Programmeerimine

7. loeng

Täna loengus

- Listi mõiste
- Järjendid ja massiivid
- Listid, ennikud ja sõned
- Massiivide töötlemine
- Jadaiterator ja määratud tsüklid
- Eeldefineeritud operatsioonid järjenditel

Listi mõiste Pythonis

- Lihtmuutuja
- List kui andmekogum
- Tühi list

Listi mõiste Pythonis

- Lihtmuutuja
- List kui andmekogum
- Tühi list

Listi mõiste Pythonis

- Lihtmuutuja

```
>>> x = 3
>>> x = 5
>>> x
5
```

- List kui andmekogum
- Tühi list

Listi mõiste Pythonis

- Lihtmuutuja

```
>>> x = 3
>>> x = 5
>>> x
5
```

- List kui andmekogum

- Tühi list

Listi mõiste Pythonis

- Lihtmuutuja

```
>>> x = 3
>>> x = 5
>>> x
5
```

- List kui andmekogum

```
>>> joogid = ['tee', 'kohv', 'mahl']
>>> temperatuurid = [80, 60, 20]
>>> koos = ['tee', 80, 'kohv', 60, 'mahl', 20]
```

- Tühi list

Listi mõiste Pythonis

- Lihtmuutuja

```
>>> x = 3
>>> x = 5
>>> x
5
```

- List kui andmekogum

```
>>> joogid = ['tee', 'kohv', 'mahl']
>>> temperatuurid = [80, 60, 20]
>>> koos = ['tee', 80, 'kohv', 60, 'mahl', 20]
```

- Tühi list

Listi mõiste Pythonis

- Lihtmuutuja

```
>>> x = 3
>>> x = 5
>>> x
5
```

- List kui andmekogum

```
>>> joogid = ['tee', 'kohv', 'mahl']
>>> temperatuurid = [80, 60, 20]
>>> koos = ['tee', 80, 'kohv', 60, 'mahl', 20]
```

- Tühi list

```
>>> tühi = []
```

Listi mõiste Pythonis

- Lihtmuutuja

```
>>> x = 3
>>> x = 5
>>> x
5
```

- List kui andmekogum

```
>>> joogid = ['tee', 'kohv', 'mahl']
>>> temperatuurid = [80, 60, 20]
>>> koos = ['tee', 80, 'kohv', 60, 'mahl', 20]
```

- Tühi list

```
>>> tühi = []
>>> joogid
['tee', 'kohv', 'mahl']
>>> temperatuurid
[80, 60, 20]
>>> tühi
[]
```

