Kurs: Programiranje igara sa programskim jezikom Pajton
004 čas: Program Pogodi_broj.py
Teme: grananje, while petlja, logički operatori, planiranje programa

Kod za program Pogodi_broj.py

```
#Pogodi_broj
#Kompjuter bira slucajan broj izmedju 1 i 100
#igrac pokusava da pogodi broj a kompjuter obavestava
#igraca da li je pokusaj preveliki ili premali ili je pogodio
import random
print("\tDobrodosli u 'Pogodi broj'!")
print("\nJa cu zamisliti broj izmedju 1 i 100.")
print("Pokusaj pogoditi u sto manje pokusaja.\n")
#postavljanje inicijalnih vrednosti
broj = random.randint(1, 100)
broj_igrac = int(input("Pokusaj da pogodis: "))
pokusaj = 1
#petlja za pogadjanje
while broj_igrac != broj:
    if broj_igrac > broj:
        print("Nize...")
    else:
        print("Vise...")
    broj_igrac = int(input("Pokusaj da pogodis: "))
    pokusaj += 1
print("Pogodio si! Tacan broj je", broj)
print("A trebalo je samo", pokusaj, "pokusaja!\n")
```

001 009 003 005 007

Generisanje slučajnih brojeva

program Bacanje_kockica.py

```
#Bacanje_kockica
#Prikazuje kreiranje slucajnog broja
import random
#generise slucajan broj izmedju 1 - 6
kocka1 = random.randint(1, 6)
kocka2 = random.randrange(6) + 1
zbir = kocka1 + kocka2
print("Dobili ste", kocka1, "i", kocka2, "sto daje zbir od", zbir)
```

```
Dobili ste 6 i 5 sto daje zbir od 11
```

Prva linija koda sadrži import iskaz. Iskaz omogućava importovanje ili učitavanje, u ovom slučaju modula random: import random.

Moduli su fajlovi koji sadrže kod koji treba da se koristi u drugim programima. Ovi moduli obično udružuju kolekcije povezanih programa po oblasti. Modul random sadrži funkcije koje su povezane sa generisanjem slučajnih brojeva i realizacijom slučajnih rezultata.

001 Funkcija randint()

Modul random sadrži funkciju, randint(), koja pravi slučajan ceo broj. U programu se koristi: random.randint(1, 6)

Vidi se da program ne poziva direktno randit() već preko modula random. Obično, može se pozvati funkcija iz importovanog modula davanjem imena modula, tačke i poziva same fukcije. Ovaj metod pristupa se naziva **dot notacija**. Upotreba dot notacije u prethodnom primeru znači da funkcija randint() pripada modulu random.

Funkcija randint() traži dva celobrojna argumenta i vraća slučajnu celobrojnu vrednost između tih argumenata. Tako da pridodavanjem vrednosti 1 i 6 u funkciju, garantovano se može dobiti ili 1, 2, 3, 4, 5 ili 6. Pošto kocka ima šest mogućih vrednosti to je ono što treba da se desi.

002 Funkcija randrange()

Modul random takođe sadrži funkciju randrange(), koja proizvodi slučajno generisan ceo broj. Postoji nekoliko načina za pozivanje randrange(), ali najjednostavniji je korišćenje jednog pozitivnog celobrojnog argumenta. Pozvana na ovaj način, funkcija vraća slučajan ceo broj iz opsega od 0 (uključujući i 0) pa sve do tog broja (ne uključujući taj broj). Tako da pozivom na random.randrange(6) proizvodi ili 0, 1, 2, 3, 4 ili 5. Pošto je potrebno uključiti i 6 kao moguć broj pri bacanju kocke, samo se doda 1 na izlaz funkcije: `kocka2 = random.randrange(6) + 1`
Sada kocka2 može dobiti kao izlaz ili 1, 2, 3, 4, 5 ili 6.

