Kurs: Programiranje igara sa programskim jezikom Pajton
010 čas: Program Iks_oks.py
Teme: funkcije, globalne promenjive

Kreiranje funkcija

Do sada su se koristile ugrađene funkcije (len(), range()). U Pajtonu je moguće napraviti sopstvene funkcije. Korisničke funkcije rade na istim principima kao i ugrađene, koje su došle zajedno sa samim programskim jezikom. Kreiranje sopstvenih funkcija donosi razne pogodnosti: deljenje koda u manje delove koji se lakše održavaju, dugački kodovi se teško čitaju i razumeju, programi sastavljeni od funkcija se lakše pišu i održavaju.

program Instrukcije.py

Često je neophodno pre početka korišćenja nekog programa napisati dobro uputstvo ili instrukcije za korišćenje aplikacije. Sledeći kod proizvodi instrukcije. Za to se koristi korisnička funkcija koja prikazuje instrukcije.

```
#Instrukcije
#Prikazuje korisnicke funkcije
def instrukcije():
    """Prikazuje uputstva za igru."""
    print(
    """
    Dobrodosli u najveci intelektualni izazov svih vremena: Iks oks.
    Ovo ce biti obracun izmedju tvog ljudskog mozga i mog silicijumskog procesora.
    Unosom broja izmedju 0 i 8, ti pravis potez. Broj odgovara
    poziciji cifre na sledecoj tabeli:
    0 | 1 | 2
    ---------
    3 | 4 | 5
    ---------
    6 | 7 | 8
    Spremi se covek. Poslednja borba pocinje.\n
    """
    )
print("Ovo su instrukcije za Iks oks igru:")
instrukcije()
print("Evo ih ponovo:")
instrukcije()
print("Do sada si verovatno shvatio sustinu igre.")
```

001 Definisanje funkcije

Linija : `def instrukcije():` je početak definicije nove funkcije.

Ova linija kaže računaru da blok koda koji sledi se koristi sav zajedno kao funkcija instrukcije(). Ovo je način imenovanja bloka iskaza. To znači kada se svaki put pozove funkcija instrukcije() u ovom programu, startuje blok koda.

Linija i njen blok su definicija funkcije. Oni definišu šta funkcija radi, ali oni ne startuju funkciju. Kada računar vidi definiciju funkcije, on pravi zabelešku da ova funkcija postoji da bi mogao kasnije da je i upotrebi. Funkciju startuje poziv funkcije.

Svaka korisnički definisana funkcija startuje sa def, pa imenom funkcije i parom zagrada i dvotačkom. Posle toga su uvučene linije iskaza, telo funkcije. Ime funkcije bi trebalo da kratko opiše šta funkcija zapravo radi.

002 Dokumentovanje funkcije

Funkcije imaju poseban mehanizam koji omogućava da se dokumentuju sa nečim što se zove docstring (ili dokumentacioni string). Kreiran je sledeći docstring za instrukcije():

```
"""Prikazuje uputstva za igru."""
```

Docstring je obično trostruki navodnik, i ako se koristi mora biti u prvoj liniji funkcije. Za jednostavne funkcije je dovoljno kao jedna rečenica koja opisuje šta funkcija radi. Funkcije rade sasvim dobro i bez docstring, ali njihovo korišćenje se smatra dobro idejom. Tako se pravi navika komentarisanja sopstvenog koda i opisuje se posao koji funkcija obavlja. Takođe, docstring funkcije se može pojaviti kao interaktivna dokumentacija dok se kuca poziv funkcije.

003 Pozivanje korisnički kreirane funkcije

Pozivanje korisnički kreirane funkcije se izvodi isto kao i ugrađene funkcije. Koristi se ime funkcije i par zagrada. U programu se poziva funkcija kao: instrukcije() . Ovime se kaže kompjuteru da izvrši funkciju koja je prethodno definisana.

004 Razumevanje apstrakcija

Pisanje i pozivanje funkcija je praktična vežba nečega što se zove apstrakcija. Apstrakcija omogućava pogled na širu perspektivu bez zadiranja u detalje. Tako, u ovom programu, samo se koristi funkcija instrukcije() bez brige o detaljima prikaza teksta. Jedino šta se treba uraditi jeste poziv funkcije.

Parametri i vraćene vrednosti

program Primi_vrati.py

```
#Primi_vrat
#Prikazuje parametre i vracene vrednosti
def prikazi(poruka):
    print(poruka)
def daj_mi_pet():
    pet = 5
    return pet
def pitanje_da_ne(pitanje):
    """Pita da ili ne pitanje."""
    odziv = None
    while odziv not in ("da", "ne"):
        odziv = input(pitanje).lower()
    return odziv

prikazi("Ovo je poruka za tebe.\n")
broj = daj_mi_pet()
print("Ovo sam dobio od daj_mi_pet():", broj)
odgovor = pitanje_da_ne("\nMolim uneti 'da' ili 'ne': ")
print("Hvala na unosu:", odgovor)
```

```
Ovo je poruka za tebe.

Ovo sam dobio od daj_mi_pet(): 5

Molim uneti 'da' ili 'ne': da
Hvala na unosu: da
```

005 Primanje informacija preko parametara

Prva funkcija u nizu je prikazi(). Ona prihvata vrednost i štampa je. Prima vrednost preko svog parametra. Parametri su zapravo imena promenjivih unutar zagrada hedera funkcije:

```
def prikazi(poruka):
```