Listi elemendid

- Indeksid
- Elementide arv

Listi elemendid

- Indeksid
- Elementide arv

Listi elemendid

- Indeksid

```
>>> temperatuurid = [80, 60, 20]
```

- Elementide arv

Listi elemendid

- Indeksid

```
>>> temperatuurid = [80, 60, 20]
```

80	60	20
0	1	2

- Elementide arv

Listi elemendid

- Indeksid

```
>>> temperatuurid = [80, 60, 20]
```

80	60	20
0	1	2

```
>>> temperatuurid[0]
80
```

- Elementide arv

Listi elemendid

- Indeksid

```
>>> temperatuurid = [80, 60, 20]
```

80	60	20
0	1	2

```
>>> temperatuurid[0]
80
>>> temperatuurid[2]
20
```

- Elementide arv

Listi elemendid

- Indeksid

```
>>> temperatuurid = [80, 60, 20]
```

80	60	20
0	1	2

```
>>> temperatuurid[0]
80
>>> temperatuurid[2]
20
```

- Elementide arv

Listi elemendid

- Indeksid

```
>>> temperatuurid = [80, 60, 20]
```

80	60	20
0	1	2

```
>>> temperatuurid[0]
80
>>> temperatuurid[2]
20
```

- Elementide arv

```
>>> len(temperatuurid)
3
```

Listi elemendid

- Negatiivne indeks

Listi elemendid

- Negatiivne indeks

Listi elemendid

- Negatiivne indeks

80	60	20
-3	-2	-1

Listi elemendid

- Negatiivne indeks

80	60	20
-3	-2	-1

```
>>> temperatuurid[-1]
20
```

Listi elemendid

- Negatiivne indeks

80	60	20
-3	-2	-1

```
>>> temperatuurid[-1]
20
>>> temperatuurid[-3]
80
```

Listi elemendid

- Negatiivne indeks

80	60	20
-3	-2	-1

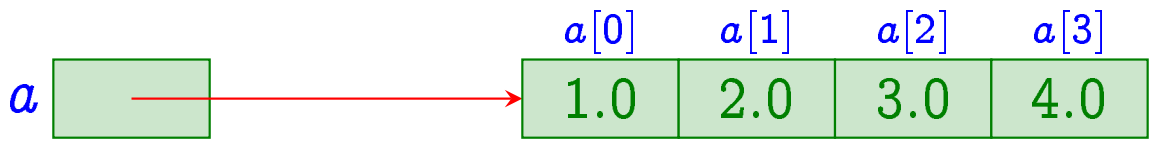
```
>>> temperatuurid[-1]
20

>>> temperatuurid[-3]
80

>>> temperatuurid[-4]
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    temperatuurid[-4]
IndexError: list index out of range
```

List on muteeritav

$$a = [1.0,\ 2.0,\ 3.0,\ 4.0]$$
$$a\,[3] = 3.1$$
$$b = [0,\ 0,\ 0,\ 0,\ 0]$$
$$b\,[3] = 6$$
$$b\,[0] = b\,[3] + 3$$

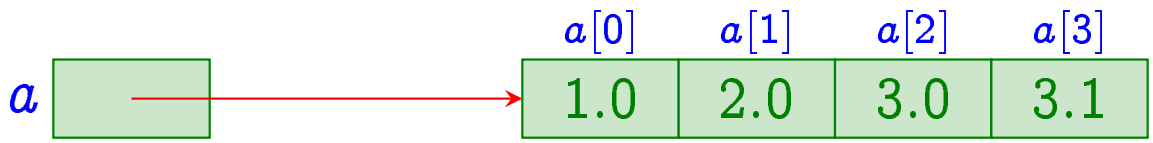
List on muteeritav

$a = [1.0, 2.0, 3.0, 4.0]$
$a[3] = 3.1$
$b = [0, 0, 0, 0, 0]$
$b[3] = 6$
$b[0] = b[3] + 3$



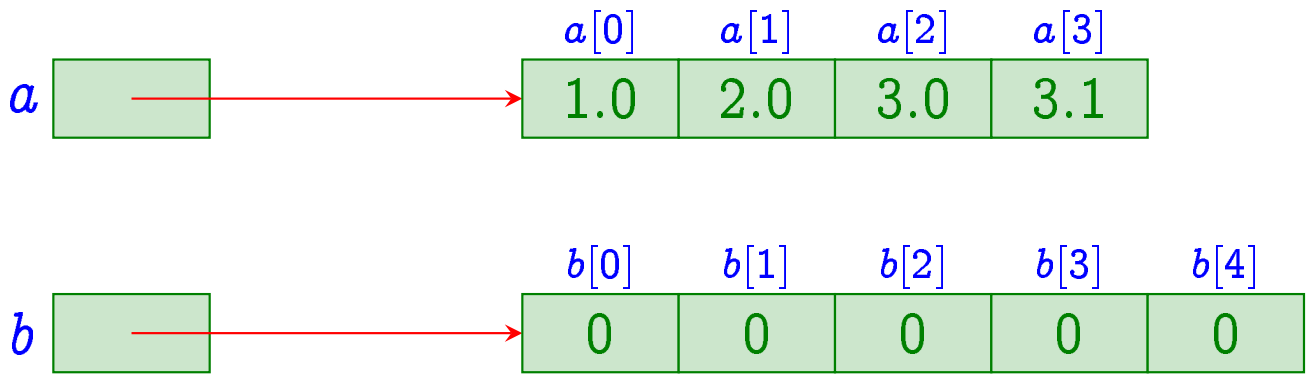
List on muteeritav

$a = [1.0, 2.0, 3.0, 4.0]$
$a[3] = 3.1$
$b = [0, 0, 0, 0, 0]$
$b[3] = 6$
$b[0] = b[3] + 3$



