Programmeerimine

3. loeng

Täna loengus

- Tõeväärtustüüp ja loogilised avaldised
- Hargnemisdirektiivid
  - Lihtne if-lause
  - if-else-lause
  - Üldkujuline if-lause
- Tsüklidirektiivid
  - Eelkontrolliga tsükkel
  - Tsükli kontrolldirektiivid
- Erindid
  - Lihtne try-lause

Loogilised avaldised

Loogilised avaldised

- Avaldisi, mille väärtus on tõeväärtustüüpi, nimetatakse loogilisteks avaldisteks.
- Loogilised avaldised koosnevad muutujatest, tõeväärtuskonstantidest, relatsioonilistest- ja loogilistest operaatoritest.
- Tõeväärtustüüp Pythonis:

Tüübi nimi	*bool*
Tõene väärtus	`True`
Väär väärtus	`False`

- NB! Pythonis on tõeväärtustüüp täisarvutüübi alamtüüp, kus `False` on sama mis `0` ja `True` on sama mis `1`.

Relatsioonilised operaatorid

- Relatsioonilised operaatorid võimaldavad võrrelda sama tüüpi väärtusi.

Operaator	Kirjeldus
==	võrdus
!=	mittevõrdus
<	väiksem
<=	väiksem või võrdne
>	suurem
>=	suurem või võrdne

- Relatsioonilised operaatorid on defineeritud kõigil baastüüpidel.
- NB! Lisaks võivad nad olla defineeritud ka mittebaastüüpidel, kuid sel juhul on nende tähendus keerulisem.

Relatsioonilised operaatorid

- NB! Ujukomaarvude võrdlemisel tuleb arvestada, et tegemist ei ole reaalarvudega ja kõik tehted ujukomaarvudel on ligikaudsed.
- Näiteks:

```
0.1+0.1+0.1-0.3        ⟹  5.551115123125783e-17
0.1+0.1+0.1-0.3 == 0   ⟹  False
```

- Ujukomaarvude võrdsuse või mittevõrdsuse kontrolliks tuleks enamik juhtudel operaatorite `==` ja `!=` asemel kasutada arvude vahe nulliga läheduse kontrolli; so.

$$abs(x - y) <= eps$$

kus *eps* on mingi nullilähedane arv ja näitab millise täpsuseni soovime ujukomaarvudega opereerida.

Relatsioonilised operaatorid

- Sõnede võrdlemine toimub leksikograafilises järjekorras.
- Tähemärkide võrdlemisel võrreldakse märkide ASCII või Unicode väärtusi.
- Numbreid tähistavad märgid on loomulikus järjekorras.
  - Märk `'0'` on koodiga `48`.
  - Märk `'1'` on koodiga `49`, jne.
- Ladina tähestiku suurtähed `'A'`, `'B'`, ..., `'Z'`.
  - Täht `'A'` on koodiga `65`, täht `'B'` koodiga `66`, jne.
- Ladina tähestiku väiketähed `'a'`, `'b'`, ..., `'z'`.
  - Täht `'a'` on koodiga `97`, täht `'b'` koodiga `98`, jne.
- NB! Suurtähed on väiketähtedest väiksema koodiga.

Loogilised operaatorid

- Keerukamate loogiliste avaldiste moodustamine toimub loogiliste operaatorite abil.

Operaator	Kirjeldus
`not`	loogiline eitus (unaarne)
`and`	loogiline JA
`or`	loogiline VÕI

- Loogiliste operaatorite formaalseks spetsifitseerimiseks kasutatakse tõeväärtustabeleid.
- Loogiline eitus:

x	`not` x
`True`	`False`
`False`	`True`

Loogilised operaatorid

- Loogiline JA (konjunktsioon):

x	y	x and y
True	True	True
True	False	False
False	True	False
False	False	False

- Loogiline VÕI (disjunktsioon):

x	y	x or y
True	True	True
True	False	True
False	True	True
False	False	False

Operaatorite prioriteedid

Astendamine	`**`
Unaarsed operaatorid	`-, +`
Multiplikatiivsed operaatorid	`*, /, //, %`
Aditiivsed operaatorid	`+, -`
Võrdlusoperaatorid	`<, <=, >, >=`
Võrdusoperaatorid	`==, !=`
Loogiline eitus	`not`
Loogiline JA	`and`
Loogiline VÕI	`or`

Loogiliste avaldiste väärtustamine

- Sarnaselt aritmeetiliste avaldistega, väärtustatakse ka loogilistes avaldistes üldjuhul operaatori argumendid enne operaatori väärtustamist.
- Erandiks on operaatorid `and` ja `or`, millede korral kasutatakse nn. kärbetega väärtustamist (*ingl. shortcut evaluation*).
  - Kõigepealt väärtustatakse esimene argument ja kui selle põhjal on võimalik leida kogu väärtus, siis teist argumenti ei väärtustata.
  - Operaatori `and` korral, kui esimene argument on `False`, siis tulemuseks on `False`.
  - Operaatori `or` korral, kui esimene argument on `True`, siis tulemuseks on `True`.

