

Pogojni stavek

Osnove programiranja

Nejc Ilc

Kaj naj danes oblečem?

Vsem znana jutranja dilema, tudi za Praskeža

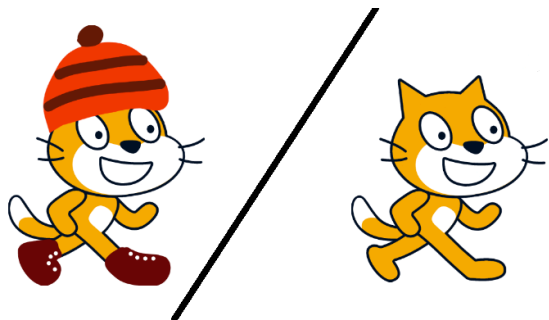
Začnimo preprosto:

- recimo, da poznamo samo zunanjo temperaturo in
- Praskež ima v omari samo zimsko opremo (toplo kapo s cofom in zimske škornje).

Praskež se mora odločiti. Noče, da ga zebe. Še manj pa, da mu je vroče. Pomagajmo mu!

Ali si bo Praskež nadel kapo in obul škornje je sedaj odvisno od določenega pogoja - temperature.

Recimo, da je primeren prag pri 5 °C. Če je zunaj manj stopinj, naj uporabi kapo in škornje, sicer ne.



Kaj naj danes oblečem?

obleke-v1

Narišimo v Scratchu in napišimo v Pythonu



Klikni sliko za ogled v brskalniku

Python nima preoblek tako kot Praskež, oranžno bismo pa je enako. Spoznajmo **če** (if).

```
videz = 'navaden'
odgovor = input('Koliko stopinj je zunaj? ')
odgovor = float(odgovor)
if odgovor < 5:
    print('Zima! Rabim kapo in tople škornje.')
    videz = 'kapa in škornji'
print('Zdaj grem lahko ven! Moj videz:', videz)
```

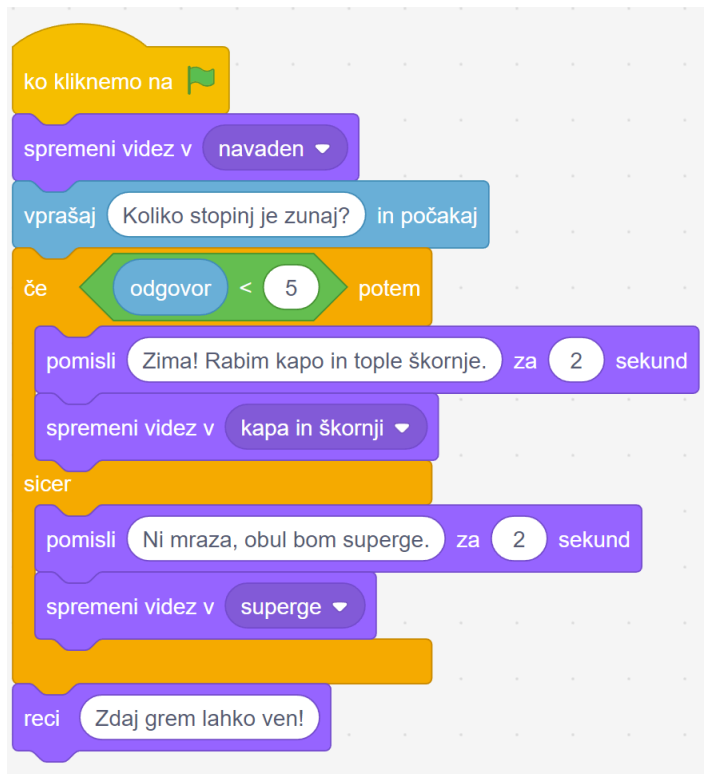
Preizkus:

```
Koliko stopinj je zunaj? 3
Zima! Rabim kapo in tople škornje.
Zdaj grem lahko ven! Moj videz: kapa in škornji
```

Kaj naj danes oblečem?

obleke-v2

Praskež potrebuje obutev, tudi če ni mraza. V omaro mu dodamo superge.



