SIMULACIJA PARKIRALIŠTA

Milak, Juraj

SIMULACIJA PARKIRALIŠTA

Milak, Juraj

VELEUČILIŠTE U KARLOVCU

STROJARSKI ODJEL

PREDDIPLOMSKI STRUČNI STUDIJ MEHATRONIKE

JURAJ MILAK

SIMULACIJA PARKIRALIŠTA

ZAVRŠNI RAD

KARLOVAC, 2021.

VELEUČILIŠTE U KARLOVCU

STROJARSKI ODJEL

PREDDIPLOMSKI STRUČNI STUDIJ MEHATRONIKE

JURAJ MILAK

SIMULACIJA PARKIRALIŠTA

ZAVRŠNI RAD

Mentor: dr.sc. Adam Stančić, v.pred.

KARLOVAC, 2021.

Izjavljujem da je sam ovaj rad izradio uz pomoć znanja i vještina koje sam stekao tijekom svojeg studija na Veleučilištu, te uz navedene izvore i literaturu.

Zahvaljujem se svojim kolegama i profesorima koji su mi pomagali tijekom studiranja.

Posebno se zahvaljujem svojoj obitelji koja mi je uvijek bila podrška i koja me motivirala kroz cijelo moje obrazovanje.

Zahvaljujem se svojem mentoru dr.sc. Adam Stančić, v.pred. koji mi je pomogao svojim savjetima i naputcima u izradi ovog rada.

Juraj Milak

SADRŽAJ

SAŽETAK

Simulacijom parkirališta će se prikazati kako definirati moguće situacije koje se mogu pojaviti u realnom okruženju te kako organizirati, unositi, ažurirati i brisati prikupljene podatke. Osnovnim programskim funkcijama i logičkim naredbama će se moći vidjeti kako se podaci spremaju u direktorije i kako im korisnik može pristupiti. Isto tako, pojavit će se situacije koje bi mogle poremetiti tijek programa, stoga će biti demonstrirana rješenja za takve slučajeve. Također je bitno da program bude što razumljiviji i što pregledniji za eventualno daljnje proširivanje ili nadogradnju.

Ključne riječi: Logičke funkcije, Python, upravljanje podacima

SUMMARY

The car park simulation will show how to define possible situations that may occur in a real environment and how to organize, enter, update and delete collected data. Using basic functions i logical commands it will be able to see how data is being saved into directories i how can user approach them. Also, some situations will show up that could potentially disrupt the flow of a program. Therefore, solutions shall be demostrated for avoiding those cases. That is very important because the goal of a program is that it doesn't disrupt the work of a person that is using it. Also, its important to make program understandable and transparent for eventual expansion.

Keywords: Logical functions , Python, manipulating the data

Popis slika:

1. UVOD

Cilj prezentiranog rada je da se kroz primjer simulacije pokaže kako se vodi evidencija podataka simuliranog parkirališta. U simulaciji biti će prezentirani postupci unosa, ažuriranja, brisanja i prikaza podataka koji se odnose na vozila.

Problem programa koji prikupljaju informacije je često količina podataka i njihova organizacija. Stoga je potrebno pronaći model koji će korisniku i programeru olakšati manipulaciju prikupljenim podacima. Programer ima zadatak automatizirati spremanje podataka u određene liste ili direktorije i omogućiti korisnicima da s lakoćom mogu pristupiti tim podacima.

Spremanje i prikazivanje podataka će se prikazati pomoću jednostavne simulacije parkirališta napisane u programskom jeziku Python. Korištenjem simulacije će se dodavati vozila na parkiralište i tražit će se unos informacija u vozilu (u ovom primjeru model vozila i registraciju). Te informacije će se pohraniti rječnik (engl. *dictionary*). Rječnik ima funkciju organiziranja većeg broja parametara u jedan ključ (engl. *key*). Ključevi će u simulaciji predstavljati broj parkirnog mjesta i unutar svakog ključa će se nalaziti podaci o vozilu koje je parkirano. Za simulaciju u Pythonu koristit će se kombinacija osnovnih petlji (*for, while*) i logičkih operacija (*if, elif, else*).

2. PRIKUPLJANJE, SPREMANJE I ZAŠTITA PODATAKA

Podaci su u današnje vrijeme jedan od najbitnijih aspekta društva. Korisni su za osobne i poslovne potrebe. Bez obzira na djelatnost vrlo je bitno i korisno voditi evidenciju ili statistiku o prikupljenim informacijama. Primjerice, kod osobnog primjera, osoba na mobitelu može pratiti koliko vremena provodi koristeći određene aplikacije, koji postupci troši najviše baterije ili internet prometa, neprekidno se prati broj koraka koja osoba napravi u danu, kontrolira se unos kalorija itd. Prikupljeni podaci se spremaju i kasnije se mogu vizualizirati u grafu ili dijagramu ili spremati u tablice radi analize kako bi se dobio detaljniji uvid.

U poslovnom svijetu je od iznimne važnosti prikupljanje podataka i informacija vezanih za poslovanje i stanje na tržištu (evidencija radnog vremena, financijske statistike itd.). Iz tog razloga kompanije imaju svoje timove ili unajmljuju vanjske firme za vođenje evidencija, prikupljanja i zaštite podataka i informacija. Kompanija na osnovu tih podataka može modificirati određene planove i strategije u cilju povećanja profita ili poboljšanja efektivnosti rada tvrtke.

Danas postoji mnogo načina prikupljanja podataka kao što su ankete, upiti, praćenje poslovanja, analize i sinteze prethodno prikupljenih informacija itd. Primjerice, kompanija Google prikuplja podatke o tome što korisnici najčešće pretražuju i koliko vremena se zadržavaju na određenim internetskim stranicama. Na osnovu toga, Google će stavljati na vrh tražilice one rezultate koje najviše odgovaraju „profilu“ korisnika. Isto tako, Google će pomoću tih podataka odrediti koje reklame i oglase prikazati korisniku. Na primjer ako korisnik dosta provodi vremena na stranicama vezano za automobile, Google-ov servis AdSense će to zabilježiti, pohraniti i prikazivati reklame i oglase vezane za automobile. Prikupljanje, obrada i analiza podataka je vrlo bitno za tvrtke i poslovne subjekte jer tako se mogu prilagođavati publici, poboljšati u nekim aspektima što na kraju dovodi do razvoja poslovnih aktivnosti i rasta prihoda [1].

2.1 Baza podataka

Baza podataka je naziv za organizirani skup podataka. Znatno je lakše analizirati, pretraživati i upravljati podacima korištenjem određenog sustava za upravljanje bazom podataka. Na primjer, elektrana koristi bazu podataka kako bi lakše vodila evidenciju o potrošnji struje kod različitih korisnika i na osnovu toga izračunala koliki račun treba naplatiti [2].

Bazom podataka se upravlja pomoću DMBS-a (engl. *Database Management System*). DMBS zajedno sa podacima čini ono što se zove baza podataka. Baze podataka danas najčešće koriste SQL (engl. *Structured Query Language*). SQL je skriptni jezik koji služi za unos, brisanje, uređivanje i upravljanje podacima [3]. Postoji mnogo modela baza podataka. Odabrani model ovisi o tome kako organizacija ili tvrtka želi upravljati tim podacima: Neki od najčešćih modela su:

Relacijski model – model koji pomoću tablica kreira „odnose“ između raznih vrsta podataka. To je jedan od najefikasnijih modela kako pristupiti strukturnim podacima [2].

Objektni model – podaci se spremaju u obliku objekata i pomoću određenih alata za takve modele, dodjeljuju se klase, atributi i metode koje određuju kako upravljati podacima [2].

Hijerarhijski model – koristi se odnos „nadređeni-podređeni“. Baza izgleda poput stabla s čvorovima gdje se spremaju različiti tipovi podataka [2].

Najčešći poslovni modeli upravljanja podacima i bazama su:

Model za distribuciju – u ovom modelu podaci nisu na jednom mjestu i dijele se unutar organizacije. Najčešće su to podaci s lokalnih računala i uređaja [2].

Model oblaka (engl. *cloud*) – Baza se nalazi u *cloud-u* (oblaku). Cloud je u principu model gdje veći broj računala preko mreže dijele raspoložive računalne resurse kao što su jezgre CPU-a (engl. *Control Processing Unit*), radna memorija, memorija za pohranu podataka itd.. Podaci se spremaju i dohvaćaju s različitih medija za pohranu podataka putem mreže [2].

2.2 Zaštita podataka

Kako se tehnologija telekomunikacijskih i računalnih mreža razvija, tako je potrebna sve veća briga o zaštiti podataka i za osobne i poslovne potrebe. Danas se potrebno pridržavati odredbi i regulativa koje se odnose na zaštitu podataka. Na primjer ako se korisnik želi registrirati na neki servis, potrebno je unijeti lozinku. U početnim fazama informatizacije bilo je dovoljno unijeti bilo koju lozinku (s obzirom na duljinu i korištene znakove) jer regulacija i zaštita podataka nije bila toliko razvijena, ali posljedica ovakvog pristupa je njihov lakši proboj. Zato se danas često može vidjeti kako se od korisnika traži određen broj znakova, kao i određeni tip znakova (barem jedan broj, poseban znak, veliko slovo itd.). Time se otežava neovlaštenim korisnicima da probiju tu lozinku. Još jedna dodatna mjera sigurnosti je dvostruka potvrda. To znači da nakon unosa lozinke, korisniku dolazi poruka na mobilni uređaj da potvrdi prijavu ili da unese određeni kod koji je poslan u poruci.

Kod baza podataka, vrlo je bitno ograničiti ono čemu korisnici mogu pristupiti u bazi. Postoji mnogo različitih načina kako se podaci čuvaju i štite, ali najčešći je pomoću enkripcije (kao jedan od mogućih načina zaštite podataka).

Enkripcija je postupak pretvaranja informacija u kodirani oblik koji se može dekriptirati (prevodi se u jasan i razumljiv oblik) jedino s posebnim ključem za dekripciju. Kriptografski ključ se generira prije ili tijekom postupka enkripcije. Isto tako, pored enkripcije se koristi i sustav definiranja prava pristupa i akcija koje korisnik provodi nad podacima. Postoje dvije vrste enkripcije [4]:

Simetrična (privatni ključ) – za enkripciju i dekripciju se koristi jedan identičan ključ. Ovu metodu je najbolje koristi u zatvorenom sustavu. Dijeljenje tog ključa s drugima može dovesti do neovlaštenog pristupa informacijama [4].

Asimetrična (javni ključ) – koristi upareni javni i privatni ključ koji su matematički povezani. Ponašaju se kao par lokot-ključ gdje se pomoću javnog ključa (lokot) mogu kriptirati podaci, ali samo sa privatnim ključem (ključ) se ti podaci mogu dekriptirati [4].

3. RAČUNALNO PROGRAMIRANJE

Programiranje se može definirati kao postupak rješavanja problema uz pomoć posebnog jezika kojeg zovemo programski jezik. Programski jezik sadrži skup simbola i pravila koje računalo razumije i time obnaša određenu funkciju i kao rezultat dobijemo “program” [5].

Programski jezik se sastoji od:

Sintakse - način kombiniranja različitih simbola programskog jezika [5].
Semantike – programski izrazi definirani programskim jezikom [5].

3.1. Povijest programiranja

Programiranje kao takvo se prvi put spominje krajem 19. stoljeća. Točnije, 1843. godine kada je Ada Lovelace razvila prvi strojni algoritam. Njezin algoritam je služio za računanje Bernoullijevih brojeva. Napisan je na komadu papira pošto računala nisu postojala u to vrijeme. Mnogi danas smatraju da je upravo taj zapis temelj programskih jezika [6].

