Zavod sv. Stanislava

Škofijska klasična gimnazija

PROGRAMIRANJE V PYTHONU

Programiranje raziskovalca datotek

Maturitetna seminarska naloga iz Informatike

Kandidat: Adam Dominko

Mentor: Valentin Sojar

Ljubljana Šentvid, april 2026

Povzetek

Seminarska naloga obravnava problem neurejenosti digitalnih podatkov in zamudno ročno razvrščanje datotek. Rešitev je izvedena v programskem jeziku *Python* z uporabo knjižnic *os* in *shutil*, ki omogočata avtomatizirano upravljanje sistemskih poti. Razviti program uporabniku omogoča pregled vsebine map, ročno premikanje elementov ali popolnoma samodejno razvrščanje datotek v mape glede na njihove končnice. Rezultat je delujoče orodje v terminalu, ki dokazuje, da lahko s preprosto logiko in avtomatizacijo bistveno povečamo učinkovitost pri delu z računalnikom.

Abstract

This seminar paper addresses the issue of digital clutter and the time-consuming process of manual file organization. The solution is implemented in *Python*, utilizing the *os* and *shutil* libraries for automated system path management. The developed program enables users to browse folder contents, move items manually, or perform fully automated file sorting based on file extensions. The result is a functional terminal-based tool, demonstrating that simple logic and automation can significantly increase efficiency in daily computer tasks and data management.

Ključne besede

Python, avtomatizacija, upravljanje datotek, shutil, os, time.

Kazalo vsebine

Kazalo slik

Kazalo tabel

Stvarno kazalo

1 UVOD

1.1 Vpeljevanje v problem

Za seminarsko nalogo sem želel narediti program, ki pa bo tudi za nekaj uporaben. Ker imamo na svojih napravah veliko map in datotek, ki dostikrat niso urejene, sem se domislil, da bi za ta problem naredil rešitev. Pogosto se zgodi, da datoteko težko najdemo, ker ne vemo točnega imena ali pa ne vemo, v kateri mapi se nahaja. Odločil sem se, da naredim program Raziskovalec datotek, ki bo pomagal pri organizaciji datotek, hkrati pa bo imel tudi nekaj osnovnih funkcionalnosti, kot so kopiranje, premikanje in pregled vsebine map. Ročno urejanje lahko pri velikemu številu datotek vzame veliko časa, zato je smiselno imeti program, ki to vsaj delno avtomatizira.

1.2 Predstavitev problema

Sprva sem želel narediti raziskovalca datotek, vsaj malo podobnega tistemu, ki ga ima operacijski sistem *Windows*. Poleg kopiranja, premikanja datotek pa sem želel dodati tudi funkcionalnost organiziranja datotek. Neorganizirano strukturo map, podmap in datotek sem želel urediti z nekaj kliki. Želel sem narediti program, ki ima tudi dober grafični uporabniški vmesnik in bi imel veliko podobnosti z s programom Raziskovalec na operacijskem sistemu *Windows*. Ko sem se lotil, sem ugotovil, da je naloga zelo zahtevna in obširna ter, da potrebnega znanja nimam. Odločil sem se, da poizkusim program še vseeno narediti, vendar brez zahtevnejših delov, le osnovne funkcionalnosti, brez grafičnega uporabniškega vmesnika, niti in ostalih , kar stvar spravi na bolj osnoven nivo programiranja, ki pa se ga lahko lotim. Cilj je bil torej narediti poenostavljenega raziskovalca datotek z dodatno funkcijo organizacije datotek v novo strukturo.

1.3 Uporabljena tehnologija

Pri izdelavi programa je bila uporabljena naslednja strojna in programska oprema.

Tabela 1: Uporabljena strojna oprema

Naprava	Prenosnik HP Victus
Centralna procesna enota	Intel® Core™ i5-12500H
Delovni Pomnilnik	16 GB 4800 MT/s
Grafična procesna enota	Nvidia GeForce RTX 3060 (prenosnik)
Zunanji pomnilnik	1 TB SSD

Tabela 2: Uporabljena programska oprema

Operacijski sistem	Windows 11 Pro
Python	Programski jezik za razvoj aplikacije in obdelavo podatkov
Modul os	Pregled map in delo z datotečnim sistemom
Modul time	Zakasnitev za boljšo preglednost
Modul shutil	Varno premikanje in organizacija datotek
Razvojno okolje	Urejevalnik kode VS Code

