# Programiranje v Pythonu

## Reševanje sudokujev

**Maturitetna seminarska naloga iz informatike**

Kandidat: Ana Čefarin

Mentor: Helena Starc Grlj

Ljubljana Šentvid, marec 2024

Škofijska klasična gimnazija


## Povzetek

Za maturitetno seminarsko nalogo sem izdelala program za reševanje sudokujev. Program sem napisala v programskem jeziku Python. Kot urejevalnik sem uporabila Visual Studio Code in IDLE. Uporabnik za uporabo programa na začetku vnese števila, na koncu pa mu program vrne vstavljen sudoku z eno dodatno številko kot rešitev. Poleg programa sem izdelala še spletno stran, kjer je dostopen program kot tudi osnutek pred izdelavo in dokumentacija. Spletna stran je bila napisana v html, pri tem pa je bil ponovno uporabljen vmesnik Visual Studio Code.

## Abstract

I created a program for solving sudoku puzzles for my graduation thesis. I wrote the program in the Python programming language. As an editor, I used Visual Studio Code and IDLE. To use the program, the user initially enters numbers, and at the end, the program returns the inserted sudoku with one additional number as a solution. In addition to the program, I also created a website where the program is accessible, as well as the pre-production draft and documentation. The website was written in Html, and the Visual Studio Code framework was used again.



## Ključne besede

- Sudoku
- Programiranje
- Miselne igre
- Reševanje

# Kazalo vsebin



# Kazalo slik



# Stvarno kazalo

# 1 UVOD

## 1.1 Predstavitev problema

V prostem času ljudje radi posežemo po križankah, sudokujih, nonogramih in drugih miselnih igrah. Poskušamo najti rešitev in nadaljevati z reševanjem, pa ne znamo naprej, na primer, ko rešujemo sudoku in nismo prepričani katera številka sodi kam, ko imamo dilemo. Takrat bi nam prav prišel kakšen namig za reševanje, da bi lažje nadaljevali z igro. Zato sem se odločila napisati program, ki uporabniku pomaga pri reševanju sudokujev. S tem, ko programu podamo nam že znane številke, nam ta vrne eno dodatno rešeno polje. Tako si lahko pomagamo s podano številko in nadaljujemo igro. Če nam to ne pomaga, pa lahko program ponovno prosimo, da nam poda še eno številko in nam s tem še dodatno olajša delo.

Osnovno znanje programiranje sem se naučila pri urah informatike v šoli. Za dodatna znanja, ki jih bom potrebovala, se bom naučila preko spletnih portalov za učenje programiranja v Pythonu.

## 1.2 Uporabljena strojna oprema

Za nalogo sem uporabila naslednje računalnike:

*Tabela 1: Strojna oprema - prvi računalnik*

| | |
|---|---|
| Procesor | Apple M1 |
| RAM | 8 GB |
| Grafična kartica | Integrirana 8-core GPU |
| Zaslon | 13,3 inch |

*Tabela 2: Strojna oprema - drugi računalnik*

| | |
|---|---|
| Procesor | Intel(R) Core(TM) i5-2500 CPU @ 3,3 GHz |
| RAM | 8 GB |
| Grafična kartica | integrirana |
| Zaslon | 24 inch |

*Tabela 3: Strojna oprema - tretji računalnik*

| | |
|---|---|
| Procesor | 2,5 GHz Dual-Core Intel Core i5 |
| RAM | 16 GB |
| Grafična kartica | Intel HD Grephics 4000 1536 MB |
| Zaslon | 27 inch |

## 1.3 Uporabljena programska oprema

Za sam projekt sem uporabljala urejevalnik kode Visual Studio Code in celotno kodo napisala s tem programom.
Dokumentacijo sem napisala v programu Microsoft Word for Mac, verzija 16.80.

*Tabela 4: Programska oprema*

| | |
|---|---|
| Operacijski sistem | MacOS Sonoma 14.0 |
| Programski jezik | Python 3.12.1 |
| Urejevalnik | Visual Studio Code |
| Urejevalnik besedila | Microsoft Word for Mac 16.80 |
| Grafični vmesnik | Aqua |
| FTP prenos datotek na strežnik | Commander One PRO |
| Spletni brskalnik | Safari |

# 2 TEORETIČNI DEL

## 2.1 Opis igre

Za začetek moramo razumeti, kako sudoku sploh deluje in kako se ga rešuje. Sudoku je igra, kjer imamo kvadrat razdeljen na 81 enakih polji. Ta so razdeljena na enote po 9 kvadratkov, ki skupaj tvorijo kvadrat. Na začetku igre so v nekaterih poljih že podana števila, število števil, ki so podana, je odvisno od zahtevnosti sudokuja. Cilj igre je v vsako polje zapisati število od 1 do 9, tako da se ta ne ponavljajo znotraj stolpca, vrstice in podenote (9 kvadratkov).