Parametar ima vrednost poslatu od poziva funkcije, preko argumenata poziva funkcije. Zato, kada se pozove prikazi(), poruka ima dodeljenu vrednosti kroz argument `"Ovo je poruka za tebe.\n"`
U delu programa posle funkcija, taj deo se često naziva glavni (main) deo programa, poziva se funkcija prikazi preko:

```
prikazi("Ovo je poruka za tebe.\n")
```

Kao rezultat, poruka dobija vrednost stringa `"Ovo je poruka za tebe.\n"` . Zatim se funkcija startuje sa poruka kao parametrom tj kao promenjivom. Linija `print(poruka)` štampa string.
Da se nije poslala vrednost u poruka, pojavila bi se greška. Funkcija prikazi() traži tačno jednu vrednost argumenta.
Iako funkcija priakazi() ima samo jedan parametar, funkcije mogu imati veliki broj parametara. Da bi se definisala funkcija sa više parametara, u zagradi se navode imena paramatara razdvojene zarezima.

006 Vraćanje informacija preko povratnih vrednosti

Sledeća funkcija, daj_mi_pet(), vraća vrednost preko iskaza return:

```
return pet
```

Kada se ova linija startuje, funkcija prosleđuje vrednost u pet nazad u onaj deo programa koji je pozvao tu funkciju, i zatim se funkcija završava. Funkcija se uvek završava po izvršavanje return iskaza.
Sada je na delu programa koji je pozvao funkciju da prihvati vrednost koja se vraća i da uradi nešto sa time. Sledeći je deo main programa koji je pozvao funkciju:

```
broj = daj_mi_pet()
print("Ovo sam dobio od daj_mi_pet():", broj)
```

Način prihvatanja povratne vrednosti funkcije je dodelom rezultata poziva funkcije promenjivoj broj. Tako da kada se funkcija završi, broj dobija povratnu vrednost od daj_mi_pet(), koja je jednaka 5. Sledeća linija štampa broj i time se pokazuje da je povratna vrednost dobra.
Može se proslediti više od jedne vrednosti iz funkcije. Samo treba izlistati sve vrednosti koje se žele vratiti odvojene zarezima. Za to je potrbno imati dovoljno promenjivih da dohvati te vrednosti. Ako ih nema dovoljno javlja se greška u programu.

007 Enkapsulacija

Možda se ne vidi razlog vraćanja vrednosti pri korišćenju korisničkih funkcija. Zašto se samo ne iskoristi promenjiva pet u main delu programa? Problem je što pet ne postoji izvan funkcije daj_mi_pet(). Nijednoj promenjivoj kreiranoj u funkciji, uključujući i njene parametre, ne može se direktno pristupiti izvan te funkcije. Ovo pravilo se naziva **enkapsulacija**. Enkapsulacija pomaže da se deo koda realizuje nezavisno sakrivanjem ili enkapsulacijom detalja. Zato se koriste parametri i povratne vrednosti: zbog komunikacije preko samo preko onih podataka koji se moraju razmeniti. Uz to, ne mora se voditi računa o promenjivima koje se krairaju unutar funkcije, u odnosu na ostatak programa. Kod velikih programa, ovo je značajan doprinos.
Enkapsulacija zvuči kao apstrakcija. Oni su usko povezani. Enkapsulacija je princip apstrakcije. Apstrakcija olakšava brigu o detaljima. Enkapsulacija sakriva detalje.

008 Dobijene i vraćene vrednosti u istoj funkciji

Poslednja funkcija u programu je pitanje_da_ne() koja prima jednu vrednost a vraća drugu. Prima pitanje a vraća odziv od korisnika, tj "da" ili "ne". Funkcija prima pitanje preko svog parametra:

```
def pitanje_da_ne(pitanje):
```

gde pitanje dobija vrednost argumenta prosleđenog funkciji. U ovom slučaju, to je string:

```
"\nMolim uneti 'da' ili 'ne': "
```

Sledeći deo funkcije koristi ovaj string i traži od korisnika povratnu informaciju:

```
odziv = None
    while odziv not in ("da", "ne"):
        odziv = input(pitanje).lower()
```

Petlja while će stalno postavljati pitanje sve dok korisnik ne unese da, DA, NE ili ne. Funkcija uvek konvertuje korisnički unos u mala slova.

Na kraju, kada je korisnik uneo validan odgovor, funkcija šalje string nazad u deo programa koji je i pozvao funkciju: `return odziv` a funkcija se zatim i završava.

U main delu programa, vraćena vrednost je dodeljena promenjivoj odgovor i odštampana:

```
odgovor = pitanje_da_ne("\nMolim uneti 'da' ili 'ne': ")
print("Hvala na unosu:", odgovor)
```

009 Ponovna upotreba softvera (software reuse)

Funkcije se lako mogu ponovo koristiti u programima. Npr, pošto je postavljanje pitanja korisniku često, može se funkcija pitanje_da_ne() korisiti u drugim programima bez dodatnog kodiranja. Ovo se naziva ponovna upotreba softvera.

Dobre strane ponovne upotrebe softvera:

- Povećava produktivnost kompanija (deo koda je već napisan)
- Povećava kvalietet softvera (deo koda je već testiran)
- Daje uniformnost softverskih proizvoda (korišćenjem istog korisničkog interfejsa korisnici su već naviknuti na rad sa softverom)
- Povećava performanse softvera (proveren efikasan deo softvera)

Jedan način je kopiranje funkcija u drugi program. Ali postoji i bolji način. Programer pravi svoje module i importuje svoje funkcije u nov program, kao što se importuju standradni Pajton moduli i koriste njihove funkcije.

