Vizualizacija izoploha u diskretnom N-dimenzionalnom prostoru primjenom programskog jezika Python

Prpić, Ivan

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Ivan Prpić

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Izv. prof. dr. sc. Vladimir Milić, mag. ing.
Dr. sc. Martina Odeljan, mag. ing.

Student:

Ivan Prpić

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru izv. prof. dr. sc. Vladimiru Miliću i komentorici dr. sc. Martini Odeljan na usmjeravanju i podršci prilikom izrade završnog rada te na savjetima i velikom strpljenju. Također, zahvaljujem se svim kolegama i kolegicama koji su mi pripomogli.

Ivan Prpić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Ivan Prpić** JMBAG: **0035234381**

Naslov rada na hrvatskom jeziku: **Vizualizacija izoploha u diskretnom N-dimenzionalnom prostoru primjenom programskog jezika Python**

Naslov rada na engleskom jeziku: **Visualization of isosurfaces in a discrete N-dimensional space using the Python programming language**

Opis zadatka:

Python je objektno orijentirani programski jezik koji se učinkovito koristi za bilo koje vrste računalnih programa koji ne trebaju izravan pristup hardveru računala. Između ostalih, nudi mnoge programske alate za izradu grafičkih korisničkih sučelja (engl. *Graphical User Interface* - GUI) koji se često razvijaju u edukacijske i znanstveno-istraživačke svrhe jer krajnjim korisnicima pružaju vizualnu interakciju čime im se omogućuje više razmišljanja o samom problemu nego o programskom jeziku.
U radu je potrebno:

1. Osmisliti i primjenom programskog jezika Python izraditi sučelje za vizualizaciju izoploha koje su zadane u diskretnom N-dimenzionalnom prostoru. U sučelju treba omogućiti korisniku da potrebne vrijednosti fizikalnih veličina učita iz relevantnih baza podataka koje mogu biti dostupne u obliku tekstualne datoteke u CSV formatu ili sl.
2. Pristup izradi sučelja mora biti objektno orijentiran te je u završnom radu potrebno opisati sve koncepte objektnog programiranja koji su primijenjeni prilikom izrade sučelja.
3. U izradi sučelja potrebno je:
   a. Koristiti standardnu Pythonovu biblioteku za izradu GUI-a Tkinter.
   b. Za rješavanje problema interpolacije koristiti biblioteke Numpy i Scipy. Razmotriti i usporediti različite metode interpolacije koje su ugrađene u Numpy i Scipy.
   c. Za prikaz dijagrama koristiti biblioteku Matplotlib. Na dijagramima prikazati sve relevantne veličine te omogućiti i neke grafičke značajke, kao npr. mogućnost zumiranja prikaza i sl.
4. U izrađenom korisničkom sučelju analizirati odabrani tehnički proces.
5. Napisati odgovarajuće zaključke i dati preporuke za budući rad.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2023.	**1. rok:** 22. i 23. 2. 2024. **2. rok (izvanredni):** 11. 7. 2024. **3. rok:** 19. i 20. 9. 2024.	**1. rok:** 26. 2. – 1. 3. 2024. **2. rok (izvanredni):** 15. 7. 2024. **3. rok:** 23. 9. – 27. 9. 2024.
Zadatak zadao:	Komentor:	Predsjednik Povjerenstva:
Izv. prof. dr. sc. Vladimir Milić	Dr. sc. Martina Odeljan	Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
h	$\frac{\mathrm{kJ}}{\mathrm{kg}}$	Specifična entalpija
p	bar	Tlak
P	W	Snaga
P_{t}	W	Ukupna snaga turbine
P_{t1}	W	Snaga visokotlačne trubine
P_{t2}	W	Snaga niskotlačne turbine
q_{m}	$\frac{\mathrm{kg}}{\mathrm{s}}$	Maseni protok
s	$\frac{\mathrm{kJ}}{\mathrm{kgK}}$	Specifična entropija
$\mathrm{s}^{'}$	$\frac{\mathrm{kJ}}{\mathrm{kgK}}$	Specifična entropija vrele kapljevine
$\mathrm{s}^{''}$	$\frac{\mathrm{kJ}}{\mathrm{kgK}}$	Specifična entropija suhozasićene pare
T	℃	Temperatura
v	$\frac{\mathrm{m}^3}{\mathrm{kg}}$	Specifični volumen
x	$\frac{\mathrm{kg}}{\mathrm{kg}}$	Sadržaj pare
Φ	W	Izmjenjivani toplinski tok
Φ_{dov}	W	Dovođeni toplinski tok
Φ_{kond}	W	Izmjenjivani toplinski tok pri kondenzaciji
η	-	Termički stupanj djelovanja