Rad sa if iskazom

Grananje je osnovni deo programiranja kompjutera. U osnovi ono znači odlučivanje o izboru jedne ili druge putanje. Kroz if iskaz, programi se granaju prema delu koda ili ga preskaču, u zavisnosti od određenih uslova koji su postavljeni.

program Lozinka.py

Ovaj program koristi if iskaz za simulaciju procedure logovanja visoko osiguranog kompjuterskog sistema. Program dopušta pristup korisniku ako je uneo korektnu lozinku.

```
#Lozinka
#Prikazuje upotrebu if iskaza
print("Dobrodosli u System Security Inc.")
print("-- sigurnost je nase srednje slovo\n")
lozinka = input("Unesi tvoju lozinku: ")
if lozinka == "tajna":
    print("Pristup Dozvoljen")
```

```
Dobrodosli u System Security Inc.
-- sigurnost je nase srednje slovo

Unesi tvoju lozinku: tajna
Pristup Dozvoljen
```

003 if iskaz

Ključni deo programa je if iskaz:

```
if lozinka == "tajna":
    print("Pristup Dozvoljen")
```

Sam iskaz se lako može razumeti prostim tumačenjem reči: ako je lozinka jednaka “tajna“, onda se štampa “Pristup Dozvoljen“ i program nastavlja prema sledećem iskazu. Ali, ako nije jednak sa “tajna“, program samo nastavlja prema sledećem iskazu posle if iskaza.

004 Kreiranje uslova

If iskaz ima uslove. Uslov je samo izraz koji je ili tačan (true) ili netačan (false). Pajton ima svoje ugrađene vrednosti koje predstavljaju tačnost ili netačnost. True predstavlja tačnost a False netačnost. Svaki uslov se mora proceniti kao jedno od ova dva stanja. U programu, uslov koji je korišćen u if iskazu je: lozinka == “tajna“. To znači da je lozinka jednaka sa, ili da se odnosi na vrednost, “tajna“. Uslov se procenjuje sa ili True ili False, u zavisnosti od vrednosti u promenjivoj lozinka. Ako je vrednost u promenjivoj lozinka jednaka sa “tajna“, onda je uslov True, inače je False.

005 Operatori upoređivanja

Uslovi su često kreirani upoređivanjem vrednosti. Vrednosti se mogu porediti korišćenjem operatora poređenja. U programu je korišćen jedan od operatora poređenja, a to je operator ekvivalencije(==).

> Operator ekvivalencije se predstavlja kao dva znaka ==. Operator dodele se označava sa =. Rezultat Bulovog izraza sa operatorom ekvivalencije može biti True ili False. Izraz sa operatorom dodele dodeljuje vrednost sa desne strane jednakosti promenjivoj sa leve strane jednakosti.

Relacioni operatori (operatori upoređivanja) su > (veće od), < (manje od), >= (veće ili jednako sa), <= (manje ili jednako sa), == (ekvivalentan sa), != (nije ekvivalentan sa).
Korišćenjem operatora upoređivanja mogu se uporediti int i float brojevi a stringovi bazirano na njihovom alfabetskom redosledu. Isto tako ne mogu se upoređivati tipovi podataka koji nemaju uporedivu definiciju redosleda (npr, 10 i “pet“).

006 Upotreba uvlačenja (indentation) za kreiranje blokova

U telu if iskaza, blok linija se uvlači za jedan tabulator. Blok predstavlja jednu ili više linija koje su uvučene za istu veličinu tabulatora i na taj način čine jednu strukturu.
Blokovi se mogu koristiti kao deo if iskaza. Oni su iskaz ili grupa iskaza koji se izvršavaju ako je uslov ispunjen (True). U programu Lozinka, blok je samo jedna linija sa jednim iskazom:

```
print("Pristup Dozvoljen")
```

Pošto blokovi mogu sadržavati neograničeni broj iskaza, može se dodati nov iskaz:

```
if lozinka == "tajna":
    print("Pristup Dozvoljen")
    print("Dobrodosli! Vi ste sigurno neko veoma bitni!")
```

Upotreba reči else

Kada je potrebno da program nešto izabere bazirano na uslovu, tada se koristi reč else zajedno sa reči if.

program Dozvoljeno_nedozvoljeno.py

Ovo je realnije rešenje programa Lozinka pošto se uzima u obzir i reakcija koda na pogrešno ukucanu lozinku.

```
#Dozvoljeno_nedozvoljeno
#Prikazuje upotrebu reči else
print("Dobrodosli u System Security Inc.")
print("-- sigurnost je nase srednje slovo\n")
lozinka = input("Unesi tvoju lozinku: ")
if lozinka == "tajna":
    print("Pristup Dozvoljen")
else:
    print("Pristup Zabranjen")
```

```
Dobrodosli u System Security Inc.
-- sigurnost je nase srednje slovo

Unesi tvoju lozinku: tajna
Pristup Dozvoljen
```

```
Dobrodosli u System Security Inc.
-- sigurnost je nase srednje slovo

Unesi tvoju lozinku: ne znam
Pristup Zabranjen
```

