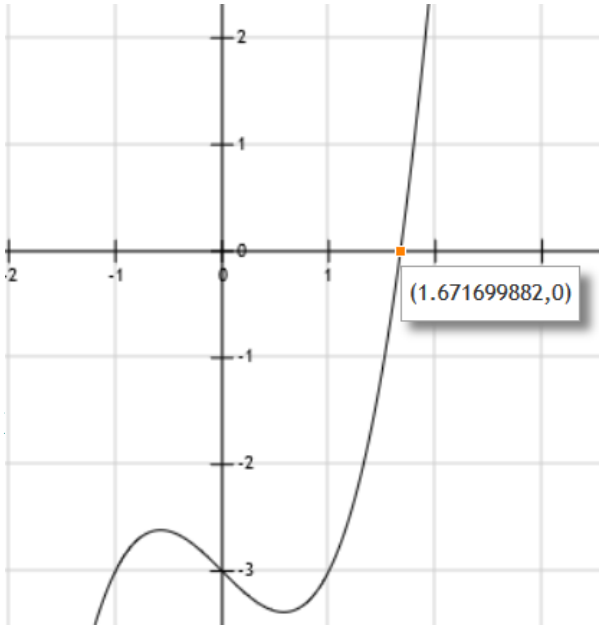



Interval se deli na pola x= (x1 + x2) / 2

| | | |
|---|---|---|
| Ako je | \| x2 - x1 \| ≤ ε | tada je x aproksimacija traženog korena. |
| Ako je | f( x )=0 | to je traženi koren |
| Inače: | f(x1) · f( x ) < 0 | i tada je novi interval ( x1, x ) |
| | | inače je novi interval ( x , x2 ) |

# Rešenje u Python-u

```
import math
def f(x): return x**3-x-3
iter=0 # brojac iteracija
maxiter=20 # maksimalni broj iteracija
eps=1.e-10 # eps je dopusteno odstupanje
print("\nunesite granice intervala ")
x1=float(input("leva  granica = "))
x2=float(input("desna granice = "))
print('Br.it.',' '*6,'x1',' '*12,'x2',' '*12,'
while ( abs(x2-x1) > eps  and  iter<=maxiter )
    xs = (x1 + x2) / 2.0
    if ( f(x1) * f(xs) < 0 ): x2 = xs
    else:                     x1 = xs
    print('{:>02d}  {:>14.10f}  {:>14.10f}  {:14.10f}
{:>14.10f}'.format(iter,x1, x2, xs,f(xs)))
    iter+=1
    print('Koren = {:>14.10} \n'.format(xs) )
```




```
# Bisekcija koriscenjem scipy
import scipy.optimize as optimize
def func(x): return x**3-x-3
print(optimize.bisect(func, 1, 2)) # 1.6716998816573323
```

# Rezultat izvršenja programa

```
unesite granice intervala
leva  granica = 1
desna granice = 2
Br.it.        x1              x2              xs              f(xs)
00    1.5000000000    2.0000000000    1.5000000000   -1.1250000000
01    1.5000000000    1.7500000000    1.7500000000    0.6093750000
02    1.6250000000    1.7500000000    1.6250000000   -0.3339843750
03    1.6250000000    1.6875000000    1.6875000000    0.1179199219
04    1.6562500000    1.6875000000    1.6562500000   -0.1128845215
05    1.6562500000    1.6718750000    1.6718750000    0.0012931824
06    1.6640625000    1.6718750000    1.6640625000   -0.0561003685
07    1.6679687500    1.6718750000    1.6679687500   -0.0274799466
08    1.6699218750    1.6718750000    1.6699218750   -0.0131124929
09    1.6708984375    1.6718750000    1.6708984375   -0.0059144357
10    1.6713867188    1.6718750000    1.6713867188   -0.0023118221
11    1.6716308594    1.6718750000    1.6716308594   -0.0005096188
12    1.6716308594    1.6717529297    1.6717529297    0.0003917071
13    1.6716918945    1.6717529297    1.6716918945   -0.0000589746
14    1.6716918945    1.6717224121    1.6717224121    0.0001663616
15    1.6716918945    1.6717071533    1.6717071533    0.0000536923
16    1.6716995239    1.6717071533    1.6716995239   -0.0000026414
17    1.6716995239    1.6717033386    1.6717033386    0.0000255254
18    1.6716995239    1.6717014313    1.6717014313    0.0000114420
19    1.6716995239    1.6717004776    1.6717004776    0.0000044003
20    1.6716995239    1.6717000008    1.6717000008    0.0000008794
Koren =    1.671700001
```

