Grananje u programu

Slajdovi za predmet Osnove programiranja

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2022.

Ciljevi

- razumevanje grananja u programu i Pythonove `if` naredbe
- dvostruko grananje i `if-else` naredba
- višestruko grananje i `if-elif-else` naredba
- pojam obrade izuzetaka (exceptions)
- pisanje koda za obradu izuzetaka

Ciljevi

- Bulovi izrazi i `bool` tip podataka
- kreiranje algoritama koji uključuju elemente kontrole toka
- uključujući nizove grananja i ugnježđeno grananje

Jednostavno grananje

- do sada smo videli programe koji predstavljaju proste sekvence instrukcija koje slede jedna drugu
- to nam nije dovoljno za rešavanje svih problema
- nekada je potrebno izmeniti tok instrukcija da bismo rešili problem

Primer: temperatura

- podsetimo se primera za konverziju temperature $^oC \rightarrow {}^oF$

```
# convert.py

def main():
    celsius = eval(input("Unesite temperaturu C >> "))
    fahrenheit = 9.0/5 * celsius + 32
    print("Temperatura je ", fahrenheit,
        "stepeni Farenhajta.")

main()
```

Primer: temperatura $_2$

- sada treba izmeniti program tako da ispiše upozorenje za ekstremne vrednosti temperature
- temperatura iznad 90^oF ili ispod 30^oF će biti ekstremna

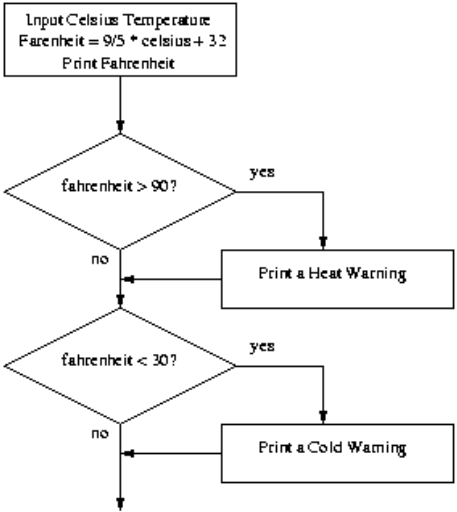
Primer: temperatura $_3$

1. unesi temperaturu u ^{o}C
2. izračunaj temperaturu u ^{o}F
3. ispiši temperaturu u ^{o}F
4. ako je $f \geq 90$
   ispiši upozorenje o visokoj temperaturi
5. ako je $f \leq 30$
   ispiši upozorenje o niskoj temperaturi

Primer: temperatura 4

- ovaj algoritam ima dva grananja na kraju
- uvlačenje teksta sugeriše da se dati korak treba izvršiti samo u slučaju da je ispunjen uslov iz prethodnog reda

Primer: temperatura [5]



Primer: temperatura $_6$

```
# convert2.py

def main():
    c = eval(input("Unesite temperaturu u C: "))
    f = 9.0 / 5 * c + 32
    print("Temperatura je", f, "stepeni Farenhajta.")
    if f >= 90:
        print("Baš je vrućina!")
    if f <= 30:
        print("Brrrrr. Dobro se obuci!")

main()
```

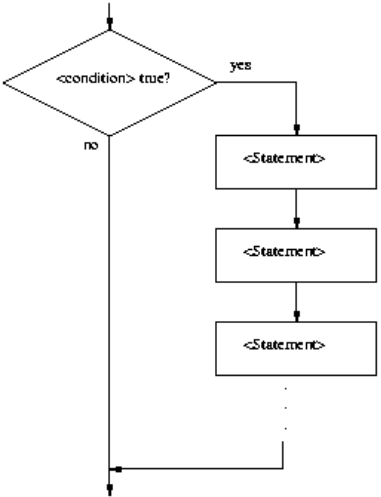
Naredba `if`

- naredba `if` služi za implementaciju grananja
- `if <uslov>:`
  `<telo>`
- `telo` je niz naredbi koje su uvučene u odnosu na `if` izraz