2 TEORETIČNA REŠITEV

2.1 Analiza problema in zbranih podatkov

Pri delu z računalnikom se hitro srečamo z ogromnim številom datotek, ki sproti nastajajo ali pa jih prenašamo od drugod. Te datoteke so zelo različnih vrst, od dokumentov v *.docx* formatu, slik v *.jpg,* do raznih video posnetkov v *.mp4* in zvočnih posnetkov v *.mp3* formatu. Če datotek ne pospravljamo sproti, te pogosto končajo v eni sami mapi (večinoma "Prenosi"), kar hitro privede do nepreglednosti. Iskanje določene datoteke v takšnem neredu postane zamudno in utrujajoče. Čeprav operacijski sistemi, kot so *Windows*, vključujejo raziskovalce datotek, ta še vedno zahteva veliko ročnega dela na tem področju. Če želimo mapo zares urediti, moramo sami ustvarjati podmape, brati končnice vsake datoteke posebej in jih z miško prestavljati. To je pri desetih datotekah enostavno, pri stotih pa postane opravilo, ki se mu raje izognemo. Večina iskalnikov v sistemih sicer omogoča iskanje po imenu, a če se imena datoteke ne spomnimo točno, nam to ne pomaga veliko.

Ko sem razmišljal, kako bi to poenostavil, sem ugotovil, da bi bila najboljša rešitev avtomatizacija. Namesto da uporabnik sam pregleduje lastnosti datotek (kot so končnice), lahko to namesto njega naredi preprost program. Moja rešitev je program, ki v trenutku prebere celotno vsebino mape in datoteke samodejno razvrsti v skupine (npr. slike v podmapo Slike, ki jo naredi, dokumente v mapo Dokumenti). S tem odpade potreba po ročnem klikanju, uporabnik pa dobi pregledno urejeno mapo v le nekaj sekundah.

2.2 Razlaga in kritično ovrednotenje rezultatov

Pri preizkušanju programa sem ugotovil, da *Python* s knjižnicama *os* in *shutil* omogoča zelo zanesljivo delo z datotekami. Ukazi, ki sem jih uporabil, omogočajo hitro prebiranje seznama datotek in njihovo varno prestavljanje, kar pomeni, da lahko celoten proces organizacije izvedem programsko, brez ročnega klikanja. Čeprav sem sprva načrtoval grafični uporabniški vmesnik, sem ugotovil, da za samo funkcionalnost to ni nujno, ukazna vrstica (terminal) deluje hitreje in bolj neposredno, predvsem pa je veliko lažje napisati kodo.

Glavna prednost moje rešitve je v hitrosti in uporabnosti. Program v manj kot sekundi pregleda celotno mapo in razvrsti datoteke, za kar bi ročno potreboval najmanj nekaj minut. Poleg tega se s tem izognemo napakam, ki bi jih pri ročnem sortiranju lahko naredili, kot je npr. spregledana datoteka ali premikanje v napačno mapo, to pa se programu ne zgodi, saj ima napisana pravila za končnice, po katerih organizira. Vendar pa ima takšna rešitev tudi svoje omejitve oziroma slabosti. Ena izmed njih je, da program razvrsti le tiste datoteke, ki jih ima vnaprej določene v kodi (npr. *.jpg*, *.pdf*). Če naleti na format, ki ga ne pozna, tista datoteka ostane na starem mestu. Druga omejitev je prepisovanje datotek, saj program ne preverja, ali v ciljni mapi že obstaja datoteka z istim imenom, kar bi v teoriji lahko privedlo do prepisa podatkov.