SAŽETAK

U ovom završnom radu prikazana je primjena Pythona kao objektno orijentiranog programskog jezika za izradu grafičkog korisničkog sučelja (engl. Graphical User Interface – GUI) koji omogućuje vizualizaciju izoploha zadanih u diskretnom N-dimenzionalnom prostoru. Budući da je izrada samog sučelja temeljena na objektno orijentiranoj paradigmi, u radu su opisani temeljni koncepti poput objekata, klase, atributa, metoda, nasljeđivanja, enkapsulacije i polimorfizma koji su primijenjeni, ali također i ugrađeni u korištene Pythonove module, pakete i biblioteke. Nadalje, detaljno je prikazana izrada korisničkog sučelja korištenjem biblioteke Tkinter s naglaskom na interakciju sučelja i „back-end" dijela. Na primjeru dva tehnička procesa iz područja termodinamike prikazana je funkcionalnost razvijenog korisničkog sučelja te su dane upute za njegovo pravilno korištenje.

Ključne riječi: objektno orijentirano programiranje, grafičko korisničko sučelje, Tkinter, SciPy, Matplotlib

SUMMARY

In this thesis, the application of Python as an object-oriented programming language for developing a graphical user interface (GUI) that enables the visualization of isopleths in a discrete N-dimensional space is presented. Since the interface creation is based on the object-oriented paradigm, the thesis describes fundamental concepts such as objects, classes, attributes, methods, inheritance, encapsulation, and polymorphism, which have been applied but are also embedded in the Python modules, packages, and libraries used. Furthermore, the development of the user interface using the Tkinter library is presented in detail, emphasizing the interaction between the interface and the back-end part. The functionality of the developed user interface is demonstrated through two technical processes in the field of thermodynamics, and instructions for its proper use are provided.

Key words: object-oriented programming, Graphical User Interface, Tkinter, SciPy, Matplotlib

1. UVOD

Svjedočimo dugogodišnjem razvoju programskih jezika i numeričkih metoda te njihovom boljem napretku i višoj uporabi pri rješavanju jednostavnih do složenih zadataka. Programiranje je prema definiciji „ priređivanje ili odabir skupa naredbi prema kojima će, ovisno o promjeni određenih ulaznih vrijednosti, djelovati neki automatski uređaj, stroj, sustav ili dr.“ [1]. Uporaba programskih jezika postala je nezamjenjivi dio svakodnevnice te se konstantnim razvojem već postojećih paketa unapređuje razvoj programskih jezika i njihova pristupačnost javnosti. Jedan od popularnih jezika je Python.

Python je opće namjenski jezik „visoke razine“. Razvijen je krajem 1980 – ih od strane Guido van Rossuma, a jednostavnošću i čitljivošću stekao je svoju popularnost. Često korišten od strane početnika u programiranju, no i dalje vrlo moćan alat za razne industrije i aplikacije. S druge strane područje termodinamike našlo se u čovjekovoj povijesti kao nezamjenjiva grana fizike koja proučava izmjenu topline i mehaničkog rada između sustava i okoline te pretvorbu i prijenos energije. Integracija ta dva područja je od velike važnosti jer omogućava preciznije i brže proračune te olakšava simulaciju i vizualizaciju procesa koji su ključni za znanstvena istraživanja i industrijske primjene. Stoga, povezivanje različitih grana inženjerstva pridonosi ne samo razvoju tehnologije, već unaprijeđenju znanosti.