007 Pogled na else

U programu se koristi jedna if-else struktura:

```
if lozinka == "tajna":
    print("Pristup Dozvoljen")
else:
```

```
    print("Pristup Zabranjen")
```

Ako lozinka nije ekvivalentna sa “tajna“, program štampa poruku “Pristup Zabranjen“. U if-else strukturi samo će se realizovati jedan od blokova koda. Ako je uslov netačan, realizuje se onaj blok koji se nalazi u telu iskaza else. Bitno je da su if i else na istoj visini uvlačenja.

Upotreba elif iskaza

Program može izabrati između nekoliko mogućnosti ako pored if postoji i elif iskaz. Ovaj način se koristi kada postoji jedna promenjiva koja se želi uporediti sa većim brojem različitih vrednosti.

program Kompjuter_raspolozenja.py

Ovaj program reaguje na raspoloženje korisnika i prikazuje korisnikovo raspoloženje.

```
#Kompjuter_raspolozenja
#Prikazuje upotrebu elif iskaza
import random
print("Osecam tvoju energiju. Tvoje prave emocije se prelivaju po mom ekranu.")
print("Ti si...")
raspolozenje = random.randint(1, 3)
if raspolozenje == 1:
    #sretan
    print( \
        """
        -----------
        |         |
        | O     O |
        |    <    |
        |         |
        | .     . |
        |  `...`  |
        -----------
        """)
elif raspolozenje == 2:
    #miran
    print( \
        """
        -----------
        |         |
        | O     O |
        |    <    |
        |         |
        | ------  |
        |         |
        -----------
        """)
elif raspolozenje == 3:
    #tuzan
    print( \
        """
        -----------
        |         |
        | O     O |
        |    <    |
        |         |
        |   .'.   |
        |  '   '  |
        -----------
        """)
else:
    print("Ilegalna vrednost raspolozenja! (Mora da si u zaista losem raspolozenju).")
print("...danas.")
```

```
Osecam tvoju energiju. Tvoje prave emocije se prelivaju po mom ekranu.
Ti si...

       ----------
       |        |
       | o    o |
       |   <    |
       |        |
       |  .'.   |
       |  '.'   |
       ----------

...danas.
```

008 elif iskaz

Jedan if iskaz sa elif iskazima može sadržati niz uslova koje program procenjuje. U programu Kompjuter_raspolozenja, linije koje sadrže različite uslove su:

```
if raspolozenje == 1:
elif raspolozenje == 2:
elif raspolozenje == 3:
```

Vidi se da samo prvi uslov je posle if a svi ostali posle elif iskaza. U programu može postoji neograničeni broj elif iskaza.

Izdvajanjem uslova iz koda, vidi se smisao strukture: testiranje promenjive raspolozenje prema tri različite vrednosti.

> Važna odlika if i elif iskaza kada se jednom uslov ispuni je da kompjuter izvrši odgovarajući blok i izlazi iz iskaza. To najčešće znači da će se izvršiti samo jedan blok iako su možda više uslova ispunjeni. U kompleksnijim programima, moguće je napraviti iskaze sa više od jednim uslovom koji su tačni istovremeno. U takvim slučajevima, izvršiće se samo onaj blok koji pripada prvom iskazu koji je postao tačan.

U programu Kompjuter_raspolozenja, ako se desi da nijedan od uslova nije ispunjen, poslednji else iskaz će se izvršiti i ispisaće se odgovarajuća poruka na ekranu. Međutim, to se nikada neće desiti pošto je kod u programu osigurao da mora biti ispunjen jedan od uslova. Bez obzira na to, postavljen je kod i za else iskaz iako je opcion.

> Iako nije neophodno da se koristi krajnji else iskaz, njegovo postojanje je uvek dobra ideja. On se koristi kao poslednja linija reagovanja kada nijedan od uslova nije ispunjen unutar prethodnih iskaza. Čak i kada programer misli da će neki od uslova uvek biti True, uvek se koristi krajnji else iskaz za prihvatanje “nemogućih“ slučajeva.