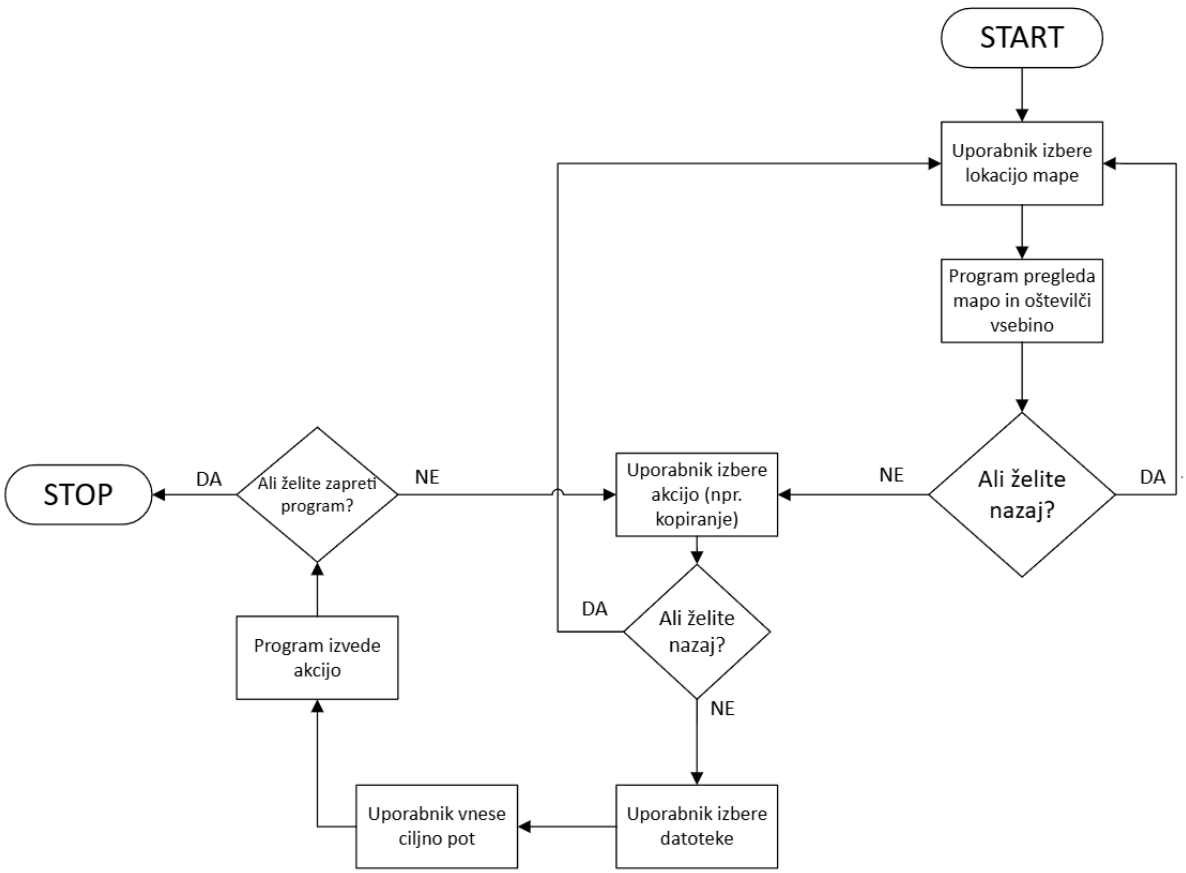
Rezultati mojega dela kažejo, da je takšna avtomatizacija izjemno uporabna za organiziranje neurejenih map. Čeprav je program preprost in nima zapletenega vmesnika, svojo glavno nalogo opravi dobro, je pa res zelo preprost in ima, z več znanja *Pythona*, veliko možnosti za napredek.

2.3 Opis rešitve

Moja rešitev je v *Pythonu* napisan program, ki deluje kot pomožni raziskovalec datotek. Zaradi tega in njegove preprostosti sem ga poimenoval Raziskovalec datotek Mini. Namesto zapletenih grafičnih oken program uporablja *terminal,* kjer se izpišejo preprosti teksti, preko katerih uporabnik ukazuje, kaj naj se z datotekami zgodi. Glavni cilj programa je avtomatizacija organizacije datotek, poleg tega pa tudi osnovni procesi kot so kopiranje ipd. Program je razdeljen na več manjših delov oziroma funkcij, ki jih na koncu glavna funkcija programa vse prikliče. Vsaka funkcija ima svojo nalogo. Na začetku je funkcija, ki zahteva pot do mape in se ne konča dokler ne dobi prave poti, oziroma poti, ki obstaja. Glavni meni je bistvo uporabniškega vmesnika, saj preko njega uporabnik lahko daje ukaze programu. Meni ima 5 možnosti, uporabnik s tem da vnese eno izmed številk 1-5 izbere možnost, kot je na primer samodejno sortiranje ali pa premikanje. Preden pa uporabnik lahko izbira med datotekami in mapami pa program prebere mapo, ki jo uporabnik izbere, s pomočjo *os* modula (*os.listdir*). Nato datoteke in mape izpiše v *terminal*.