Službene reči kao argumenti i difolt vrednosti parametara

Davanje vrednosti kroz argumente za parametre omogućava davanje informacija funkcijama. Pajton omogućava kontrolu i fleksibilnost preko načina na koji se dodaju informacije, kroz difolt vrednosti parametara i službene reči koje su argumenti.

program Rodjendanske_zelje.py

```
#Rodjendanske_zelje
#Prikazuje argumente kao sluzbene reci i difolt vrednosti parametara
def rodjendan1(ime, godine):
    print("Sretan rodjendan,", ime, "!", "Cujem da si napunio", godine, "danas.\n")
def rodjendan2(ime = "Milan", godine = 1):
    print("Sretan rodjendan,", ime, "!", "Cujem da si napunila", godine, "danas.\n")

rodjendan1("Milan", 1)
rodjendan1(1, "Milan")
rodjendan1(ime = "Milan", godine = 1)
rodjendan1(godine = 1, ime = "Milan")
rodjendan2()
rodjendan2(ime = "Zorana")
rodjendan2(godine = 12)
```

```
rodjendan2(ime = "Zorana", godine = 12)
rodjendan2("Zorana", 12)
```

010 Pozicioni parametri i pozicioni argumenti

Ako se samo izlista niz imena promenjivih u hederu funkcije, kreiraju se pozicioni parametri:

```
def rodjendan1(ime, godine):
```

Alo se pozove funkcija samo sa nizom vrednosti, kreiraju se pozicioni argumenti:

```
rodjendan1("Milan", 1)
```

Korišćenje pozicionih parametara i pozicionih argumenata znači da parametri dobijaju njihove vrednosti bazirano samo na poziciji poslatih vrednosti. Prvi parametar dobija prvo poslatu vrednost, drugi parametar dobija drugu poslatu vrednost itd.

Sa ovakvim pozivom funkcije, to znači da ime dobija “Milan“ a godine dobija 1. Ovo rezultuje u poruci: Sretan rodjendan, Milan ! Cujem da si napunio 1 danas. Ako se zamene pozicije dva argumenta, parametri dobijaju različite vrednosti. Tako da sa pozivom:

```
rodjendan1(1, "Milan")
```

ime dobija prvu vrednost 1, a godine dobija drugu vrednost “Milan“. Kao rezultat, dobija se izlaz: Sretan rodjendan, 1 ! Cujem da si napunio Milan danas.

Postoje i drugi načini za kreiranje liste parametara i liste argumenata u programima.

011 Pozicioni parametri i službene reči kao argumenti

Pozicioni parametri dobijaju vrednosti poslate njima po redosledu (poziciji), osim ako se funkciji kaže da bude drugačije. Može se reći funkciji da se dodele određene vrednosti posebnim parametrima, bez obzira na redosled, ako se koriste službene reči kao argumente. Sa ovakvim argumentima, koriste se imena pravih parametara iz hedera funkcije za povezivanje vrednosti sa parametrom. Tako da, pozivanjem funkcije rodjendan1() sa

```
rodjendan1(ime = "Milan", godine = 1)
```

ime dobija “Milan“ a godine dobija 1 a funkcija prikazuje poruku

Sretan rodjendan, Milan ! Cujem da si napunio 1 danas.

Isti rezultat se dobija i bez službenih reči kao argumenata samo slanjem ovih vrednosti po datom rasporedu. Ali stvar je u tome što za rad sa službenim rečima kao argumentima, redosled nije bitan, već službene reči povezuju vrednosti sa parametrima. Tako da poziv

```
rodjendan1(godine = 1, ime = "Milan")
```

takođe daje Sretan rodjendan, Milan ! Cujem da si napunio 1 danas. iako su vrednosti poređane po suprotnom redosledu.

Službene reči kao argumenti dopuštaju dodavanje vrednosti u bilo kojem redosledu. Ali njihov najveći doprinos je u jasnoći. Kada se vidi poziv funkcije korišćenjem službenih reči kao argumenata, dobija se bolje razumevanje šta vrednosti predstavljaju.

Problem je ako se želi koristiti istovremeno i pozicioni argumenti i službene reči kao argumenti. Kada se jednom koriste službene reči kao argumenti, svi preostali argumenti u pozivu moraju biti službene reči kao argumenti. Zato treba koristiti samo jedan način pozivanja funkcije.

012 Difolt vrednosti parametara

Na kraju, postoji opcija dodeljivanja difiolt vrednosti parametrima, vrednosti koje su dodeljene parametrima ako nikakva vrednost nije pridodata njima. To je urađeno u slučaju funkcije rodjendan2(). Heder funkcije izgleda:

```
def rodjendan2(ime = "Milan", godine = 1):
```

Što znači da ako nikakva vrednost nije data za ime, ona dobija “Milan“. Ako nikakva vrednost nije data za godine, ona dobija 1. Tako da poziv: rodjendan2(), ne proizvodi grešku, umesto toga, difolt vrednosti se dodeljuju parametrima, a na izlazu se dobija:

Sretan rodjendan, Milan ! Cujem da si napunila 1 danas.