Klikni sliko za ogled v brskalniku

Spoznamo **blok *sicer*** (`else`) pogojnega stavka.

```
videz = 'navaden'
odgovor = input('Koliko stopinj je zunaj? ')
odgovor = float(odgovor)
if odgovor < 5:
    print('Zima! Rabim kapo in tople škornje.')
    videz = 'kapa in škornji'
else:
    print('Ni mraza, obul bom superge.')
    videz = 'superge'
print('Zdaj grem lahko ven! Moj videz:', videz)
```

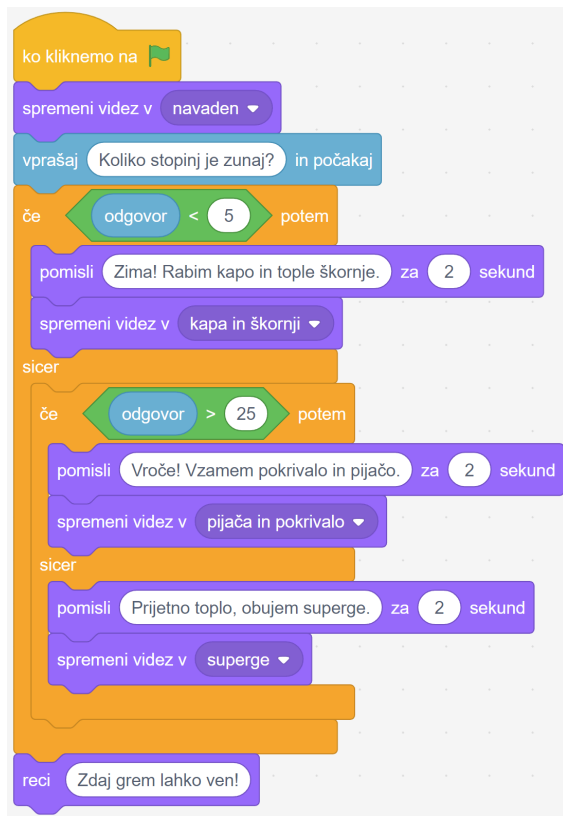
Preizkus:

```
Koliko stopinj je zunaj? 10
Ni mraza, obul bom superge.
Zdaj grem lahko ven! Moj videz: superge
```

Kaj naj danes oblečem?

obleke-v3.0

Kaj pa, ko je vroče? Veliko tekočine in pokrivalo na glavo!



Spoznamo **ugnezdeni** pogojni stavek.

```
videz = 'navaden'
odgovor = input('Koliko stopinj je zunaj? ')
odgovor = float(odgovor)
if odgovor < 5:
    print('Zima! Rabim kapo in tople škornje.')
    videz = 'kapa in škornji'
else:
    if odgovor > 25:
        print('Vroče! Vzamem pokrivalo in pijačo.')
        videz = 'pijača in pokrivalo'
    else:
        print('Prijetno toplo, obujem superge.')
        videz = 'superge'
print('Zdaj grem lahko ven! Moj videz:', videz)
```

Kaj naj danes oblečem?

obleke-v3.1

Scratch tega ne pozna ...

Ugnezdjeni pogojni stavek s prejšnje strani:

```
videz = 'navaden'
odgovor = input('Koliko stopinj je zunaj? ')
odgovor = float(odgovor)
if odgovor < 5:
    print('Zima! Rabim kapo in tople škornje.')
    videz = 'kapa in škornji'
else:
    if odgovor > 25:
        print('Vroče! Vzamem pokrivalo in pijačo.')
        videz = 'pijača in pokrivalo'
    else:
        print('Prijetno toplo, obujem superge.')
        videz = 'superge'
print('Zdaj grem lahko ven! Moj videz:', videz)
```

Spoznamo **sicer-če** (`elif`) in zapišimo drugače.

```
videz = 'navaden'
odgovor = input('Koliko stopinj je zunaj? ')
odgovor = float(odgovor)
if odgovor < 5:
    print('Zima! Rabim kapo in tople škornje.')
    videz = 'kapa in škornji'
elif odgovor > 25:
    print('Vroče! Vzamem pokrivalo in pijačo.')
    videz = 'pijača in pokrivalo'
else:
    print('Prijetno toplo, obujem superge.')
    videz = 'superge'
print('Zdaj grem lahko ven! Moj videz:', videz)
```

✨ Odlično, program je zdaj malo bolj pregleden!