1943. godine se pojavljuje Assembly jezik. Najviše se koristio za EDSAC (engl. *Electronic Delay Storage Automatic Calculator*). To je pojednostavljeni jezik za računala da bi im se preciznije davali zadaci i naredbe [6].

Jedan od prvih „modernijih” programskih jezika je razvijen 1953. godine. FORTRAN (engl. *Formula Translation*) programski jezik je razvila tvrtka IBM. FORTRAN je služio za rješavanje kompleksnijih znanstvenih, matematičkih statističkih i drugih problema. Također je jedan od najstarijih jezika koji se koristi i danas [6].

Kako je vrijeme prolazilo, razvijeno je sve više programskih jezika. Neki do njih su:

BASIC - prvi pojednostavljeni programski jezik kako bi studenti mogli lakše naučiti osnove programiranja [6].

Pascal – isto se najviše koristio za učenje programiranja. Kompanija Apple ga je koristila u svojim počecima za razvoj software-a [6].

C – ovo je prvi „pravi” programski jezik razvijen 1972. godine, C je prvi program koji je bio bliži ljudskom jeziku nego strojnom kodu. Prvotna svrha mu je bila da bi se UNIX mogao koristiti na računalima različite arhitekture. Mnogi jezici koji su došli kasnije su

uzeli princip rada iz C-a. Postoji velik broj programskih jezika koji se baziraju na C-u. Oni skupa čine nešto što se zove „C obitelj programskih jezika“, a najpoznatiji su C# i C++ [6].

Python – jedan od najjednostavnijih programskih jezika. Lako se čita i treba manje linija koda nego većina ostalih jezika za obavljanje zadataka [6].

Java – Javu je razvila kompanija Sun Microsystems. Originalno je bila namjena za uređaje za kabelsku televiziju i manje uređaje. Kasnije je Java implementirana na računala korištena u mrežnom okruženju. Danas se Java koristi u velikom broju računala, uređaja i pametnim telefonima [10].

Javascript - koristi se za poboljšanje rada web stranica i preglednika. Javascript poboljšava interaktivnost web stranica prilikom određenog unosa od strane korisnika.

Ruby – objektno orijentirani jezik baziran na Perl-u i koristi se za izradu web aplikacija.

PHP – skriptni serverski jezik koji se koristi za programiranje i izradu dinamičkih web stranica.

Danas je potreba za programskim rješenjima toliko razvijena da svaka tvrtka treba usluge programera za izradu računalnih aplikacija, web stranica te ostalih programskih rješenja. Jezici su danas sve jednostavniji za korištenje i razumijevanje kako bi korisnici lakše ušli u svijet programiranja. Neki jezici se koriste više od ostalih, npr. JavaScript, PHP, C# i Python [9].

3.2. Programski jezik Python

Python je programski jezik opće namjene. Jezik je vrlo fleksibilan što omogućuje različite programske stilove (objektno orijentirano, strukturno i aspektno orijentirano programiranje). Takva prednost čini Python jednim od najpopularnijih programskih jezika. Najviše se koristi u Linux (operativni sustav) okruženju.

Python je interpreterski jezik (izvršava kod u realnom vremenu), što ga čini sporijim od ostalih jezika koji su kompajlerski (npr. C, C++). Za razlikovanje programskih blokova, Python koristi „uvlačenje“ programskog koda umjesto vitičastih zagrada i nekih ključnih riječi koje koriste ostali programski jezici.

Zajednica oko Python-a konstantno raste. Pošto je Python open-source (originalni kod u kojem je Python napisan je dostupan svima za modificiranje i distribuciju), postoji mnogo različitih modula i dodataka koji su razvijeni i implementirani u jezik od strane korisnika.

Neke od najčešćih namjena programskog jezika Python su:

Rad i upravljanje algoritmima umjetne inteligencije – programeri preferiraju Python zbog jednostavnosti i fleksibilnosti. Razvijen je velik broj paketa koji se koriste za rješenja koja upravljaju umjetnom inteligencijom [7].

Analiza podataka – pošto se u današnje vrijeme prikuplja velika količina podatke u različitim djelatnostima i aktivnostima, potrebno je odgovarajuće rješenje za manipuliranje, sortiranje i opću organizaciju tih podataka. Zbog svoje jednostavnosti i odlične podrške, Python se često koristi za tu svrhu [7].

Vizualizacija podataka – Isto kao i kod analize, vizualizacija koristi prednosti Python-a i modula za vizualizaciju [7].

Izrada aplikacija – Python isto ima široku namjenu. Od čitanja i izrade direktorija za datoteke, do aplikacija za pregledavanje video zapisa i slušanje glazbe. Nadalje, Python se koristi za izradu grafičkog sučelja i može koristiti API (*Application Programming Interface*) operativnih sustava [7].

Automatizacija i skriptiranje – Python se može koristiti za skriptiranje (najčešće u Linux okruženju). Skriptiranje je samo po sebi jezik koji se koristi za automatizirane procese (ispis raznih informacija o sustavu, provjera grešaka, upravljanje procesima i signalima, administriranje operativnog sustava itd.) [7].

4. SIMULACIJA PARKIRALIŠTA

Jedan od razloga zašto Python koristan za učenje jest, kako je već rečeno, jednostavnost i lakoća učenja programiranja i uvid kako komunikacija između računala i korisnika općenito funkcionira. Simulacija parkirališta je jedan od takvih projekta. Preko simulacije se može vidjeti funkcioniranje petlji, logičkih operacija, kako spremiti određene podatke, kako izbjeći situacije koje bi mogle poremetiti program.

Simulacija ima zadatak voditi evidenciju o parkiralištu, bilježiti koja su mjesta popunjena i koja je pozicija svih parkiranih vozila. Program zapisuje model i registraciju automobila, zatim sprema te podatke u rječnik koji će se definirati na početku same simulacije (zvat će se „spot_info“).

Rječnik (*dictionary*) - vrsta strukturirane kolekcije podataka u Pythonu. Rječnik koristi par ključ-vrijednost gdje do vrijednosti može doći preko odgovarajućeg ključa gdje se ta vrijednost nalazi. Rječnik se označava vitičastim zagradama. Točka-zarez dijeli ključ od vrijednosti. Ključ također može za vrijednost sadržavati liste ili druge rječnike. U prezentiranoj simulaciji koristiti će se kombinacija rječnika u rječniku.

{key1 ; value1, key2 ; value2, …} – primjer rječnika.
{key1 ; {key1 : value1}} – primjer rječnika u rječniku (rječnik zapisan vrijednost).

U rječniku će se za svako vozilo kreirati „manji“ rječnik (rječnik naveden kao vrijednost nadređenog rječnika) koji će sadržavati informacije o vozilu koje je parkirano. Drugim riječima, svako vozilo će imati svoj rječnik sa informacijama i svi ti „manji“ rječnici će biti u jednom većem rječniku. To nam omogućuje preglednost i lakše upravljanje podacima. Simulacija će na početku tražiti korisnika da unese veličinu parkirališta, odnosno broj parkirnih mjesta i zatim će biti prikazane opcije glavnog izbornika:

1. **Add a vehicle (Dodaj vozilo)**
2. **Remove a vehicle (Ukloni vozilo)**
3. **Check occupied parking spots (Provjeri zauzeta parkirna mjesta)**
4. **Empty the parking lot (Isprazni parkiralište)**
5. **Change parking size (Promjeni veličinu parkirališta)**
6. **Quit (Izlaz)**

Add a vehicle – pokrenut će zasebnu funkciju za dodavanje vozila na parkiralište. Funkcija će korisnika prvo tražiti uplatu za ulazak na parkiralište, a zatim će korisnik odabrati slobodno parkirno mjesto.

Remove a vehicle – pokrenut će funkciju koja će uklanjati vozila s parkirališta. Ovaj dio simulacije se isto obavlja uz pomoć zasebne funkcije koja će tražiti broj parkirnog mjesta koje korisnik želi osloboditi. Time će se obrisati podaci o vozilu koje je bilo parkirano na tom mjestu i označiti ga kao slobodno.

Check occupied parking spots - prikazuje podatke o vozilima koja su na parkiralištu.

Empty the parking lot - pomoćna opcija. Briše sve podatke koji se nalaze u prethodno definiranom rječniku i time se parkiralište označava kao prazno.

Change parking size – mogućnost izmjene ukupnog broja parkirnih mjesta na parkiralištu.

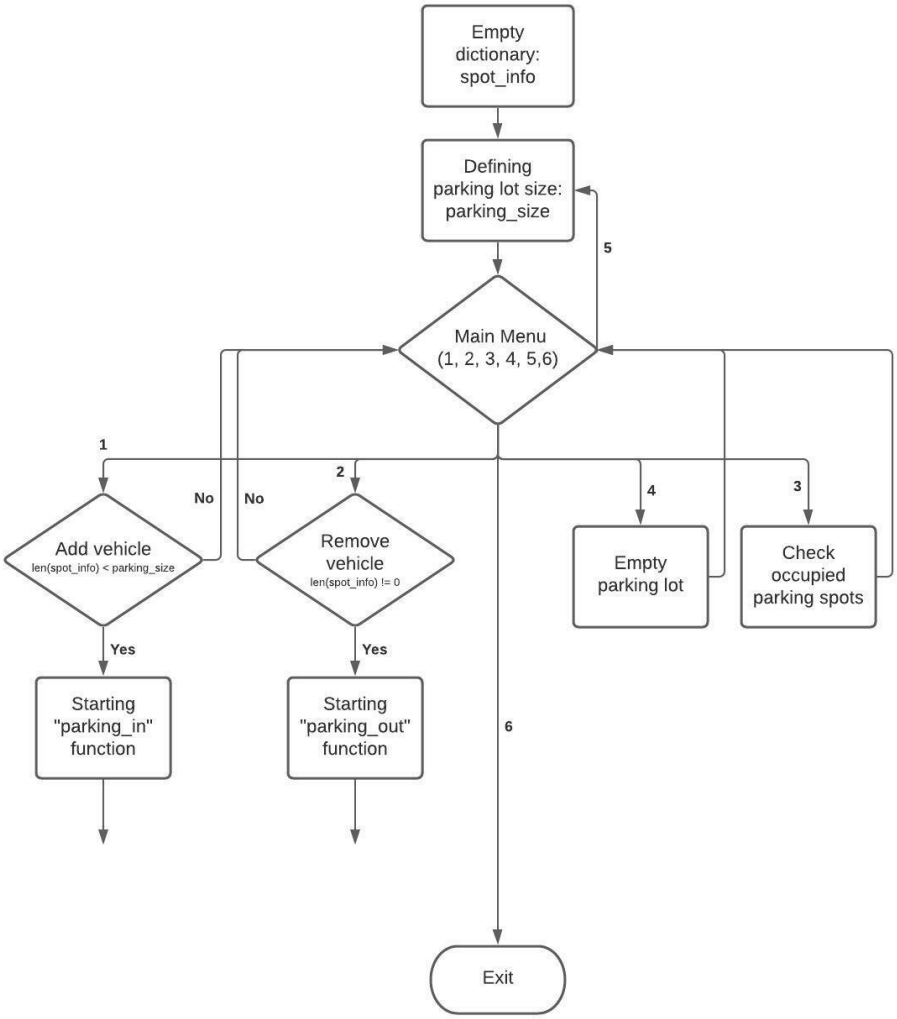
Quit - izlaz iz simulacije.