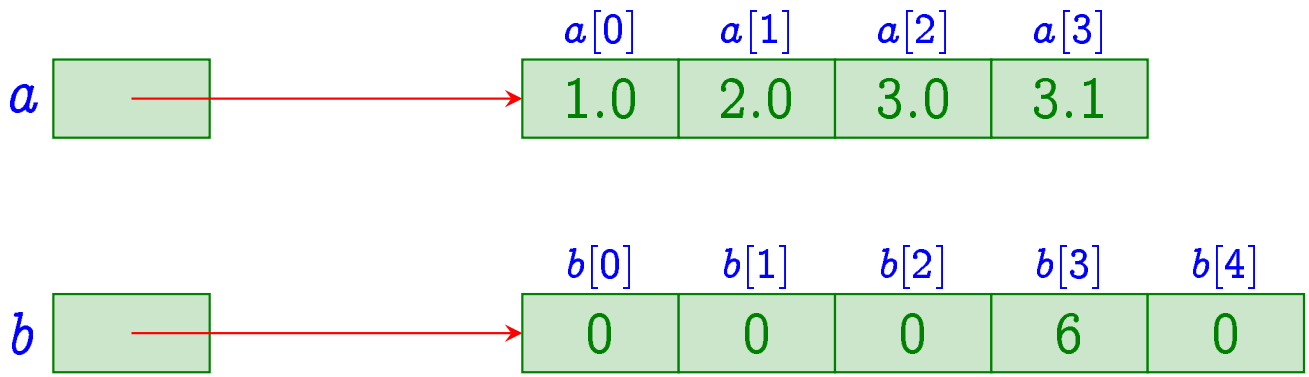
List on muteeritav

$a = [1.0, 2.0, 3.0, 4.0]$
$a[3] = 3.1$
$b = [0, 0, 0, 0, 0]$
$b[3] = 6$
$b[0] = b[3] + 3$



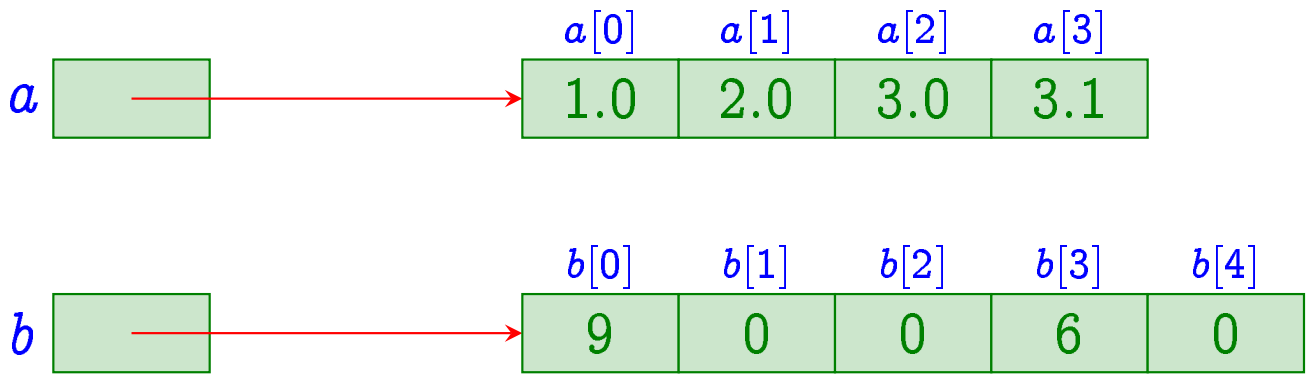
List on muteeritav

$a = [1.0, 2.0, 3.0, 4.0]$
$a[3] = 3.1$
$b = [0, 0, 0, 0, 0]$
$b[3] = 6$
$b[0] = b[3] + 3$



List on muteeritav

$a = [1.0, 2.0, 3.0, 4.0]$
$a[3] = 3.1$
$b = [0, 0, 0, 0, 0]$
$b[3] = 6$
$b[0] = b[3] + 3$



Listi läbivaatus

```
temperatuurid = [80, 60, 20]
i = 0
while i < len(temperatuurid):
  print(i, temperatuurid[i])
  i += 1
```

- Nullime 50st suuremad temperatuurid ¡4-¿[]

Listi läbivaatus

```
temperatuurid = [80, 60, 20]
i = 0
while i < len(temperatuurid):
  print(i, temperatuurid[i])
  i += 1
```

```
0 80
1 60
2 20
```

- Nullime 50st suuremad temperatuurid ¡4-¿[]

Listi läbivaatus

```
temperatuurid = [80, 60, 20]
i = 0
while i < len(temperatuurid):
  print(i, temperatuurid[i])
  i += 1
```

```
0 80
1 60
2 20
```