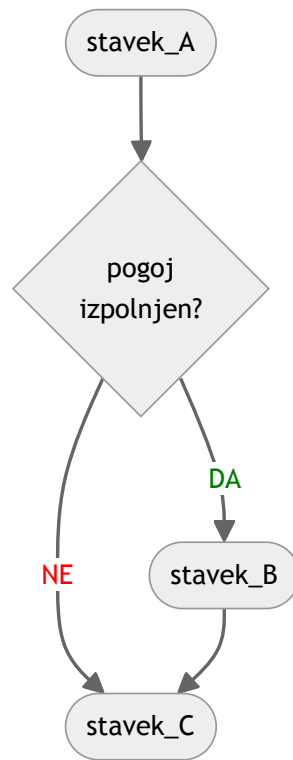
if

Uporabimo, ko želimo izvesti blok stavkov pod določenim pogojem

```
stavek_A

if pogoj:
    # Ta stavek izvedemo, ko je `pogoj` resničen
    stavek_B
    # Lahko dodamo stavke ...

# Zaključimo blok pogojnega stavka
# Naslednji stavek se izvede ne glede na `pogoj`
stavek_C
```



Zamikanje

Kako povemo, kateri stavki se izvedejo ob izpolnjenem pogoju?

```
stavek_A

if pogoj:
    # Ta stavek izvedemo, ko je `pogoj` resničen
    stavek_B
    # Lahko dodamo stavke ...

# Zaključimo blok pogojnega stavka
# Naslednji stavek se izvede ne glede na `pogoj`
stavek_C
```

V Pythonu bloke stavkov že na daleč opazimo, saj se od drugih delov kode ločijo po **zamiku**!

- Po bontonu en zamik naredimo tako: 4 × `preslednica`.
- Thonny in ostala razvojna okolja pritisk tipke `Tab` spremenijo v presledke.
- Skrivnost: poskusite kombinacijo `Shift` + `Tab`

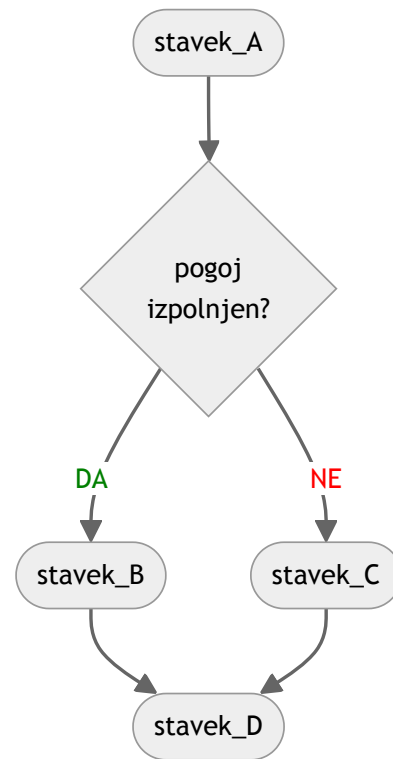
if-else

Uporabimo, ko želimo izvesti blok stavkov pod določenim pogojem. Poleg tega želimo, da se drugi blok stavkov izvede samo takrat, ko ta pogoj ni resničen.

```
stavek_A

if pogoj:
    # Ta stavek izvedemo, ko je `pogoj` resničen
    stavek_B
    # Lahko dodamo stavke ...
else:
    # Ta stavek izvedemo, ko `pogoj` ni resničen
    stavek_C
    # Lahko dodamo stavke ...

# Zaključimo blok pogojnega stavka
# Naslednji stavek se izvede ne glede na `pogoj`
stavek_D
```