Opisana simulacija se može izvesti na više načina. Način koji je ovdje odabran izveden je uz pomoć dvije funkcije i glavnog sučelja. Program je podijeljen na tri različite .py (Python) datoteke i pomoću glavnog sučelja se dohvaća željena funkcija *(„add vehicle”***,** *„remove vehicle”*).

Taj način omogućava lakše snalaženje u programskom kodu te je lakše naći greške i situacije koje se ne bi smjele dogoditi. Simulacija sama po sebi nije komplicirana i zahtjeva osnovno znanje programiranja. U opisanom primjeru će se uglavnom koristiti petlje, logičke operacije, funkcije i rječnici.

5. GLAVNI IZBORNIK SIMULACIJE

Glavni izbornik se nalazi u zasebnoj datoteci i pomoću njega će se dohvaćati ostale funkcije koje se koriste u simulaciji. Na slici 1 je opisan dijagram toka glavnog izbornika. Prikazan je postupak kako će se odvijati sve opcije glavnog izbornika pomoću petlje. U programskom kodu je potrebno definirati nekoliko parametara (ubaciti potrebne module i varijable) prije nego što izvršavanje koda dođe do samog glavnog izbornika (slika 2). U glavnom izborniku, program će tražiti od korisnika da izabere željenu opciju. Kasnije će biti opisano kako se manipulira korisnikovim unosima i sprječavaju moguće greške.



Slika 1: Blok dijagram glavnog izbornika

```
# Definira se prazni rječnik gdje će se
# spremati podaci o vozilima na parkiralištu.
spot_info = {}

print("\n----Welcome to parking simulation----")
input("\tPress 'Enter' to begin")

# Varijabla za broj parkirnih mjesta na parkiralištu.
parking_size = int
# Petlja koja će tražiti da korisnik unese broj parkirnih mjesta.
while True:
    try:
        parking_size = int(input("\nDefine the size of parking lot: "))
        break
    except ValueError:
        # Osigurava da unos bude isključivo broj.
        print("Wrong input! Please, enter a number.")

# Varijabla koja označava unos koji će
# korisnik unijeti u glavnom izborniku.
mm = ''

# Petlja ima uvjet da će se prekinuti tek kada korisnik unese broj 6.
while mm != '6':
    print("\n--Main Menu--")
    print("\t1. Add a vehicle"
          "\n\t2. Remove a vehicle"
          "\n\t3. Check occupied parking spots"
          "\n\t4. Empty the parking lot"
          "\n\t5. Change parking size"
          "\n\t6. Quit")
    # Ispisuje omjer slobodnih mjesta i ukupnog broja mjesta na parkiralištu.
    print(f"\nParking lot: ({len(spot_info)}/{parking_size})")
    # Traži se unos od korisnika.
    mm = input("\nEnter a number (1-6): ")
```

Slika 2: Glavni izbornik

5.1 Prazni rječnik

Za početak se definira prazni rječnik (slika 3). Umjesto rječnika se mogu koristiti liste, ali pošto se vodi računa o evidenciji parkiranih vozila, lakše je to obaviti uz pomoć rječnika.

spot_info = {} – prazan rječnik za spremanje informacija o parkiranim vozilima.

- oznaka za komentar [8].

```
# Definira se prazni rječnik gdje će se
# spremati podaci o vozilima na parkiralištu.
spot_info = {}
```

Slika 3: Priprema rječnika

Svaki rječnik će označavati parkirno mjesto i u njemu će se zapisivati podaci o vozilu. Zatim se ta parkirna mjesta spremaju u jedan zajednički rječnik koji se naziva *"spot_info".* Rječnik omogućuje bolju organiziranost jer svako parkirno mjesto ima svoj „mali" rječnik sa ključevima koji označavaju informacije o vozilu koje se dohvaćaju. Nakon što je veličina parkirališta definirana, ispisat će se poruka dobrodošlice (slika 4) i korisnika će se tražiti da pritisne „Enter" za ulazak u glavni izbornik.

```
print("\n----Welcome to parking simulation----")
input("\tPress 'Enter' to begin")
```

Slika 4: Poruka dobrodošlice

print() - naredba koja ispisuje određenu poruku ili neku drugu varijablu [8].
input() - naredba koja traži korisnikov unos [8].

5.2 Broj parkirnih mjesta

Prije definiranja glavnog izbornika, potrebno je definirati broj parkirnih mjesta. Prvo se definira prazna varijabla *„parking_size".* (slika 5)Pošto se traži broj parkirnih mjesta, potrebno je osigurati da ta varijabla bude cijeli broj (eng. *integer*).

```
# Varijabla za broj parkirnih mjesta na parkiralištu.
parking_size = int
# Petlja koja će tražiti da korisnik unese broj parkirnih mjesta.
while True:
    try:
        parking_size = int(input("\nDefine the size of parking lot: "))
        parking_size = abs(parking_size)
        break
    except ValueError:
        # Osigurava da unos bude isključivo broj.
        print("Wrong input! Please, enter a number.")
```

Slika 5: Definiranje veličine parkirališta

Zatim se navedena varijabla uvodi u *„while"* petlju. Tom petljom će se osigurati da unos bude isključivo broj. Ta petlja će se koristiti još par puta u simulaciji. Ako korisnik unese vrijednost koja nije broj, pomoću kombinacije naredbi *„try"* i *„except"*, petlja će ispisati poruku da je unos pogrešan i ponovo tražiti unos vrijednosti od korisnika. Sve dok korisnik ne unese broj, izvršavanje petlja se neće prekinuti. Međutim, svaki put kada korisnik unese vrijednost kroz *„input()"* naredbu, Python taj unos detektira kao niz znakova (engl. *string*). Zato je potrebno pretvoriti uneseni niz znakova u broj.

To se postiže tako da se ispred *„input()”* naredbe napiše *„int()”*. Time će se osigurati da se unos pretvori u broj. Nadalje, ako je korisnik unio broj, izvršavanje petlje će se prekinuti uz pomoć *„break”* naredbe. Parametar „*abs*“ osigurava da uneseni broj ne bude negativan.

parking_size - varijabla koja će označavati broj parkirnih mjesta.
int - označava da je varijabla cijeli broj.
abs – apsolutna vrijednost.
while – petlja koja će izvršavati set naredbi dok god je uvjet uz petlju valjan (slika 6) [8].

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Slika 6: Primjer *„while“* petlje

break – prekida petlju [8].
int(input()) – korisnikov unos pretvara u broj [8].

Za ovu „while“ petlju se postavlja uvjet *„True”*. To je standardni uvjet koji se koristi u *„while”* petlji ako se ne traži neki specifični uvjet. Izvršavanje petlje se prekida dok se *„True”* ne transformira u *„False”* ili dok se izvršavanje ne prekine uz pomoć *„break”* naredbe. Ovom petljom se želi osigurati da veličina parkirališta bude isključivo broj. Ako korisnik unese slova ili druge znakove, naredba *„int()”* to neće znati pretvoriti u broj i program će javiti grešku. Da bi se to izbjeglo, koriste se naredbe *„try”* i *„except”.* Te naredbe služe za testiranje bloka koda i sprječavaju „rušenje“ programa.

try – testira blok koda [8].
except – rješava grešku [8].
except ValueError – greške vezane za vrijednost [8].

Naredba *„try”* će testirati da li je unos korisnika broj. Ako nije, aktivirat će se naredba *„except ValueError”* koja rješava greške s vrijednostima i sprječava „rušenje“ programa. Ispisat će se poruka za krivi unos i petlja će se vratiti na početak.

5.3. Glavna petlja izbornika

Nakon definiranja broja parkirnih mjesta, simulacija ulazi u glavnu petlju i glavni izbornik. Prije petlje se varijabla *“mm”* definira kao prazan niz. Ta varijabla označava korisnikov unos u glavnom izborniku (slika 7).

```
# Petlja ima uvjet da će se prekinuti tek kada korisnik unese broj 6.
while mm != '6':
    print("\n--Main Menu--")
    print("\t1. Add a vehicle"
          "\n\t2. Remove a vehicle"
          "\n\t3. Check occupied parking spots"
          "\n\t4. Empty the parking lot"
          "\n\t5. Change parking size"
          "\n\t6. Quit")
    # Ispisuje omjer slobodnih mjesta i ukupnog broja mjesta na parkiralištu.
    print(f"\nParking lot: ({len(spot_info)}/{parking_size})")
    # Traži se unos od korisnika.
    mm = input("\nEnter a number (1-6): ")
```

Slika 7: Početak petlje glavnog izbornika

mm – varijabla za opcije glavnog izbornika.

Uvjet za prekid izvršavanja petlje će biti *„mm != ‘6’ ”* („!=“ označava nejednakost). Dokle god korisnikov unos nije jednak 6, petlja se neće prekinuti. Prvo će se ispisati naslov glavnog izbornika *(Main Menu)* i sve prisutne opcije u simulaciji. Broj 6 označava izlaz iz aplikacije (*Quit)*. Zato kad korisnik unese vrijednost „6”, treba osigurati da se izvršavanje petlje prekine.

Prije nego će se tražiti unos od korisnika, radi preglednosti, definirat će se prikaz trenutnog broja slobodnih mjesta na parkiralištu. Kada korisnik završi s dodavanjem vozila, prekinut će se izvršavanje funkcije i korisnik će se vratiti na glavni izbornik. Također bi bilo poželjno da postoji prikaz koliko je mjesta na parkiralištu popunjeno (korisno ako parkiralište ima velik broj parkirnih mjesta).
Usporedbom broja ključeva rječnika s ukupnim brojem parkirnih mjesta dobiva se uvid o popunjenosti parkirališta (slika 8).

```
# Ispisuje omjer slobodnih mjesta i ukupnog broja mjesta na parkiralištu.
print(f"\nParking lot: ({len(spot_info)}/{parking_size})")
```

Slika 8: Omjer praznih mjesta i ukupnog broja mjesta parkirališta

print(f"message: {}") – "f" označava formatirani niz znakova. Drugim riječima, u poruci se može navesti vrijednost nekih varijabli koje je prethodno definirano. Varijablu se stavlja u vitičaste zagrade [8].

\n – novi red [8].

\t – tab (kartica) [8].

len() – označava duljinu niza, broj predmeta u listi ili rječniku [8].

5.4. Opcije glavnog izbornika

Izbornik ima 6 opcija od kojih je jedna za izlaz iz glavne petlje. Za svaku opciju se formira logička *„if"* funkcija koja odrađuje skup naredbi ovisno o opciji koju je korisnik odabrao. Pošto broj „6" označava prekid izvršavanja petlje, nije potrebno definirati logičku funkciju za taj slučaj. U izborniku će biti ukupno 6 funkcija, 5 za opcije samog izbornika i jedna za pogrešan unos (*else*) (slika 9).

```
# Logička funkcija za dodavanje vozila na parkiralište.
    if mm == '1':

#  Logička funkcija za micanje vozila s parkirališta.
    elif mm == '2':

# Uvid u informacije o vozilima.
    elif mm == '3':

# Opcija za resetiranje simulacije
    elif mm == '4':

# Opcija za promjenu broja parkirnih mjesta
    elif mm == '5':

# Aktivirat će se ako se unese pogrešan unos
    else:
        print("Wrong input!")
```

Slika 9: Opcije glavnog izbornika

if – logička operacija "ako". Izvršit će određeni skup naredbi ako je postavljeni uvjet zadovoljen [8].

else – koristi se u kombinaciji s *„if"* operacijom. Izvršava naredbe ako uvjet/uvjeti nisu zadovoljeni [8].

elif – (else if) – koristi se u kombinaciji s *„if"* funkcijom ako imamo neki drugi uvjet koji aktivira drugačiji slijed naredbi [8].