Kada se jednom dodele difolt vrednosti parametru u listi, mora da se dodele difolt vrednosti za sve parametre u listi posle njega. Zato, ovakav heder funkcije je dobar:

def majmun(banane = 100, sa_repom = “da“, ujak = “majmunov“)

a nije dobar i izazvaće poruku o grešci:

def majmun(banane = 100, sa_repom, ujak)

Može se dodati sitnica koja prevazilazi difolt vrednosti bilo kojeg ili svih parametara. Sa pozivom:

```
rodjendan2(ime = "Zorana")
```

difolt vrednost za ime je prevaziđeno. ime dobija “Zorana“, godine i dalje dobijaju svoju difolt vrednost 1, i na ekranu se prikazuje:

Sretan rodjendan, Zorana ! Cujem da si napunila 1 danas.

Sa pozivom funkcije: `rodjendan2(godine = 12)`

difolt vrednost od godine je prevaziđena. godine dobija vrednost 12, dok ime dobija njenu difolt vrednost “Milan“:

Sretan rodjendan, Milan ! Cujem da si napunila 12 danas.

Sa pozivom: `rodjendan2(ime = "Zorana", godine = 12)`

obe difolt vrednosti su prevaziđene. ime dobija “Zorana“ a godina dobija 12. Poruka se prikazuje:

Sretan rodjendan, Zorana ! Cujem da si napunila 12 danas.

Sa pozivom: `rodjendan2("Zorana", 12)`

dobija isti rezultat kao u prethodnom pozivu. Obe difolt vrednosti su prevaziđene. ime dobija “Zorana“ a godine dobija 12.

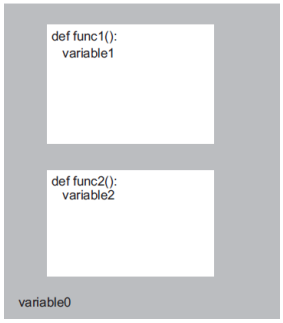
Difolt vrednosti su dobre ako postoji funkcija gde skoro svaki put kada se pozove, neki parametar šalje istu vrednost.

Globalne promenjive i konstante

Zbog dejstva enkapsulacije, funkcije su nezvisne jedna od druge i od glavnog dela programa. Jedini način da se njima dopreme podaci je preko njihovih parametara, a jedini način da se iz njih dobiju podaci je preko njihovih povratnih vrednosti. Ali postoji još jedan način da funkcije međusobno dele informacije a to je kroz globalne promenjive.

013 Dosezi

Dosezi (scopes) predstavljaju različite delove programa koji su međusobno odvojeni. Svaka funkcija koja se definiše, ima sopstveni doseg. Iz tog razloga funkcije ne mogu direktno pristupiti promenjivima drugih funkcija.



Na slici se vidi program sa tri različita dosega. Prvi je definisan sa funkcijom func1(), drugi sa funkcijom func2() a treći je globalni doseg (jedini koji svaki program automatski dobija). U ovom programu, kod koji ne pripada ni jednoj od funkcija nalazi se u globalnom dosegu. Svaka promenjiva koja se kreira u globalnom dosegu se naziva globalna promenjiva, dok svaka promenjiva koja se kreira unutar funkcije se naziva lokalna promenjiva (za tu funkciju).
Pošto je variable1 definisana unutar func1(), ona je lokalna promenjiva koja se koristi samo u dosegu funkcije func1(). Promenjivoj variable1 se ne može prići iz bilo kog drugog dosega. To znači da ne postoji naredba u func2() kojom se može dobiti vrednost iz variable1, a takođe ne postoji ni komanda kojom se može prići ili modifikovati vrednost u njoj iz globalnog dosega.
Ako dve promenjive imaju ista imena unutar dve različite funkcije, to su dve potpuno različite promenjive bez ikakve međusobne veze.

program Globalni_pristup.py

```
#Globalni_pristup
#Prikazuje globalne promenjive
def ucitava_globalne():
    print("Unutar lokalnog dosega u ucitava_globalne(), vrednost je:", vrednost)

def senka_globalna():
    vrednost = -10
    print("Unutar lokalnog dosega u senka_globalna(), vrednost je:", vrednost)

def izmeni_globalnu():
    global vrednost
    vrednost = -10
    print("Unutar lokalnog dosega u izmeni_globalnu(), vrednost je:", vrednost)

vrednost = 10
print("U globalnom dosegu, vrednost je postavljena na:", vrednost, "\n")
ucitava_globalne()
print("Dok u globalnom dosegu, vrednost je jos uvek:", vrednost, "\n")
senka_globalna()
print("Dok u globalnom dosegu, vrednost je jos uvek:", vrednost, "\n")
izmeni_globalnu()
print("U globalnom dosegu, vrednost je promenjena na:", vrednost)
```

```
U globalnom dosegu, vrednost je postavljena na: 10

Unutar lokalnog dosega u ucitava_globalne(), vrednost je: 10
Dok u globalnom dosegu, vrednost je jos uvek: 10

Unutar lokalnog dosega u senka_globalna(), vrednost je: -10
Dok u globalnom dosegu, vrednost je jos uvek: 10

Unutar lokalnog dosega u izmeni_globalnu(), vrednost je: -10
U globalnom dosegu, vrednost je promenjena na: -10
```

014 Čitanje globalne promenjive unutar funkcije

Vrednost globalne promenjive se može čitati unutar bilo kojeg dosega unutar programa. Zato funkcija ucitava_globalne() može da odštampa sadržaj promenjive vrednost.
Iako se može pročitati sadržaj globalne promenjive unutar bilo koje funkcije, ipak se ne može promeniti taj sadržaj direktno. Zato bi se javila greška unutar funkcije ucitava_globalne() kada bi se napisalo: vrednost += 1.
Ideja je da se sadržaj globalne promenjive može pročitati u bilo kojem dosegu unutar programa ali se ne može i modifikovati direktno, bez posebnih komandi.