# Aritmetika s realnim brojevima na računaru

- Da bi se u potpunosti shvatila greška zaokruživanja mora se poznavati način pohranjivanja takvih brojeva na računaru.

  Npr. broj s fiksnim zarezom 13.524 je isto što i broj s pokretnim zarezom $.13524*10^2$ ili .13524E2

- Različiti računari koriste za prikaz različite tehnike ali se danas mogu sresti uglavnom nekoliko standardnih načina prikaza.
- Kod svih je osnovna ideja ista. Binarni prikaz broja se sastoji od predznaka, dela koji sadrži podatak koji se odnosi na eksponent (često se naziva karakteristikom), te delu u kojem se nalaze značajne znamenke normalizovanog binarnog broja (često se naziva mantisa).

**Primer:**

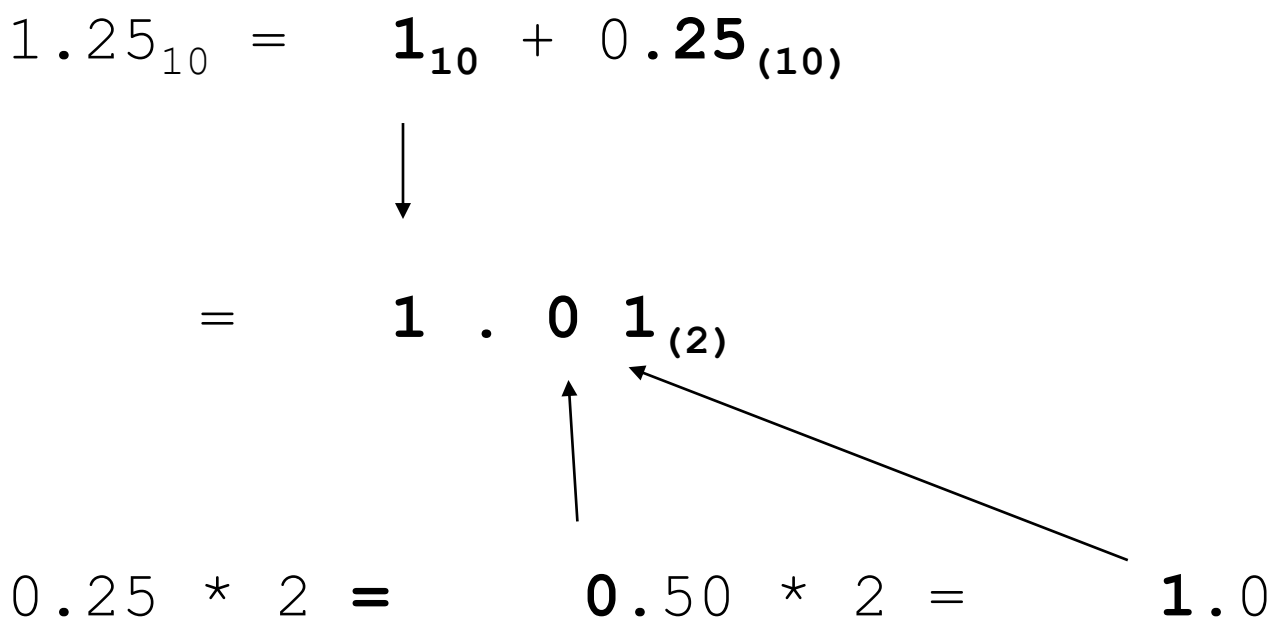
$\mathbf{5.75}_{10} = \quad \mathbf{5} * 10^0 \quad + \mathbf{7} * 10^{-1} + \mathbf{5} * 10^{-2} =$