Prvi slučaj (*mm == '1'*) će aktivirati skup naredbi za pokretanje funkcije pomoću koje se dodaju vozila na parkiralište. Unutar te logičke funkcije se koriste još dvije logičke funkcije (slika 10).

```
# Logička funkcija za dodavanje vozila na parkiralište.
    if mm == '1':
        print("\nChecking..."))
        # Logička funkcija za ulaz u funkciju za dodavanje vozila
        # Aktivirat će se ako parkiralište nije puno.
        if len(spot_info) < parking_size:
            print("Everything OK!"
                  "\nStarting...")
            # Uzima se funkcija "parking_in" iz "Adding_cars.py" datoteke.
            from Adding_cars import parking_in
            # Pokretanje funkcije s postavljenim parametrima.
            parking_in(spot_info, parking_size)
        # Aktivirat će se ako nema slobodnih mjesta na parkiralištu
        # Vraća korisnika na glavni izbornik.
        else:
            print("Parking lot is full.")
            input("Press 'Enter' to return to main menu")
```

Slika 10: Opcija za dodavanje vozila

Prva logička podfunkcija će pokrenuti funkciju za dodavanje vozila. Prvo će usporediti broj vozila na parkiralištu i ukupni broj parkirnih mjesta i aktivirat će se samo ako je broj vozila na parkiralištu manji od ukupnog broja parkirnih mjesta. Time se sprječava dodavanje vozila na parkiralište ako nema slobodnih mjesta. Ako je to slučaj, umjesto *„if"* funkcije, aktivirat će se *„else"* funkcija koja će upozoriti korisnika simulacije da je parkiralište puno i vratit će korisnika na glavni izbornik.

Ako ima slobodnih mjesta potrebno je pokrenuti funkciju za dodavanje vozila. Da bi se funkcija pokrenula, prvo je treba „dohvatiti" iz datoteke u kojoj se nalazi. Za to se koriste *„from"* i *„import"* naredbe. Iz *„Adding_cars.py"* datoteke se povlači *„parking_in"* funkcija (slika 11). Zatim se funkcija pokreće s zadanim parametrima. U ovom slučaju, rječnik (*spot_info*) i broj parkirnih mjesta (*parking_size*).

```
# Uzima se funkcija "parking_in" iz "Adding_cars.py" datoteke.
from Adding_cars import parking_in
# Pokretanje funkcije s postavljenim parametrima.
parking_in(spot_info, parking_size)
```

Slika 11: Dodavanje i pokretanje funkcije

funkcija – blok naredbi koji se izvršava tek kada se funkcija pozove [8].
def function(parametar1, parametar2, ...) – definiranje funkcije [8].
function(parametar1, parametar2, ...) – pozivanje funkcije [8].
return () – vraća vrijednost iz funkcije [8].

Drugi slučaj (*mm == '2'*) će pokrenuti funkciju za uklanjanje vozila s parkirališta (slika 12). Funkcija se poziva na isti način kao i u prvom slučaju. Isto tako, postavljaju se dvije podfunkcije. Pošto se radi o uklanjanju vozila s parkirališta, kako bi se pokrenula odgovarajuća funkcija, parkiralište ne smije biti prazno i zato se to postavlja kao uvjet njenog izvršavanja. Ovdje se ne mora uspoređivati broj vozila na parkiralištu i broj parkirnih mjesta. Dovoljno je definirati da broj vozila ne smije biti jednak nuli, odnosno da rječnik ne smije biti prazan.

```
#  Logička funkcija za micanje vozila s parkirališta.
    elif mm == '2':
        print("\nChecking...")
        # Aktivirat će se ako parkiralište nije prazno.
        if len(spot_info) != 0:
            print("Everything OK!"
                  "\nStarting...")
            # Uzima se funkcija "parking_out" iz "Remove_cars.py" datoteke.
            from Remove_cars import parking_out
            parking_out(spot_info, parking_size)
        # Aktivirat će se ako je parkiralište prazno
        # Vraća korisnika na glavni izbornik.
        else:
            print("Parking lot is empty!")
            input("Press 'Enter' to return to main menu")
```

Slika 12: Opcija za uklanjanje vozila

Ako taj navedeni uvjet nije zadovoljen, pokreće se *„else"* funkcija i vraća korisnika na početak glavnog izbornika. Ako je uvjet zadovoljen, pokreće se funkcija *„parking_out"* iz *„Remove_cars.py"* datoteke sa zadanim parametrima.

Treći slučaj (*mm == '3'*) će pokrenuti funkciju za uvid u evidenciju o parkiranim vozilima. Tu se prikazuju podaci o vozilima na parkiralištu. U ovom slučaju dva podatka: model vozila i registracija (slika 13).

```
# Uvid u informacije o parkiranim vozilima.
    elif mm == '3':
        # Ispisat će poruku da je parkiralište prazno
        # ako nema informacija u rječniku.
        if len(spot_info.keys()) == 0:
            print("\nThere are no vehicles on the parking lot!")
        # Aktivirat će se ako ima podataka u rječniku.
        else:
            # Petlja koja ispisuje podatke o vozilima i njihova parkirna
              mjesta.
            for spot, spot_i in sorted(spot_info.items()):
                # Definira se svaki podatak o vozilu (marka i registracija).
                veh_n = spot_i['veh']
                veh_r = spot_i['reg']
                # Ispisuju se podaci o svakom vozilu na parkiralištu.
                print(f"\nSpot {spot}:")
                print(f"\tVehicle: {veh_n.title()}")
                print(f"\tRegistration: {veh_r.upper()}")

        # Povratak na glavni izbornik
        input("Press 'Enter' to return to main menu")
```

Slika 13: Opcija za uvid u informacije o parkiranim vozilima

Za uvid u stanje parkirališta, također se koriste podfunkcije. Za situaciju ako je rječnik prazan, koristi se *„if"* funkcija koja ima uvjet da duljina ključeva bude jednaka nuli, odnosno da je rječnik prazan (slika 14). Ako je to slučaj, ispisat će se poruka da je parkiralište prazno i tražit će se da korisnik da pritisne *„Enter"* za povratak na glavni izbornik.

```
# Ispisat će poruku da je parkiralište prazno
# ako nema informacija u rječniku.
if len(spot_info.keys()) == 0:
    print("\nThere are no vehicles on the parking lot!")
```

Slika 14: Ispis praznog rječnika

Ako rječnik nije prazan tada je potreban programski kod za ispisivanje svih podataka o vozilima. Za ispisivanje svake stavke u rječniku, najbolje je koristiti *„for"* petlju (slika 15). Pomoću te petlje se jednostavno i pregledno mogu ispisati podaci o svakom vozilu na parkiralištu.

```
else:
    # Petlja koja ispisuje podatke o vozilima i njihova parkirna mjesta.
    for spot, spot_i in sorted(spot_info.items()):
        # Definira se svaki podatak o vozilu (marka i registracija).
        veh_m = spot_i['veh']
        veh_r = spot_i['reg']
        # Ispisuju se podaci o svakom vozilu na parkiralištu.
        print(f"\nSpot {spot}:")
        print(f"\tVehicle: {veh_m.title()}")
        print(f"\tRegistration: {veh_r.upper()}")
```

Slika 15: Petlja za ispis podataka iz rječnika

for – petlja koja ispisuje ili izvršava skup naredbi za svaki predmet u varijabli. Npr. ispisuje svaki predmet u listi posebno dok ne dođe do kraja liste (slika 16) [8].

```
cars = ["audi", "nissan", "citroen"]
for x in cars:
  print(x)
```

Slika 16: Primjer *„for“* petlje

spot – varijabla za broj parkirnog mjesta
spot_i – informacije o parkiranom vozilu

Za prikaz informacija koriste se dvije varijable: *„spot”* i *„spot_i”.* Varijabla *“spot”* će označavati broj parkirnog mjesta gdje se vozilo nalazi, a varijabla *„spot_i”* će označavati informacije o tom vozilu. Kao što je navedeno, svako vozilo ima svoj zasebni rječnik, te će petlja ispisati njihov sadržaj.

Potrebno je za svaki podatak o vozilu kreirati varijablu - u ovom slučaju dvije varijable: model vozila i registracija. Naziv varijabli može biti proizvoljan, a u simulaciji se koriste varijable *„veh_n”* (model vozila) i *„veh_r”* (registracija). Pošto se radi o rječniku, potrebno je povezati te varijable s odgovarajućim ključevima gdje se nalaze podaci koje se želi ispisati. Ključevima su dani nazivi *„veh“* (model vozila) i *„reg“* (registracija). To su ključevi koji će sadržavati podatke o vozilima. Svako vozilo će imati svoj rječnik s ova dva ključa.

veh_m – model vozila.
veh_r – registracija vozila.

Nakon toga se ispisuju traženi podaci - ispisuje se broj parkirnog mjesta i podaci o vozilu na tom parkirnom mjestu. Definiran je prikaz podataka tako da se za tip vozila prvo slovo ispisuje kao veliko slovo (*.title()*), a kod prikaza registracije sva slova su velika (*.upper()*).

x.title() – pretvara prvo slovo varijable (x) u veliko slovo [8].
x.upper() – pretvara sva slova varijable (x) u velika slova [8].

Petlja će se izvršavati dok se ne ispišu podaci o svakom parkirnom mjestu na kojem se nalazi vozilo. Kada se ispiše zadnje parkirno mjesto, izvršavanje petlje će se prekinuti i od korisnika će se tražiti da pritisne *„Enter"* da se vrati na glavni izbornik.

Četvrti slučaj (*mm == '4'*) se koristi za resetiranje simulacije. Najlakši način da se to postigne jest da se obrišu svi podaci u rječniku (slika 17).

```
# Opcija za resetiranje simulacije
    elif mm == '4':
        # Briše se cijeli rječnik
        spot_info.clear()
        print("Parking lot has been restarted")
        print(f"Parking lot: ({len(spot_info)}/{parking_size})")
        input("Press 'Enter' to return to main menu")
```

Slika 17: Opcija za brisanje podataka iz rječnika

Rječnik se briše pomoću *„.clear()"* naredbe. Zatim se ispisuje potvrda da je parkiralište prazno i na kraju se traži od korisnika da pritisne *„Enter"* za povratak na glavni izbornik.

Peti slučaj (*mm == '5'*) je za promjenu ukupnog broja parkirnih mjesta. U ovom slučaju se samo kopira *„while"* petlja s početka simulacije (slika 18).