015 Sakrivanje globalne promenjive unutar funkcije

Ako se promenjivoj unutar funkcije da isto ime kao globalnoj promenjivoj, kaže se da je lokalna promenjiva sakrila (shadowing) globalnu promenjivu. Izgledaće kao da je moguće promeniti vrednost globalne promenjive, ali samo je moguće promeniti vrednost lokalne promenjive. Upravo to se i desilo unutar funkcije senka_globalna(). Kada je lokalnoj promenjivoj vrednost dodeljeno -10, time nije promenjen sadržaj globalne promenjive vrednost.

Sakrivanje globalne promenjive unutar funkcije nije dobra ideja u programiranju jer se lako javlja greška u razumevanju sadržaja globalne i lokalne promenjive istog imena.

016 Izmena globalne promenjive unutar funkcije

Da bi se moglo potpuno pristupiti globalnoj promenjivoj unutar funkcije, koristi se rezervisana reč global, kao unutar funkcije izmeni_globalnu():

global vrednost

U tom momentu, funkcija ima potpun pristup globalnoj promenjivoj vrednost, pa se sadržaj promenjive i menja sa: vrednost =-10.

017 Kada se koriste globalne promenjive i konstante

Globalne promenjive čine program nerazumljivijim, pošto je teško voditi računa o modifikaciji njihovog sadržaja.

Globalne konstante (ili samo konstante) čine program manje konfuznim.

018 Planiranje igre Iks_oks

Prikazati uputstva za igru
Odrediti ko prvi igra
Kreirati praznu iks oks tabelu
Prikazati tabelu
Sve dok se ne zna pobednik ili sve dok nije nerešeno
    Ako je potez igrača
        Dobiti potez igrača
        Updejtovati izgled tabele sa tim potezom
    Inače
        Izračunati potez kompjutera
        Updejtovati tabelu sa tim potezom
    Prikazati tabelu
    Prikazati da je redosled poteza za drugog igrača
Čestitati pobedniku ili prograsiti nerešeno

Koristiće se po jedan znak za potez igrača i kompjutera: “X“ ili “0“. Nepopunjeno mesto će se prikazati sa “ “. Sama tabela treba da bude lista pošto će se sadržaj tabele stalno menjati. Pošto u tabeli postoji devet mesta, to znači da i lista treba da je devet elemenata dugačka.

0	1	2
3	4	5
6	7	8

Svaki broj u polju odgovara poziciji u listi koja predstavlja tabelu.

019 Kreiranje liste funkcija

Funkcija	Opis funkcije
prikaz_uputstva()	Prikazuje uputstva za igru.
pitanje_da_ne(pitanje)	Pita da ili ne pitanje. Prima pitanje. Vraća ili “d“ ili “n“.
pitanje_broj(pitanje, nisko, visoko)	Pita za broj unutar opsega. Prima pitanje, niži broj i veći broj. Vraća broj unutar opsega od nisko do visoko.
znak()	Određuje ko počinje prvi. Vraća kompjuter znak i igračev znak.
nova_tabela()	Kreira novu, praznu tabelu za igru. Vraća tabelu.
prikaz_tabele(tabela)	Prikazuje tabelu na ekranu. Prima tabelu.
dozvoljeni_potezi(tabela)	Kreira listu dozvoljenih poteza. Prime tabelu. Vraća listu dozvoljenih poteza.
pobednik(tabela)	Određuje pobednika u igri. Prima tabelu. Vraća znak, “Nereseno“ ili None.
potez_igrac(tabela, igrac)	Dobija potez od igraca. Dobija tabelu i igračev znak. Vraća igračev potez.
potez_kompjuter(tabela, kompjuter, igrac)	Računa potez kompjutera. Dobija tabelu, znak kompjutera i znak igrača. Vraća potez kompjutera.
sledeci_potez(potez)	Održava redosled poteza baziran na tekućem potezu. Dobija znak. Vraća znak.
cestitka_pobedniku(pobednik, kompjuter, igrac)	Čestita pobedniku ili objavljuje nerešeno. Dobija znak pobednika.