$\mathbf{1} * 2^2 + \mathbf{0} * 2^1 + \mathbf{1}*2^0 + \mathbf{1}*2^{-1} + \mathbf{1}*2^{-2} =$

$\mathbf{1\,0\,1\,.\,1\,1}_2 = \mathbf{1\,.0\,1\,1\,1}_2 * 2^2$

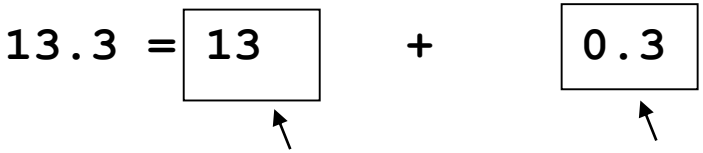
# Primer pretvaranja decimalnog broja u binarni

- Celobrojni deo dekadskog broja pretvara se u binarni uzastopnim deljenjem, a decimalni uzastopnim množenjem s 2, gde celobrojni deo dobijenih proizvoda čini znamenke binarnog razlomka.



# Primer pretvaranja decimalnog broja u binarni

- Primer pretvaranja decimalnih brojeva koji se ne mogu prikazati konačnim brojem binarnih frakcija




```
1 1 0 1  . 0 1 0 0 1 1 0 0 1 ...
.3 * 2   =0.6
.6 * 2     =1.2
.2 * 2       =0.4
.4 * 2         =0.8
.8 * 2           =1.6
.6 * 2             =1.2
.2 * 2               =0.4
.4 * 2                 =0.8
.8 * 2                   =1.6 . . .
```

- Treba uočiti da se konačni decimalni razlomak prikazuje kao beskonačni periodički binarni razlomak.

# Realni brojevi standardne preciznosti - IEEE754

- Koristi se 4 okteta (32 bita = 1+8+23)
- Realni broj se pohranjuje u obliku $x= (-1)^P 2^{K-127} (1.M)$


P je predznak (P = 1: negativan broj; P = 0: pozitivan broj)

- Karakteristika: binarni eksponent + 127 (time se omogućuje prikaz negativnog eksponenta bez upotrebe tehnike dvojnog komplementa)

  Raspon karakteristike: K [0,255].

  Raspon binarnog eksponenta BE [-126,127], slučaj K=0 i K=255 biće objašnjeni u nastavku lekcije.
- Mantisa je oblika 1+m gde vodeća jedinica (skriveni bit) ne ulazi u prikaz

# Posebni slučajevi: prikaz broja 0, +∞ i -∞, NaN

- Kada bi vodeća znamenka normaliziranog broja uvek bila 1, ne bi bilo moguće prikazati broj 0
- Koristi se sledeći dogovor: kada je K=0 i svi bitovi mantise postavljeni na 0, radi se o prikazu realnog broja 0.
- U računaru mogu postojati brojevi "+0" i "-0"

**0000 0000 0000 0000 0000 0000 0000 0000 → +0**
**1000 0000 0000 0000 0000 0000 0000 0000 → -0**

- Međutim, prilikom poređenja tih vrednosti, smatra se da su jednake.
- Prikaz +∞ i -∞
  K=255 i svi bitovi mantise su postavljeni na 0, radi se o prikazu +∞ ili -∞.

**0111 1111 1000 0000 0000 0000 0000 0000 → +∞**

**1111 1111 1000 0000 0000 0000 0000 0000 → -∞**

- Prikaz NaN (*not a number*):
  K=255 i postoje bitovi mantise koji nisu 0.

0111 1111 1**1**00 0000 0000 0000 0000 0000 → NaN

# Posebni slučajevi: denormalizirani broj

- Kada je K=0 i postoje bitovi mantise koji nisu 0, radi se o "denormalizovanom broju".
- Ne podrazumeva se skriveni bit, te se smatra da je vodeći bit mantise 0.
- Vrednost eksponenta je fiksirana na -126 (ne koristi se izraz K=binarni eksponent+127).