Izvršavanje `if` naredbe

- prvo se izračuna uslov
- ako je uslov ispunjen, niz naredbi u telu se izvršava, i program se dalje izvršava nakon `if` naredbe
- ako uslov nije ispunjen, niz naredbi u telu se preskače, i program se izvršava nakon `if` naredbe

Izvršavanje `if` naredbe $_2$



Izvršavanje `if` naredbe 3

- telo `if` naredbe izvršava se ili se ne izvršava u zavisnosti od uslova
- u svakom slučaju izvršavanje se nastavlja od naredbe koja sledi posle `if`
- ovo je jednostavno ili jednostruko grananje

Pisanje uslova za `if`

- za početak, pišemo jednostavne izraze poređenja
- `<izraz> <relop> <izraz>`
- `relop` je oznaka za relacioni operator

python	**matematika**	**značenje**
`<`	$<$	manje od
`<=`	$\leq$	manje ili jednako
`==`	$=$	jednako
`>=`	$\geq$	veće ili jednako
`>`	$>$	veće od
`!=`	$\neq$	nije jednako

= ¡ ==

- za poređenje koristi se == umesto =
- znak = je rezervisan za operaciju dodele vrednosti
- uobičajena greška je da pišemo = prilikom poređenja

Poređenje stringova

- možemo porediti brojeve i stringove
- kada poredimo stringove, koristi se leksikografski redosled
- tj. stringovi se sortiraju prema Unicode kodovima znakova
  - velika slova su „manja“ od malih, dakle
    `"Bbbb" < "aaaa"`

Bulovi izrazi

- uslov je Bulov izraz (George Boole 1815-1864)
- Bulov izraz može imati samo dve vrednosti,
  - tačno (uslov je ispunjen)
  - netačno (uslov nije ispunjen)
- Python ima konstante `True` i `False`
- u nekim jezicima 0 označava netačno, a 1 tačno

Bulovi izrazi 2

- Bulovi izrazi su tipa `bool`
- dve moguće vrednosti: `True` i `False`

```
>>> 3 < 4
True
>>> 3 * 4 < 3 + 4
False
>>> "hello" == "hello"
True
>>> "Hello" < "hello"
True
```

Primer: kvadratna jednačina

- podsetimo se programa za rešavanje kvadratne jednačine

```
import math

def main():
    a, b, c = eval(input(
        "Unesite koeficijente (a, b, c): "))
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("\nRešenja su:", root1, root2)

main()
```

Primer: kvadratna jednačina $_2$

- ako je $b^2 - 4ac < 0$, program će pući

```
Unesite koeficijente (a, b, c): 1,1,2

Traceback (most recent call last):
  File "/home/branko/pajton/quadratic.py", line 13, in -toplevel-
    main()
  File "/home/branko/pajton/quadratic.py", line 8, in main
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

Primer: kvadratna jednačina 3

- možemo da proverimo ovaj uslov; prvi pokušaj:

```
# quadratic2.py
import math

def main():
    a, b, c = eval(input(
        "Unesite koeficijente (a, b, c): "))

    discrim = b * b - 4 * a * c
    if discrim >= 0:
        discRoot = math.sqrt(discrim)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nRešenja su:", root1, root2)
```

Primer: kvadratna jednačina $_4$

- prvo računamo diskriminantu $b^2 - 4ac$ i proveravamo da li je nenegativna
- ako jeste, program nastavlja sa telom if naredbe i izračunavamo koren
- u suprotnom slučaju program se završava ne ispisujući nikakvu poruku!
- skoro da je ovo lošije nego prethodna greška, jer korisnik nema obaveštenje o tome šta se desilo

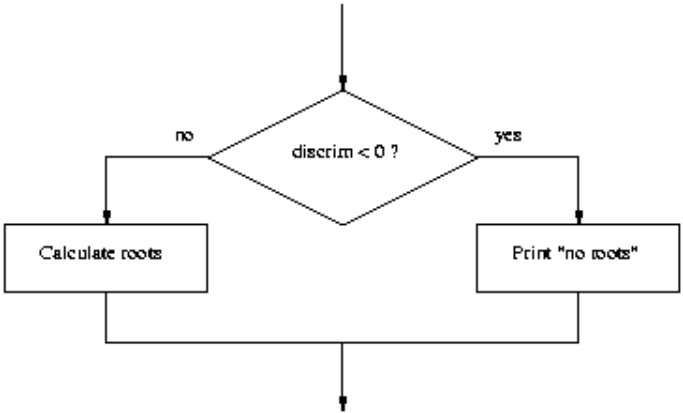
Primer: kvadratna jednačina $_5$

- mogli bismo dodati još jedan `if` na kraj programa:

```
if discrim < 0:
    print("Nema realnih korena!")
```

- ovo će raditi ali nije elegantno
- imamo dva uslova koji su međusobno isključiva

Dvostruko grananje



Dvostruko grananje $_2$

- u Pythonu možemo napraviti dvostruko grananje dodavanjem `else` klauzule na kraj `if` klauzule
- ovo se zove `if-else` naredba:

```
if <uslov>:
    <naredbe>
else:
    <naredbe>
```

Dvostruko grananje $_3$

- kada Pyhon naiđe na `if-else` naredbu
- prvo se izračunava uslov
- ako je uslov ispunjen, izvršavaju se naredbe ispod `if`
- ako uslov nije ispunjen, izvršavaju se naredbe ispod `else`
- u oba slučaja izvršavanje se nastavlja sa naredbama posle `if-else`

Primer: kvadratna jednačina

```
# quadratic3.py
import math

def main():
    a, b, c = eval(input(
        "Unesite koeficijente (a, b, c): "))
    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nNema realnih korena!")
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nRešenja su:", root1, root2)

main()
```

Primer: kvadratna jednačina $_2$

```
>>>
Unesite koeficijente (a, b, c): 1,1,2

Nema realnih korena!

>>>
Unesite koeficijente (a, b, c): 2, 5, 2

Rešenja su: -0.5 -2.0
```

Višestruko grananje

- poslednji program i dalje ima problema!

```
Unesite koeficijente (a, b, c): 1,2,1

Rešenja su: -1.0 -1.0
```

Višestruko grananje $_2$

- program radi ispravno, ali može da zbuni korisnika – izgleda kao da je greškom dva puta ispisao istu vrednost
- dva ista korena dobijaju se kada je diskriminanta jednaka 0
- tada koren ima vrednost $-b/2a$
- treba nam trostruko grananje!

Višestruko grananje 3

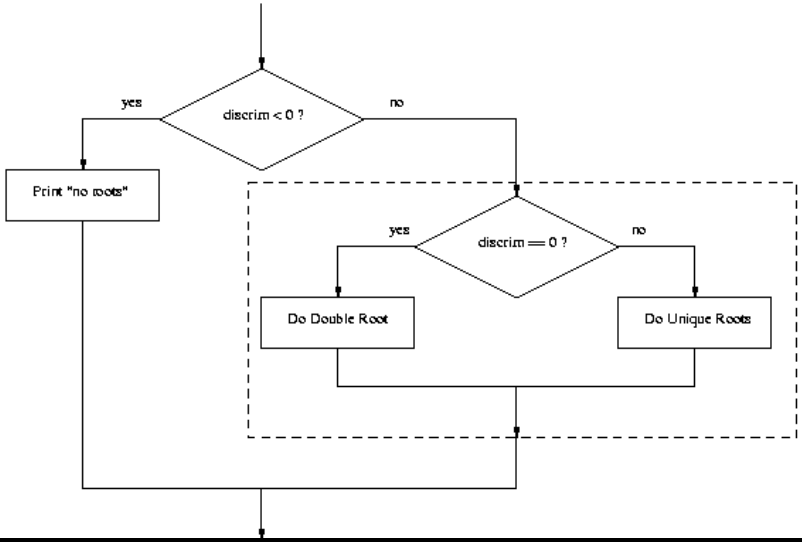
```
izračunaj vrednost diskriminante
   kada je < 0: slučaj kad nema realnih korena
   kada je = 0: slučaj kad ima jedan koren
   kada je > 0: slučaj kad ima dva realna korena
```

- ovo možemo rešiti sa dva `if`-a, jedan unutar drugog
- umetanje jedne složene naredbe unutar druge zove se ugnježđavanje

Višestruko grananje 4