Kreiranje ciklusa

Ciklusi razvijaju jednostavnu ideju: sve dok je neki uslov tačan, ponovi obradu podataka.

program Sim_konver_sa_3god.py

```
#Simulator_konverzacije_sa_trogodisnjakom
#Prikazuje upotrebu while ciklusa
print("\tDobrodosli u 'Simulator_konverzacije_sa_trogodisnjakom'\n")
print("Pokusaj zaustaviti ludilo.\n")
odgovor = ""
while odgovor != "zato":
    odgovor = input("Zasto?\n")
print("A. Dobro.")
```

```
        Dobrodosli u 'Simulator_konverzacije_sa_trogodisnjakom'

Pokusaj zaustaviti ludilo.

Zasto?
Haj
Zasto?
Pa, zdravo
Zasto?
Ovaj
Zasto?
Pa...
Zasto?
Aha
Zasto?
zato
A. Dobro.
```

009 while petlja

U programu se koristi while petlja:

```
while odgovor != "zato":
    odgovor = input("Zasto?\n")
```

Vidi se sličnost sa if iskazom (moglo bi se reći da je samo došlo do zamene if službene reči sa while). Nije samo u tome sličnost već i u načinu funkcionisanja. U oba iskaza, ako je uslov tačan, blok obrade podataka (u slučaju petlje se naziva telo petlje) se izvršava. U while iskazu, kompjuter testira uslov i konstantno izvršava telo petlje , sve dok uslov ne postane netačan.

U programu telo petlje se sastoji samo od jedne linije koda: `odgovor = input("Zasto?\n")` koja će se izvršavati sve dok se ne unese odgovor: zato. Tada, program izvršava sledeći iskaz: `print("A. Dobro.")`

010 Inicijalizacija stražar promenjive

Često se while petlje kontrolišu pomoću stražar (sentry) promenjive, promenjive koja se koristi u uslovu i upoređuje sa nekim vrednostima. Stražar promenjiva pomaže u formiranju barijere oko tela while petlje. U programu, stražar promenjiva je odgovor. Koristi se u uslovu i upoređuje sa stringom “zato“ svaki put pre nego se izvrši telo petlje.

Bitno je inicijalizovati stražar promenjivu. Najčešće se stražar promenjive inicijalizuju baš pre same petlje. U programu je linija pre petlje: `odgovor = ""`.

Često je dobro inicijalizovati stražar promenjivu na neku vrstu prazne vrednosti. U programu je to prazan string. Inače se postavljaju granične vrednosti kao 0, -1 ili ‘q’ (quit).

011 Provera stražar promenjive

Trebalo bi osigurati da while uslov proceni na vrednost True u nekom trenutku; inače, telo petlje se neće nikad izvršiti:

```
odgovor = "zato"
while odgovor != "zato":
    odgovor = input("Zasto?\n")
```

Pošto je stražar promenjivoj odgovor inicijalno dodeljena vrednost “zato“, baš pre while petlje, telo petlje se neće nikada izvršiti, jer se uslov za ulazak u petlju nikad neće ispuniti.

012 Updejtovanje stražar promenjive

Kada se jednom uspostavi uslov, inicijalizuje stražar promenjiva i osigura da će se pod nekim uslovima telo petlje izvršavati, aktivna petlja je spremna. Sledeće šta treba uraditi je osigurati da će se petlja u nekom trenutku i završiti.

Ako se napravi petlja koja se nikad neće završiti to se zove beskonačna petlja:

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

Ovde je programer zaboravio da napravi kod koji će menjati vrednost promenjive brojac unutar tela petlje. Dakle, vrednosti uslova se moraju menjati pod uticajem koda unutar tela petlje. Ako se nikada ne menjaju, stvara se beskonačna petlja.

Izbegavanje beskonačne petlje

Jedan tip beskonačne petlje je gde stražar petlje se nikada ne updejtuje. Ali postoje podmuklije forme beskonačne petlje.