1.1. Motivacija i ciljevi

Razvoj softverskih alata za rješavanje određenih problema često zahtjeva kombinaciju programerskog razmišljanja i razumijevanje problema. Programiranje omogućava automatizaciju repetitivnih zadataka i proračuna, čime se uspješno štedi vrijeme i smanjuje mogućnost pogreške. Motivacija za stvaranje aplikacije bila je osobna želja za učenjem programiranja te osnova za stvaranje složenijih alata koji mogu pomoći programerima i inženjerima da modeliraju i analiziraju kompleksne procese uz minimalan napor. Razvijanjem ove aplikacije istražuje se kako programski jezici poput Pythona mogu pripomoći rješavanju realnih problema na efikasan način, pružajući korisnicima pouzdane rezultate.

Cilj aplikacije je pokazati prednosti programiranja u rješavanju tehničkih problema i spajanja različitih grana inženjerstva. Aplikacija provodi jednostavne izračune, no demonstrira kako programski alati mogu ubrzati radne procese i olakšati razumijevanje kompleksnih problema, čime se omogućava da korisnici stave fokus na ključne dijelove analize, a aplikacija preuzima odgovornost tehničkih detalja.

Važno je naglasiti da ova aplikacija nije namijenjena za rješavanje sveobuhvatne problematike desnokretnih kružnih procesa s parom. Kao takva, nije preporučena za korištenje u akademskom ili profesionalnom okruženju, niti je službeno priznata od strane autora, mentora, Katedre za tehničku termodinamiku ili Fakulteta strojarstva i brodogradnje. Ovaj rad služi primarno kao edukativno sredstvo programiranja i osnova za daljnji razvoj složenijih alata.

1.2. Struktura rada i korištena literatura

Rad je podijeljen u nekoliko poglavlja koja postupno obrađuju temeljne koncepte objektno orijentiranog programiranja (OOP) i primjenu Pythonovih biblioteka za razvoj grafičkog korisničkog sučelja (GUI). U poglavlju 2 obrađene su osnove OOP-a, s fokusom na stvaranje klasa, atributa i metoda. Na praktičnim primjerima iz vlastitog koda implementiranim u Pythonu prikazuje se kako ti koncepti mogu biti iskorišteni za razvoj aplikacije. Razmatranja u potpoglavljima 2.1.1. i 2.1.2. najvećim su dijelom temeljena na literaturi [2], gdje su detaljno opisane ključne metododologije i principi objektno orijentiranog pristupa.

Nastavak poglavlja 2 bavi se izradom grafičkog korisničkog sučelja korištenjem Tkinter biblioteke koja je dio standardne Pythonove biblioteke. Korištena literatura [3] pruža detaljan pregled mogućnosti Tkintera, dok su prikazani primjeri i teorijski koncepti dodatno podržani praktičnim implementacijama iz literature.

Poglavlje 3 opisuje tehničke procese iz područja termodinamike, gdje se aplikacija primjenjuje na dva konkretna primjera. Razmatranja u ovom dijelu djelomično su temeljena na literaturi [4], koja pruža teorijske osnove za termodinamičke procese. Detaljno su prikazani koraci potrebni za korištenje razvijenog korisničkog sučelja. Poseban naglasak stavljen je na povezanost između "back-end" dijela aplikacije i korisničkog sučelja, što omogućuje jednostavnu interakciju korisnika s aplikacijom.

Poglavlje 4 obrađuje korištenje SciPy biblioteke za numeričke metode, s naglaskom na interpolacijske funkcije. Razmatra se primjena funkcije interp1d, koja omogućuje linearne i nelinearne interpolacije u jednom koraku, te splrep i splev, koji omogućuju složenije splajnove za obradu višedimenzionalnih podataka.

Korištena literatura obuhvaća relevantne izvore koji se odnose na objektno orijentirano programiranje i razvoj GUI aplikacija. U literaturi [4] obrađuju se temelji termodinamike, dok [2] pokriva koncept Pythonovog OOP-a, a [3] detaljno prikazuje izradu GUI aplikacija koristeći Tkinter.