- Nullime 50st suuremad temperatuurid

Listi läbivaatus

```
temperatuurid = [80, 60, 20]
i = 0
while i < len(temperatuurid):
  print(i, temperatuurid[i])
  i += 1
```

```
0 80
1 60
2 20
```

- Nullime 50st suuremad temperatuurid

```
i = 0
while i < len(temperatuurid):
  if temperatuurid[i] > 50:
    temperatuurid[i] = 0
  i += 1
print(temperatuurid)
```

Listi läbivaatus

```
temperatuurid = [80, 60, 20]
i = 0
while i < len(temperatuurid):
  print(i, temperatuurid[i])
  i += 1
```

```
0 80
1 60
2 20
```

- Nullime 50st suuremad temperatuurid

```
i = 0
while i < len(temperatuurid):
  if temperatuurid[i] > 50:
    temperatuurid[i] = 0
  i += 1
print(temperatuurid)
```

```
[0, 0, 20]
```

Järjendid

Põhimõisted

- Järjend on mingite elementide lõplik jada.
- Massiiv (ingl. *array*) on sama tüüpi elementidega ja fikseeritud suurusega järjend, kus elementidele juurdepääs (indekseerimine) toimub konstantse ajaga.
- Massiivi elemendid on identifitseeritavad täisarvuliste indeksite kaudu.
- Klassikaliselt on massiivid muteeritavad (ingl. *mutable*); so. mingi elemendi muutmisel, muudetakse selle väärtus olemasolevas massiivis.
- Mõnedes keeltes võivad massiivid olla mittemuteeritavad (ingl. *immutable*); so. elemendi "muutmisel" luuakse uus massiiv.

Järjendid

Järjendid Pythonis

- Pythonis klassikalisi massiive keeles otseselt ei ole.
- Selle asemel on kaks üldist järjenditüüpi: listid ja ennikud.
  - Sarnaselt massiividega toimub elementide indekseerimine täisarvudega ning konstantse ajaga.
  - Mõlemal juhul põhierinevuseks, et *elemendid ei pea olema sama tüüpi*.
  - Listide korral ka see, et suurus ei ole fikseeritud (so. saab lisada uusi elemente ning olemasolevaid eemaldada).
- Lisaks on ka sõned Pythonis järjenditüübiks.

NB!

Edaspidi nimetame massiivideks liste, kus kõik elemendid on sama tüüpi.

Järjendid Pythonis

Listid

- List on dünaamilise pikkusega muteeritav järjend.
- Esitatakse elementide loendina kandiliste sulgude vahel.

$$[\ expr_1, expr_2, \ldots, expr_n\ ]$$

- Listiavaldis tekitab n-elemendilise listi, kus elementide väärtused on vastavalt avaldiste $expr_i$ väärtused.
- Avaldised $expr_i$ võivad olla erinevat tüüpi.

NB!

Teisetes keeltes nimetatakse listideks tavaliselt dünaamilise pikkusega järjendeid, kus elementidele juurdepääs toimub järjestikku (so. lineaarse ajaga).

Järjendid Pythonis

Ennikud

- Ennik (ingl. *tuple*) on fikseeritud pikkusega mittemuteeritav järjend.
- Esitatakse elementide loendina (ümar-)sulgude vahel.

$$(expr_1, expr_2, \ldots, expr_n)$$

- Ennikuavaldis tekitab n-elemendilise korteeži, kus elementide väärtused on vastavalt avaldiste $expr_i$ väärtused.
- Avaldised $expr_i$ võivad olla erinevat tüüpi.

Sõned

- Sõne (ingl. *string*) on tähemärkidest koosnev mittemuteeeritav järjend.
- Esitatakse ülakomade või jutumärkide vahel.

Järjendid Pythonis

Näiteid listidest ja ennikutest

```
a = [1, 7, 3, 5]
b = [2, 8, 0, 2]
d = ["Hello", 1, 5]          # erinevat tüüpi elementidega
e = [ ]                      # tühi list

f = (1, 'World', a)
g = (3,)                     # üheelemendiline ennik
h = ()                       # tühi ennik
```

Järjendid Pythonis

Järjendi elementidele juurdepääs

$$seqName\,[index\,]$$

- *seqName* on järjendimuutuja nimi (või suvaline avaldis, mis väärtustub järjendiks).
- *index* on täisarvuline avaldis, mis näitab mitmendale elemendile juurdepääsu soovime.
- Järjendite indekseerimine algab nullist.
- Listide korral võib indeksavaldist kasutada ka L-väärtusena (so. omistuslause vasakul poolel).