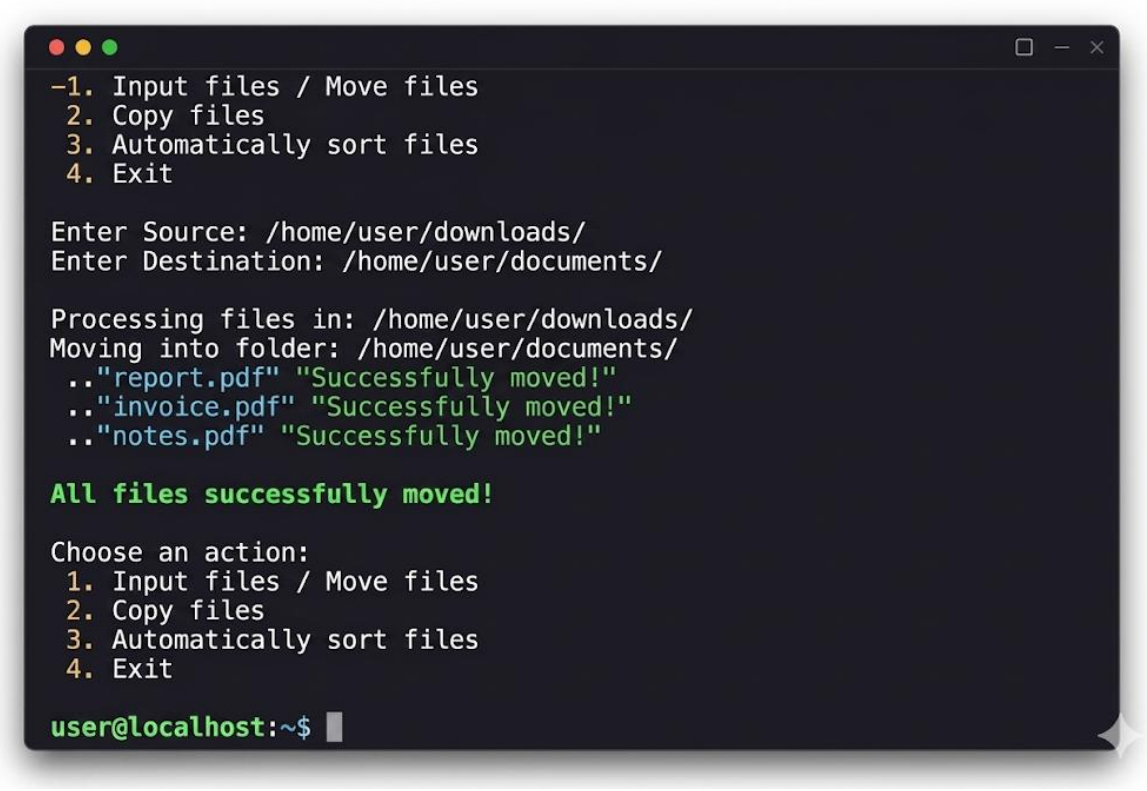
2.4 Diagram poteka

Diagram poteka aplikacije prikazuje zaporedje delovanja programa od zagona do zaključka. Uporabnik najprej izbere mapo, ki jo želi pregledati, nato program analizira datoteke in njihove lastnosti. Na podlagi uporabnikove izbire se izvede iskanje ali organizacija datotek. Diagram omogoča boljše razumevanje delovanja aplikacije.



Slika 1: Diagram poteka programa

Pomemben del diagrama je zanka, ki omogoča, da se po vsaki izvedeni akciji (kopiranje, premikanje, razvrščanje) uporabnik vrne v glavni meni, namesto da bi se program zaprl in bi ga moral uporabnik še enkrat zagnati.



Slika 2: Primer končnega izdelka

3 PRAKTIČNA REŠITEV

3.1 Potrebna oprema in pogoji za delovanje

Program sem razvijal na prenosnem računalniku z operacijskim sistemom *Windows 11*, a ga je možno zagnati tudi na sistemih *Linux* ali *MacOS*. Od strojne opreme potrebujem le osnovni procesor, nekaj delovnega pomnilnika in seveda prostor na disku, kjer se nahajajo datoteke, ki jih želimo urejati.

Glede programske opreme je edini pogoj, da imamo nameščen *Python*. Za samo pisanje kode sem uporabljal *Microsoftov* urejevalnik *Visual Studio Code*, ker mi omogoča hiter pregled nad mapami in ima vgrajen terminal s pomočjo katerega lahko uporabljam program. Omeniti moram, da program ne potrebuje nobene dodatne namestitve knjižnic s spleta, saj sem uporabil samo tiste module, ki so že standardno zraven ob namestitvi Pythona. To so *os* za delo s sistemom, *shutil* za premikanje datotek in *time* za nadzor hitrosti izpisa potrditvenih sporočil.

3.2 Struktura programa

Na vrhu kode sem uvozil potrebne module. Uporabil sem tri osnovne: *os, shutil* in *time*. Knjižnica os je potrebna za delo z operacijskim sistemom, saj mi omogoča, da program "vidi" mape, preveri, če pot obstaja, in prebere imena datotek. *Shutil* sem dodal, ker poskrbi za dejansko premikanju oziroma urejanje datotek. Modul *time* pa sem uporabil za en sam ukaz (*sleep*), ki programu pove, naj za sekundo počaka, preden nadaljuje, kar uporabniku omogoči, da sploh prebere potrditvena sporočila na zaslonu.

3.2.1 Splošna struktura

Program sem si zamislil tako, da je razdeljen na več funkcij, saj je bolj enostavno in tudi bolj pregledno. Takšna struktura je tudi bolj smiselna, saj vsak del kode opravlja točno določeno nalogo, kar mi je med programiranjem zelo olajšalo iskanje napak. Program je v osnovi sestavljen iz dveh delov in sicer pomožnih funkcij, ki skrbijo za posamezne akcije (izpis, kopiranje, razvrščanje), in glavne funkcije (*glavni_program*), ki vse to povezuje v smiselno celoto in skrbi za uporabniški meni. Na koncu kode prikličem funkcijo *glavni_program*, ki zažene program in kliče vse ostale funkcije.