Za reševanje obstaja veliko različnih tehnik, ki so odvisne od posameznika.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | | 5 | 8 | 3 | | 2 | |
| | 5 | 9 | 2 | | | 3 | | |
| 3 | 4 | | | | 6 | 5 | | 7 |
| 7 | 9 | 5 | | | | 6 | 3 | 2 |
| | | 3 | 6 | 9 | 7 | 1 | | |
| 6 | 8 | | | | 2 | 7 | | |
| 9 | 1 | 4 | 8 | 3 | 5 | | 7 | 6 |
| | 3 | | 7 | | 1 | 4 | 9 | 5 |
| 5 | 6 | 7 | 4 | 2 | 9 | | 1 | 3 |

*Slika 1: Primer enostavnega sudokuja*

## 2.2 Namen in cilj

Namen moje seminarske naloge je olajšati igranje sudokuja s podajanjem namigov.

Cilj je ustvariti uporabniku prijazen program, ki od igralca zahteva vnos njegove igre sudoku in mu nato program vrne namig v obliki ene dodatne številke v tabeli. Poleg tega uporabniku izpiše katero število je dodal in v katero vrstico ter stolpec.

## 2.3 Že obstoječi programi

Na internetu sem zasledila kar nekaj spletnih strani in You Tube posnetkov, ki prikazujejo izdelavo programa, ki reši sudoku. Vsi ti programi so zasnovani tako, da ti v celoti rešijo sudoku. Svoj program pa sem nadgradila tako, da ti ne reši vsega. Poleg tega sem ga naredila uporabniku bolj prijaznega, saj ti na začetku izpiše kaj moraš kot uporabnik storiti, kar pri drugih programih nisem zasledila. Prav tako ti pri izpisani rešitvi (namigu) napiše v točno katero celico ti je dopisal število, zato, da uporabniku ni treba primerjati vsake celice in gledati, kje nastopa novo število.

## 2.4 Ideja pred izdelavo programa

Sam program vključuje izvajanje algoritma za zapolnitev praznih celic ob upoštevanju pravil igre. Izdelave programa se bom lotila tako, da bom začela z izdelavo mreže. Ker vpisujemo številke od 1 do 9, bodo 0 predstavljale prazna polja.

Napisala bom funkcijo za iskanje naslednje prazne celice v mreži, ki vrne vrstico in stolpec prve prazne celice, na katero naleti.

Funkcija bo preverila, ali se lahko število v skladu s pravili igre vstavi v določeno celico. Preveriti mora, da ne krši nobenih pravil igre (se ne ponovi v stolpcu, vrstici ali podenoti).

Program bo zaključil z delovanjem, ko ne bo ostalo nič več praznih celic.

Ko bom uspela napisati program, ki pravilno reši celoten sudoku in uporabniku izpiše vse rešitve, se bom lotila tega, da uporabniku ne vrne celotno rešenega sudokuja, ampak da doda k že v naprej podanim številkam samo eno dodatno in pri tem ostane.

Če si uporabnik želi oziroma potrebuje še eno številko, program ponovno vrne samo eno dodatno rešitev in ne celotne. Polje, v katerem vrne rešitev, je naključno izbrano in nad tem uporabnik nima vpliva, tako ne more sam določiti, kje želi prikazano rešitev. Namen je spodbuditi uporabnika k razmišljanju in ne, da samo ponudi točno določeno željeno rešitev.

Na koncu, ko bo program napisan in bo deloval tako, kot sem si to zamislila, se bom lotila estetskega urejanja. S tem bom zagotovila, da je program razumljiv in enostaven za uporabnika, hkrati pa vizualno privlačen.