program Bitka_poraz.py

Heroj je okružen armijom trolova, scenario iz RPG igara. Program opisuje borbu, udarac po udarac, sve dok heroj ne pobedi trola, ali onda zadobija više povreda. Na kraju, završava se uvek sa smrti heroja. Da li tako i treba?

```
#Bitka_poraz
#Prikazuje beskonacnu petlju
print("Tvoj usamljeni heroj je okruzen armijom trolova.")
print("Njihova raspadajuca zelena tela, prostiru se i spajaju sa horizontom.")
print("Tvoj heroj podize svoj mac u zadnjoj borbi za zivot.\n")
zdravlje = 10
trolovi = 0
ostecenje = 3
while zdravlje != 0:
    trolovi += 1
    zdravlje -= ostecenje
    print("Tvoj heroj zamahuje i pobedjuje zlog trola, " \
        "ali zadobija", ostecenje, "poena ostecenja.\n")
print("Tvoj heroj se borio hrabro i porazio je", trolovi, "trolova.")
print("Ali, tvoj heroj je porazen u borbi.")
```

Desiće se da se startovanjem programa pokreće beskonačna petlja koja se mora prekinuti stopiranjem izvršavanja samog programa.

013 Praćenje rada programa

Na prvi pogled, program ima logičke greške. Dobar način da se otkriju ovakve greške je praćenje (tracing) izvršenja programa. Praćenje znači simulacija rada programa i isto izvršavanje kodova kao što to program radi, praćenjem svakog iskaza i vođenjem računa o vrednostima dodeljenim promenjivima. Na ovaj način se može proći programom, razumeti šta se tačno dešava u svakoj tački programa i otkriti okolnosti koje izazivaju grešku u kodu.

zdravlje	trolovi	ostecenje	zdravlje != 0
10	0	3	True
7	1	3	True
4	2	3	True
1	3	3	True
-2	4	3	True
-5	5	3	True
-7	6	3	True

Ovde se može zaustaviti praćenje jer izgleda da postoji beskonačna petlja. Pošto vrednost zdravlje je negativno (i nije jednako 0) u poslednja tri ciklusa, uslov je i dalje True. Problem je što zdravlje nikada nije palo na 0. Samo je raslo u negativnom smeru svaki put kada se petlja izvrši. Zato, uslov neće nikada postati False i petlja se neće završiti.

014 Kreiranje uslova koji će omogućiti vrednost False

Uz osiguravanje da će se vrednosti u uslovu while petlje promeniti, treba osigurati i da će na kraju krajeva taj uslov postati False; inače, i dalje će postojati beskonačna petlja.

Ispravka u programu: while zdravlje > 0:

Sada je izlaz programa:

```
Tvoj usamljeni heroj je okruzen armijom trolova.
Njihova raspadajuca zelena tela, prostiru se i spajaju sa horizontom.
Tvoj heroj podize svoj mac u zadnjoj borbi za zivot.

Tvoj heroj zamahuje i pobedjuje zlog trola, ali zadobija 3 poena ostecenja.

Tvoj heroj zamahuje i pobedjuje zlog trola, ali zadobija 3 poena ostecenja.

Tvoj heroj zamahuje i pobedjuje zlog trola, ali zadobija 3 poena ostecenja.

Tvoj heroj zamahuje i pobedjuje zlog trola, ali zadobija 3 poena ostecenja.

Tvoj heroj se borio hrabro i porazio je 4 trola.
Ali, tvoj heroj je porazen u borbi.
```

Sada praćenje rada programa pokazuje sledeće:

zdravlje	trolovi	ostecenje	zdravlje != 0
10	0	3	True
7	1	3	True
4	2	3	True
1	3	3	True
-2	4	3	False

Rad sa vrednostima kao uslovima

Ako se postavi pitanje o proceni 35 + 2, logičan odgovor je 37. Ali ako se postavi pitanje o proceni 37, kao tačno ili netačno, odgovor neće biti lak. Ideja o posmatranju bilo koje vrednosti kao ili tačne ili netačne je dopuštena u Pajtonu. Bilo koja vrednost, bilo kojeg tipa, se može smatrati kao tačno ili netačno.

program Francuski_restoran.py

```
#Francuski_restoran
#Prikazuje tretiranje vrednosti kao uslova
print("Dobrodosli u Chateau D' Hrana")
print("Izgleda da smo puni gostiju veceras.\n")
novac = int(input("Sa koliko eura castis sefa sale? "))
if novac:
    print("A, setio sam se slobodnog stola. Ovuda, molim Vas.")
else:
    print("Molim Vas sacekajte. Ovo moze potrajati.")
```

```
Dobrodosli u Chateau D' Hrana
Izgleda da smo puni gostiju veceras.

Sa koliko eura castis sefa sale? 5
A, setio sam se slobodnog stola. Ovuda, molim Vas.
```

015 Tumačenje bilo koje vrednosti kao True ili False

Ovaj koncept se ogleda u sledećoj liniji koda:

```
if novac:
```

Vidi se da novac nije upoređen ni sa jednom vrednosti. Novac jeste uslov. Kada se desi procena broja kao uslova, 0 je False a bilo šta drugo jeste True. Zato je prethodna linija koda ekvivalentna sa:

```
if novac != 0:
```

Prva verzija je jednostavnija i elegantnija i treba je koristiti.