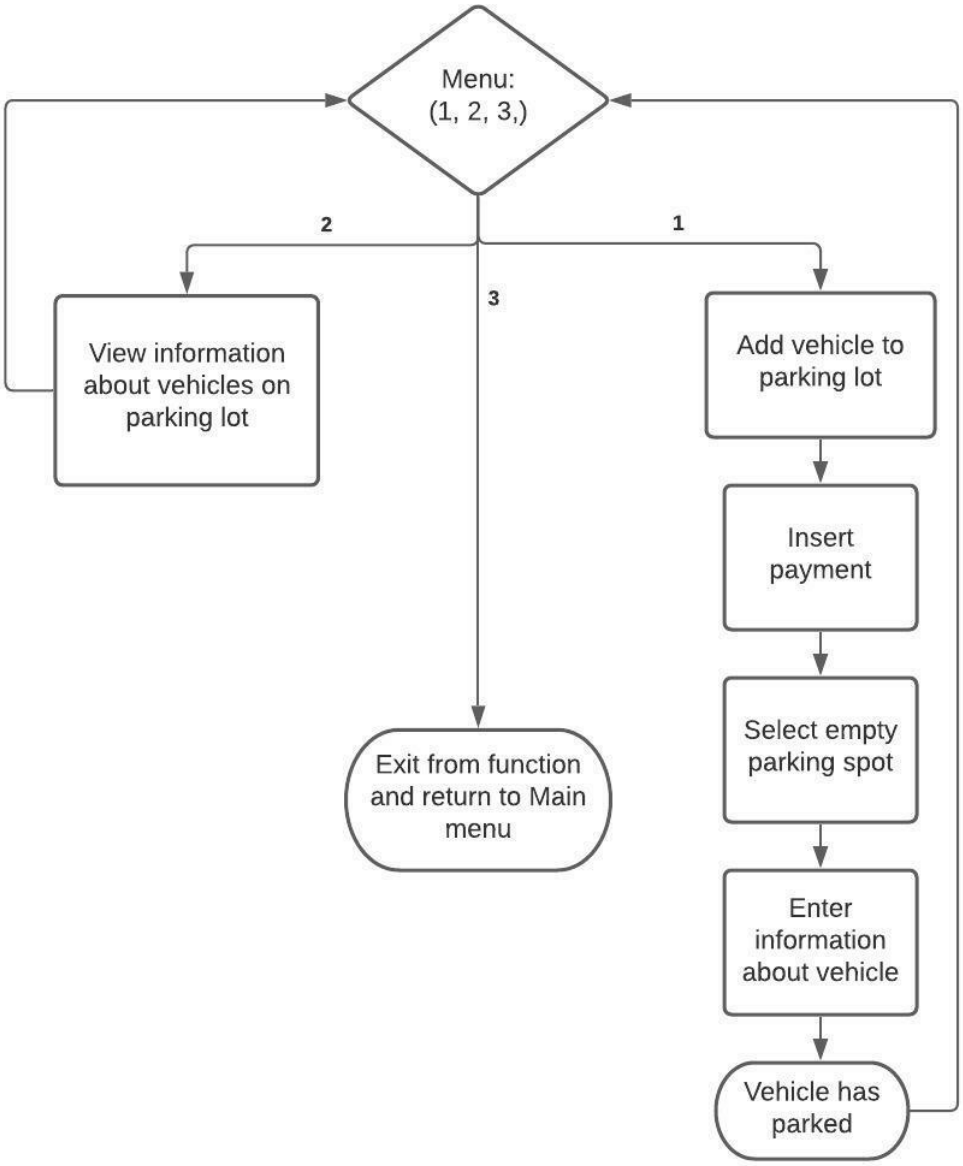
```
# Opcija za promjenu broja parkirnih mjesta
    elif mm == '5':
        # Petlja će se odvijati dok korisnik ne unese broj
        while True:
            try:
                parking_size = int(input("\nDefine the size of parking
                                         lot:"))
                parking_size = abs(parking_size)
                if parking_size < len(spot_info):
                    print("Number of occupied parking spots is bigger than
                          lot size!")
                else:
                    break
            except ValueError:
                # Osigurava da unos bude isključivo broj.
                print("Wrong input! Please, enter a number.")
```

Slika 18: Opcija za promjenu ukupnog broja parkirnih mjesta

Petlja za promjenu broja parkirnih mjesta će se ponavljati sve dok korisnik ne unese cijeli pozitivan broj. Ako je unos vrijednost koja se ne može pretvoriti u broj, petlja će javiti grešku i od korisnika će se tražiti ponovni unos.

6. DODAVANJE VOZILA NA PARKIRALIŠTE

U ovom poglavlju će biti opisan postupak dodavanja vozila na parkiralište. Biti će opisan postupak uplate za pristup parkiralištu i odabir praznog parkirnog mjesta i spremanje podataka o vozilu u rječnik. Blok dijagram (slika 19) prikazuje izbornik s tri opcije od kojih je jedna (opcija „3“) za izlaz iz funkcije za dodavanje vozila i povratak na glavni izbornik. Opcija „1“ se koristi za dodavanje vozila na parkiralište a opcija „2“ za uvid u informacije o parkiranim vozilima.



Slika 19: Blok dijagram funkcije za dodavanje vozila

6.1. Početni izbornik

Funkcija za dodavanje vozila ima 2 parametra: broj zauzetih parkirnih mjesta (*spot_info*) i ukupni broj parkirnih mjesta (*parking_size*). Funkcija ima svoj izbornik (slika 20) s tri opcije (dodavanje vozila, uvid u informacije o vozilima na parkiralištu i izlaz iz funkcije).

```
def parking_in(spot_info, parking_size):
    print("\n--Adding vehicles--")
    # Glavna petlja
    while True:
        print("\n1. Add vehicle"
              "\n2. Parking info"
              "\n3. Exit to main menu")
        print(f"Parking lot: ({len(spot_info)}/{parking_size})")
        # Traži se unos od korisnika.
        user = input("Enter a number (1-3): ")

        # Ako korisnik odabere opciju '1' i ako ima slobodnih parkirnih
          mjesta.
        if user == '1' and len(spot_info) < parking_size:

        # Ako korisnik odabere '1' i ako nema slobodnih parkirnih mjesta.
        elif user == '1' and len(spot_info) == parking_size:
             print("\nAll spots are occupied!")
             input("Press 'Enter' to return")

        # Uvid u informacije o parkiranim vozilima.
        elif user == '2':

        # Izlaz iz glavne petlje i same funkcije.
        elif user == '3':
             break

        # Za krivi unos korisnika.
        else:
             print("Wrong input!")

    # izlaz iz funkcije s dva parametra.
    return spot_info, parking_size
```

Slika 20: Izbornik funkcije za dodavanje vozila

Kao što se vidi iz priloženog, za prvu opciju (*1. Add vehicle*) postoje dvije logičke *„if”* funkcije. Razlog tome je što će biti dva različita uvjeta. Jedan za nepopunjeno parkiralište i drugi za popunjeno parkiralište. Ovaj drugi slučaj neće dopustiti korisniku da dodaje nova vozila na parkiralište ako nema slobodnih parkirnih mjesta (ako je broj ključeva u rječniku jednak ukupnom broju parkirnih mjesta). Ispisat će se poruka da je parkiralište puno i vratiti će korisnika na početni izbornik ove funkcije (slika 21).

```
elif user == '1' and len(spot_info) == parking_size:
    print("\nAll spots are occupied!")
    input("Press 'Enter' to return")
```
Slika 21: Slučaj ako je parkiralište puno

Pošto su te obje logičke funkcije vezane za prvu opciju u izborniku, potreban je zajednički uvjet, a to je da korisnikov unos bude broj '1'. Ako parkiralište nije popunjeno i korisnik odabere opciju '1', tada se pokreće prva logička *„if"* funkcija i počinje proces dodavanja vozila na parkiralište (slika 22). Simulacija će prvo tražiti uplatu od korisnika za pristup parkiralištu, a nakon toga će korisnik odabrati slobodno parkirno mjesto i na kraju će tražiti unos podataka o tom vozilu.

```
if user == '1' and len(spot_info) < parking_size:
    # Plaćanje za ulaz u parkiralište

    # Odabir parkirnog mjesta

    # Zapis podataka o parkiranom vozilu
```
Slika 22: Opcija za proces dodavanja vozila

Treća funkcija je za uvid o podatke o parkiranim vozilima i ima samo jedan uvjet (unos treba biti '2'). Kao i u glavnom izborniku, koristi se logička podfunkcija za dva slučaja: ako je parkiralište prazno (ispisat će da nema vozila na parkiralištu) i ako na parkiralištu ima vozila (ispisat će podatke o svakom vozilu na parkiralištu). Isto tako, koristit će se identična *„for"* petlja koja će ispisivati podatke o svakom pojedinačnom vozilu (slika 23).

```
# Uvid u informacije o parkiranim vozilima
elif user == '2':
    if len(spot_info.keys()) == 0:
        # Ako je rječnik prazan, tj. ako je parkiralište prazno
        print("\nThere are no vehicles on the parking lot!")
    else:
        for spot, spot_i in sorted(spot_info.items()):
            # Petlja koja ispisuje podatke o svim popunjenim mjestima
            veh_n = spot_i['veh']
            veh_r = spot_i['reg']
            print(f"\nSpot {spot}:")
            print(f"\tVehicle: {veh_n.title()}")
            print(f"\tRegistration: {veh_r.upper()}")
```
Slika 23: Uvid u podatke o parkiranim vozilima

Zadnja opcija ('3') se koristi za izlaz iz funkcije (slika 24). Koristi se jednostavna logička funkcija s jednim uvjetom i uz naredbu *„break"* koja će prekinuti glavnu petlju funkcije i potom će se pokrenuti naredba *„return"* koja će izaći iz funkcije sa spremljenim parametrima (evidencija o parkiranim vozilima i ukupan broj mjesta na parkiralištu). Time

korisnik može postojeće podatke iz funkcije za dodavanje vozila koristiti dalje u simulaciji. Parametar petlje *„else”* će ispisati poruku za pogrešan unos za svaki unos koji nije broj od 1 do 3 (slika 25).

```
# Izlaz iz funkcije
elif user == '3':
    break

# izlaz iz funkcije s dva parametra
return spot_info, parking_size
```

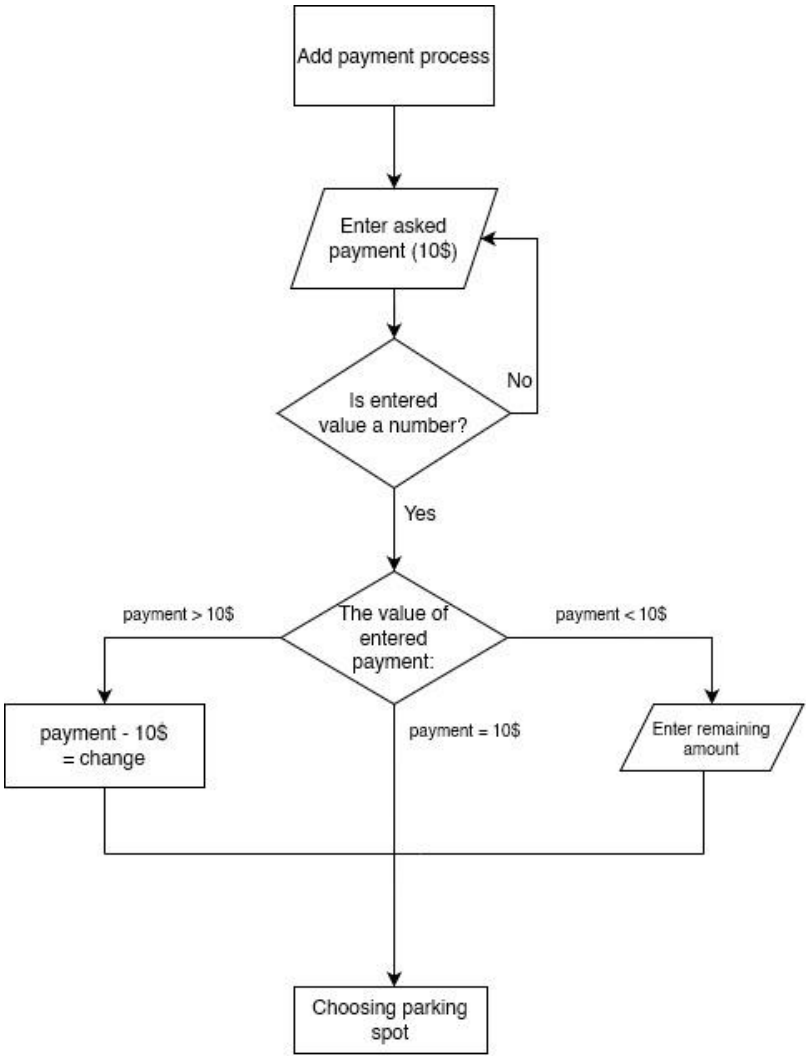
Slika 24: Izlaz iz petlje i funkcije

```
else:
    print("Wrong input!")
```

Slika 25: Slučaj ako korisnik unese krivi unos

6.2. Unos uplate za ulaz na parkiralište

Ako su uvjeti za prvu opciju (Add vehicle) zadovoljeni, pokrenut će se prva logička *„if"* operacija i počet će postupak dodavanja vozila na parkiralište. Dijagram na slici 26 prikazuje postupak uplate.