# 3 PRAKTIČNI DEL

Commented [HG1]: Od tu je naslov Praktični del (glavni naslov)povečaj sliko, da bo pisava programa izgledala približno enako velika po vsej dokumentaciji

## 3.1 Izdelava programa

Problema sem se lotila tako, da sem najprej definirala svojo tabelo, ki je v tem primeru polje veliko 9 x 9 prostorov. Takšno polje sem ustvarila v programu za testiranje njegovega delovanja.

```
"""
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
"""
```

*Slika 2: Primer sudoku polja*

V postopku izdelave programa sem pogosto med ukaze vstavljala *»print«,* zato, da sem vedela, ali mi program izpiše tisto, kar si želim.

V vsaki funkciji, ki sem jo napisala, sem morala definirati, da uporabljam le števila od 1 do 9.

Prva funkcija *»prosto_mesto (tabela, vrstica, stolpec, stevilo)«* je namenjena temu, da v tabeli najde mesta, ki so prazna. To stori tako, da se premika po vrsticah in stolpcih. Prazna mesta so zapisana z ničlo (0). Poleg vrstice in stolpca sem morala v tabeli definirati še manjše enote, ki so velike 3 x 3 prostorov. Funkcija deluje tako, da vzame *»stevilo«* in ga postavi v »tabelo« (neko polje na tabeli), nato pa preveri, da se na tem mestu predhodno ne pojavlja že kakšna druga številka (1-9), ali se številka, ki jo program želi postaviti, že nahaja v isti vrstici, stolpcu ali manjši podenoti. Če ne krši nobenega od teh pogojev, program izpiše *True*, v nasprotnem primeru pa *False*.

```
def prosto_mesto(tabela, vrstica, stolpec, stevilo): #to je samo za iskanje prostega mest
    # zato, da preveri ali je številka že v vrstici
    if stevilo in tabela[vrstica]:
        return False

    # preveri še stolpec in pazi !!
    if stevilo in [tabela[i][stolpec] for i in range(9)]: #število mora biti med 1 in 9
        return False

    # preveri v kvadratkih
    prva_vrstica, prvi_stolpec = 3 * (vrstica // 3), 3 * (stolpec // 3)
    for i in range(prva_vrstica, prva_vrstica + 3):
        for j in range(prvi_stolpec, prvi_stolpec + 3):
            if tabela[i][j] == stevilo:
                return False
```

*Slika 3: Funkcija za iskanje prostega mesta*

Na sliki je prikazan končni del programa z nekaj komentarji. Iz teorije je nastal takšen program v praksi, ki pa je na začetku predstavljal težave, saj sem v program narobe vstavljala besedi vrstica in stolpec oz. sem ju zapisala v napačnem zaporedju in zaradi tega program ni deloval.

Naslednja funkcija je *»resitev(tabela)«*, ta funkcija določa rešitve sudokuja. Deluje tako, da se vrača nazaj in to ponavlja za vsako polje, dokler ne doseže konca. Najde prazno celico (zapisano z 0) in vanjo vstavi številko od 1 do 9, brez, da krši pogoje prejšnje funkcije. Ko najde prvo ustrezno številko, jo vpiše v celico in se premakne k naslednji. To nadaljuje skozi celotno tabelo. V primeru, da naleti na problem (v določeno celico ni mogoče vstaviti nobene pravilne številke), se vrne nazaj in poskusi z drugo številko. To ponavlja, dokler ne reši pravilno celotne tabele. Ko funkciji to uspe, vrne *True*, v nasprotnem primeru vrne *False*.

```
if resitev(tabela):
    dodatno_stevilo = novo_stevilo(prvotna_tabel
    if dodatno_stevilo:
        vrstica, stolpec = dodatno_stevilo
        prvotna_tabela[vrstica][stolpec] = tabel
        print("\nŠtevilo:", tabela[vrstica][stol
        print("Vrstica:", vrstica + 1)  # izpiše
        print("Stolpec:", stolpec + 1)  # izpiše
    else:
        print("\nSudoku je že rešen.") #če je že
else:
    print("\nSudoku ni rešljiv.") # če ni rešliv,
```

Del te funkcije predstavlja tudi bolj interaktivni del, kjer je zapisano to, kar uporabnik prejme kot povratno informacijo, tudi v primeru, če njegov sudoku ni rešljiv.

Funkcijo *»print_tabela(tabela)«* sem naredila zaradi estetskega videza. Njen namen je narediti presledke med številkami v vsaki vrstici tabele za boljšo preglednost in lažjo berljivost.

```
def print_tabela(tabela):
    for vrstica in tabela:
        print(" ".join(map(str, vrstica))) #če javlja na
```

*Slika 4: Naslednji del programa*

Pri pisanju te funkcije sem si veliko pomagala z internetom, da sem se naučila in posledično lahko uporabila ukaze, ki jih pred tem nisem poznala.