program Iks_oks.py

```
#Iks_oks
#Igra iks_oks kompjuter protiv coveka
X = "X"
O = "O"
PRAZNO = " "
NERESENO = "NERESENO"
BROJ_POLJA = 9

def prikaz_uputstva():
    """Prikazuje uputstvo za igru."""
    print(
    """
    Dobrodosli u najveci intelektualni izazov: Iks-oks.
    Ovo je obracun izmedju tvog ljuskog mozga i mog silikonskog procesora.

    Potez se pravi unosom broja između 0 i 8.
    Broj odgovara poziciji polja u tabeli kao na slici:

                    0 | 1 | 2
                    ---------
                    3 | 4 | 5
                    ---------
                    6 | 7 | 8
```

```
    Spremi se covece. Konacni obracun samo sto nije poceo.\n
    """)

def pitanje_da_ne(pitanje):
    """Pita da ili ne pitanje."""
    odgovor = None
    while odgovor not in ("d", "n"):
        odgovor = input(pitanje).lower()
    return odgovor

def pitanje_broj(pitanje, nisko, visoko):
    """Pita za broj unutar opsega."""
    odgovor = None
    while odgovor not in range(nisko, visoko):
        odgovor = int(input(pitanje))
    return odgovor

def znak():
    """Odredjuje ko pocinje prvi."""
    pocinje_prvi = pitanje_da_ne("Da li hoces da imas prvi potez? (d/n): ")
    if pocinje_prvi == "d":
        print("\nPocni prvi. To je sigurnije.")
        igrac = X
        kompjuter = O
    else:
        print("\nTvoja hrabrost je tvoj kraj... Ja pocinjem prvi")
        kompjuter = X
        igrac = O
    return kompjuter, igrac

def nova_tabela():
    """Kreira praznu tabelu."""
    tabela = []
    for polje in range(BROJ_POLJA):
        tabela.append(PRAZNO)
    return tabela

def prikaz_tabele(tabela):
    """Prikazuje tabelu na ekranu."""
    print("\n\t", tabela[0], "|", tabela[1], "|", tabela[2])
    print("\t", "---------")
    print("\t", tabela[3], "|", tabela[4], "|", tabela[5])
    print("\t", "---------")
    print("\t", tabela[6], "|", tabela[7], "|", tabela[8], "\n")

def dozvoljeni_potezi(tabela):
    """Kreira listu dozvoljenih poteza."""
    potezi = []
    for polje in range(BROJ_POLJA):
        if tabela[polje] == PRAZNO:
            potezi.append(polje)
    return potezi

def pobednik(tabela):
    """Odredjuje pobednika u igri."""
    POTEZI_ZA_POBEDU = ((0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6),
                   (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4, 6))
    for red in POTEZI_ZA_POBEDU:
        if tabela[red[0]] == tabela[red[1]] == tabela[red[2]] != PRAZNO:
            pobednik = tabela[red[0]]
            return pobednik
```

```
    if PRAZNO not in tabela:
        return NERESENO

    return None

def potez_igrac(tabela, igrac):
    """Dobija potez od igraca."""
    dozvoljen = dozvoljeni_potezi(tabela)
    potez = None
    while potez not in dozvoljen:
        potez = pitanje_broj("Koji je tvoj potez? (0 - 8):", 0, BROJ_POLJA)
        if potez not in dozvoljen:
            print("\nTo polje je vec zauzeto, tupavi covek. Izaberi drugo polje\n")
    print("Dobro...")
    return potez

def potez_kompjuter(tabela, kompjuter, igrac):
    """Racuna potez kompjutera."""
    tabela = tabela[:]
    NAJBOLJI_POTEZI = (4, 0, 2, 6, 8, 1, 3, 5, 7)
    print("Dajem broj polja", end=" ")

    #ako kmpjuter moze pobediti, uradi taj potez
    for potez in dozvoljeni_potezi(tabela):
        tabela[potez] = kompjuter
        if pobednik(tabela) == kompjuter:
            print(potez)
            return potez
        #ovaj potez nije dobar
        tabela[potez] = PRAZNO

    #ako igrac moze pobediti, blokiraj potez
    for potez in dozvoljeni_potezi(tabela):
        tabela[potez] = igrac
        if pobednik(tabela) == igrac:
            print(potez)
            return potez
        #ovaj potez nije dobar
        tabela[potez] = PRAZNO

    #posto niko ne pobedjuje u sledecem potezu, izabrati bilo koje slobodno polje
    for potez in NAJBOLJI_POTEZI:
        if potez in dozvoljeni_potezi(tabela):
            print(potez)
            return potez

def sledeci_potez(potez):
    """Promena na potezu."""
    if potez == X:
        return O
    else:
        return X

def cestitka_pobedniku(najbolji, kompjuter, igrac):
    """Cestita pobedniku."""
    if najbolji != NERESENO:
        print(najbolji, "pobeda!\n")
    else:
        print("Nereseno!\n")

    if najbolji == kompjuter:
        print("Kao sto sam i predvideo covek, ponovo sam bio bolji.  \n" \
```

```
            "Ovo je dokaz da su racunari pametniji od ljudi.")

    elif najbolji == igrac:
        print("Ne, ne! Ovo nije istina! Prevario si me, covek. \n" \
              "Samo ovaj put! Ja, kompjuter, zaklinjem se!")

    elif najbolji == NERESENO:
        print("Bio si veoma sretan, covek, i nekako si se izvukao sa neresenim.  \n" \
              "Slavi danasnji dan... ovo je tvoj najveci uspeh koji ces ikada
postici.")

def main():
    prikaz_uputstva()
    kompjuter, igrac = znak()
    na_potezu = X
    tabela = nova_tabela()
    prikaz_tabele(tabela)

    while not pobednik(tabela):
        if na_potezu == igrac:
            potez = potez_igrac(tabela, igrac)
            tabela[potez] = igrac
        else:
            potez = potez_kompjuter(tabela, kompjuter, igrac)
            tabela[potez] = kompjuter
        prikaz_tabele(tabela)
        na_potezu = sledeci_potez(na_potezu)

    najbolji = pobednik(tabela)
    cestitka_pobedniku(najbolji, kompjuter, igrac)

main()
```

020 Postavka programa

Postavljene su globalne konstante. To su vrednsoti koje će više od jedne funkcije da koriste. Sa njima je rad funkcija jasniji a kasnije je lakše i promeniti njihove vrednosti ako bude bilo potrebno.

```
X = "X"
O = "O"
PRAZNO = " "
NERESENO = "NERESENO"
BROJ_POLJA = 9
```

Konstanta X ima znak “X“ koji označava prvi potez u igri. Konstanta PRAZNO je prazno polje u tabeli. NERESENO je mogući rezultat igre. BROJ_POLJA je ukupan broj polja u tabeli.

