Vead ja erindid

Viga *(error)* võib programmi kirjutades tekkida erinevatel hetkedel ning vastavalt sellele jaotatakse klassikaliselt vead kolme klassi.

Süntaksiviga

Süntaksiviga *(syntax error)* – programmeerija poolt tehtud õigekirjaviga, kus eksitakse keelekonstruktsioonide kasutamise vastu (ka puuduvad erisümbolid), tehakse „näpukaid“ keelesõnade kirjutamisel, jäetakse muutujaid deklareerimata/algväärtustamata jne. Need vead on reeglina avastatavad enne programmi käivitamist leksilise- või süntaksianalüsaatori poolt (mis töötas interpretaatori ja kompilaatori koosseisus). Kompileeritavatel keeletel ei teki masinkoodis faili seni, kuni süntaksivead pole kõrvaldatud. Interpreteeritavatel keeltel võidakse täita osa programmist enne vea avastamist. Pythoni programm ei tohiks ka enne käivituda, sest baitkoodi loomiseks peaks programmi lähtetekst juba veavaba olema. Süntaksivigu aitab kompilaator ja interpretaator avastada, kuid parandamise kohustus jääb programmeerijale. Et tavaliselt väljastatakse ka veateade, tasub neid lugeda ja meeles pidada. Alati ei pruugi teade siiski 100% adekvaatne olla.

Semantikaviga

Semantika vead *(semantic error)* – need on teisisõnu loogikavead. Sel juhul on programmeerija oma mõttes ja algoritmis probleemi valesti lahendanud. Võib ütelda, et programm töötab, kuid ei tee päris seda, mida ta tegema peaks. Ka siin võib olla tegemist näpukaga (nt unustatakse tehete järjekorra määramiseks olulised sulud, kasutatakse vale valemit), kuid mõni loogikaviga võib olla väga raskesti avastatav – kui juba kogu algoritmi kavandamine on olnud vale. Neid vigu aitab avastada testimine – programmi käivitamine erinevate lähteandmetega ja tulemuste kontroll.

Täistmisaegne viga

Täitmisaegsed vead *(runtime error)* – programmi täistmise ajal tekib viga, mis põhjustab programmi töö katkemise. Need vead on tihti seotud programmi töö jooksul tekkivate muutujate väärtustega. Tüüpilised näited oleksid nulliga jagamine, ruutjuur negatiivsest arvust, ebasobivat tüüpi sisend, puuduv fail jms. Seda tüüpi vigu tuleb programmi kirjutades ette näha ja kirjutada programmi sisse vigade töötlus, et kasutajal oleks „pehmem maandumine“. Nende vigade töötlemise osas on ka kõige rohkem muutunud põhimõtted keelte arenedes.

Vead Pythonis

Pythoni ametlik materjal jaotab vigu kahte klassi:

Süntaksiviga

Need on igasugused õigekirja vead, kuid mitte muutujate algväärtustamise, andmetüüpide kokkusobimise jms sorti vead (võrdluseks eespool toodud süntaksivigade kirjeldusele).

Erind / erand (exception)

Eestikeelse mõistena on kasutusel **erand**, **erandolukord** vastavalt *vallaste.ee*-le. Aga ka sõna **erind**. Siia liigituvad vead, mis Pythonis avastatakse programmi täitmise ajal, kuid see ei kattu päris eespool kirjeldatud täitmisaegsete vigadega. Mõnede näidetena võiks tuua, et erindiks on ka sobimatute andmetüüpide koos kasutamine ja muutuja algväärtustamise (sisuliselt muutuja deklaratsiooni) puudumine. Kompileeritavate keelte puhul tuvastatakse viimased just kompileerimise käigus (kui seda on oluliseks peetud).

Kui programmi täidetakse, siis võib mõne lause täitmine põhjustada **erindi tingimuse** *(exception condition)*. Kui interpretaator avastab ja tunneb ära vea, siis ta **loob erindi** *(raising, triggering, throwing or generating an exception)*. See tähendab, et ta annab käsuvoole (käskude täitmise järjekorrale) teada, et midagi on viltu. Kui programmeerija on programmis mingi erinditöötluse ettenäinud, käivitub see ja programm ei lõpeta veaga oma tegevust. Erindi töötlemise käigus võib vastavalt olukorrale teha väga erinevaid toiminguid alates ignoreerimisest ja lõpetades programmi katkestamisega (vea logimine, situatsiooni parandamine, võimalus otsast alustada jms).