Tingimuslaused

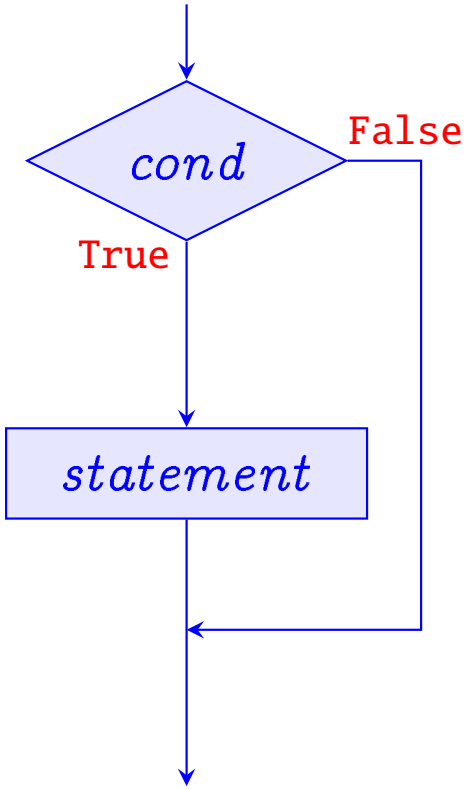
Tingimuslaused

- Lihtne if-lause:

  if *cond*:

  statement

- Tingimus *cond* on loogiline avaldis.
- Lause *statement* täidetakse ainult juhul, kui tingimus *cond* on tõene.
- *statement* võib olla kas üksik lause või lausete plokk.
- Pythonis on plokk määratud taandega.



Tingimuslaused

Näide – kahe arvu järjestamine

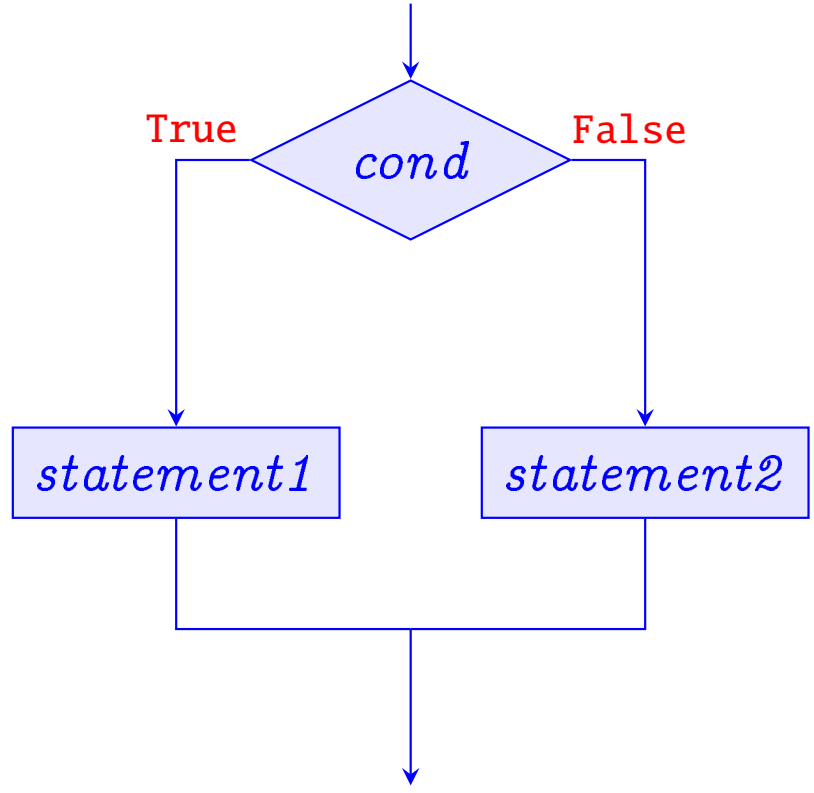
```
arv1 = int(input("Sisesta esimene arv: "))
arv2 = int(input("Sisesta teine arv: "))

# Kui esimene on suurem, siis vahetada
if arv1 > arv2:
    tmp = arv1
    arv1 = arv2
    arv2 = tmp

print("Arvud kasvavalt:", arv1, "ja siis", arv2)
```

Tingimuslaused

- if-else-lause:

```
if cond:
    statement1
else:
    statement2
```

- Kui tingimus *cond* on tõene, siis täidetakse lause *statement1*.
- Kui tingimus *cond* on väär, siis täidetakse lause *statement2*.