Pravila o tome šta čini vrednost True ili False su jednostavna: bilo šta prazno ili vrednost 0 je False, bilo šta drugačije ima vrednost True.

Namerno kreiranje beskonačne petlje

Ovo su beskonačne petlje koje imaju uslov za prekid rada ugrađen unutar tela petlje.

Program Falicni_brojac

```
#Falicni_brojac
#Prikazuje iskaze break i continue
brojac = 0
while True:
    brojac += 1
    #zavrsi petlju ako je brojac > 10
    if brojac > 10:
        break
    #preskoci 5
    if brojac == 5:
        continue
    print(brojac)
```

```
1
2
3
4
6
7
8
9
10
```

016 Upotreba iskaza break za iskakanje iz petlje

Ako se petlja postavi kao: while True:, to teoretski znači da će petlja raditi večito, osim ako ne postoji izlazni uslov u telu petlje. U programu on postoji:

```
#zavrsi petlju ako je brojac > 10
if brojac > 10:
       break
```

Pošto je brojac povećano za 1 svaki put kada započne novi ciklus petlje, na kraju krajeva će jednom doći do 11. Kada se to desi, aktivira se iskaz break i program iskače iz petlje.

017 Upotreba iskaza continue za skok na vrh petlje

U linijama:

```
#preskoci 5
if brojac == 5:
       continue
```

Iskaz continue označava skok na vrh petlje. Na vrhu petlje, while uslov je testiran i u petlju se ponovo ulazi ako je procenjen uslov na True. Tako da, kada je brojac jednak 5, program ne izvršava print(brojac) iskaz. Umesto toga, ide nazad na vrh petlje tako da se 5 ne prikazuje na ekranu.

018 Razlozi kada da se koriste break i continue

Iskazi break i continue se mogu koristiti u svakoj petlji. Njihovo korišćenje nije ograničeno na upotrebu u namerno napravljenim beskonačnim petljama. Ali, treba ih pametno koristiti. Oba iskaza ne pomažu da se vidi tok petlje i ne pomažu razumevanju po kojim vrednostima se petlja završava.

U Pajtonu, postoje trenuci kada je namerna beskonačna petlja jasnija od tradicionalne petlje. U ovim slučajevima, kada je nesigurno pisati petlju sa običnim uslovima, progameri koriste namerne beskonačne petlje.

Korišćenje složenih uslova

Do sada su se koristila upoređivanja sa tačno dve vrednosti. Oni se nazivaju prosti uslovi. Moguće je kombinovati proste uslove pomoću logičkih operatora pa se kreiraju složeni (compaund) uslovi. Tako program pravi odluke na osnovu upoređivanja više grupa vrednosti.

program Ekskluzivna_mreza.py

Ovaj program simulira elitnu kompjutersku mrežu kojoj samo odabrana nekolicina ima mogućnost pristupa. Članstvo čini samo nekoliko najboljih svetskih programera.

Kao i u realnom svetu, svaka osoba mora uneti korisničko ime (username) i lozinku (password) i svako ima svoj sigurnosni nivo. Član mora uneti oba podatka ili se neće moći logovati (pristupiti) na mrežu. Sa uspešnim logovanjem, član je pozdravljen.