Näide

```
a    = [1, 7, 3, 5]
b    = a[0] + a[3]        # b = 6
a[2] = 0                  # a = [1, 7, 0, 5]
```

Järjendid Pythonis

Järjendi elementidele juurdepääs

- Kasutada võib ka negatiivseid indekseid, misjuhul indekseeritakse paremalt.
  - Indeksiga -1 on parempoolseim element.
- Indeks i peab olema vahemikus $-N \leq i \leq N-1$ (kus N on listi elementide arv), vastasel korral on täitmisaegne viga.

Näide

```
a = [1, 7, 3, 5]
d = a[-1]          # d = 5
e = a[-4]          # e = 1
```

Jadaiteraator

- Pythonis on for-tsükkel üldine jadaiteraator:

  for *var* **in** *list*:
      body

- *var* on tsüklimuutuja ning *list* on mingi jada.
- Tsüklimuutuja saab algul väärtuseks jada esimese elemendi väärtuse ja igal järgneval iteratsioonisammul jada järgmise elemendi väärtuse.
- Tsükkel lõpeb, kui jadas rohkem elemente pole.

Jadaiteraator

Näide – jada elementide väljatrükkimine

```
for c in "Hello, World!":
    print(c)

for i in [1,7,2,3,12]:
    print(i)

for e in (3,8,2,15):
    print(e)
```

Määratud tsüklid

- Määratud tsükli tarvis saab jada genereerimiseks kasutada funktsiooni *range*.
- Kõige üldisem versioon, $range(start, stop, step)$, saab kolm täisarvulist argumenti:
  - argument *start* on jada esimene element;
  - argument *stop* on jada ülemine piir (kõik jada elemendid on sellest "väiksemad");
  - argument *step* on samm, mille võrra jada järgmine element on eelmisest "suurem".
- Kaheargumendilisena, $range(start, stop)$, on *step* üks.
- Üheargumendilisena, $range(stop)$, on *start* väärtuseks null.

$$
\begin{aligned}
range(5) &\Longrightarrow [0,1,2,3,4] \\
range(1,6) &\Longrightarrow [1,2,3,4,5] \\
range(1,6,2) &\Longrightarrow [1,3,5] \\
range(6,1,-2) &\Longrightarrow [6,4,2]
\end{aligned}
$$

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

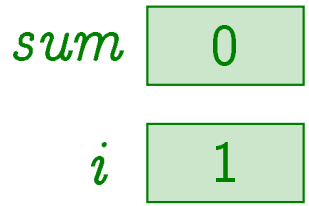
Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum 0

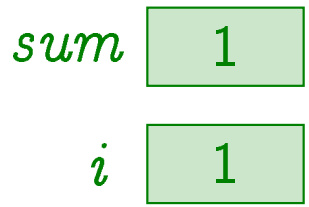
Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```



Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i
...
```



Määratud tsüklid

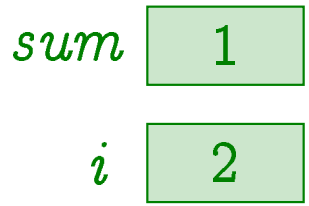

```
sum = 0
for i in range(1,5):
    sum = sum + i
```

...

sum	1
i	1

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```



Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	3
i	2

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	3
i	2

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	3
i	3

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	6
i	3

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	6
i	3

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	6
i	4

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	10
i	4

Määratud tsüklid


```
sum = 0
for i in range(1,5):
    sum = sum + i

...
```

sum	10
i	4

Määratud tsüklid

```
sum = 0
for i in range(1, 5):
    sum = sum + i
```


...

sum	10
i	4

Massiivide töötlemine

Massiivi elementide summeerimine

```
def sumListValues(a):
    sum = 0
    for elem in a:
        sum += elem
    return sum
```

Näide

```
A = [7, 2, 4]
s = sumListValues(A)
print(s)                                    # Trükib: 13
```

Massiivide töötlemine

Massiivi elementide summeerimine (ver. 2)

```
def sumListValues(a):
    sum = 0
    for i in range(len(a)):
        sum += a[i]
    return sum
```

Näide

```
A = [7, 2, 4]
s = sumListValues(A)
print(s)                                    # Trükib: 13
```

Massiivide töötlemine

Massiivi negatiivsete elementide loendamine