Slika 26: Dijagram toka za unos uplate

Prvo što će simulacija tražiti u tom slučaju, jest da korisnik plati za ulaz na parkiralište kako bi se rampa na ulazu mogla podići. Potrebno je napraviti varijablu koja će definirati cijenu parkiranja. Varijabla će se zvati *„payment"* i označavat će početnu (glavnu) uplatu.

```
# Petlja koja osigurava da je unesena uplata broj (float).
while True:
    try:
        print("\nTicket price 10.00$")
        # Traži unos uplate od korisnika.
        payment = float(input('Please enter your payment: '))
        payment = abs(payment)
        print("\nProcessing...")
        break
    except ValueError:
        # Izvršit će se ako unos nije broj,
        # tj. ako se unos ne može pretvoriti u integer.
        print('Wrong input!')
```

Slika 27: Unos uplate

Cijena za ovaj primjer će biti 10.00$ i korisnika će se tražiti da uplati točan iznos. Ponovo će se koristiti petlja koja će tražiti isključivo zapis u obliku broja. Međutim, ovaj put neće tražiti cijeli broj, nego decimalni broj (*float*) (slika 27).

float() – decimalni broj [8].

Razlog korištenja decimalnog broja je mogućnost da korisnik koristi dolare i cente prilikom plaćanja usluge parkiranja. Zato se obavezno mora koristiti realni broj za prikaz cijene parkiranja vozila. U petlji za unos uplate se definira parametar za apsolutnu vrijednost kako bi se izbjegla situacija unosa negativnog broja.

Nakon što korisnik unese vrijednost koja će se moći pretvoriti u decimalni broj, petlja za uplatu će prekinuti izvršavanje i postupak će se nastaviti. Međutim, prije nego simulacija krene dalje potrebno je predvidjeti par situacija. Neće se uvijek dogoditi da korisnik unese točno traženu uplatu. Događat će se da korisnik uplatiti više ili manje od traženog iznosa za parkiranje te treba u obzir uzeti navedene situacije (slika 28).

```
# Ako je korisnik uplati manje od traženog.
if payment < 10:
    # Set naredbi za rješavanje ovog slučaja.

# Ako je korisnik prvobitno točno uplatio traženi iznos
elif payment == 10:
    print("Payment successful!")

# Ako je korisnik uplatio više od traženog.
elif payment > 10:
    print("Payment successful!")
    print(f"Your change: {format(float(payment - 10), '.2f')}$")
```

Slika 28: Slučajevi koji ovise o unesenoj uplati

Potrebno je definirati tri logičke funkcije za sljedeće situacije: za manju uplatu, za točnu uplatu i za veću uplatu. Zadnja dva slučaja je lako riješiti. Ako korisnik unese točno traženi iznos (*payment == 10*), pripadajuća logička funkcija će samo ispisati da je uplata uspješna i simulacija će se nastaviti. Ako korisnik uplati više od traženog iznosa (*payment > 10*), pripadajuća logička funkcija će prvo vratiti ostatak tako da se od unesene uplate oduzme traženi iznos i potom će uplata biti uspješna i simulacija će se nastaviti odvijati. Kod prvog navedenog slučaja (*payment < 10*) je potrebno formirati skup naredbi koji će korisnika tražiti ponovnu uplatu sve dok se ne zadovolji zadani iznos.

```
if payment < 10:
    # Traži se da korisnik uplati ostatak.
    print(f"\nYou've added {format(float(payment), '.2f')}$")
    # Izračunava se koliko je ostalo za uplatiti.
    rest = 10 - payment
    # Petlja će se pokrenuti ako korisnik još uvijek nije uplatio dovoljno
      novaca.
    while rest != 0:
        while True:
            # Petlja koja osigurava da je unesena uplata broj (float).
            try:
                rest2 = float(input(f"\nYou still need to add
                                    {format(float(rest), '.2f')}$: "))
                rest2 = abs(rest2)
                print("Processing...")
                break
            except ValueError:
                print('Wrong input!')
        # Ponovo se računa razlika.
        rest = rest - rest2
        # Ako je korisnik uplatio više nego što je potrebno.
        if rest < 0:
            print(f"Your change is {format(abs(float(rest)), '.2f')}$.")
            break
```

Slika 29: Slučaj ako korisnik unese manje od tražene uplate

Za slučaj kad je uplata manja od tražene, koristi se skup naredbi koji će izračunati koliko korisnik još treba uplatiti (slika 29). Prvo će se ispisati poruka u kojoj je navedeno koliko je korisnik uplatio. Taj iznos uplate treba formatirati tako da bude decimalni broj s dvije decimale.

format – formatira vrijednost i sprema u niz [8].

format(float(payment), '.2f') – "payment" varijabla se formatira da ima dvije decimale (10$ = 10.00$) [8].

Nakon toga, formira se formula koja će izračunati koliko korisnik još treba uplatiti. Od tražene cijene se oduzme korisnikova uplata i rezultat se sprema u varijablu *„rest”.* Dalje je potrebno korisnika tražiti da uplati preostali iznos. Rješenje je *„while”* petlja koja će se ponavljati dok korisnik ne uplati ostatak (tj. dok ostatak ne bude jednak nuli).

```
while rest != 0:
    while True:
        # Petlja koja osigurava da je unesena uplata broj (float).
        try:
            rest2 = float(input(f"\nYou still need to add{format(float(rest),
            '.2f')}$: "))
            rest2 = abs(rest2)
            print("Processing...")
            break
        except ValueError:
            print('Wrong input!')
    # Ponovo se računa razlika.
    rest = rest - rest2
    # Ako je korisnik uplatio više nego što je potrebno.
    if rest < 0:
        print(f"Your change is {format(abs(float(rest)), '.2f')}$.")
        break
```

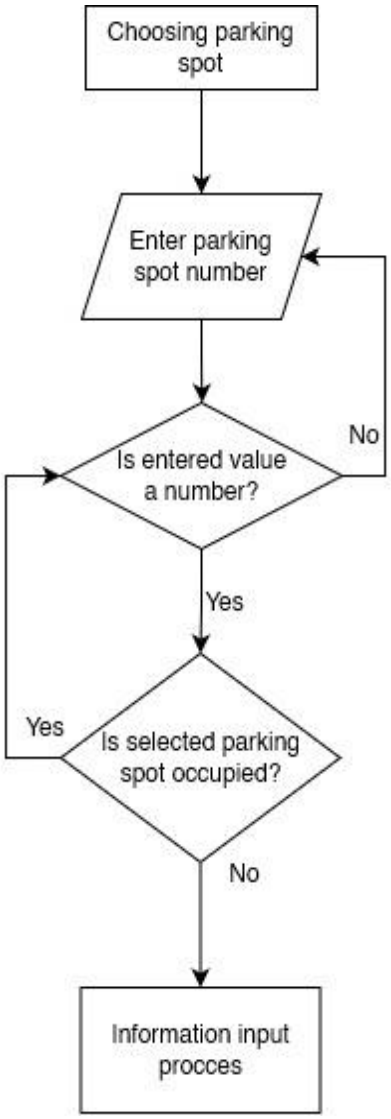
Slika 30: Petlja za unos preostale uplate

Unutar petlje za slučaj kada je uplata manja od tražene će biti još jedna „manja“ petlja (slika 30) koja će osigurati da unos bude isključivo decimalni broj. Korisnik treba uplatiti ostatak i ta vrijednost će se spremati u novu varijablu *„rest2”.* Nakon toga se izračunava ostatak. Od prvobitnog ostatka „*rest*“ se oduzima novi korisnikov unos „*rest2*” i rezultat se sprema u varijablu *„rest”.* Ako je rezultat nula, uplata je potpuna i izvršavanje petlje će se prekinuti.

Ako kojim slučajem rezultat bude negativan, to znači da je korisnik uplatio više nego što je potrebno i pokreće se „manja“ *„if”* funkcija koja će riješiti navedeni slučaj (*if rest < 0*). Korisniku će biti vraćen ostatak, ispisat će se poruka gdje je naveden iznos koji se vraća korisniku i izvršavanje petlje za uplatu će se prekinuti korištenjem *„break”* naredbe.

6.3. Odabir parkirnog mjesta

Nakon što je uplata uspješno izvršena, simulacija će tražiti od korisnika da odabere željeno parkirno mjesto. Dijagram (slika 31) prikazuje proces odabira parkirnog mjesta. Korisnika će se tražiti da prvo odabere broj parkirnog mjesta, zatim će se uz pomoć petlji i logičkih funkcija vršiti provjera tog parkirnog mjesta (da li je parkirno mjesto prazno ili ne).



Slika 31: Dijagram toka za odabir parkirnog mjesta

Ovdje će se koristiti petlja koja će spriječiti da korisnik odabere parkirno mjesto koje je zauzeto ili koje ne postoji (slika 32) . Za provjeru da li je parkirno mjesto slobodno, potrebno je provjeriti rječnik (*spot_info*) u kojem se nalaze podaci o zauzetim parkirnim mjestima.

```
# Petlja za odabir slobodnog parkirnog mjesta
while True:
    # Petlja koja će traćiti isključivo broj (int).
    while True:
        try:
            # Traži se korisnik odabere broj parkirnog mjesta.
            add_veh = int(input("\nChoose your parking spot: "))
            add_veh = abs(add_veh)
            print("Checking...")
            break
        except ValueError:
            print("Wrong input!")
    # Ako je odabrano mjesto postoji na parkiralištu i da je slobodno.
    if spot_info.get(add_veh) is None and add_veh <= parking_size:
        print(f"Spot {add_veh} is available.")
        # Odabrano mjesto se sprema i rječnik.
        spot_info[add_veh] = {}
        break
    # Ako mjesto postoji na parkiralištu i ako je zauzeto.
    elif spot_info.get(add_veh) is not None and add_veh <= parking_size:
        print(f"Spot {add_veh} is already occupied!")
    # Ako odabrano mjesto ne postoji na parkiralištu.
    else:
        print("That spot doesn't exist.")
```

Slika 32: Postupak odabira parkirnog mjesta

U ovom slučaju se koriste parametri koje funkcija za dodavanje vozila traži (*spot_info, parking_size*). Parametar *„spot_info"* će biti prazan ako prethodno nisu dodavana vozila na parkiralište. Kreira se *„while"* petlja koja će provjeravati da li je parkirno mjesto prazno i odmah ispod nje još jedna petlja koja traži od korisnika da odabere broj željenog parkirnog mjesta.

```
while True:
    try:
        # Traži se korisnik odabere broj parkirnog mjesta.
        add_veh = int(input("\nChoose your parking spot: "))
        print("Checking...")
        break
    except ValueError:
        print("Wrong input!")
```