Gosti takođe mogu pristupiti mreži ali njihov sigurnosni nivo je najniži.

```
#Ekskluzivna_mreza
#Prikazuje logicke operatere i slozene uslove
print("\tEkskluzivna Racunarska Mreza")
print("\t\tSamo za clanove!\n")
sigurnost = 0
username = ""
while not username:
    username = input("Username: ")
password = ""
while not password:
    password = input("Password: ")
if username == "Jovan" and password == "tajna":
    print("Cao Joco.")
    sigurnost = 5
elif username == "Betmen" and password == "dzoker":
    print("Hej Betko.")
    sigurnost = 3
elif username == "Bil Gejts" and password == "malo nezno":
    print("Gde si Bili!")
    sigurnost = 3
elif username == "Tramp" and password == "najbolji":
    print("Ajde Doni!")
    sigurnost = 3
elif username == "gost" or password == "gost":
    print("Dobro dosao, gostu.")
    sigurnost = 1
else:
    print("Login nije uspeo. Ti nisi ekskluzivan.\n")
```

```
    Ekskluzivna Racunarska Mreza
            Samo za clanove!

Username: Betmen
Password: dzoker
Hej Betko.
```

```
    Ekskluzivna Racunarska Mreza
            Samo za clanove!

Username: ne znam
Password: gost
Dobro dosao, gostu.
```

```
    Ekskluzivna Racunarska Mreza
            Samo za clanove!

Username: Ja
Password: ne znam
Login nije uspeo. Ti nisi ekskluzivan.
```

019 Logički operator not

Program je takav da se očekuje da korisnik mora nešto da unese kao podatak za username. Ako se samo pritisne ENTER za username javlja se petlja koja nastavlja da traži nekakav unos za username:

```
username = ""
```

```
while not username:
    username = input("Username: ")
```

U while uslovu, koristi se logički operator not. U Pajtonu, stavljanje not ispred uslova kreira nov uslov koji se procenjuje suprotno od originalnog uslova. To znači da not username je True kada je username False, i obrnuto.

Pošto je username inicijalizovan na prazan string u programu, on počinje kao False. To čini da je not username True kada petlja startuje po prvi put. Onda, program dobija vrednost za username od korisnika. Ako korisnik pritisne ENTER, username je i dalje prazan string, pa petlja nastavlja sa radom. To znači, sve dok je username prazno, petlja nastavlja sa radom i korisnik će uvek dobijati poruku da unese username koji je validan.

Pa kada konačno unese nešto, username više nije prazan string što čini da username se proceni na True a not username se proceni na False. Kao rezultat toga, petlja se završava. Isto se dešava i pri unosu za password.

020 Logički operator and

Ako član želi da se loguje na ekskluzivnu mrežu, član mora uneti username i password koji se prepoznaju. Ako se prepozna samo jedno, član se neće prepoznati:

```
elif username == "Bil Gejts" and password == "malo nezno":
```

U ovoj liniji se nalazi jedan složeni uslov koji se sastoji od dva prosta uslova. Prosti uslovi su udruženi u jedan izraz upotrebom and operatora. Složeni uslov kao i prosti može biti ili True ili False.

Korišćenjem operatora and, samo ako su oba prosta uslova True, onda će i složeni uslov biti True.

A	B	A and B
T	T	T
T	F	F
F	T	F
F	F	F

021 Logički operator or

Ako gost želi da se loguje na ekskluzivnu mrežu, gost mora uneti username ili password koji se prepoznaju. Dovoljno je da se prepozna samo jedno, i gost će se prepoznati:

```
elif username == "gost" or password == "gost":
```

U ovoj liniji se nalazi jedan složeni uslov koji se sastoji od dva prosta uslova. Prosti uslovi su udruženi u jedan izraz upotrebom or operatora. Složeni uslov kao i prosti može biti ili True ili False.

Korišćenjem operatora or, makar da je jedan od dva prosta uslova True, onda će i složeni uslov biti True.

A	B	A and B
T	T	T
T	F	T
F	T	T
F	F	F

Planiranje u programiranju

Dosadašnji programi su bili jednostavni. Planiranje ovakvih programa liči na preterenu radnju. Ali, planiranje programa čak i prostih, uvek rezultuje u sačuvanom vremenu.

U programiranju se koristi nekoliko alata i tehnika za planiranje procesa programiranja.

022 Kreiranje algoritma sa pseudokodom

Algoritam je set jasnih, lako praćenih instrukcija za dostizanje cilja. Algoritam je i konkretna lista koraka koji se prate po utvrđenom redosledu.

