Muutujad ja andmetüübid

Muutuja *(variable)* on mäluaadrssiga määratud koht arvuti mälus, mis sisaldab teadaolevat või ka tundmatut hulka informatsiooni (nn väärtust) ja millele saab tavaliselt viidata nime (identifikaator) abil. Muutuja mõistet kasutatakse ka matemaatikas, kus tal on pigem abstraktne tähendus – mingi märk või üksik täht, mida seostatakse väärtusega.

Muutujal IT mõttes on nimi, mäluaadress, andmetüüp ja väärtus.

Nime (*name, identifier*) järgi saab programmeerija muutujaga „suhelda“, st sinna väärtust hoiule panna (omistada) või seal hoitavat väärtust küsida-kasutada. Nime kaudu suhtleb programmeerija arvuti muutmälus olevate mälupesadega, seosed muutuja nime ja tema väärtuse paiknemiskoha vahel tekitatakse siis, kui konkreetne muutuja kasutusele võetakse.

Mäluaadress (*memory adress*) on reaalne kohta muutmälus, kus muutuja väärtus programmi töö ajal tegelikult asub. Mitmed programmeerimiskeeled annavad võimaluse ka aadressi kaudu muutuja väärtusele ligi pääseda, aga see on üsna ebaturvaline lähenemine. Mõistlikum on nime ja aadressi vahelise seose „meeles pidamine“ jätta arvuti hooleks.

Andmetüüp (*data type*) määrab, milliseid väärtuseid saab vastavat tüüpi muutujas talletada. Kui tahame arvutada, siis tuleb kasutada arvandmeid; kui tahame tekste meeles pidada ja nendega mingeid operatsioone teha, siis on tegemist tekstiliste andmetega. Pythoni lihtandmetüübid jaotatakse tekstandmeteks, nn stringideks (ka sõne) ja arvandmeteks. Arvud omakorda jagunevad järgmiselt:

- `int` *(integers)* – täisarv
    - `bool` *(Boolean values)* – loogikaväärtus on täisarvu alaliik
- `float` *(floating point numbers)* – ujukomaarv
- `complex` *(complex numbers)* – kompleksarv – koosneb kahest ujukomaarvust – nn reaal- ja imaginaarosast.

Andmetüübid on Pythonis dünaamilised *(dynamically typed)*, st et eelnevat muutujate deklareerimist ei nõuta. Muutuja „tekib“ siis, kui talle antakse väärtus ning muutuja tüüp lähtub omistatud väärtusest. Samas ei ole hea praktika kasutada sama nimega muutujat kord täisarvu ja siis hoopis ujukomaarvu hoidmiseks. Igakordsel väärtuse omistamisel paigutatakse muutuja mälus uude kohta.

Väärtus (*value*) on suurus/andmed, mida muutuja „sees“ meeles peetakse; väärtus peab sobima kokku andmetüübiga. Muutuja kasutusele võtmiseks tuleb muutujale omistada tema tüübiga sobiv väärtus. St kui soovid kasutada muutujat ujukomaarvude hoidmiseks, tuleks talle alguses ka ujukomaarv väärtuseks anda.

Muutujate nimed

Muutujatele sobivate nimede valimine on programmeerija jaoks oluline osa programmi kavandamisel. Ühelt poolt seab kasutatav programmeerimiskeel tehnilised piirangud, teisalt on mitmesuguseid soovitusi, mida järgida tuleb. Soovitused võivad erineda nii keeleti (keelte kasutuse kohta on koostatud üldisi Stiiliraamatuid *(style guide)*) kui ka firmade lõikes – st tarkvarafirma on kehtestanud "mängureeglid" oma projektidele. Selliste reeglite eesmärk ei ole kedagi ahistada, vaid tõsta programmikoodi inimloetavust kõigi töötajate-programmeerijate jaoks.

Pythonis kehtivad muutujanimedele järgmised reeglid (mis on üsna tüüpilised ja levinud ka teistes keeltes):

Nimi, üldisemalt identifikaator, koosneb **tähtedest, numbritest** ja **allkriipsudest.** Esimene sümbol peab olema täht või allkriips *(alphabetic)*, sellele võivad järgneda nii **allkriipsud**, **tähed** kui **numbrid** *(alphanumeric)*. Python on **tõstutundlik** *(case sensitive)*, st suur- ja väiketähed on erinevad (muutujanimed arv ja Arv on erinevad). Oluline on ka see, et tähti tuleks valida nn ladina tähtede hulgast (`a..z` ja `A..Z`), mis teisisõnu tähendab, et näiteks tähed õ, ä, ö ja ü ei ole soovitatavad. Sõltuvalt keele- ja kultuuritaustast võivad nad osutuda lausa ohtlikeks.