```
if discrim < 0:
    print("Nema realnih korena!")
else:
    if discrim == 0:
        root = -b / (2 * a)
        print("Ima jedan koren:", root)
    else:
        # odradi dva korena
```

Višestruko grananje 5



Višestruko grananje 6

- ako nam treba petostruko grananje, imali bismo ugnježdene if-else naredbe na četiri nivoa!
- umesto rogobatnog uvlačenja, možemo koristiti `if-elif-else` naredbu

```
if <uslov1>:
    <naredbe>
elif <uslov2>:
    <naredbe>
elif <uslov3>:
    <naredbe>
...
else:
    <naredbe>
```

Višestruko grananje 7

- samo jedan blok naredbi će biti izvršen
- izračunavaju se uslovi, jedan-po-jedan, i traži se prvi koji je `True`
- ako se nađe takav uslov, izvršava se blok naredbi ispod njega
- ako se ne nađe, izvršava se blok ispod `else`
- `else` klauzula nije obavezna!

Višestruko grananje 8

```
import math
def main():
    a, b, c = eval(input(
        "Unesite koeficijente (a, b, c): "))
    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nNema realnih korena!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nIma jedan koren:", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nRešenja su:", root1, root2)
```

Obrada izuzetaka

- izbegavali smo grešku u programu (izračunavanje korena negativnog broja) tako što smo koristili grananje
- u mnogim slučajevima – grananje koristimo da se zaštitimo od grešaka
  - možda retkih ali ipak mogućih

Obrada izuzetaka 2

- u prethodnom primeru proveravali smo podatke pre nego što pozovemo `sqrt`
- funkcija može da proveri da li postoji greška i vrati neku specijalnu vrednost da naznači da je došlo do greške
- neka druga funkcija za računanje korena mogla bi da vrati -1 da označi grešku
- pošto koren nikad nije negativan, ova vrednost je jedinstvena

Obrada izuzetaka $_3$

- provere u programu mogu biti toliko česte da je teško pratiti algoritam
- umesto brojnih `if`-provera, možemo koristiti mehanizam obrade izuzetaka (exception handling)

Obrada izuzetaka 4

- „izvrši ove naredbe, i ako se neki problem pojavi obradi ga ovako“
- ne treba nam eksplicitna provera podataka pri svakom koraku

```
try:
    # operacije koje mogu da
    # izazovu grešku
    # ...
except ValueError:
    # obradi grešku ovako
```

Primer: kvadratna jednačina

```
import math

def main():
    try:
        a, b, c = eval(input(
            "Unesite koeficijente (a, b, c): "))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nRešenja su:", root1, root2)
    except ValueError:
        print("\nNema realnih korena")
```

Obrada izuzetaka

- kada Python naiđe na `try` naredbu, pokušaće da izvrši sve naredbe u njenom telu
- ako nema greške, izvršavanje se nastavlja iza try/except
- ako ima greške, traži se onaj `except` sa odgovarajućim tipom greške
- šta je tip greške?

```
...
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

Obrada izuzetaka 2

- try/except se može upotrebiti za hvatanje bilo koje greške
- u slučaju prethodnog primera mogu se javiti sledeće greške:
- unošenje pogrešnog broja podataka (traže se tri) – `unpack tuple of wrong size`
- unošenje identifikatora umesto broja – `NameError`
- unošenje neispravnog Python izraza – `TypeError`

Obrada izuzetaka 3

```
try:
    a, b, c = eval(input(
        "Unesite koeficijente (a, b, c): "))
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("\nRešenja su:", root1, root2)
except ValueError as excObj:
    if str(excObj) == "math domain error":
        print("Nema realnih korena!")
    else:
        print("Pogrešan broj koeficijenata.")
except NameError:
    print("Niste uneli tri broja.")
except TypeError:
    print("Na ulazu nisu sve brojevi.")
except SyntaxError:
    print("Unos je neispravan. Nedostaje zarez?")
except:
    print("Nešto nije u redu!")
```