Tingimuslaused

Näide – kahest arvust maksimaalse leidmine

```
arv1 = int(input("Sisesta esimene arv: "))
arv2 = int(input("Sisesta teine arv: "))

if arv1 < arv2:
    max = arv2
else:          # arv1 >= arv2
    max = arv1

print("Maksimaalne on arv", max)
```

Tingimuslaused

Näide – arvude võrdlemine (v. 1)

```
arv1 = int(input("Sisesta esimene arv: "))
arv2 = int(input("Sisesta teine arv: "))

if arv1 > arv2:
    print("Esimene arv on suurem")
else:
    if arv2 > arv1:
        print("Teine arv on suurem")
    else:
        print("Arvud on võrdsed")
```

Tingimuslaused

- Kolme või enama variandiga hargnemiseks saab kasutada "if-else-if-redelit":

```
if cond1:
    statement1
elif cond2:
    statement2
  ⋮
elif condN:
    statementN
else:
    statementN+1
```

- Võrreldes üksteisesse sisestatud if-else-lausetega, võivad kõik harud olla sama taandega.

Tingimuslaused

Näide – arvude võrdlemine (v. 2)

```
arv1 = int(input("Sisesta esimene arv: "))
arv2 = int(input("Sisesta teine arv: "))

if arv1 > arv2:
    print("Esimene arv on suurem")
elif arv2 > arv1:
    print("Teine arv on suurem")
else:
    print("Arvud on võrdsed")
```

Tsüklidirektiivid

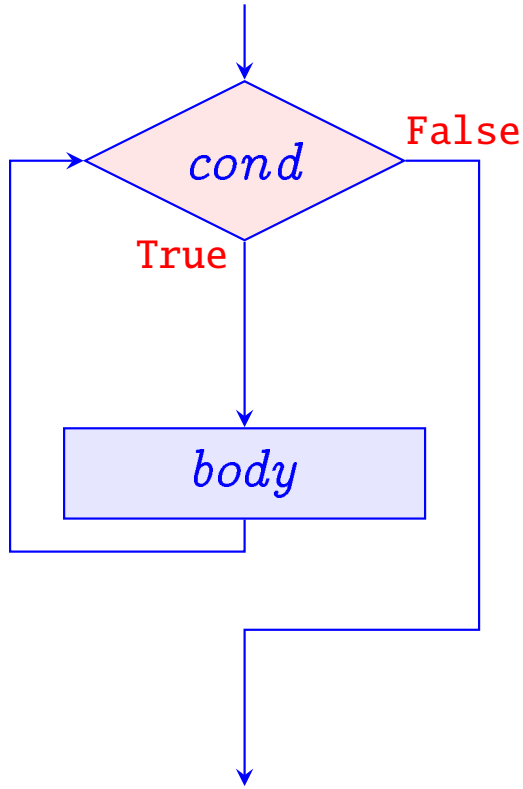
Eelkontrolliga tsükkel

- while-tsükkel:

  while *cond*:

  body

- Tingimus *cond* on loogiline avaldis.
- Tsüklikeha *body* võib olla kas üksik lause või lausete plokk.
- Kui tingimus on tõene, siis täidetakse tsüklikeha üks kord ja kontrollitakse tingimust uuesti.
- Protsessi korratakse kuni tingimus on väär, misjuhul tsüklikeha rohkem ei täideta, vaid jätkatakse tsüklile järgnevate lausete täitmisega.



Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

sum 0

Eelkontrolliga tsükkel

$$sum = 0$$
$$i = 1$$
$$\textbf{while } i < 5\text{:}$$
$$\qquad sum = sum + i$$
$$\qquad i = i + 1$$

$$\ldots$$

sum	0
i	1

Eelkontrolliga tsükkel

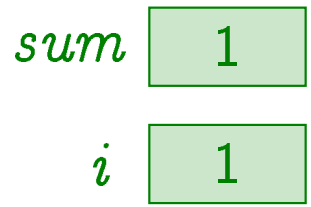

```
sum = 0
i = 1
while i < 5:
    sum = sum + i
    i = i + 1

...
```

sum	0
i	1

Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...



Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

sum	1
i	2

Eelkontrolliga tsükkel


```
sum = 0
i = 1
while i < 5:
    sum = sum + i
    i = i + 1

...
```

sum	1
i	2

Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

sum	3
i	2

Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

sum	3
i	3

Eelkontrolliga tsükkel


```
sum = 0
i = 1
while i < 5:
    sum = sum + i
    i = i + 1

...
```

sum	3
i	3

Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

sum	6
i	3

Eelkontrolliga tsükkel


```
sum = 0
i = 1
while i < 5:
    sum = sum + i
    i = i + 1

...
```

sum	6
i	4

Eelkontrolliga tsükkel


```
sum = 0
i = 1
while i < 5:
    sum = sum + i
    i = i + 1

...
```

sum	6
i	4

Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

sum	10
i	4

Eelkontrolliga tsükkel

$sum = 0$
$i = 1$
while $i < 5$:
    $sum = sum + i$
    $i = i + 1$

...

sum	10
i	5

Eelkontrolliga tsükkel

```
sum = 0
i = 1
while i < 5:
    sum = sum + i
    i = i + 1
```


...

sum	10
i	5

Eelkontrolliga tsükkel

Näide – kolmega jaguvate arvude trükkimine

```
i = 0
while (i < 10):
    if i % 3 == 0:
        print(i)
    i += 1
```

Eelkontrolliga tsükkel

Näide – faktoriaal

```
n = 5
fact = 1
while n > 1:
    fact *= n
    n    -= 1
print('5! =', fact)
```

Eelkontrolliga tsükkel

- Reeglina tuleb *tsüklimuutujat* tsüklikehas muuta selliselt, et "vahe" tsüklitingimusega väheneks.
- Vastasel korral võib tsükkel mittetermineeruda.

Eelkontrolliga tsükkel

- Reeglina tuleb *tsüklimuutujat* tsüklikehas muuta selliselt, et "vahe" tsüklitingimusega väheneks.
- Vastasel korral võib tsükkel mittetermineeruda.

Näide – lõpmatu tsükkel (1)

```
i = 0
while (i < 10):
    print(i)
```

Eelkontrolliga tsükkel

- Reeglina tuleb *tsüklimuutujat* tsüklikehas muuta selliselt, et "vahe" tsüklitingimusega väheneks.
- Vastasel korral võib tsükkel mittetermineeruda.

Näide – lõpmatu tsükkel (2)

```
i = 0
while (i < 10):
    print(i)
    i = i - 1
```

Tsükli kontrolldirektiivid

- Eelkontrolliga tsüklis kontrollitakse jätkutingimust enne igat tsüklikeha täitmist.
- Mõnikord on loomulikum kontrollida tingimust tsüklikeha keskel, lõpus või isegi mitmes kohas.
- Tingimusi saab kontrollida if-lausega ning tsüklikeha täitmist on võimalik katkestada kontrolldirektiividega **break** ja **continue**.
- Käsk **break** lõpetab koheselt tsükli täitmise ning programm jätkab tsüklile järgneva lause täitmisega.
- Käsk **continue** lõpetab tsüklikeha täitmise ning täitmist jätkatakse tsüklitingimuse kontrollimisega; kui see on tõene, siis jätkatakse tsükli täitmist edasi.
- NB! Üksteisesse sisestatud tsüklite korral mõjutavad käsud **break** ja **continue** ainult sisemise tsükli täitmist.

Tsükli kontrolldirektiivid

Näide – sisendi korrektsuse kontrollimine (v. 1)

```
while True:
    arv = int(input("Sisesta positiivne täisarv: "))
    if arv > 0:
        break
    print("Sisestatud arv ei olnud positiivne!")
```

Erindid

Erindid

- Lihtne try-lause:

  try:

  statement1

  except:

  statement2

- Kõigepealt täidetakse lause (lausete plokk) *statement1* .
- Kui selle täitmine õnnestus edukalt, siis on ka kogu try-lause täitmine lõpetatud.
- Kui selle täitmisel tekkis viga, siis täidetakse lause (lausete plokk) *statement2*.

Erindid

Näide – sisendi korrektsuse kontrollimine (v. 2)

```
while True:
    try:
        arv = int(input("Sisesta positiivne täisarv: "))
    except:
        print("Sisestatud tekst ei olnud arvuks teisendatav!")
        continue
    if arv > 0:
        break
    print("Sisestatud arv ei olnud positiivne!")
```

Järgmiseks korraks

- Lugeda läbi õpiku peatükid:
  - (Ptk. 3 *"Tingimus- ja korduslaused"*)
  - Ptk. 5 *"I osa kokkuvõte"*
- Kui lugemisel tekkis küsimusi, mille kohta soovite järgmises loengus vastust, siis need võib saata hiljemalt esmaspäeva lõunaks mailiga varmo.vene@ut.ee ja/või helle.hein@ut.ee.

Suur tänu osalemast

ja

kohtumiseni!