**0000 0000 0110 0000 0000 0000 0000 0000**

→ $0.11 \cdot 2^{-126}$

**0000 0000 0000 0000 0000 0000 0000 1101**

→ $0.000\ 0000\ 0000\ 0000\ 0000\ 1101 \cdot 2^{-126}$

Primetiti: 0 je poseban slučaj denormaliziranog broja

0000 0000 0000 0000 0000 0000 0000 0000

→ $0.000\ 0000\ 0000\ 0000\ 0000\ 0000 \cdot 2^{-126}$

# Raspon i preciznost realnih brojeva u pomičnom zarezu

- Raspon i preciznost realnih brojeva standardne preciznosti

  Najmanji pozitivni float broj koji se može prikazati je:

  $0.00000000000000000000001_2 * 2^{-126}$

  što iznosi

  $1.401298464324817 * 10^{-45}$

  Najveći pozitivni float broj koji se može prikazati je:

  $1.11111111111111111111111_2 * 2^{127}$

  to je približno

  $2^{128} = 3.402823669209 * 10^{38}$

Prikaz nije moguć za brojeve:

manje od $-3.4*10^{38}$

između $-1.4*10^{-45}$ i $+1.4*10^{-45}$

veće od $+3.4*10^{38}$

u preostalim intervalima moguć je prikaz brojeva uz preciznost od približno 7 cifara

# Realni brojevi dvostruke preciznosti

- Koristi se 8 okteta (64 bita = 1+11+52)
- Realni broj se pohranjuje u obliku


P je predznak (P = 1: negativan broj; P = 0: pozitivan broj)

Karakteristika: binarni eksponent + 1023, raspon karakteristike: K [0,2047].

Raspon binarnog eksponenta BE [-1022,1023]

Mantisa 52+1 vodeći bit za jedinicu (skriveni bit)

Najmanji pozitivni broj : $0.0000000000...00001_2 * 2^{-1022}$ tj. $4.9406 * 10^{-324}$

Najveći pozitivni broj: $1.11111111...11111_2 * 2^{1023}$ tj.

$2^{1024} = 1.79769313486231_6 * 10^{308}$

Preciznost prikaza broja s mantisom veličine 53 binarne cifre je:

$2^{53}$ približno odgovara $10^x$ pa je

$53 \log 2 \cong x \log 10$ , a odavde x je približno 15.95458977019

(podrazumeva se **15** prvih važećih cifara).

# Realni brojevi dvostruke preciznosti, 80bita

- Osim 64bitne postoje implementacije sa: 80, 96 i 128 bita.
- Primer paspodele ako je njegova veličina 80 bita (1+15+64)

79 78 64 63 0

**P** **Karakteristika** **Mantisa**

P je predznak (P = 1: negativan broj; P = 0: pozitivan broj)

Karakteristika: binarni eksponent + 16383 (15 bita),

raspon karakteristike: K X[0,32767].

K = 0 rezervisana je za prikaz nule, K = 32767 rezervisana je za prikaz ∞

Raspon binarnog eksponenta BEX[-16382,16383]

Mantisa 64+1 bit.

Najmanji pozitivni broj (!=0): $1.0_2 * 2^{-16382} = 3*10^{-4931}$

a najveći je: $1.1111.....111111_2 * 2^{16382}$ približno $2^{16383} = 6*10^{4931}$

**Tačnost prikazivanja brojeva**

$2^{64}$ približno je $10^x$ , pa je 64 log 2 približno jednako x log 10

odavde je x približno 19.2659 tj. približno 19 prvih važećih tačnih cifara.

# Greške

- Glupe greške (blunders) : čovek koji je pripremio program ili parametre za programski jezik je sklon greškama. Obratiti pažnju na mogućnosti programskog jezika, npr. sledeći kod je precizan za množenje

```
x = 1;
y = 1.0;
print ( 0, x, y);
for i in range(1,800):
    x = x*2;
    y = y/2;
    print ( i, x, y);
```

- Greške zbog konverzije ulaza: zbog "šuplje" brojne prave beskonačni skup realnih brojeva mora se konvertovati u konačni skup brojeva u računaru.
- Ako se na ulazu pokuša uneti broj manji od najmanjeg mogućeg javlja se *eksponent underflow* i broj se prisilno postavlja na nulu.

```
i=float(input("unesite mali broj ")) # uneti 1e-1024
print(i)                             # 0.0
```

  - Kod nekih računara ovo se dojavljuje kao greška. Ima računara koja razlikuju *negativ underflow* od *pozitive underflow*.