3.2.2 Pomožne funkcije

Na začetku kode sem napisal funkcijo, ki ob zagonu programa Mini raziskovalec datotek prikaže ime programa in nekakšno navodilo »vnesite pot in začnite urejati«, naj uporabnik vnese pot želene mape.

```
def prikazi_uvod():
    print("\n*********************************************")
    print(" MINI RAZISKOVALEC             ")
    print(" Vnesite pot in začnite urejati.        ")
    print("*********************************************\n")
```

Slika 3: Uvodna funkcija

Sledi funkcija za izpis datotek v mapi, ki smo jo določili. Ta prebere podatke z diska in jih pretvori v pregleden in oštevilčen seznam. Za branje podatkov sem uporabil *os.listdir, os* prikliče modul za datoteke, *listdir* pa poskrbi, da se izpišejo mape in datoteke iz lokacije, ki smo jo določili (*trenutna_pot*) in seznam shrani v spremenljivko *vsebina*. Funkcijo sem začel z *try*, če slučajno ni možno prebrati podatkov, zaradi napačne poti ali pa česa drugega, vrne napako, ki nam da vedeti da smo vnesli napačno pot in v tem primeru vrne prazen seznam, da se izognem napaki *TypeError*. Če je mapa vsebina prazna, pa lahko takoj zaključimo funkcijo in ne potrebujemo naslednje zanke *for*.

```
try: #poskuša prebrati vsebino v mapi
    vsebina = os.listdir(trenutna_pot) #os.listdir vrne seznam vseh datotek in map v trenutni poti
    #os je vgrajena knjižnica za delo z datotekami, listdir pa ukaz znotraj knjižnice, ki pogleda v
except:  #če ne more prebrati vsebine vrne to:
    print("NAPAKA: Pot ni dostopna ali ne obstaja!")
    return [] #vrne prazen seznam, če bi vrnila None potem bi dobil TypeError

if not vsebina: #če je mapa prazna lahko takoj zaključimo in ne potrebujemo zanke for
    print("Mapa je prazna.")
    return []
```

Slika 4: Prvi del funkcije izpisi_datoteke

Zanka *for* pa gre čez seznam datotek in map ter jih oštevilči in izpiše. Najprej se z zanko sprehodi čez datoteke in napiše polno pot, torej trenutno pot združi z imenom datoteke. Tukaj sem uporabil *os.path.join,* ki to zelo priročno združi. Potem pa z *os.path.isdir* preveri ali je polna pot pot mape in če je, shrani »MAPA« v spremenljivko tip, če pa ni mapa pa z *else* shrani »DAT« v spremenljivko tip. Na koncu funkcije print poskrbi, da program izpiše seznam datotek in map z indeksom, tipom ter imenom. Na koncu funkcija vrne vsebino.

```
# gre čez seznam datotek in jih oštevilči ter izpiše
i = 0
for ime in vsebina:  #z zanko se sprehodimo čez vsebino torej datoteke in ime je spremenljivka ki
    polna_pot = os.path.join(trenutna_pot, ime) #to da pot mape in ime datoteke skupaj
    if os.path.isdir(polna_pot): #os.path.isdir preveri ali je polna_pot mapa
        tip = "[MAPA]"
    else:          #če ni mapa je datoteka
        tip = "[DAT]"
    print(i, ":", tip, ime)
    i = i + 1

print("----------------------------------------------------")
return vsebina
```

Slika 5: Drugi del funkcije izpisi_datoteke

Sledi funkcija *upravljaj_datoteko*, ki skrbi za kopiranje in premikanje datotek. Najprej izpiše vse možne akcije. Vpraša nas, ali želimo iti nazaj in potem pa naš vnos, shrani v vnos_indeks, in nas zato pogojni stavek vrne nazaj, saj ustreza pogoju. Potem sledi pogojni stavek, kjer je pogoj, da je vneseni indeks številka s pomočjo *.isdigit*. Potem pretvori vneseno številko v *integer* in če je ta večji od dolžine seznama vrne napako, če pa ne vnesemo nič, pa tudi vrne napako. V *izbrano_ime* se shrani ime datoteke ali mape na mestu vpisane številke. Potem v spremenljivko *cilj* vnesemo ciljno pot ali »n« za nazaj.

```python
def upravljaj_datoteko(pot, seznam, akcija):
    print() # Prazen prostor za preglednost
    print("---", akcija.upper(), "---") #spremeni male tiskane v velike tiskane
    vnos_indeks = input("Vpiši številko datoteke (ali 'N' za NAZAJ): ")

    if vnos_indeks.lower() == 'n':  #če pritisneš N se funkcija zaključi in sevrne nazaj na glavni meni
        return

    if vnos_indeks.isdigit(): #preveeri ali je vnos res številka
        indeks = int(vnos_indeks) #spremeni vpisano število v integer da lahko uporabimo kot indeks v se
        if indeks >= len(seznam): #če je vnesena številka večja kot zadnji indeks na seznamu vrne napako
            print("Napaka: Številka ni na seznamu!")
            return
    else:   #če ni številka vrne napako
        print("Napaka: Vpisati moraš številko!")
        return

    izbrano_ime = seznam[indeks] #iz seznama vzame ime datoteke ali mape na mestu vpisane številke
    print("Izbrali ste datoteko:", izbrano_ime)
    cilj = input("Vnesi ciljno pot (ali 'N' za nazaj): ") #vnesemo cilno pot kamor želimo premakniti/kop

    if cilj.lower() == 'n': #preveri če je uporabnik vpisal n, dela pa tudi N (zato lower)
        return
```