Obrada izuzetaka 2

```
except ValueError as excObj:
    if str(excObj) == "math domain error":
        print("Nema realnih korena!")
    else:
        print("Pogrešan broj koeficijenata.")
```

- možemo dodeliti identifikator izuzetku (`as excObj`)
- možemo koristiti tu promenljivu da se „raspitamo“ o greškama

```
except:
    print("Nešto nije u redu!")
```

- ovaj `except` može da obradi bilo koji izuzetak
- ako se on ne obradi nekim prethodnim `except`-om

Primer: najveći od tri

- treba nam program koji traži najveći od tri broja

```
def main():
    x1, x2, x3 = eval(input("Unesite tri broja: "))
    # ovde nedostaje kod koji određuje najveći broj
    print("Najveća vrednost je", maks)
```

Primer: najveći od tri $_2$

- ovo liči na trostruko grananje gde treba izvršiti jedno od ova tri:

```
maks = x1
maks = x2
maks = x3
```

- samo nam treba pravi uslov za svaku od ovih naredbi

Strategija 1: poredi svakog sa svima

- slučaj kada je `x1` najveći

```
if x1 >= x2 >= x3:
    maks = x1
```

- mnogi jezici ne dozvoljavaju ovakva višestruka poređenja
- Python to dozvoljava – testira da li važi $x_1 \geq x_2 \geq x_3$

Dva pitanja kad pišemo uslove

1. kada je uslov ispunjen, da li je blok naredbi odgovarajuća akcija?
   - x_1 je barem velik kao x_2 i x_3, pa možemo reći da je najveći x_1
   - treba uvek voditi računa o graničnim vrednostima
2. suprotno pitanje: da li je uslov ispunjen u svim slučajevima kada je x_1 najveći?
   - recimo da je $x_1 = 5, x_2 = 2, x_3 = 4$
   - x_1 je najveći, ali ne važi $x_1 \geq x_2 \geq x_3$

Strategija 1: poredi svakog sa svima

- možemo da rastavimo ove uslove sa and

```
if x1 >= x2 and x1 >= x3:
    maks = x1
elif x2 >= x1 and x2 >=x3:
    maks = x2
else:
    maks = x3
```

- poredimo svaku vrednost sa svima ostalima

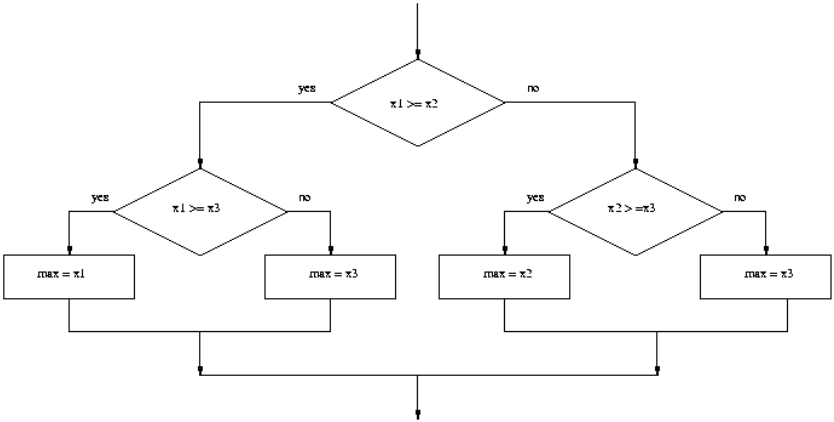
Strategija 1: poredi svakog sa svima $_2$

- šta ako treba da odredimo najvećeg od 5 brojeva?
- trebaće nam 4 testa, svaki sa po 4 uslova spojenih `and`-om
- ...grozno!