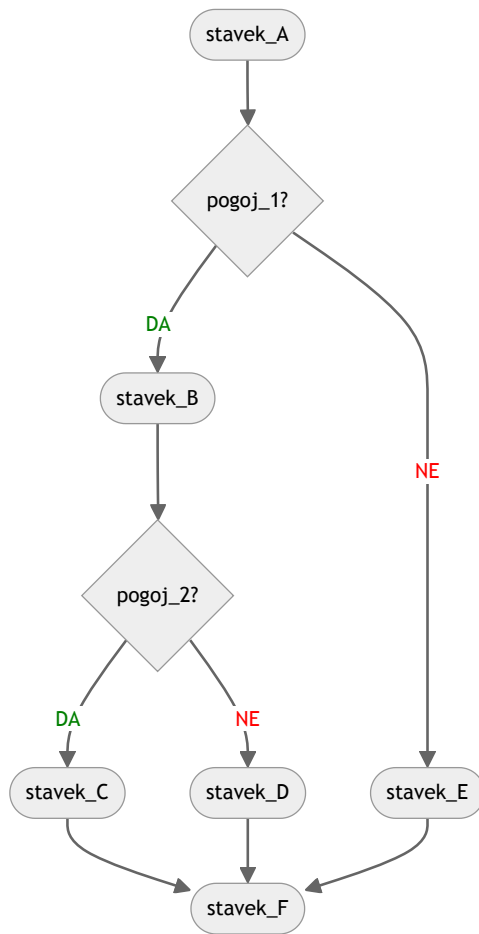


Ugnezdeni pogojni stavek

Naredimo hierarhijo blokov pogojnih stavkov. Pazimo na dodatne zamike.

```
stavek_A
if pogoj_1:
    # Če je `pogoj_1` resničen
    stavek_B
    if pogoj_2:
        # Če sta resnična `pogoj_1` in `pogoj_2`
        stavek_C
    else:
        # Če je `pogoj_1` resničen in `pogoj_2` ni
        stavek_D
else:
    # Če `pogoj_1` ni resničen
    stavek_E

# Naslednji stavek se izvede ne glede na pogojni stavek
stavek_F
```

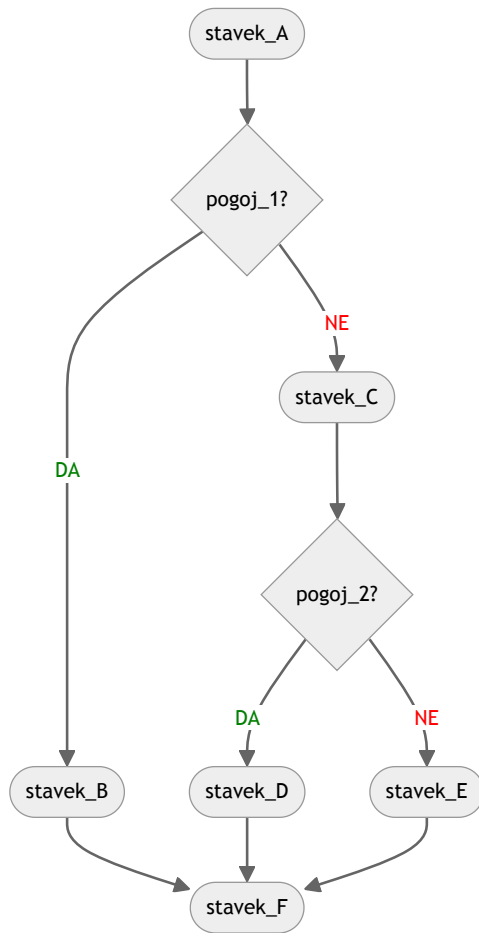


Ugnezdimo drugam

Kaj se zgodi, če pogojni stavek ugnezdimo v blok `else` ?

```
stavek_A
if pogoj_1:
    # Če je `pogoj_1` resničen
    stavek_B
else:
    # Če `pogoj_1` ni resničen
    stavek_C
    if pogoj_2:
        # Če `pogoj_1` ni resničen, `pogoj_2` pa je
        stavek_D
    else:
        # Če sta `pogoj_1` in `pogoj_2` neresnična
        stavek_E

# Naslednji stavek se izvede ne glede na pogojni stavek
stavek_F
```