Slika 6: Prvi del funkcije upravljaj_datoteko

V drugem delu te funkcije pa se v spremenljivko *nova_pot* shrani cilj kamor želim premakniti ali kopirati datoteko. Shrani se s pomočjo *os.path.join*, ki združi pot in ime datoteke. Nižje je pogojni stavek, ki preveri, če ciljna lokacija obstaja, in nato izvede akcijo, kopiranje ali premikanje s pomočjo *shutil.copy* in *shutil.move* iz stare lokacije na novo. Če pa ciljna pot ne obstaja vrne napako.

```python
    stara_pot = os.path.join(pot, izbrano_ime) #združi naslov trenutne mape in ime izbrane datoteke v
    nova_pot = os.path.join(cilj, izbrano_ime) #združi naslov ciljne mape in ime datoteke, da se ve, k

    if os.path.exists(cilj): #preveri če mapa v katero želimo kopirati/premakniti sploh obstaja, če ne
        if akcija == "kopiraj":
            shutil.copy(stara_pot, nova_pot) #shutil.copy kopira datoteko iz stare_pot v nova_pot
        else:
            shutil.move(stara_pot, nova_pot) #shutil.move premakne datoteko iz stare_pot v nova_pot
        print("Uspešno izvedeno!")
    else:
        print("Napaka: Ciljna pot ne obstaja!")
```

Slika 7: Drugi del funkcije upravljaj_datoteko

Pomembna funkcionalnost programa pa je samodejno razvrščanje datotek, kar pa se zgodi v funkciji *samodejno_razvrsti*. Najprej z *os.listdir* dobimo seznam datotek in map, potem pa z zanko *for* razvrsti datoteke. Na začetku preveri ali je polna pot mapa, in če je, jo preskoči saj obdeluje le datoteke. Z *os.path.splitext* pa pridobimo ime končnice datoteke (npr. .jpg). Sledi kategorizacija, ki s pomočjo pogojev razvrsti datoteke po mapah glede ime končnice datoteke. Naslednji pogojni stavek preveri (*os.path.exists(pot_mape)*) če je mapa že določena in se nato datoteka premakne v to mapo (npr. .pdf se premakne v DOKUMENTI, če ta mapa že obstaja) in če mapa še ne obstaja, naredi novo primerno mapo (*os.mkdir)* in premakne datoteke *shutil.move*.

3.2.3 Jedro programa

Zadnja funkcija pa je tista, ki je osnova programa in sicer *glavni_program*. Ta funkcija tudi kliče vse ostale funkcije po potrebi. Na začetku kliče funkcijo za uvod *prikazi_uvod*, potem pa pripravi prazno spremenljivko v kateri bo lokacija mape. Zanka *while* se ponavlja, dokler ne dobi veljavne poti.

```
def glavni_program():
    prikazi_uvod() #prikliče funkcijo ki prikaže uvodni napis

    moja_pot = "" #pripravi prazno spremelnljivko v kateri bo lokacija mape
    while moja_pot == "": #zanka se ponavlja dokler uporabnik ne vnese veljavne poti
        vnos_poti = input("Vpiši pot do mape, ki jo želiš upravljati: ")

        # Preverimo, če vnesena pot sploh obstaja na računalniku
        if os.path.exists(vnos_poti):
            moja_pot = vnos_poti
            print("Pot uspešno sprejeta!")
        else:
            print("NAPAKA: Ta pot ne obstaja. Poskusi ponovno.") #in še enkrat vpišemo pot
```

Slika 8:Prvi del funkcije glavni_program

Dodal sem tudi da po vsakem ukazu oziroma izvedeni akciji ponovno prikaže glavni meni, ker imam možnosti kopiraj, premakni, samodejno razvrsti, spremeni trenutno mapo te izhod. Možnost spremeni trenutno mapo omogoča, da vnesem novo absolutno pot mape, ki jo želim preurejati.

```
#glavni meni
print("\nGLAVNI MENI:")
print("1 - Kopiraj")
print("2 - Premakni")
print("3 - Samodejno razvrsti mapo")
print("4 - Spremeni trenutno mapo")
print("5 - IZHOD")
```