Slika 33: Petlja za odabir parkirnog mjesta

Odabrano parkirno mjesto će se spremiti u varijablu *„add_veh"* i izvršavanje petlje za odabir parkirnog mjesta će se prekinuti i nastavit će se izvršavati petlja „iznad" nje koja će provjeriti da li je odabrano parkirno mjesto prazno. Vrijednost *„add_veh"* varijable se uspoređuje kroz *„if"* logičke funkcije (slika 34). Potrebno je osigurati da je odabrano parkirno mjesto slobodno te spriječiti odabir mjesta koje je već zauzeto. Ako se funkcija za dodavanje vozila pokreće prvi put, sva mjesta će biti slobodna, ali kasnije, kada će parkiralište biti popunjeno, potrebno je spriječiti ovu neželjenu situaciju. Potrebno je

definirati tri logičke funkcije: za slobodno parkirno mjesto, za zauzeto parkirno mjesto i za slučaj ako korisnik odabere broj parkirnog mjesta koji ne postoji.

```
# Ako je odabrano mjesto postoji na parkiralištu i da je slobodno.
if spot_info.get(add_veh) is None and add_veh <= parking_size:
    print(f"Spot {add_veh} is available.")
    # Odabrano mjesto se sprema i rječnik.
    spot_info[add_veh] = {}
    break

# Ako mjesto postoji na parkiralištu i ako je zauzeto.
elif spot_info.get(add_veh) is not None and add_veh <= parking_size:
    print(f"Spot {add_veh} is already occupied!")

# Ako odabrano mjesto ne postoji na parkiralištu.
else:
    print("That spot doesn't exist.")
```

Slika 34: Slučajevi za odabrano parkirno mjesto

Prva *„if"* funkcija će se pokrenuti ako je parkirno mjesto slobodno. Navode se dva uvjeta. Prvi uvjet će biti da podaci o parkirnom mjestu nisu u rječniku za informacije o zauzetim parkirnim mjestima (*spot_info*). Uvjet se provjerava pomoću *„.get"* naredbe koja traži ključ u rječniku (*spot_info*) koji ima vrijednost varijable *„add_veh"*. Ako *„add_veh"* nije u navedenom rječniku (*is None*), to znači da je to odabrano parkirno mjesto slobodno. Drugi uvjet je da odabrani broj (*add_veh*) nije veći od ukupnog broja parkirnih mjesta - drugim riječima, da to parkirno mjesto postoji. Ako su ta dva uvjeta zadovoljena, ispisat će se poruka da je odabrano mjesto slobodno i vrijednost varijable *„add_veh"* će se dodati u rječnik (*spot_info*) i pomoću „*break*" naredbe, izvršavanje petlje za provjeru dostupnosti parkirnog mjesta će se prekinuti.

spot_info.get(add_veh) – *„.get"* traži zadani ključ unutar zagrada [8].
spot_info[add_veh] = {} – dodavanje vrijednosti u rječnik. U uglate zagrade se upisuje ključ, a poslije znaka jednakosti se upisuje vrijednost koja se pridodaje tom ključu. Vitičaste zagrade u ovom slučaju označavaju prazan rječnik i on se sprema pod ključ *„add_veh"*. [8].

Druga logička *„if"* funkcija će se pokrenuti ako odabrani broj (*add_veh*) postoji u rječniku (*spot_info*) - drugim riječima, parkirno mjesto je zauzeto. Ta funkcija će ispisati poruku da je mjesto zauzeto i petlja će se ponoviti. Također, drugi uvjet (da je varijabla „*add_veh*" manja od ukupnog broja parkirnih mjesta „*parking_size*") je isti kao i u prvoj logičkoj funkciji.

Opcija *„else”* u petlji za provjeru dostupnosti parkirnog mjesta će se pokrenuti ako odabrano parkirno mjesto ne postoji, tj. korisnikov odabir (*add_veh*) je veći od ukupnog broja parkirnih mjesta. Ispisat će se poruka da to mjesto ne postoji i petlja za provjeru dostupnosti parkirnog mjesta će se ponoviti. Kada se odabere slobodno parkirno mjesto, izvršavanje petlje za provjeru dostupnosti parkirnog mjesta se prekida i kreće zadnji dio postupka dodavanja vozila na parkiralište.

6.4. Unos informacija o vozilu

Nakon što je odabrano željeno parkirno mjesto, rampa će se podignuti i kada vozilo dođe na parkirno mjesto, senzor će se aktivirati i označiti to mjesto kao zauzeto i ispisat će se poruka da je vozilo uspješno parkirano. Simulacija će zatim tražiti unos podataka o vozilu (slika 35).

```
# Rampa se podiže i vozilo ide na parkirno mjesto.
print("\nRamp is being lifted.")
print("Vehicle has parked successfully")
print(f"Spot {add_veh} is occupied")

# definiraju se varijable za marku vozila i registraciju i traži se unos.
veh_name = input("\nEnter vehicle name: ")
veh_reg = input("Enter vehicle registration: ")

# vrijednosti se spremaju u rječnik za odabrano parkirno mjesto.
# Ključevi imaju naziv 'veh' za marku vozila i 'reg' za registraciju.
spot_info[add_veh]['veh'] = veh_name
spot_info[add_veh]['reg'] = veh_reg

# ispisuje se broj parkirnog mjesta i informacije o parkiranom vozilu.
print(f"Spot {add_veh} info:")
print(f"\tVehicle name: {veh_name.title()}"
      f"\n\tVehicle registration: {veh_reg.upper()}")
```

Slika 35: Redoslijed naredbi za unos informacija o parkiranom vozilu

NAPOMENA: Ova simulacija ne obuhvaća simulaciju senzora i dizanja rampe.

U prezentiranom primjeru tražit će se samo model vozila i registracija (slika 36). Broj podataka se naravno može povećati ovisno o potrebama (informacije o vozaču, vrijeme dolaska na parkiralište, parking zona itd.).

```
# Rampa se podiže i vozilo ide na parkirno mjesto.
print("\nRamp is being lifted.")
print("Vehicle has parked successfully")
print(f"Spot {add_veh} is occupied")
```

```
# definiraju se varijable za marku vozila i registraciju i traži se unos.
veh_name = input("\nEnter vehicle name: ")
veh_reg = input("Enter vehicle registration: ")
```

Slika 36: Oznaka da je vozilo parkirano i upit za unos informacija o vozilu

Prvo treba definirati varijable za podatke koji će se unijeti (*veh_name* i *veh_reg*). Zatim se traži unos podataka o vozilu i ti podaci se spremaju pod ključevima koji imaju naziv *„veh“* (model vozila) i *„reg“* (registracija) i na kraju se ti ključevi spremaju u rječnik za odabrano parkirno mjesto (*add_veh*) koji se nalazi u glavnom rječniku (*spot_info*) (slika 37).

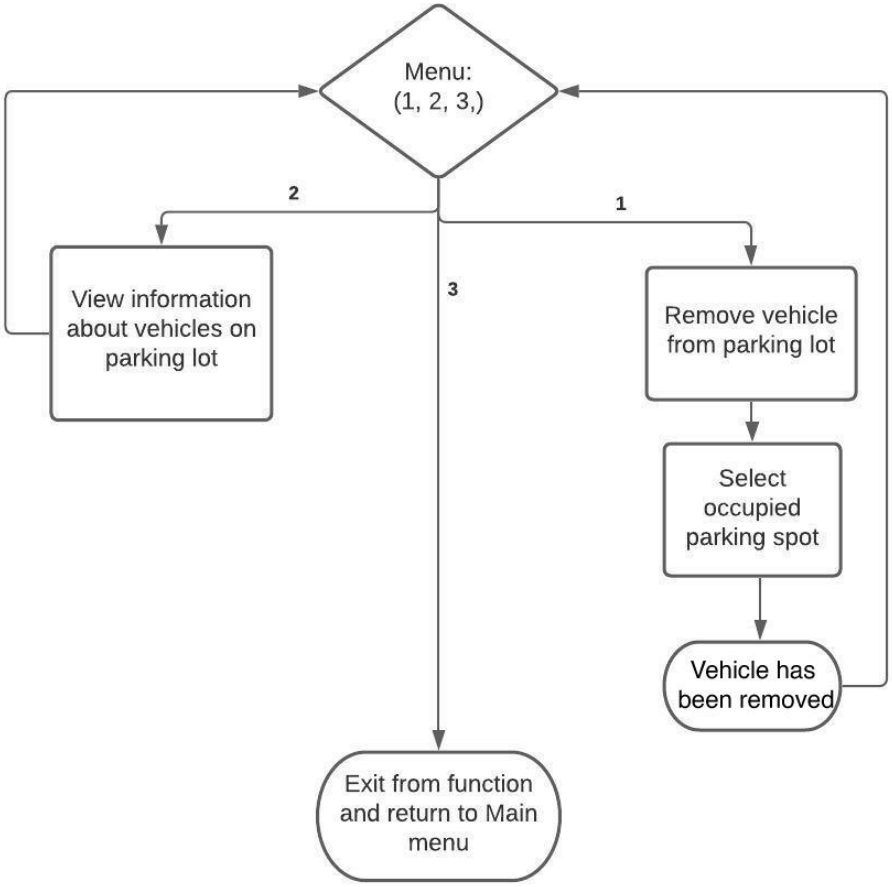
```
# vrijednosti se spremaju u rječnik za odabrano parkirno mjesto.
# Ključevi imaju naziv 'veh' za marku vozila i 'reg' za registraciju.
spot_info[add_veh]['veh'] = veh_name
spot_info[add_veh]['reg'] = veh_reg

# ispisuje se broj parkirnog mjesta i informacije o parkiranom vozilu.
print(f"Spot {add_veh} info:")
print(f"\tVehicle name: {veh_name.title()}"
      f"\n\tVehicle registration: {veh_reg.upper()}")
```

Slika 37: Spremanje podataka o vozilu u rječnik

Na kraju, simulacija ispisuje broj parkirnog mjesta i informacije o vozilu koje je tamo parkirano. Ova operacija ujedno označava kraj postupka dodavanja vozila na parkiralište i glavna petlja funkcije vraća korisnika na početni izbornik. Kasnije se može dodavati još vozila sve dok se parkiralište ne popuni ili dok korisnik ne odluči izaći iz funkcije.

7. UKLANJANJE VOZILA S PARKIRALIŠTA

U ovom poglavlju će biti prikazan postupak uklanjanja vozila s parkirališta. Funkcija za uklanjanje vozila će također imati svoj izbornik, ali će biti „manja“ od funkcije za dodavanje vozila jer je potrebno samo odabrati broj zauzetog parkirnog mjesta koje korisnik želi osloboditi. Dijagram (slika 38) je sličan kao i kod funkcije za dodavanje vozila. Jedina razlika je što ovdje nema procesa unosa uplate.