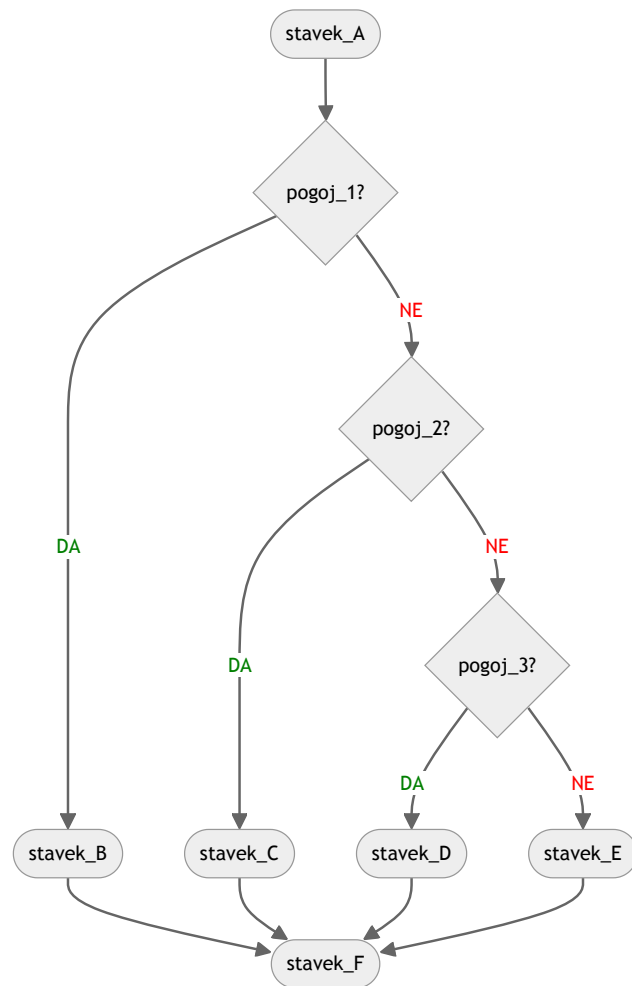


if-elif-...-elif-else

Imamo več pogojev – ko je prvi od njih resničen, izvedemo njegov blok.

```
stavek_A
if pogoj_1:
    # Če je `pogoj_1` resničen
    stavek_B
elif pogoj_2:
    # Če `pogoj_1` ni resničen, `pogoj_2` pa je
    stavek_C
elif pogoj_3:
    # `pogoj_3` resničen, ostala dva ne
    stavek_D
else:
    # Če noben pogoj ni resničen
    stavek_E

# Naslednji stavek se izvede ne glede na pogojni stavek
stavek_F
```



Primerjalni operatorji

Uporabimo jih za sestavljanje pogoja.

operator	opis	primeri	
<code>==</code>	enak	<code>+0 == -0 → True</code>	<code>'Ana' == 'ana' → False</code>
<code>!=</code>	ni enak	<code>1 != 1 → False</code>	<code>'Ana' != 'ana' → True</code>
<code>></code>	večji	<code>1 > 1 → False</code>	<code>'Anna' > 'Ana' → True</code>
<code>>=</code>	večji ali enak	<code>1 >= 1 → True</code>	<code>'ABC' >= 'ABD' → False</code>
<code><</code>	manjši	<code>0.33 < 1/3 → True</code>	<code>'A' < 'a' → True</code>
<code><=</code>	manjši ali enak	<code>0.0 <= -0.0 → True</code>	<code>'1a' <= '2a' → True</code>

Vsi primeri v tabeli so **logični izrazi**. Rezultat logičnega izraza je tipa `bool`.

Logični operatorji

`1 + 2**3` je matematični oz. aritmetični izraz. Logični izraz je sestavljen iz pogojev in logičnih operatorjev.

not

Pomeni "ne", zanikanje. Ima prednost pred `and` in `or`.

`not True → False`

`not False → True`

and

Pomeni "in", hkratnost. Ima prednost pred `or`.

`True and True → True`

`True and False → False`

`False and False → False`

or

Pomeni "ali", izbirnost.

`True or True → True`

`True or False → True`

`False or False → False`

Logični izraz lahko vsebuje kombinacijo logičnih operatorjev, denimo:

`True or False and not False → True`

Nizanje operatorjev

Kako preverimo, ali je vrednost `x` med 10 in 20?

Ena možnost je ta:

```
x > 10 and x < 20
```

Nam bolj naraven pa je zapis, ki uporablja nizanje operatorjev:

```
10 < x < 20
```

Operatorja vsebovanosti


Python pozna tudi operatorja `in` in `not in`, ki povesata, ali je neka stvar vsebovana v drugi. Poglejmo si to na primeru nizov.

```
'bar' in 'rabarbara' → True
```

```
'b' not in 'RaBarBara' → True
```

Na ta način lahko preverimo, ali niz vsebuje določen drug niz oziroma določeno črko. Uporabna sta tudi pri rečeh, ki jih za zdaj še ne poznamo (seznami, terke, množice, slovarji).