021 Funkcija znak()

Funkcija pita igrača da li želi da ima prvi potez i vraća koji znak dobija kompjuter a koji igrač. Onaj ko igra prvi dobija znak X, inače 0.

```
def znak():
    """Odredjuje ko pocinje prvi."""
    pocinje_prvi = pitanje_da_ne("Da li hoces da imas prvi potez? (d/n): ")
    if pocinje_prvi == "d":
        print("\nPocni prvi. To je sigurnije.")
        igrac = X
        kompjuter = O
    else:
        print("\nTvoja hrabrost je tvoj kraj... Ja pocinjem prvi")
        kompjuter = X
        igrac = O
    return kompjuter, igrac
```

Unutar funkcije se poziva funkcija pitanje_da_ne().

022 Funkcija nova_tabela()

Funkcija kreira novu praznu tabelu (koja je zapravo lista), sa svih devet elemenata postavljenih na PRAZNO i takvu listu i vraća na mesto poziva:

```
def nova_tabela():
    """Kreira praznu tabelu."""
    tabela = []
    for polje in range(BROJ_POLJA):
        tabela.append(PRAZNO)
    return tabela
```

023 Funkcija prikaz_tabele()

```
def prikaz_tabele(tabela):
    """Prikazuje tabelu na ekranu."""
    print("\n\t", tabela[0], "|", tabela[1], "|", tabela[2])
    print("\t", "---------")
    print("\t", tabela[3], "|", tabela[4], "|", tabela[5])
    print("\t", "---------")
    print("\t", tabela[6], "|", tabela[7], "|", tabela[8], "\n")
```

Funkcija prikazuje tabelu koja je njen ulaz. Pošto elementi na tabeli mogu biti “ “, “X“ ili “0“, funkcija može štampati svaki od njih. Svako polje u tabeli ima svoju poziciju. Na početku igre sve pozicije su popunjene sa “ “.

024 Funkcija dozvoljeni_potezi()

```
def dozvoljeni_potezi(tabela):
    """Kreira listu dozvoljenih poteza."""
    potezi = []
    for polje in range(BROJ_POLJA):
        if tabela[polje] == PRAZNO:
            potezi.append(polje)
    return potezi
```

Funkcija dobija tabelu a vraća listu dozvoljenih poteza. Ovu funkciju koriste druge funkcije. Koristi je funkcija potez_igrac() da bi osigurala da igrač bira dozvoljen potez. Takođe je koristi potez_kompjuter() funkcija da bi računar uzeo u obzir samo dozvoljene poteze pri odlučivanju o potezu.

Dozvoljeni potez je predstavljen brojem polja u kojem je prazno mesto. Npr, ako je centralno polje prazno, unos broja 4 bi bio dozvoljen potez. Ako su samo polja na uglovima slobodna, lista slobodnih poteza bi bila: [0, 2, 6, 8].

Ova funkcija prolazi petljom kroz listu tabele. Svaki put kada pronađe prazno polje, dodaje njegov broj u listu dozvoljenih poteza. Zatim tu listu vraća kao listu potezi (dozvoljeni potezi).

025 Funkcija pobednik()

```
def pobednik(tabela):
    """Odredjuje pobednika u igri."""
    POTEZI_ZA_POBEDU = ((0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6),
                        (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4, 6))
    for red in POTEZI_ZA_POBEDU:
        if tabela[red[0]] == tabela[red[1]] == tabela[red[2]] != PRAZNO:
            pobednik = tabela[red[0]]
            return pobednik

    if PRAZNO not in tabela:
        return NERESENO

    return None
```

Funkcija dobija listu tabela i vraća promenjivu pobednik. Postoje četiri moguće vrednosti za vraćanje. Funkcija vraća X ili 0 ako je dobijen pobednik (onaj koji je prvi igrao ili drugi). Ako je svako polje popunjeno a nema pobednika, vraća se NERESENO. I ako za sada nema pobednika a ostalo je makar jedno nepopunjeno polje, vraća se None.
U funkciji se definiše konstanta, POTEZI_ZA_POBEDU, koja predstavlja svih osam načina na koji se može postići pobeda u igri (tri ista znaka u redu). Svaki način pobede je predstavljen ntorkom. Svaka ntorka je niz tri pozicije u tabeli koji predstavljaju pobednički niz u redu. Npr, prva ntorka u nizu (0, 1, 2) predstavlja gornji red, polja 0, 1 i 2. Sledeća ntorka je srednji red itd.
Zatim, koristi se for petlja za prolazak kroz sve moguće načine pobede traženjem da li bilo koji ugrač ima tri znaka u redu. Iskaz if proverava da li tri polja koja se posmatraju sadrže istu vrednost i da li nisu prazna. Ako je to tačno to znači da je u nizu tri X ili tri 0 i postoji pobednik. Dodeljuje se taj znak promenjivoj pobednik.
Ako nema pobednika, funkcija istražuje da li postoji još praznih polja na tabeli. Ako ih nema, igra je nerešena i vraća se NERESENO.
Ako igra nije nerešena, funkcija se nastavlja. Najzad, ako niko od igrača nije pobedio i igra nije nerešena, onda još uvek nema pobednika, pa se vraća None.

