18. Napake, Izjeme. Ukazi: try, except, raise.

18.1. Odpravljanje napak pri programiranju; Napake

Pri vsakem delu, tudi pri programiranju, delamo napake. S tem moramo računati in biti na napake pripravljeni. Ena pomembnejših veščin programiranja je veščina odpravljanja napak. Napake so zoprne, ker jih ne moremo predvideti. Če bi jih predvideli, bi se jim lahko izognili. Zavedati se moramo, da lahko pri odpravljanju obstoječih napak vnašamo nove napake.

Daljši in bolj ko je zapleten program, več napak lahko pričakujemo in teže jih odpravimo. Če napak ne odpravljamo sproti, jih je lahko na koncu preveč in se v njih utopimo, program pa kljub nenehnemu odpravljanju napak nikoli ne deluje prav.

18.2. Osnovni tipi napak

Napake v programu lahko v grobem razdelimo v tri skupine:

- Napake pred izvajanjem
- Napake med izvajanjem
- Logične oz. vsebinske napake

18.3. Opomba: Ta pogled na napake je zelo poenostavljen, vendar nima smisla prezgodaj skrbeti za vse detajle.

18.4. Napake pred izvajanjem (sintaktične napake).

Napake pred izvajanjem so sintaktične. Sintaktična napaka se zgodi, če naš program ni napisan v skladu s sintakso (skladnjo) jezika. Takšno napako zazna tolmač še preden se progam izvede, saj programa ali njegovega dela ne razume in ga tudi ne zna interpretirati. Vzroki sintaktičnih napak so lahko različni. Npr.

- Nepoznavanje sintakse jezika
- Uporaba rezerviranih besed v duge namene.
- Napačna uporaba tabulatorjev
- Zatipk
- …

18.5. Napake med izvajanjem (semantične napake)

To je katerakoli napaka, zaradi katere se delovanje sintaktično pravilnega programa med tekom prekine. Tipična napaka med izvajanjem je napaka, pri kateri uporabimo vrednost spremenljivke preden ji kakšno vrednost priredimo ali pa pokličemo funkcijo na način, ki ga ob definiciji nismo predvideli.

18.6. Logična oziroma vsebinska napaka (pragmantična napaka)

Če sintaktično pravilni program deluje, vendar ne vrne rezultata, ali pa je rezultat napačen, je prisotna logična napaka. Z drugimi besedami: vsebinska napaka

pomeni, da je napaka že v algoritmu ali pa gre na napako v programiranju, ki ni povzorčila izjeme, ampak le napačno računanje.

18.7. Defenzivno programiranje

Najboljša strategija za odpravljanje napak je defenzivno programiranje. Pri defenzivnem programiranju poskušamo vnaprej zmanjšati možnost, da bi se pojavila kakšna zahrbtna napaka. To med drugim pomeni, da damo napaki čimprej priložnost, da se pokaže, sami pa si zmanjšujemo možnosti, da bi kasneje vpeljevali napake.

18.8. Nekaj nasvetov za defenzivno programiranje

1. Program sproti skrbno komentiramo. S tem ohranjamo pregled nad programom tudi na daljši časovni rok. Če delamo v skupini, komentarji pomagajo drugim članom skupine razumeti naše razmišljanje ob programiranju.
2. Izogibamo se nepotrebnim in zavajajočim komentarjem.
3. Programe pišemo v majhnih sklopih, vsak sklop takoj preverimo.
4. Uporabljamo pravilo 7 vrstic. Sklop naj bi ne bil dosti daljši od sedmih vrstic in naj po možnosti s komentarji vred ne seže čez en zaslon.
5. Pri pisanju funkcije preverimo, ali formalni parametri zadoščajo deklariranim pogojem. Ne zaupamu tistemu, ki bo funkcijo poklical.
6. Pri klicu funkcije preverimo, ali vrednosi dejanskih parametrov zadoščajo deklariranim pogojem. Ne zaupamo tistemu, ki je funkcijo napisal.
7. Na kritičnih mestih si izpisujemo vrednosti spremenljivk. Ko smo z izdelkom zadovljni, lahko testne stavke print zakomentiramo.
8. Pri daljših programih si lahko pomagamo s posebno spremenljivko, ali parametrom, ki uravnava “zgovornost” programa ali funkcije.
9. Pri programiranju računamo, da bomo morali program pisati dalj časa z daljšimi prekinitvami.
10. To pomeji, da bo zanami program bral ne le računalnik, ampak tudi človek,
11. Spremenljivke poimenujemo tako, da laho njihov pomen kasneje zlahka razberemo.

18.9. Odpravljanje sintaktičnih napak

Pri programiranju uporabljamo samo tiste konstrukte jezika, ki jih dobro razumemo. Tolmač točno pokaže mesto sintaktične napake in izpiše pojasnilo zanjo. Če nam ne uspe odpraviti napake, naredimo kopijo programa v novi datoteki in eksperimentiramo z njo toliko časa, dokler ne spoznamo vzroka za napako. To pomeni, da v kopiji postopoma brišemo stavke, ki na napako ne morejo vplivati.

Druga možnost je, da poskušamo v novi, testni datoteki rekreirati isto napako.

Odpravljanje napak z ugibanjem ni priporočljiva metoda, saj brez razumevanja vzroka za napako kasneje, v podobnih situacijah, ne bomo nič na boljšem

V skrajnem primeru sumljivi del programa sprogramiramo ponovno na drug način.