# Greške

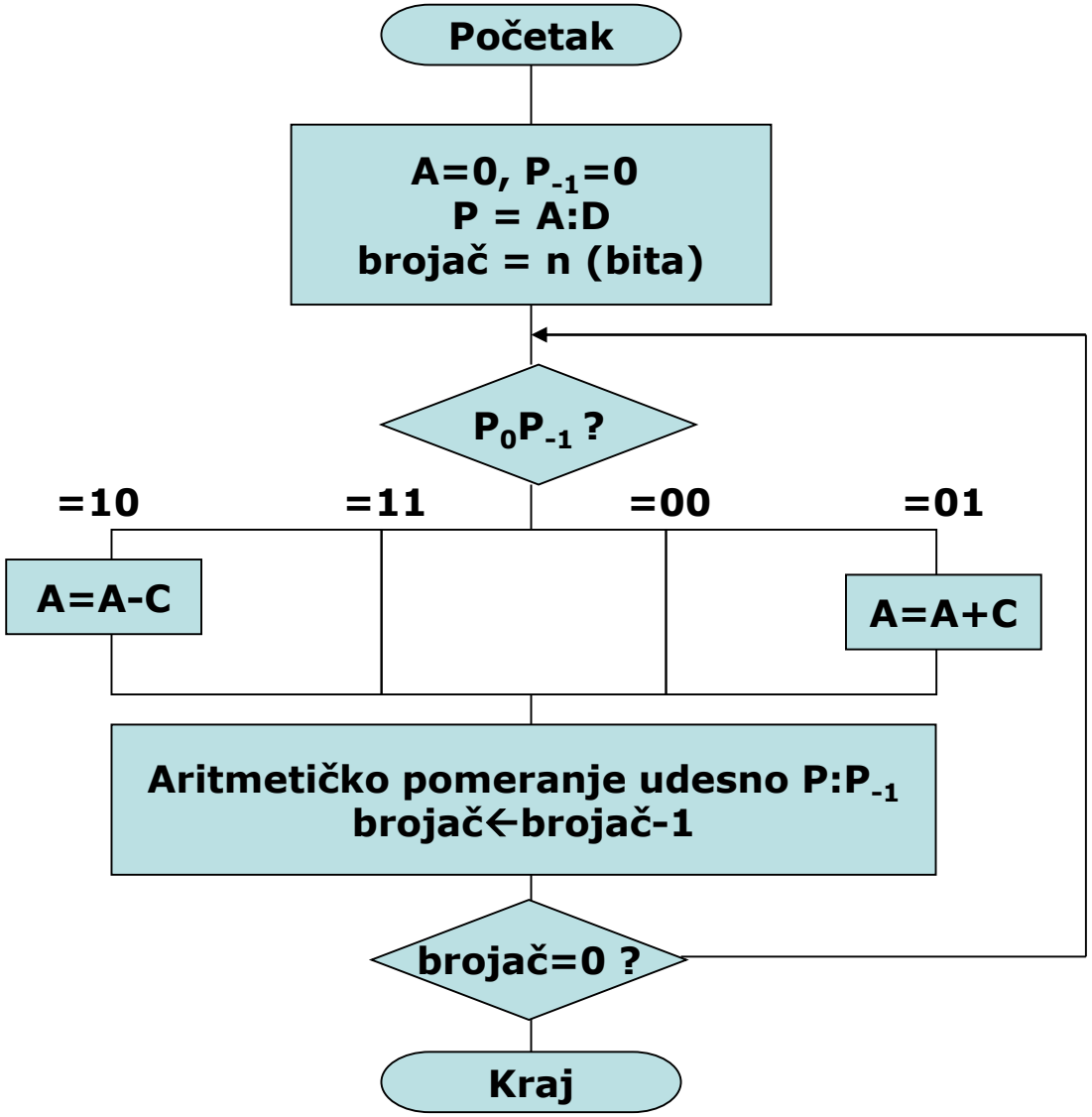
- Slično se događa ako se zada broj veći od najvećeg mogućeg.
  - Ova pojava se naziva *eksponent overflow* i obično se dojavljuje kao *run time error*.
  - Kod nekih tipova računara se svaki broj koji bi trebao biti prevelik postavlja na najveću moguću vrednost.

```
i=float(input("unesite veliki broj "))    #unesite 1e+2048
print(i)                                  #inf
```