Funkcijo *»novo_stevilo(tabela)«* sem naredila zaradi nadgradnje programa. Njen namen je iskanje prve proste celice (na začetku, preden je sudoku rešen) v katero vstavi namig (število).

```
def novo_stevilo(tabela):
    for i in range(9):
        for j in range(9):
            if tabela[i][j] == 0:
                return i, j
    return None #ne vrne nič, če n
```

*Slika 5: Naslednji del*

Namen *»main()«* je, da poveže vse predhodne funkcije in uporabnikov input. Deluje tako, da po tem, ko uporabnik vpiše podatke, začne delovati funkcija *»prosto_mesto (tabela, vrstica, stolpec, stevilo)«* in ko najde rešitev, uporabi funkcijo *»novo_stevilo(tabela)«*. Izpiše prvotno tabelo z dodatkom nove številke in položaja nove številke. Prav tako pa preveri ali rešitev obstaja, na kar opozori uporabnika naslednji pogojni stavek v funkciji, ki je odgovoren za podajanje odgovora v obliki povedi uporabniku.

Zadnji del programa preverja neposredno izvajanje programa po skripti.

## 3.2 Postopni programi do končnega programa

```
board = [
   [0,0,7,0,0,6,0,1,0],
    [0,4,0,0,0,0,0,9,0],
    [8,0,0,0,5,0,6,0,4],
    [0,1,0,0,0,5,7,0,2],
    [0,0,0,0,0,0,0,6,0],
    [0,0,3,0,8,0,0,0,0],
    [0,2,0,0,0,7,4,0,5],
    [1,0,0,2,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,9]
]

def solve(a):
    f = f_empty(a)
    if not f:
        return True
    else:
        r, c = f

    for i in range(9): # 1-9, polja
        if v(a, i, (r, c)):
            a[r][c] = i

            if s(a):
                return True

            a[r][c] = 0

    return False
```

*Slika 6: Del prvega programa*

Del prvega programa, pri katerem uporabnik ni mogel spremeniti tabele, ker je bila le ta zapisana znotraj programa.

Prav tako sem se pri pisanju tega programa lotila pisanja spremenljivk s kraticami v obliki ene črke, kar me je kasneje zmedlo in sem se iz tega naučila, da bom uporabljala bolj pomenske kratice za spremenljivke, ki mi bodo direktno povedale kaj ta spremenljivka je.

```
def solve_sudoku(board):
    for i in range(9):
        for j in range(9):
            if board[i][j] == 0:
                for num in range(1, 10):
                    if is_valid_move(board, i, j, num):
                        board[i][j] = num
                        if solve_sudoku(board):
                            return True
                        board[i][j] = 0
                return False
    return True
```

*Slika 7: Del drugega programa*

Del drugega programa, kjer sem spremenljivke pisala z angleškimi kraticami, kar se je kasneje izkazalo, da ni idealno, zato sem začela uporabljati cele slovenske besede.

## 3.3 Navodila za uporabnika

Za odpiranje datotek uporabi IDLE ali kateri koli drugi program, ki podpira Python datoteke (npr. Visual Studio Code).

Ko je program odprt ga zaženemo. V datoteki se bodo izpisala navodila za uporabo, kjer je podan primer sudokuja (kako mora igledati ta, ki ga želimo vstaviti).

Za vsako novo vrstico lahko uporabimo tipko ENTER brez strahu, da se bo program sprožil. Ta se sproži šele po vnosu zadnje vrstice.

```
========== RESTART: /Users/anacetarin/Downloads/Program1.py ==========
Navodila:
Vstavi svoj sudoku. Prazna polja zapiši z 0. Med števili zapiši presledke.
Ko enkrat nadaljuješ na naslednjo vrstico se ne moreš vrniti in popraviti.
Na koncu vsake vrstice, z izjemo zadnje mora biti presledek.
Primer zapisa sudokuja:
0 0 9 0 0 3 5 6 0
4 0 0 2 0 0 0 0 0
0 0 0 8 0 9 0 1 2
3 4 0 0 0 0 2 8 0
0 0 0 0 0 6 0 0 5
0 5 2 3 0 0 0 0 0
6 9 4 7 5 0 0 2 3
0 2 1 9 3 0 6 7 8
8 0 0 6 1 0 0 0 0
```