Slika 38: Blok dijagram funkcije za uklanjanje vozila

7.1. Početni izbornik

Postupak uklanjanja vozila s parkirališta je sličan kao i postupak dodavanja. Umjesto da se vozila dodaju u rječnik (*spot_info*), u ovom slučaju se brišu iz njega (slika 39).

```
def parking_out(spot_info, parking_size):
    print("\n--Removing vehicles--")
    # Glavna petlja
    while True:
        print("\n1. Remove vehicle"
              "\n2. Parking info"
              "\n3. Exit to main menu")
        print(f"Parking lot: ({len(spot_info)}/{parking_size})")
        # Traži unos od korisnika
        user = input("Enter a number (1-3): ")

        # Ako korisnik odabere opciju '1' i ako parkiralište nije prazno.
        if user == '1' and len(spot_info) != 0:

        # Ako korisnik odabere opciju '1' i ako je parkiralište prazno.
        elif user == '1' and len(spot_info) == 0:
            print("Parking lot is empty!")
            input("Press 'Enter' to return")
        # Uvid u informacije o parkiranim vozilima.
        elif user == '2':
            # Ako je rječnik prazan, tj. ako je parkiralište prazno.
            if len(spot_info.keys()) == 0:
                print("There are no vehicles on the parking lot!")
            else:
                # Petlja koja ispisuje podatke o svim popunjenim mjestima.
                for spot, spot_i in sorted(spot_info.items()):
                    veh_n = spot_i['veh']
                    veh_r = spot_i['reg']
                    print(f"\nSpot {spot}:")
                    print(f"\tVehicle: {veh_n.title()}")
                    print(f"\tRegistration: {veh_r.upper()}")
            # Povratak na izbornik.
            input("Press 'Enter' to return")
        # Izlaz iz funkcije.
        elif user == '3':
            break
        # Za krivi unos korisnika.
        else:
            print("Wrong input!")
    # izlaz iz funkcije s dva parametra.
    return spot_info, parking_size
```

Slika 39: Funkcija za uklanjanje vozila s parkirališta

Kao i u prethodnoj funkciji, potrebno je napraviti izbornik s mogućim opcijama koje korisnik može odabrati (slika 40). Prva opcija je za uklanjanje vozila s parkirališta, druga za uvid u informacije o vozilima na parkiralištu i treća je za prekid glavne petlje i izlaz iz funkcije.

```
# Glavna petlja
while True:
    print("\n1. Remove vehicle"
          "\n2. Parking info"
          "\n3. Exit to main menu")
    print(f"Parking lot: ({len(spot_info)}/{parking_size})")
    # Traži unos od korisnika
    user = input("Enter a number (1-3): ")
```

Slika 40: Glavna petlja funkcije za uklanjanje vozila s parkirališta

Za prvu opciju (uklanjanje vozila) su definirane dvije logičke funkcije. Prva koja će pokrenuti postupak uklanjanja vozila s parkirališta i druga koja će upozoriti korisnika da je parkiralište prazno i vratiti ga nazad na izbornik (slika 41). Da bi parkiralište bilo prazno, rječnik mora biti prazan, što se i postavlja kao uvjet.

```
# Ako korisnik odabere opciju '1' i ako parkiralište nije prazno.
if user == '1' and len(spot_info) != 0:
        # Postupak micanja vozila s parkirališta.

# Ako korisnik odabere opciju '1' i ako je parkiralište prazno.
elif user == '1' and len(spot_info) == 0:
        print("Parking lot is empty!")
        input("Press 'Enter' to return")
```

Slika 41: Prva opcija s dva slučaja

Druga opcija je uvid u informacije o parkiranim vozilima (slika 42). Potpuno je identična kao i u prethodnoj funkciji.

```
# Uvid u informacije o parkiranim vozilima
elif user == '2':

    # Ako je rječnik prazan, tj. ako je parkiralište prazno
    if len(spot_info.keys()) == 0:
        print("There are no vehicles on the parking lot!")

    else:
        # Petlja koja ispisuje podatke o svim popunjenim mjestima
        for spot, spot_i in sorted(spot_info.items()):
            veh_n = spot_i['veh']
            veh_r = spot_i['reg']
            print(f"\nSpot {spot}:")
            print(f"\tVehicle: {veh_n.title()}")
            print(f"\tRegistration: {veh_r.upper()}")

    # Povratak na izbornik
    input("Press 'Enter' to return")
```

Slika 42: Uvid u informacije o parkiranim vozilima

Treća opcija se koristi za izlaz iz petlje i iz same funkcije (slika 43). Funkcija sprema zadane parametre u rječnik (*spot_info*), a zadnja opcija se koristi za krivi unos korisnika.

```
# Izlaz iz funkcije
    elif user == '3':
        break

# izlaz iz funkcije s dva parametra
return spot_info, parking_size
```

Slika 43: Opcija za izlaz iz petlje i funkcije

7.2. Postupak uklanjanja vozila s parkirališta

Ako su prvobitni uvjeti zadovoljeni (izabrana je opcija „1“ na glavnom izborniku i rječnik nije prazan), pokrenut će se postupak za uklanjanje vozila s parkirališta. Postupak započinje ulaskom u *„while”* petlju koja će se izvršavati sve dok odabrano vozilo ne napusti parkiralište (slika 44).

```
if user == '1' and len(spot_info) != 0:
    # Petlja za odabir zauzetog parkirnog mjesta
    while True:
        # Petlja koja će tražiti isključivo broj (int).
        while True:
            try:
                # Traži se da korisnik odabere broj parkirnog mjesta.
                remove_veh = int(input("\nEnter a spot number
                                        you want to free:"))
                remove_veh = abs(remove_veh)
                print("Checking...")
                break
            except ValueError:
                print("\nWrong input!")
        # Ako je odabrano mjesto zauzeto i postoji na parkiralištu
        if spot_info.get(remove_veh) is not None and
        remove_veh <= parking_size:
            print(f"\n{spot_info[remove_veh]['veh'].title()} -
                    ({spot_info[remove_veh]['reg'].upper()})"
                    f" has left the parking lot!")
            del spot_info[remove_veh]
            break

        # Ako je odabrano mjesto prazno i postoji na parkiralištu
        elif spot_info.get(remove_veh) is None and
        remove_veh <= parking_size:
            print(f"Spot {remove_veh} is already empty!")

        # Ako odabrano mjesto ne postoji na parkiralištu
        else:
            print("That spot doesn't exist.")
```

Slika 44: Postupak uklanjanja vozila s parkirališta

Na početku se koristi „manja“ petlja koja će tražiti da unos korisnika bude broj. Navedeni broj se sprema u varijablu (*remove_veh*). Varijabla označava broj parkirnog mjesta.

Ako je unos valjan, „manja“ petlja se prekida i nastavlja se glavna petlja za uklanjanje vozila. Izabrani broj (*remove_veh*) se pokušava dohvatiti u rječniku (*spot_info*). Potrebno je napraviti logičke operacije koje će obraditi obje mogućnosti: ako odabrani broj postoji u rječniku i ako ne postoji u rječniku (slika 45).

```
# Ako je odabrano mjesto zauzeto i postoji na parkiralištu
if spot_info.get(remove_veh) is not None and remove_veh <= parking_size:
    print(f"\n{spot_info[remove_veh]['veh'].title()} -
({spot_info[remove_veh]['reg'].upper()})"
          f" has left the parking lot!")
    del spot_info[remove_veh]
    break

# Ako je odabrano mjesto prazno i postoji na parkiralištu
elif spot_info.get(remove_veh) is None and remove_veh <= parking_size:
    print(f"Spot {remove_veh} is already empty!")

# Ako odabrano mjesto ne postoji na parkiralištu
else:
    print("That spot doesn't exist.")
```

Slika 45: Logičke funkcije za uklanjanje vozila s parkirališta

Obje mogućnosti će imati dva uvjeta. Jedan će se odnositi na rječnik, a drugi na ukupan broj parkirnih mjesta. Prvi slučaj traži da odabrano parkirno mjesto (*remove_veh*) postoji u rječniku i da taj broj postoji na samom parkiralištu. Ako su ti uvjeti zadovoljeni, prikazat će se poruka koja ispisuje koje vozilo s kojom registracijom napušta parkiralište. Potom će se informacije o tom vozilu izbrisati iz rječnika.

del spot_info[remove_veh] – briše ključ vrijednosti varijable u uglatim zagradama.

U drugom slučaju, prikazat će se poruka da je izabrano parkirno mjesto već prazno i da nema informacija o vozilu. Ova logička operacija će se aktivirat ako odabrani broj parkirnog mjesta ne postoji u rječniku. Treći mogući slučaj je ako broj parkirnog mjesta ne postoji na samom parkiralištu. Ispisat će se poruka da parkirno mjesto ne postoji. Nakon što korisnik odabere odgovarajuće mjesto i vozilo napusti parkiralište, podaci o vozilu će se obrisati iz rječnika i izvršavanje petlje će se prekinuti pomoću *„break”* naredbe. Korisnika se potom vraća na izbornik gdje može ponovo ukloniti vozilo s parkirališta, pregledati informacije o parkiranim vozilima ili izaći iz funkcije.

8. ZAKLJUČAK

Simulacijom parkirališta se realiziralo i prezentiralo kako funkcionira primjer vođenje evidencije pomoću programa napisanog u programskom jeziku Python. Ova simulacija pokazuje kako tijekom razvoja aplikacije treba voditi računa o mogućim situacijama na parkiralištu i mogućim problemima koje one mogu prouzročiti. Na primjer, kako riješiti slučaj sa uplatom - neće se uvijek dogoditi da korisnik uplati točan traženi iznos. Zato je potrebno predvidjeti sve situacije koje bi se mogle dogoditi. Potrebno je vrijeme da se definiraju moguće situacije te da se testira programski kod (ponašanje u skladu sa situacijom, obrada neispravnih unosa korisnika, stabilan rad simulacije). Prije pisanja programa treba razraditi kad i gdje se mogu pojaviti neželjene situacije i ishodi i definirati adekvatno rješenje koje će te ishode izbjeći.

Sama simulacija je podijeljena na tri Python datoteke zbog preglednosti programskog koda, lakšeg snalaženja u samom kodu tijekom odvijanja različitih operacija te radi lakšeg pronalaženja i praćenja grešaka. Simulacija se može dodatno nadograđivati novim funkcionalnostima npr. korisnik može tražiti nove podatke o vozilima, mogu se postaviti parkirne zone s različitim cijenama, voditi računa o tome koliko je vremena neko vozilo parkirano na parkiralištu i sl.

Literatura

[1] What Are the Methods of Data Collection?, dostupno na: https://www.lotame.com/what-are-the-methods-of-data-collection/, pristupljeno: 20.08.2021.

[2] What is a Database, dostupno na: https://www.guru99.com/introduction-to-database-sql.html, pristupljeno 20.08.2021.

[3] What Is a Database?, dostupno na: https://www.oracle.com/database/what-is-database/, pristupljeno 20.08.2021.

[4] Data Encryption: An Introduction, dostupno na: https://cloudian.com/guides/data-protection/data-encryption/, pristupljeno 20.08.2021.

[5] Computer programming, dostupno na: https://en.wikipedia.org/wiki/Computer_programming, pristupljeno 20.08.2021.

[6] History of programming languages, dostupno na: https://devskiller.com/history-of-programming-languages/, pristupljeno 20.08.2021.

[7] What is Python used for? 10 practical Python uses, dostupno na: https://www.futurelearn.com/info/blog/what-is-python-used-for, pristupljeno 20.08.2021.

[8] Python Tutorial - https://www.w3schools.com/python/, pristupljeno 21.08.2021.

[9] Top Computer Languages, dostupno na: https://statisticstimes.com/tech/top-computer-languages.php, pristupljeno 15.11.2021.

[10] Why Does Java Remain So Popular?, dostupno na: https://blogs.oracle.com/oracleuniversity/post/why-does-java-remain-so-popular, pristupljeno 08.12.2021.