2. Grafičko korisničko sučelje sa stanovišta objektno orijentiranog programiranja

2.1. Objektno orijentirano programiranje

Softverski objekti nisu neke opipljive stvari koje se mogu osjetiti ili pokupiti, već se opisuju kao modeli koji mogu odrađivati radnje i radnje mogu biti rađene nad njima. U suštini objekt je skup podataka i povezanih ponašanja. Kada se govori o objektno orijentiranom misli se na funkcionalno usmjereno modeliranje objekata. Ovo je jedna od tehnika za modeliranje složenih sustava putem opisivanja međusobno povezanih objekata, njihovih podataka i ponašanja. Objektno orijentirano programiranje (OOP) proces je pretvorbe dizajna u radni program koji ispunjava tražene naredbe.

2.1.1. Objekti i klase

Kao što je rečeno objekt je zbirka podataka s povezanim ponašanjem, a klase opisuju stvorene objekte, kako se iz nacrta stvaraju kuće, tako se iz klasa stvaraju objekti. Objekti su instance klasa koje se mogu povezati. Na primjer, ako jabuka stoji na stolu za nju se može reći da spada pod klasu jabuka. Rečeno je da je objekt zbirka podataka povezanih ponašanja, pa se postavlja pitanje koji su podaci i ponašanja povezni s objektom? Podaci predstavljaju karakteristike objekta, a klasa definira skupove karakteristika koje dijele svi objekti klase, no svaki objekt ne mora sadržavati različite vrijednosti za iste karakteristike. Uzme li se za primjer tri jabuke koje se nalaze na stolu, svaka od jabuka može imati različitu masu. Ako tri jabuke na stolu predstavljaju klasu, očigledno je da svaka instanca klase ili jabuka (objekt) sadrži podatak mase (atribut), no svaki objekt može, ali i ne mora, imati različitu vrijednost tog atributa. Stoga se atributi nazivaju članovima ili svojstvima.

Ponašanja se definiraju kao akcije koje se izvršavaju na objektu. Ponašanja koja se izvode na klasi nazivaju se metode. Metode su poput funkcija koje nešto obavljaju, no imaju dopuštenje koristiti sve informacije koje su povezane s tim objektom. Podaci u metodi su liste objekata koje trebaju biti proslijeđene u metodu koja je pozvana. Objekti koji su proslijeđeni nazivaju se argumenti. Argumenti su na raspolaganju metodi da ih koristi kako bi mogla izvršiti radnju koja joj je zadana. Dodavanjem metoda objektima stvara se sustav i proširuje međusobna interakcija objekata. Klase određuju koje podatke objekt može sadržavati i koje se metode unutar njega nalaze. Podaci za svaki objekt mogu biti različiti, iako je objekt nastao iz iste klase, te svaki objekt može drugačije reagirati na poziv iste metode zbog razlike u podacima.

Svrha modeliranja objekta je odrediti koje će javno sučelje biti za taj objekt. Sučelje se definira kao skup atributa i metoda koji drugi objekti koriste za interakciju s tim objektom. Drugim objektima ne bi trebao biti omogućen pristup unutarnjem radu objekta. Proces skrivanja podataka unutar objekta naziva se skrivanje informacija ili enkapsulacija. Podaci koji su enkapsulirani nisu automatski i skriveni podaci. Najbolji način opisivanja enkapsulacije bio bi zamisliti kutiju. Ako je kutija napravljena od drveta te se u nju pohrane određene informacije i naknadno se zakopa, informacije su skrivene i ne može im se pristupiti, no ako je kutija izrađena od prozirne plastike i ne zakopa se, informacije ili predmeti koje smo pohranili unutar nje su zaključani (inkapsulirani), no nema skrivenih informacija kada se kroz kutiju može sve vidjeti.

Pojam apstrakcije usko je vezan za enkapsulaciju i skrivanje informacija. Definicija apstrakcije je razina detalja koja najviše odgovara nekom cilju. U suštini apstrakcija je enkapsulacija podatka sa zasebnim javnim i privatnim sučeljem. Kako bi se bolje razumjela ova definicija može se zamisliti auto i vozač. Auto kao sustav sastoji se od motora, kočnica, sustava hlađenja, itd. Vozač od cijelog sustava koristi upravljački dio, kočnice i papučicu za gas. Nisu mu bitni način rada kočnica niti rad pogonskog vratila. Razmisli li se malo bolje, očigledno je automobil dio javnog sučelja, dok je vozač dio privatnog sučelja. Vozač kao dio privatnog sučelja enkaspulira podatke koji su njemu bitni (kočnica, gas, volan, itd.). U svrhu enkapsulacije mogu se koristiti *setter* i *getter* metode. Pomoću ovih metoda enkapsulira se unutarnje stanje objekta te se sigurnim načinom mijenjaju ili pristupaju atributi. *Getter* i *setter* su korisne kada je korištena logika koja je izvršena svaki put kod pristupa atributu ili njegovom mijenjanju.