*Slika 8: Navodila za uporabnika*

# 4 Zaključek

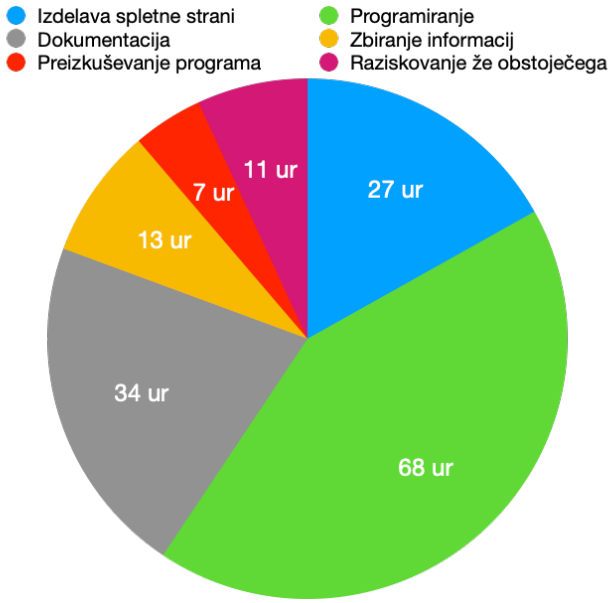
## 4.1 Sklep

Na začetuku sem bila prepričana, da izdelava programa ne bo težka, vendar sem kaj kmalu ugotovila nasprotno, saj sem na poti do cilja naletela na veliko problemov. Problemi so se mi predvsem pojavljali pri programiranju, saj sem imela omejeno znanje, kar pa sem nadgradila s pomočjo različnih spletnih strani ter tudi preko krožka programiranja. S tem sem spoznala veliko novih načinov za reševanje problemov. Na poti do končnega izdelka sem ustvarila veliko različnih programov od katerih sem se učila in so vodili do tega, da lahko na koncu oddam delujoč program na katerega sem ponosna.

## 4.2 Izboljšave

Čeprav sem ponosna na svoj izdelek, bi lahko dodelala njegovo delovanje. Na primer lahko bi naredila, namesto, da vstavi številko v prvo naslednje prazno polje, da bi izbral naključnega. Prav tako program ne izpiše celotnega novega sudokuja, ampak samo številko, ki jo je potrebno naslednjo vstaviti in v katero vrsto ter stolpec. Prav tako pa program, če ne zazna praznih polj napiše, da je sudoku rešen, tudi če je ta rešen napačno. Poleg tega pa bi lahko naredila še velik napredek na estetskem področju prikaza programa.

## 4.3 Graf porabljenega časa




*Graf: Graf porabljenega časa 1*

# 5 VIRI IN LITERATURA

Starc Grlj, H. [Ustno izročilo ali Po pripovedi]. Ljubljana, 2024
Sojer, V. [Ustno izročilo ali Po pripovedi]. Ljubljana, 2024
Data Flair (online). [uporabljeno 2023 ] https://data-flair.training/blogs/python-sudoku-game/
Programiz (online). [uporabljeno 2023] https://www.programiz.com/python-programming/dictionary
W3Schools (online). [uporabljeno 2023/24] dostopno na https://www.w3schools.com/python/default.asp
Python (online). [uporabljeno 2024] dostopno na https://docs.python.org/3/tutorial/datastructures.html
Geeks for Geeks (online). [uporabljeno 2024] https://www.geeksforgeeks.org/building-and-visualizing-sudoku-game-using-pygame/
RIN1 (online). [uporabljeno 2023/24] https://lusy.fri.uni-lj.si/ucbenik/book/index.html#
TutorialsPoint za Python (online), dostopno na https://www.tutorialspoint.com/python/index.htm

(
https://www.youtube.com/watch?v=kqtD5dpn9C8
https://www.youtube.com/watch?v=NB5LGzmSiCs
https://www.youtube.com/watch?v=sugvnHA7ElYhttps://www.youtube.com/watch?v=sugvnHA7ElY
https://www.youtube.com/watch?v=-yzfxeMBe1s
https://www.youtube.com/watch?v=tLxOx-Wl54g
https://stackoverflow.com/questions/45471152/how-to-create-a-sudoku-puzzle-in-python
)