- Akumulacione greške se prenose iterativnim prolazom kroz kod koji generiše malu grešku koja je potrebna kao ulaz u sledećem prolazu:

```
a=0.1
for i in range(0,2000000):
    a+=0.1
print(a)          #200000.10000715364

#probajte 0.1*2000000
```

# Booth-ov algoritam (C*D)



# Primer Booth-ovog algoritma (2 x (-3))

**A=0, C=$2_{(10)}$, D=$-3_{(10)}$,**
**P(proizvod)= $-3_{(10)}$ = $00001101_{(2)}$= A:D, $P_{-1}$=0**

| Operacija | P | $P_0P_{-1}$ | sledeće? | |
|---|---|---|---|---|
| | 0000 | 1101 0 | | |
| | | | 10 -> | |
| 1a. A = A – C | 1110 | 1101 0 | | oduzmi, |
| 1b. P:$P_{-1}$>>1 | 1111 | 0110 1 | | pomeri desno (aritmetički) |
| | | | 01 -> | |
| 2a. A = A + C | 0001 | 0110 1 | | saberi |
| 2b. P:$P_{-1}$>>1 | 0000 | 1011 0 | | pomeri desno (aritmetički) |
| | | | 10 -> | |
| 3a. A = A - C | 1110 | 1011 0 | | oduzmi |
| 3b. P:$P_{-1}$>>1 | 1111 | 0101 1 | | pomeri desno (aritmetički) |
| | | | 11 -> | |
| 4b. P:$P_{-1}$>>1 | 1111 | 1010 1 | | pomeri desno (aritmetički) |
| Rezultat = | 1111 | 1010 | | kraj |

# Primer Booth-ovog algoritma (5 x (-3))

$5_{(10)}$ = 00101 → C

$-3_{(10)}$ = 11101 → D

---

| A | D | | |
|---|---|---|---|
| 00000 | 11101 | 0 | |
| | | → | sub and shift |
| 11011 | 11101 | 0 | |
| 11101 | 11110 | 1 | |
| | | → | add and shift |
| 00010 | 11110 | 1 | |
| 00001 | 01111 | 0 | |
| | | → | sub and shift |
| 11100 | 01111 | 0 | |
| 11110 | 00111 | 1 | |
| | | → | shift |
| 11111 | 00011 | 1 | |
| | | → | shift |
| 11111 | 10001 | 1 | |

---

result = 11111 10001 = $-15_{(10)}$

# Booth-ov algoritam 1/2

```
def BinarniIspisBroja( broj, brojbitova):
    for i in range (brojbitova-1,-1,-1):
        if (broj & (1<<i) ): print('1',end='')
        else:                print('0',end='')
        if( i%8 == 0 ):      print(' ',end='')

def Ispisi_A_D_dodatnibit( a, d, dbit, brbita):
   BinarniIspisBroja(a,brbita); print(' ',end='')
   BinarniIspisBroja(d,brbita); print(' ',end='')
   print(dbit)

def main ():
    brojbita = 8;
    C =  2;    D = -3
    A = 0  # bice registar AD
    dodatnibit = 0
    print(' '*13,'A',' '*(brojbita-1),'D',' '*(brojbita-1),"dodatni bit" )
    print(' '*13,end='')
    Ispisi_A_D_dodatnibit(A,D,dodatnibit,brojbita)
```