Slika 9: Glavni meni

Drugi del funkcije *glavni_program* pa vsebuje pogojne stavke. S pomočjo *if* in *elif* program zažene pomožno funkcijo glede na to kaj sem izbral v glavnem meniju.

```
        izbira = input("\nIzberi možnost: ") #shrani vneseno številko v izbira

        if izbira == "1":  #če je uporabnik izbral možnost 1, pokliče funkcijo za kopiranje datoteke
            upravljaj_datoteko(moja_pot, seznam_datotek, "kopiraj")
        elif izbira == "2":
            upravljaj_datoteko(moja_pot, seznam_datotek, "premakni") #fun za premikanje
        elif izbira == "3":
            samodejno_razvrsti(moja_pot) #fun za razvrščanje datotek v podmape
        elif izbira == "4":
            nova = input("Vnesi novo pot (ali 'n' za NAZAJ): ") #če želiš vnesti novo pot
            if nova.lower() != 'n' and os.path.exists(nova): #preveri, da uporabnik ni želel nazaj in če
                moja_pot = nova
        elif izbira == "5":
            print("Zapiram program. Nasvidenje!") #izhod iz programa
            break
        else:
            print("Neveljavna izbira!") #katera druga številka kot 1-5 vrne napako

        # Program malce počaka da uporabnik lažje prebere kar napiše program, preden se vrne na glavni m
        time.sleep(1)

# Zagon programa
glavni_program()
```

Slika 10: Drugi del funkcije glavni_program

3.3 Oblika rezultatov

Glavni namen mojega programa je, da poskrbi za bolj urejen računalnik, predvsem za pregledno razporeditev datotek in map. Ko program konča svoje delo, lahko uporabnik rezultate opazi na dva načina. Prvi je izpis v terminalu urejevalnika kode. Med delovanjem program sproti izpisuje, kaj izvaja. Če na primer premaknem datoteko, se prikaže sporočilo »Uspešno izvedeno!«. Če pride do napake (npr. napačno vnesena pot do mape), program na to opozori, da vemo kaj je narobe in to lahko odpravimo. Zaradi ukaza time.*sleep* se sporočila ne izpišejo prehitro, ampak jih lahko normalno preberem, preden se prikaže glavni meni. Druga oblika rezultatov pa je vidna na disku. Ko odprem mapo, vidim, da so datoteke res premaknjene ali kopirane tja, kamor sem želel. Če uporabim funkcijo samodejnega razvrščanja, se v mapi ustvarijo podmape, kot so *SLIKE*, *DOKUMENTI* in *VIDEO*. Datoteke so nato razvrščene glede na svoje končnice, zato je mapa veliko bolj pregledna in ni več potrebe po ročnem iskanju posameznih datotek.

Končni izdelek je poenostavljen raziskovalec datotek s samodejnim sortiranjem.

```
**********************************************
 MINI RAZISKOVALEC
 Vnesite pot in začnite urejati.
**********************************************

Vpiši pot do mape, ki jo želiš upravljati: C:\Users\adamd\Downloads\test
Pot uspešno sprejeta!

--------------------------------------------------
TRENUTNA LOKACIJA: C:\Users\adamd\Downloads\test
--------------------------------------------------
0 : [MAPA] DOKUMENTI
1 : [MAPA] SLIKE
2 : [DAT] test.rar
--------------------------------------------------

GLAVNI MENI:
1 - Kopiraj
2 - Premakni
3 - Samodejno razvrsti mapo
4 - Spremeni trenutno mapo
5 - IZHOD

Izberi možnost: 1

--- KOPIRAJ ---
Vpiši številko datoteke (ali 'N' za NAZAJ): 2
Izbrali ste datoteko: test.rar
Vnesi ciljno pot (ali 'N' za nazaj): C:\Users\adamd\Downloads\
Uspešno izvedeno!

--------------------------------------------------
TRENUTNA LOKACIJA: C:\Users\adamd\Downloads\test
--------------------------------------------------
0 : [MAPA] DOKUMENTI
1 : [MAPA] SLIKE
2 : [DAT] test.rar
--------------------------------------------------

GLAVNI MENI:
1 - Kopiraj
2 - Premakni
3 - Samodejno razvrsti mapo
4 - Spremeni trenutno mapo
5 - IZHOD
```