18.10. Odpravljanje napak med izvajanjem

V Pythonu delovanje programa poteka "gnezdeno". To pomeni, da se z vsakim klicem funkcije spustimo eno raven niže. Tolmač nam v primeru napake med izvajanjem pojasni, kako je potekalo gnezdeno klicanje funkcij tako, da je na vrhu vrstica glavnega programa, ki je sprožila klic prve funkicje, spodaj pa vrstica najgloblje vgnezdene funkcije, kjer je do napake (izjeme) res prišlo. To pomeni, da moramo napako iskati od spodaj navzgor in je najpomembnejše spodnje sporočilo.

18.11. Odpravljanje logičnih napak

Vsebinske napake so najtrši oreh, saj nam tolmač ne nudi nobene pomoči, ker ne ve, kaj nas muči, saj je program sintaktično pravilen in lepo teče do konca, samo rezutat ni pravi. Obstajajo posebna orodja (razhroščevalniki), s katerimi si lahko pomagamo pri odkrivanju napak, vendar za učenca, začetnika niso najbolj primerna, saj ga obremenjujejo z dodatnim učenjem in mu jemljejo koncentracijo.

18.12. Sledenje programa

Sledenje programa je veščina, s katero "peš", torej s svinčnikom in papirjem simuliramo delovanje programa. Sledenje programa je kakor igra, v kateri poskušamo napovedati vrednosti, ki jih zavzamejo nekatere spremenljivke v določenih fazah izvajanja programa, potem pa to preverimo npr. z vstavljanjem stavkov print na primernih mestih, ali pa s klici posameznih funkcij s primernimi vrednostmi parametrov. Programirati znamo, če znamo brati delujoč program. Pomeni, da ga znamo pravilno slediti.

S sledenjem programa bi lahko vsaj teoretično našli poljubno logično napako, saj moramo le najti prvi trenutek, ko se neka vrednost spremenljivke ne ujema s tisto, ki smo je ob programiranju predvideli.

Običajno pa je dosledno sledenje programa prezahtevno in bi zanj potrošili preveč časa. Zato si raje pomagamo drugače.

18.13. Vstavljanje stavkov print

Če v daljšem programu pride do nepričakovane napke, si lahko pomagamo z vstavljanjem začasnih stavkov print. Z njimi poskušamo določiti mesto v programu, kjer se pojavi prva napaka. Pri tem je zelo učinkovita metoda "bisekcije", ko začnemo nekje na sredini, potem pa se v odvisnosti od natisnjenih vrednosti pomikamo proti mestu, kjer je nastala napaka.

18.14. Sklep

Napake so sestavni deli programiranja, zato je nujno, da jih znamo odkrivati in odpravljati.

Brez ustrezne prakse je to nemogoče doseči.

Najtežje prepoznavamo tiste napake, ki nastajajo zaradi napačnega mentalnega modela v naših glavah, ki odstopa od realnosti.

Pri učenju programiranja se zato metodi “poskusi in popravi” žal ne moremo izogniti.

18.15. Zlato pravilo: nad vsako napako posebej!

Pogosto imamo opravka z več kot eno napako. Ko odpravljamo napake, odpravljamo vsako posebej in vsakič preverimo, ali smo napako res odpravili, ali pa je bil popravek morda napačen.

18.16. Zlato pravilo: Preizkušamo z majhnimi vrednostmi

Napako v delovanju funkcije poskušamo najti tako, da je število ukazov, ki se v njej izvedejo, kar najmanjše. To pa se običajno zgodi pri majhniv vrednostih parametrov.

18.17. Zlato pravilo: vsak parameter posebej

Delovanje funkcije, ki je odvisna od več parametrov, preverjamo tako, da variiramo vsak parameter posebej.

18.18. Izjeme. Stavki try, except, raise, else, finally.

Python pozna mehanizem, ki lovi napake. Običajno se pri napaki izvajanje programa prekine. Najbolj preprosto varianto prikazuje naslednji primer pri katerem v okolju try: lahko pride do napake, ki jo "servisiramo" v okolju except. Za bolj kompliciranе uporabe tega mehanizma boste morali pogledati ustrezno dokumentacijo.

Tu gre v veliki večini primerov za napake med tekom programa, ki so izven območja, ki ga obvladuje programer in jih povzroči "okolica", oz. uporabnik programa. Npr. dobimo slabe podatke, v pomnilniku zmanja prostora,podatke hočemo brati z datoteke, ki tisti trenutek ni na voljo...

```
a = 1
b = 0
# delimo a/b in lovimo izjemo 1/0.
try:
     x = a/b
     print ("Ta stavek se ne bo natisnil.",x)
except:
     print("deljenje z 0 je prepovedano.")
     raise
print("Odstranite stavek raise in poskusite znova.")
```

Z ukazom raise lahko povzročimo napako, tudi če je ni bilo.

V resnici imamo lahko več stavkov `except`. Vsak except lahko nosi tudi informacijo o izjemi, ki jo ulovi. Na koncu je možno uporabiti še stavka `else` in `finally`. Stavek `else` se izvede, če ne pride do izjem, stavek `finally` pa v vsakem primeru.

18.19. Testiranje.

Vsak resen programski produkt moramo pred objavo testirati in odpravljati skrite napake. Za trstiranje ima python na voljo posebna orodja že v standardni knjižnici. Zelo enostaven za uporabo je modul `doctest`. Malo bolj zahteven je modul `unittest`.