Algoritam se često piše u nečemu što se naziva pseudokod. Pseudokod se piše jezikom koji može biti kombinacija maternjeg jezika i programerskih izraza. Zato bi trebalo da podseća na plan ili algoritam za realizaciju programa. Primer pseudokod algoritma za zaradu od milion dolara:

```
Ako se setiš novog i korisnog proizvoda
        Onda je to tvoj proizvod
Inače
        Prepakuj postojeći proizvod kao da je tvoj
Napravi reklamu za taj proizvod
Pokaži reklamu na TV
Stavi cenu od $100 po proizvodu
Prodaj 10 000 jedinica proizvoda
```

023 Primena metode Detaljnih koraka na algoritam

Početni algoritam nije dorađen. Često je potrebno više različitih metoda da bi se algoritam mogao implementirati u kodu. Metoda Detaljnih koraka (Stepwise refinement) je proces za ponovno pisanje algoritma kojim se on priprema za implementaciju. Često se objašnjava kao metod za detaljnu razradu početnog algoritma. Svaki korak u algoritmu se deli na više sitnijih koraka, sve dok se ne oseti da je algoritam dovoljno usitnjen da bi mogao da se primeni na kod.

Npr, u liniji pseudokoda: Napravi reklamu za taj proizvod, nejasno je kako napraviti reklamu. Korišćenjem metode Detaljnih koraka, jedan korak se deli na više manjih, pa linija pseudokoda postaje:

```
Napiši skript za reklamu proizvoda
Iznajmi TV studio za jedan dan
Unajmi produkcijsku ekipu
Unajmi publiku
Snimi reklamu
```

Ako je osećaj da su ovih pet koraka jasni i dostižni, onda je ovaj deo algoritma detaljno prikazan. Ako nema tog osećaja, onda se mora uraditi još sitnijih koraka.

Detaljna izrada programa Pogodi_broj

024 Planiranje programa

Pseudokod:

```
Izaberi slučajan broj
Dok igrač ne pogodi broj
        Neka igrač pogađa
Čestitati igraču
```

Naravno, ovo je prva skica programa i nedostaju neki elementi. Prvo, program mora reći igraču ako je pokušaj previše visok ili previše nizak. Drugo, program mora voditi računa o tome koliko pokušaja su napravljeni i da kaže igraču koji je to broj na kraju igre.

Ovo je primena Detaljnih koraka na prethodni algoritam u pseudokodu:

```
Pozdravi igrača i objasni igru
```

Izaberi slučajan broj između 1 i 100
Pitaj igrača za pokušaj pogađanja
Postavi broj pokušaja na 1
Dok igračev pokušaj ne bude identičan sa tačnim brojem
Ako je pokušaj veći od tačnog broja
Reci igraču da je smanji sledeći pokušaj
U suprotnom
Reci igraču da poveća sledeći pokušaj
Uzmi novi pokušaj pogađanja od igrača
Povećaj broj pokušaja za 1
Čestitaj igraču na pogođenom broju
Neka igrač sazna koliko je potrošio pokušaja pogađanja

Posle ovoga, osećaj je dobar za kodovanje algoritma.

program Pogodi_broj.py

```
#Pogodi_broj
#Kompjuter bira slucajan broj izmedju 1 i 100
#igrac pokusava da pogodi broj a kompjuter obavestava
#igraca da li je pokusaj preveliki ili premali ili je pogodio
import random
print("\tDobrodosli u 'Pogodi broj'!")
print("\nJa cu zamisliti broj izmedju 1 i 100.")
print("Pokusaj pogoditi u sto manje pokusaja.\n")
#postavljanje inicijalnih vrednosti
broj = random.randint(1, 100)
broj_igrac = int(input("Pokusaj da pogodis: "))
pokusaj = 1
#petlja za pogadjanje
while broj_igrac != broj:
    if broj_igrac > broj:
        print("Nize...")
    else:
        print("Vise...")
    broj_igrac = int(input("Pokusaj da pogodis: "))
    pokusaj += 1
print("Pogodio si! Tacan broj je", broj)
print("A trebalo je samo", pokusaj, "pokusaja!\n")
```

```
        Dobrodosli u 'Pogodi broj'!

Ja cu zamisliti broj izmedju 1 i 100.
Pokusaj pogoditi u sto manje pokusaja.

Pokusaj da pogodis: 50
Vise...
Pokusaj da pogodis: 75
Vise...
Pokusaj da pogodis: 85
Nize...
Pokusaj da pogodis: 80
Nize...
Pokusaj da pogodis: 78
Nize...
Pokusaj da pogodis: 77
Nize...
Pokusaj da pogodis: 76
Pogodio si! Tacan broj je 76
A trebalo je samo 7 pokusaja!
```