026 Funkcija potez_igrac()

```
def potez_igrac(tabela, igrac):
    """Dobija potez od igraca."""
    dozvoljen = dozvoljeni_potezi(tabela)
    potez = None
    while potez not in dozvoljen:
        potez = pitanje_broj("Koji je tvoj potez? (0 - 8):", 0, BROJ_POLJA)
        if potez not in dozvoljen:
            print("\nTo polje je vec zauzeto, tupavi covek. Izaberi drugo polje\n")
    print("Dobro...")
    return potez
```

Ova funkcija prima tabelu i znak igrača a vraća broj polja gde igrač želi da napravi potez. Prvo funkcija dobija listu dozvoljenih poteza za trenutnu tabelu. Zatim pita igrača za broj polja gde igrač želi sledeći potez. Petlja se nastavlja sve dok se ne dobija broj polja koje je dozvoljeno. Ako je polje dozvoljeno ono se vraća u program.

027 Funkcija potez_kompjuter()

Prima tabelu, znak kompjutera i znak igrača. Vraća potez kompjutera.
Ovo je najkompleksnija fukcija u kodu. Potrebno je uneti kod kojim će računar naizgled inteligentno izabrati najpodesniji potez u datom stanju na tabeli. Ova funkcija je podesna za dalje usavršavanje i trenutni kod je samo pokazni ne i najfunkcionalniji.
Tabela je promenjiva i ona će se menjati u ovoj funkciji dok se traži najbolji potez za kompjuter. Problem je što promena tabele bi se oslikavala u promeni u kodu kod poziva funkcije. Zato je napravljena jedna kopija liste i sve promene na toj kopiji se odnose samo na kopiju ne i na originalnu tabelu:

```
tabela = tabela[:]
```

Menjanje promenjivog parametra se smatra kreiranjem nečega što se naziva bočni efekat (side effect). Njihova pojava nije nužno loša za kod, ali generalno ni bi trebalo da se koristi. Najbolje je komunicirati sa ostatkom koda preko vraćenih vrednosti, jer na taj način je jasno koja se informacija vraća.
Osnovna strategija u potezu za kompjuter je:

1. Ako postoji potez kojim kompjuter pobeđuje u ovom potezu, kompjuter bira taj potez
2. Ako postoji potez koji omogućava igraču da u sledećem potezu pobedi, kompjuter bira taj potez
3. U drugom slučaju, kompjuter treba da izabere najbolje prazno polje za potez. Najbolje polje je u centru tabele. Sledeće najbolje polje su uglovi tabele a ostali najbolji potezi su ostala polja.

Zato se definiše ntorka koja predstavlja najbolje poteze u redu:

```
NAJBOLJI_POTEZI = (4, 0, 2, 6, 8, 1, 3, 5, 7)
    print("Dajem broj polja", end=" ")
```

Dalje, kreira se lista svih dozvoljenih poteza. U petlji, ubacuje se znak za kompjuter u svaki od praznih polja iz liste dozvoljenih poteza i proverava da li je to pobednički potez.

```
#ako kmpjuter moze pobediti, uradi taj potez
    for potez in dozvoljeni_potezi(tabela):
        tabela[potez] = kompjuter
        if pobednik(tabela) == kompjuter:
            print(potez)
            return potez
        #ovaj potez nije dobar
        tabela[potez] = PRAZNO
```

Ako kompjuter može da pobedi, to je potez koji treba napraviti. Ako je to slučaj, funkcija vraća taj potez i završava se. Inače, potez nije dobar i briše se pa se ispituje sledeći sa liste.

Ako se stiglo do ove tačke u kodu, to znači da kompjuter ne može da pobedi na svom sledećem potezu. Zato se proverava da li igrač može da pobedi u sledećem svom potezu.

```
#ako igrac moze pobediti, blokiraj potez
    for potez in dozvoljeni_potezi(tabela):
        tabela[potez] = igrac
        if pobednik(tabela) == igrac:
            print(potez)
            return potez
        #ovaj potez nije dobar
        tabela[potez] = PRAZNO
```

Kod prolazi kroz petlju liste dozvoljenih poteza, stavljajući igračev znak na svko polje, i proverava da li je to potez za pobedu. Ako igrač može da pobedi, to je potez kojim se blokira igračeva pobeda. Tada funkcija vraća taj potez i završava, inače, potez se briše i proverava sledeći dozvoljen potez na listi.

Ako se stiglo do ove tačke, to znači da nijedna strana ne može pobediti u sledećem potezu. Zato se kreće kroz listu najboljih poteza i uzima prvi dozvoljen.

```
#posto niko ne pobedjuje u sledecem potezu, izabrati bilo koje slobodno polje
    for potez in NAJBOLJI_POTEZI:
        if potez in dozvoljeni_potezi(tabela):
            print(potez)
            return potez
```

Kompjuter prolazi petljom kroz NAJBOLJI_POTEZI i čim pronađe dozvoljeni, vraća ga.

Funkcija sledeci_potez() dobija znak trenutnog poteza i vraća znak sledećeg poteza.

028 Funkcija main()

Ova funkcija je glavni deo programa. Umesto da se ostavlja na globalnom nivou, dobar način programiranja je da se enkapsulira i ovaj main deo programa. Naziv ovog dela koda je zbog tradicije main, ali ne mora da se tako i zaista naziva. Ovaj deo koda je zapravo prepisan i primenjen psaudokod pri planiranju pisanja koda.