# Booth-ov algoritam 2/2

```
    for i in range(brojbita):
        print("korak:",'{:>2}'.format(i+1));
        if ((D&1)==1 and dodatnibit==0) : #sub
            print('{:>13}'.format("sub  "), end='')
            A=A-C;
            Ispisi_A_D_dodatnibit(A,D,dodatnibit,brojbita)
        if ((D&1)==0 and dodatnibit==1) :   #add
            print('{:>13}'.format("add  "),end='')
            A=A+C;
            Ispisi_A_D_dodatnibit(A,D,dodatnibit,brojbita)
        if (D&1):   dodatnibit = 1      #shift
        else:       dodatnibit = 0
        D = D >> 1
        if(A&1):    D = ( (1 << brojbita-1)  | D)
        else:       D = ( (~(1 << brojbita-1)) & D)

        A = A >> 1
        print('{:>13}'.format("shr  "),end='')
        Ispisi_A_D_dodatnibit(A,D,dodatnibit,brojbita)

if __name__ == '__main__':main()
```

# Booth – izlaz 3/3

| | | A | D | dodatni bit |
|---|---|---|---|---|
| | | 00000000 | 11111101 | 0 |
| korak: | 1 | | | |
| | sub | 11111110 | 11111101 | 0 |
| | shr | 11111111 | 01111110 | 1 |
| korak: | 2 | | | |
| | add | 00000001 | 01111110 | 1 |
| | shr | 00000000 | 10111111 | 0 |
| korak: | 3 | | | |
| | sub | 11111110 | 10111111 | 0 |
| | shr | 11111111 | 01011111 | 1 |
| korak: | 4 | | | |
| | shr | 11111111 | 10101111 | 1 |
| korak: | 5 | | | |
| | shr | 11111111 | 11010111 | 1 |
| korak: | 6 | | | |
| | shr | 11111111 | 11101011 | 1 |
| korak: | 7 | | | |
| | shr | 11111111 | 11110101 | 1 |
| korak: | 8 | | | |
| | shr | 11111111 | 11111010 | 1 |

# MONTE KARLO - metoda

- Osnovna ideja se sastoji u sledećem:
  - cela oblast (domen) sadrži ciljnu oblast
  - generiše se veliki broj slučajnih uzoraka (tačaka) koji pripadaju celoj oblasti
  - odnos površine cele oblasti i površine ciljne oblasti je dat kao:

$$\frac{P_1}{P_2} = \frac{N}{n}$$

gde su:

| | |
|---|---|
| $P_1$ | površina cele oblasti |
| $P_2$ | površina ciljne oblasti |
| $N$ | ukupan broj generisanih uzoraka koji pripadaju celoj oblasti |
| $n$ | broj generisanih uzoraka koji pripadaju ciljnoj oblasti |

- Na ovaj način moguće je numerički izračunati npr. integrale, površine koje se mogu izračunati samo numerički (površina elipse).

# Monte Karlo metoda približnog izračunavanja broja π

Primenom Monte Karlo metode izračunati približnu vrednost broja π :

Ulazi u program:

broj slučajnih brojeva N,
ili
iterativno povečavanje N od min do max korisničke vrednosti

Izlazi iz programa:

približna vrednost broja π izračunata Monte Karlo metodom za dati ulaz

*Napomena: razmisliti o*

- *površini jediničnog kruga sa centrom u (0.5, 0.5)*
- *zapremini jedinične kugle sa centrom u (0.5,0.5,0.5)*
- *ili jednostavnije kruga ili kugle u ishodištu*

# Monte Karlo metoda približnog izračunavanja broja π

```
# posmatra se cetvrtina kruga kome je centar u ishodistu a r=1
# gadja se kvadrat sa vrhovima (0,0)-(0,1)-(1,1)-(1,0)
# pogodak je slucajna koordinata unutar cetvrtine kruga
# odnos povrsina kvadrata =1 i cetvrtine kruga r**2*Pi/4=Pi/4
# odgovara odnosu ukupno_iteracija/broj_pogodaka
# te je PI=4*broj_pogodaka/ukupno_iteracija
import math
    import random
    def f(n):
        pog = 0;
        for i in range(n):
            xrand = random.random()
            yrand = random.random()
            if((math.sqrt( (xrand**2)+(yrand**2) ))<=1):
                pog+=1
        return( 4.0*pog/n)
    print("Broj gadjanja  PI")
    print('{:>10d}     {:<5.4}'.format(1000000,f(1000000)))
```

```
Broj gadjanja  PI
   1000000     3.141
```