Üldine keelest sõltumatu soovitus nimede kohta on nende arusaadavus. Pane muutujale selline nimi, mis annab märku sellest, mis andmeid seal hoitakse. St ära kasuta nimesid `aaa`, `bbb` ja `ccc`, vaid `palk`, `maks`, `nimi` jne. Ühetähelised nimed on tavaliselt sobimatud. Loomulikult võime teha erandeid matemaatilistele üldlevinud tähistustele. Kõige tähtsam tingimus on ikka arusaadavus. Ja seda mitte ainult koodi kirjutaja jaoks. Veel üldisem nõue suuremas tarkvaraarenduses on nimede kirjutamine inglise keeles, mis aitab reeglina koodi loetavust parandada. Siiski siin kursusel ma ingliskeelseid nimesid ei nõua, sest me alles alustame programmeerimise õppimist. Aga loomulikult ka ei keela nende kasutamist.

Nimesid ei kasutata vaid muutujate tähistamiseks, vaid ka mitmete teiste programmi osade jaoks (nt alamprogrammid, konstandid jms). Nime üldisemalt kutsutakse **identifikaatoriks** *(identifier)* ja tema kasutamise reeglid ühtivad eelneva

jutuga.

Omistamine

Muutujad saavad oma väärtusi kas **sisendlausest** *(input statement)* (klaviatuurilt trükituna või failist loetuna) või **omistuslausetest** *(assignment)*. **Omistusmärgiks** on Pythonis =.

Näiteid Pythoni omistuslausetest:

```
arv = -12
sona = "kaalikas"
komaga_arv = -3.1415 * (5.0**2)
uus_string = "politsei" + "koer"
```

Python toetab ka nn **täiendatud omistamist** *(augmented assignment)*, kus omistusmärgile lisatud tehtemärk näitab muutust sama muutuja väärtuses. Näiteks:

`nimede_loendur = nimede_loendur + 1` tähendab, et hetkel kehtivat `nimede_loendur`-i väärtust suurendatakse 1 võrra ja tehte tulemus säilitatakse samas muutujas.

`nimede_loendur += 1` on täiendatud omistamine ja tähendab seda sama.

Täiendatud omistustehted on järgmised:

`+=  -=  *=  /=  %=  **=`

Lubatud on ka **mitmene omistamine** *(multiple assignment)*. See tähendab, et mitmele muutujale omistatakse sama väärtus. Näide:

```
x = y = z = 1
```

Seda võimalust ei tasuks üleekspluateerida, kuna selle all programmi loetavus kannatab ja tegemist ei ole programmeerimiskeelte jaoks tüüpilise võimalusega. Abimuutujate (ei sisalda olulisi andmeid) algväärtustamiseks võib seda siiski sobivaks pidada.

Andmetüübid

Eespool oli juba mainitud, et lihtandmetüübid võib jaotada kahte suuremasse klassi: arvandmed ja tekstandmed. Arvandmetega saab reeglina arvutada ja tekstandmeid muul moel töödelda.

Arvude hulgas eristatakse täisarve *(integer)* ja ujukomaarve *(floating point numbers)*.

Täisarvud on teisisõnu komakohtadeta arvud. Teisendusfunktsioon, mida peale sisestamist (või ka muul puhul) kasutada, on `int()`.

Ujukomaarve salvestatakse tüvenumbreid ja nn kümneastmeid kasutades. Seda teemat täpsustatakse eraldi loengus. Teisendusfunktsiooniks on `float()`.

Lisaks on võimalik Pythonis kasutada **kompleksarve** *(complex)*. Sellist tüüpi muutujas salvestatakse reaalosa ja imaginaarosa eraldi ujukomaarvudena. Kompleksarve esitatakse samuti kahes osas, kasutades imaginaarosa näitamiseks sufiksit `j`.

Näide kompleksarvu omistamisest:

```
kompleksarv = 1.5+0.5j
```

Reaal- ja imaginaarosa saab kompleksarvust `kompleksarv` kätte järgmiselt: `kompleksarv.real` ja `kompleksarv.imag`.

String on andmetüüp, mis lubab salvestada suvalist teksti. Stringieraldajatena (alguse ja lõpu märkidena) kasutatakse ülakomasid (`'Maali'`) või jutumärke (`"Maali"`). Kui stringi sees on vaja kasutada ülakoma, siis kasutatakse tema varjestamiseks **paomärki** *(escape character)* \. Näiteks: et trükkida välja `Maali't` tuleks string esitada nii: `'Maali\'t'`. Erisümbolina võib stringi sees kasutada ka reavahetuse märki, milleks on `\n`, nagu tavaks C-tüüpi keeltes.

Stringe saab liita + märgiga ja korrata täisarv korda * märgiga.

Näiteks (-> märgiga on näidatud tekkiv väärtus, see ei kuulu Python-keele õigekirja hulka):

```
"politsei" + "koer" -> "politseikoer"
4 * "hiir" -> "hiirhiirhiirhiir"
```

Stringis olevatele sümbolitele saab ükshaaval indeksi abil ligi. Esimese sümboli indeksiks on 0. Näiteks

```
sona = 4 * "hiir"
sona[4] -> "h" (teise hiire algustäht)
```

Samal viisil saab kätte ka mitu järjestikust sümbolit:

```
sona[0:2] -> "hi"
```

Standardfunktsioon `len()` leiab ja tagastab stringi pikkuse:

```
len(sona) -> 16
```

Stringide töötlemiseks on kirjeldatud mitmeid funktsioone. Mõnedega neist tutvume eraldi praktikatunnis.