Erindid on erinevat tüüpi ja igal tüübil on oma nimi. Veateates kuvatakse vastav nimi (näiteks: TypeError, NameError, ZeroDivisionError). Aadressilt http://docs.python.org/3.1/library/exceptions.html on leitav **standard-erindite** *(built-in exceptions)* täielik loetelu. Tegemist on Pythoni ametliku dokumentatsiooniga.

Võimalike vigade suhtes tuleb oma programmi kindlustada.

Klassikalisel juhul toimub täitmisaegsete vigade vältimine selliselt, et vea võimalikkust kontrollitakse enne veaolukorra tekkimist ja ei lubata täita vigu põhjustavaid lauseid (nt nulliga jagamine). Erindite püüdmise loogika on vastupidine – lastakse arvutil täita võimalikku viga põhjustav lause (nt jagamistehe) ja seejärel vaadatakse, kas viga tekkis või mitte. Klassikalist varianti saab teha suvalise keele abil, erindite püüdmiseks peavad keeles olema erivahendid. Pythonis „püütakse erindeid“ `try`-lause abil ja `except`-osas kirjeldatakse programmi käitumine veaolukorra puhul.

```
try:
    lause[d], mille täitmisel võib viga tekkida (kuid võivad olla ka
    laused, kust viga ei teki
except veatüübid:
    laused, mida tehakse siis, kui selline viga tekkis
[except veatüübid:
    laused, mida tehakse siis, kui teised vead tekkisid
...]
```

`except`-osa kutsutakse ka **erindi töötlejaks** *(exception handler)*

Kuidas **`try .. except` lause** käitub:

1. Kui `try`-osas viga ei tekkinud, täidetakse kõik antud osas olevad laused, `except`-osast hüpatakse üle ja programmi täitmine jätkub järgneva lausega
2. Kui `try`-osas tekkis viga, siis selles osas rohkem lauseid ei täideta, vaid minnakse `except`-ossa ja kui veatüüp sobib, täidetakse antud osas olevad laused. Programmi täitmine jätkub peale `try ... except`-lauset.
3. Kui `try`-osas tekkis viga, kuid except-osas ei ole sellise erindi jaoks tegevusi ettenähtud (ja antud `try` ei sisaldu teises `try`-lauses, millest abi oleks), siis lõpetab programm töö veateatega.

Lauses võib `except`-osi olla mitu, mis kirjeldavad erinevad käitumised erinevate erindite jaoks. Ühes `except`-osas võib olla loetelu erinditest, sel juhul täidetakse laused kõigi erindite jaoks. Kui erinditest on loetelu, tuleb see panna sulgudesse ja üksteisest eraldada erindid komadega.

Näide nulliga jagamisel tekkiva vea püüdmisest (jagamine_erindiga.py):

```
jagatav = eval(input("Sisesta jagatav "))
jagaja = eval(input("Sisesta jagaja "))
try:
    jagatis = jagatav /jagaja
    print("Vastus on ", jagatis)
except ZeroDivisionError:
    print("Nulliga jagamine ei ole lubatud.")
```

Valik erinditest

Järgnevalt mõned näited erinditest koos nende ametlike nimede ja tekkepõhjustega:

`NameError`

Püütakse kasutada deklareerimata / algväärtustamata muutujat, st muutujate tabelist sellist nime ei leitud

`ZeroDivisionError`

Katse jagada nulliga. Kehtib nii täis- kui ujukomaarvude jaoks.

`IndexError`

Kasutatav indeks on jada (massiivi) piiridest väljas, st sellise indeksi abil püütakse väärtust küsida.

`ValueError`

Tekib siis, kui standartfunktsioon saab argumendiks muutuja, mille tüüp või väärtus ei ole sobiv (nt negatiivne arv ruutjuure leidmisel)

`OverflowError`

Tekib, kui arimeetikaoperatsiooni tulemus on esitamiseks liiga suur. Peamiselt kontrollitakse täisarvude juures.

Kogu erindite temaatika on tunduvalt laiem, lubades ka näiteks ise uusi erindeid kirjeldada ja kasutada.