```
def countNegValues(a):
    count = 0
    for i in range(len(a)):
        if a[i] < 0:
            count += 1
    return count
```

Näide

```
A = [7, -2, 4, -1]
n = countNegValues(A)
print(n)                                    # Trükib: 2
```

Massiivide töötlemine

Massiivist otsimine

```
def findListValue(x, a):
    for i in range(len(a)):
        if a[i] == x:
            return True
    return False
```

Näide

```
A = [7, 2, 4]
if findListValue(2,A):
    print("Arv leiti!")
else:
    print("Arvu ei leitud!")
                                    # Trükib: "Arv leiti!"
```

Massiivide töötlemine

Massiivist minimaalse elemendi leidmine

```
def minListValue(a):
    min = a[0]
    for i in range(1,len(a)):
        if (a[i] < min):
            min = a[i]
    return min
```

Näide

```
A = [7, -2, 4, -1]
m = minListValue(A)
print(m)                    # Trükib: -2
```

Massiivide töötlemine

Listi ümberpööramine

```
def reverseList(a):
    n = len(a)
    b = list(range(n))
    for i in range(n):
        b[i] = a[n-1-i]
    return b
```

Näide

```
A = [7, 2, 4]
B = reverseList(A)
print(A[0])          # Trükib: 7
print(B[0])          # Trükib: 4
```

Eeldefineeritud operatsioone järjenditel

Eeldefineeritud operaatorid

- Liitmine: avaldis *s1 + s2* konkateneerib järjendid *s1* ja *s2*.
  - Järjendid *s1* ja *s2* peavad olema sama tüüpi.
- Korrutamine: avaldis *s * n* "paljundab" järjendit *n*-korda.
- Jadasse kuuluvus: avaldis *x* `in` *s* kontrollib, kas *x* on järjedi *s* element.
  - Sõnede korral võib *x* olla mitmetäheline sõne.
  - "Mittekuuluvust" saab kontrollida operaatoriga `not in`.

Näide

```
a = [1, 2, 3] + [4, 5]      # a = [1, 2, 3, 4, 5]
b = [1, 2] * 3              # b = [1, 2, 1, 2, 1, 2]
c = 3 in a                  # c = True
d = 3 in b                  # d = False
```

Eeldefineeritud operatsioone järjenditel

Eeldefineeritud funktsioone

Funktsioon	Kirjeldus
len(s)	leiab järjendi pikkuse
list(s)	teisendab järjendi listiks
tuple(s)	teisendab järjendi ennikuks
min(s)	tagastab minimaalse elemendi
max(s)	tagastab maksimaalse elemendi
sorted(s)	tagastab sorteeritud listi

Näide

```
a = list('Hello')          # a = ['H', 'e', 'l', 'l', 'o']
b = tuple('Hello')         # b = ('H', 'e', 'l', 'l', 'o')
c = min('Hello, World')    # c = ' '
d = max('Hello, World')    # d = 'r'
e = sorted('World')        # e = ['W', 'd', 'l', 'o', 'r']
```

Eeldefineeritud operatsioone järjenditel

Alamjadade selekteerimine

- Pythonis on võimalik järjendist alamjada selekteerimiseks kasutada nn. viilutamist (ingl. *slicing*).
- Viilutamine on analoogiline indekseerimisega, kuid indeksavaldises on kaks kooloniga eraldatud täisarvulist indeksit.

$$seqName\,[index1 : index2\,]$$

- *index1* on selekteeritava alamjada esimese elemendi indeks.
- *index2* on selekteeritava alamjada ülempiir.

Näide

```
a = 'myfile.txt'
b = a[2:6]      # b = 'file'
c = a[:6]       # c = 'myfile'
d = a[-3:]      # d = 'txt'
```

Eeldefineeritud operatsioone järjenditel

Eeldefineeritud meetode listidel

Meetod	Kirjeldus
$L.index(x)$	tagastab elemendi x esimese indeksi
$L.count(x)$	tagastab elemendi x esinemiste arvu
$L.append(x)$	lisab elemendi x listi L lõppu

Eeldefineeritud operatsioone järjenditel

Näide

```
L = list(range(3))          # L = [0, 1, 2]
L.append(1)                 # L = [0, 1, 2, 1]
b = L.index(2)              # b = 2
c = L.index(1)              # c = 1
```

Suur tänu osalemast

ja

kohtumiseni!