Kateri operator je torej bolj pomemben?

operator	opis	prioriteta
<code>**</code>	potenciranje	največja
<code>*</code> , <code>/</code> , <code>//</code> , <code>%</code>	množenje in deljenje	
<code>+</code> , <code>-</code>	seštevanje in odštevanje	
<code>==</code> , <code>!=</code> , <code>></code> , <code>>=</code> , <code><</code> , <code><=</code>	primerjalni operatorji	
<code>is</code> , <code>is not</code>	operatorja identitete (bomo razložili pozneje)	
<code>in</code> , <code>not in</code>	operatorja vsebovanosti	
<code>not</code>	logični ne	
<code>and</code>	logični in	
<code>or</code>	logični ali	najmanjša

False

Vse, kar je prazno ali nično, Python razume kot `False`.

Neko številčno vrednost ali niz (ali druge eksotične živali, ki jih bomo spoznali ta semester) lahko pretvorimo v logično vrednost z uporabo funkcije `bool()`.

Naslednje vrednosti se torej razumejo kot `False`:

- ničla: `0` in `0.0`
- prazen niz: `''`
- nič: `None` (to je posebna rezervirana beseda, ki pomeni "nič")
- prazen seznam `[]`, terka `()`, množica `set()`, slovar `{}`, obseg `range(0)`

True

Vse, kar ni `False` 😊

Utrdimo vse skupaj

```
>>> 1 <= 1.0 < 1.5
True
>>> 1 <= 1.0 < 1.5 == 1 + 0.5
True
>>> not 1 == 0 and 'a' != 'A'
True
>>> 0 == 0.0 and 1 == 1.1
False
>>> 1 + 1 > 1 and 'Anton' < 'Toni' and 2**5 >= 32
True
>>> 'pes' != 'mačka' or 'pes' == 'kuža'
True
>>> bool(0) or not bool(1)
False
>>> bool(1) and bool(2) and bool(3)
True
>>> bool(0) or bool(1) and bool(2)
True
>>> not bool('') or bool(1) and bool('a')
True
>>> not (bool('') or bool(1) and bool('a'))
False
>>> 'pes' in 'pesjan' and not 'love' not in 'Slovenija'
True
```

Kratek stik za radovedne

Privarčujmo nekaj časa in energije s tem trikom

Recimo, da bi radi izračunali izraz `x**y > x*y`, vendar samo, če sta tako `x` kot tudi `y` manjša od 1000. Lahko zapišemo tako (mimogrede bomo še videli, da lahko posamezne stavke med seboj ločimo s podpičjem):

```
>>> x = 3; y = 4; x < 1000 and y < 1000 and x**y > x*y
True
```

V zgornjem logičnem izrazu imamo tri trditve, med katerimi je operator `and`. Celoten logični izraz bo v tem primeru resničen (`True`) samo takrat, ko bodo resnične vse tri trditve. Prva trditev `x < 1000` je resnična, zato moramo preveriti tudi drugo trditev. Ta je prav tako resnična, saj ima `y` vrednost 4 kar je manj kot 1000. Sledi še zadnji del izraza, ki ga moramo izračunati, če želimo vedeti, ali je celoten izraz resničen ali ne. Zdaj izračunamo `x**y` in `x*y` ter ju primerjamo. Rezultat je `True`.

Kaj pa, če nastavimo tako: `x = 3` in `y = 4000`? Prva trditev, da je `3 < 1000`, je resnična. Gremo naprej. Pri drugi se zatakne, ker 4000 ni manjše od 1000. Dobimo `False`, ki pri uporabi operatorja `and` naredi kratek stik in varovalko "zabriše ven" – končamo z računanjem celotnega izraza, ker točno vemo, da bo rezultat vedno `False`. S tem se izognemo potratnemu računanju `3**4000`. Podobno velja za operator `or`, vendar ravno obratno: logični izraz računamo po kosih vse dokler ne naletimo na `True` – tam se ustavimo.

Pobegni!

igra-pobegni-v1.0

Naredimo svojo igro, pravo pustolovščino!

Znajdeš se v neznani sobi.
Boli te glava, pogled je meglen.
Ne veš, zakaj si tu.
Veš pa, da je pobeg edina rešitev.

Soba nima oken, ima pa modra vrata.
Sredi sobe je miza in nekaj na njej.

Izberi eno od možnosti:

- 1: Odpri modra vrata.
 - 2: Pogledj, kaj je na mizi.
- Izbira:



Fotografija: Maick Maciel [vir]