Slika 11: Končni izdelek

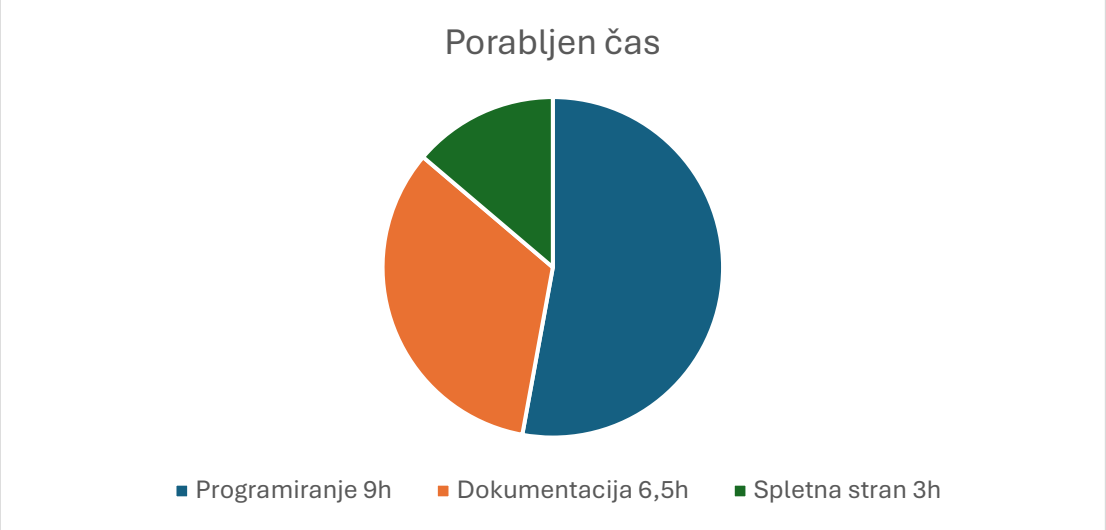
4 SKLEP

V tej nalogi sem se ukvarjal s problemom neurejenih datotek na računalniku. Velikokrat se zgodi, da imamo vse pomešano v eni mapi in potem težko najdemo, kar iščemo. Moj cilj je bil narediti program, ki bi to vsaj delno rešil in olajšal delo. Na koncu sem naredil program v *Pythonu*, ki zna pregledati mapo, premikati in kopirati datoteke ter jih tudi samodejno razvrstiti v podmape glede na končnice. Program deluje hitro in zanesljivo. Mapa, ki je bila prej neurejena, je po uporabi programa postala veliko bolj pregledna. S tem sem dosegel glavni namen naloge.

Seveda pa program ni popoln in je še veliko možnosti za izboljšave. Prepozna samo najbolj pogoste tipe datotek (npr. *.jpg, .pdf, .mp4*), zato katerih drugih, neobičajnih ne zna razvrstiti. Prav tako nima možnosti, da bi razveljavil dejanja, kar pomeni, da moraš biti pri uporabi pazljiv. Ena slabost je tudi ta, da deluje samo v terminalu, kar je lahko uporabnike malo nerodno in ne najbolj enostavno ter estetsko.

V prihodnosti bi lahko program še izboljšal. Dodal bi več tipov datotek ali pa naredil, da bi jih program sam prepoznal. Dobro bi bilo tudi dodati možnost razveljavi, da bi lahko popravil napake. Največja izboljšava pa bi bil grafični vmesnik z gumbi, da programa ne bi bilo treba uporabljati samo preko terminala. Na splošno mislim, da je program uporaben, predvsem za hitro urejanje map. Program je dobra osnova, ni zapleten, svoje delo opravi dobro ampak bi ga bilo mogoče še precej nadgraditi.

Graf 1: Delež porabljenega časa



4.1 Navodila za uporabo

Najprej zaženemo program, in izpiše se naslov ter navodilo, ki nam pove, da moramo vnesti pot do mape, ki jo želimo urejati ali pa sortirati. Nato izberemo akcijo (kopiranje, sortiranje, ...), ki jo bomo izvedli, za tem pa datoteko, na kateri želimo izvesti to akcijo. Nazadnje izberemo še ciljno mapo, kamor želimo kopirati ali premakniti datoteke. Vmes imamo večkrat možnost, da gremo s tipko »n« nazaj na glavni meni.

5 VIRI

W3Schools: Python Tutorial (online). [Uporabljeno 8. 4. 2026] Dostopno na https://www.w3schools.com/python/

GeeksforGeeks: Python | shutil.move() method (online). [Uporabljeno 8. 4. 2026]. Dostopno na https://www.geeksforgeeks.org/python-shutil-move-method/

Sweigart, A.: Automate the Boring Stuff with Python, Chapter 9 – Organizing Files (online). [Uporabljeno 8. 4. 2026]. Dostopno na https://automatetheboringstuff.com/1e/chapter9/

Zavod sv. Stanislava: Spletna učilnica (online). [Uporabljeno 10. 4. 2026]. Dostopno na https://ucilnica.stanislav.si/

Chen, A.: Python for Beginners: How to Write a Simple File Organizer Code (online). [Uporabljeno 9. 4. 2026]. Dostopno na https://blog.devgenius.io/python-for-beginners-how-to-write-a-simple-file-organizer-code-fd6a12eb4b3d