Strategija 2: stablo odlučivanja

- možemo da izbegnemo ponavljanje istih testova korišćenjem stabla odlučivanja
- počnimo od testa `x1 >= x2`
- ovo će eliminisati jednu od vrednosti kao kandidata za najvećeg
- ako je uslov ispunjen, treba još proveriti koji je veći, `x1` ili `x3`
- ako nije ispunjen, poredimo `x2` i `x3`

Strategija 2: stablo odlučivanja



Strategija 2: stablo odlučivanja

```
if x1 >= x2:
    if x1 >= x3:
        maks = x1
    else:
        maks = x3
else:
    if x2 >= x3:
        maks = x2
    else:
        maks = x3
```

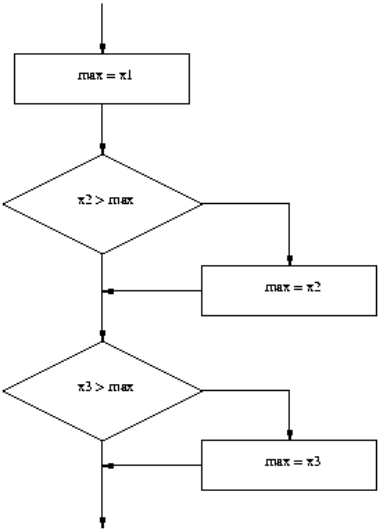
Strategija 2: stablo odlučivanja

- sa ovim pristupom uvek imamo tačno 2 poređenja bez obzira na raspored brojeva
- komplikovanije od prve strategije:
- za traženje najvećeg od 4 broja trebaće nam `if-else` u tri nivoa, sa ukupno 8 naredbi dodele

Strategija 3: sekvencijalna obrada

- šta ako dobijemo listu od 100 brojeva?
- idemo redom kroz listu i pamtimo najvećeg pronađenog do tog trenutka
- na kraju ćemo kao zapamćenog imati najveći broj

Strategija 3: sekvencijalna obrada



Strategija 3: sekvencijalna obrada

- lako se prevodi u Python kod

```
maks = x1
if x2 > maks:
    maks = x2
if x3 > maks:
    maks = x3
```

Strategija 3: sekvencijalna obrada

- ovaj postupak se ponavlja i lako se predstavlja petljom
- unesemo broj
- uporedimo ga sa trenutnim `maks`
- ako je veći, ažuriramo `maks`
- ponavljamo dok ima brojeva

Strategija 3: sekvencijalna obrada

```
n = eval(input("Koliko ima brojeva? "))

maks = eval(input("Unesite broj >> "))

for i in range(n-1):
    x = eval(input("Unesite broj >> "))
    if x > maks:
        maks = x

print("Najveći broj je", maks)
```

Strategija 4: koristi Python biblioteku

- postoji funkcija `max` koja vraća najveći broj iz date liste

```
x1, x2, x3 = eval(input("Unesite tri broja: "))
print("Najveći broj je", max(x1, x2, x3))
```

Python moduli

- fajlovi sa Python kodom se mogu upotrebiti na više načina

1. neki su predviđeni da se direktno pokreću kao programi – programi ili skriptovi
2. neki su predviđeni da se koriste kao moduli, tj. `import`-uju iz drugih programa
3. neki mogu da se koriste na oba načina

Python moduli $_2$

- kako u nekom Python fajlu znamo da li smo pokrenuti direktno (kao program), ili `import`-ovani (kao modul)?
- najčešće kada `import`-ujemo modul, ne želimo da se on izvršava

- u svakom Python fajlu možemo koristiti specijalnu promenljivu `__name__`
- u njoj je ime našeg modula (ako je fajl trenutno `import`-ovan)...
- ...ili je njena vrednost `"__main__"` ako je fajl pokrenut kao program

Python moduli 3

```
# mojmodul.py

def fun1():
    ...

def fun2():
    ...

def main():
    ...

if __name__ == "__main__":
    main()
```

Python moduli 4

```
if __name__ == "__main__":
    main()
```

- ako smo ovaj fajl pokrenuli kao Python program:
  `__name__ == "__main__"`
- ako smo ovaj fajl `import`-ovali kao modul:
  `__name__ == "mojmodul"`