Rečeno je kako su sustavi grupe udruženih objekata, gdje interakcija obuhvaća apstrakciju. No, kako razviti apstrakcije? Apstrakcije je moguće razviti pomoću dva osnovna načela: sastava i nasljedstva. Sastav je skup objekata u svrhu stvaranja novog objekta. Nasljedstvo je nalik obiteljskog stabla. Koristeći nasljedstvo neka klasa može naslijediti metode i atribute druge klase. Nasljedstvo je usko vezano za polimorfizam. Ako je želja upravljati različitim objektima na isti način jedan od najvažnijih koncepata je polimorfizam.

Polimorfizam opisuje kako se različita ponašanja i događaji odvijaju ovisno koja potklasa je korištena, bez potrebnog znanja o toj potklasi. Polimorfizam jedna je od važnijih načela objektno orijentiranog programiranja, unutar Pythona utjecaj polimorfizma može se narušiti zbog tzv. *duck typinga*. *Duck typing* omogućava korištenje svih objekata sa zadanim ponašanjem, a da pritom ne moraju biti podklase. „Ako hoda kao patka ili pliva kao patka, onda je patka“ [2]. Polimorfizam može biti jedan od razloga korištenja naslijeđivanja, no zato što se

u Pythonu svi objekti koji pružaju ispravnu radnju mogu koristiti naizmjenično[1] umanjuje se njegova nužnost. Nasljeđivanje je i dalje korisno za dijeljenje dijelova koda, no ako se dijeli iz javnog sučelja, potreban je samo *duck typing*.

2.1.2. Stvaranje objekta i klasa

Nakon što su objašnjeni osnovni koncepti, sadržaj i želje objektno orijentiranog programiranja, na primjerima se može vidjeti kako je to implementirano. Za primjer implementacije prikazat će se dio koda koji se koristi u GUI aplikaciji za crtanje grafova.
Stvaranje klase započinje jednostavnom naredbom class.

```
class Stanja_plotiranje():
   pass
```

Trenutno stvorena klasa nema funkcionalnost, budući da je kao naredba izvršavanja zadana zamjenska naredba koja preskače bilo koju radnju. Bez obzira što klasa trenutno nema svrhu moguće je prikazati odnos klase i objekta.

```
a = Stanja_plotiranje()
b = Stanja_plotiranje()
print(a)
<__main__.Stanja_plotiranje object at 0x0000026A3A88E1F0>
print(b)
<__main__.Stanja_plotiranje object at 0x0000026A3A88E700>
```

Ovim kodom se instanciraju dva objekta iz klase. Stvaranje objekta dosta je jednostavna stvar. Ispisivanjem objekata a i b dobivaju se informacije vezane za te objekte, kao što su porijeklo objekta i njihove memorijske adrese. Kako bi se klasa proširila i dobila značenje, smisao i svrhu, dodaju se atributi unutar klase.

```
class Stanja_plotiranje():
   def __init__(self, objektica, objektica1 = None):
```

[1] Misli se na *duck typing*

```
self.objektica=objektica # primjer atributa
self.objektica1 = objektica1 # primjer atributa
stanja = np.array(self.objektica[3], dtype = 'float64') # primjer lokalne varijable
```

Atributi unutar koda počinju s naredbom self te popratnim nazivom atributa. Argument self ne treba biti nužno self, to može biti bilo koja riječ, no najčešći naziv je self. On je argument metode ili atributa koji je referenca objekta na koji se metoda ili atribut poziva. Ovaj atribut potreban je kako bi se omogućio pristup varijablama i metodama povezanih s trenutnom instancom klase. Trenutno svi su atributi definirani unutar __init__ metode, koja će biti objašnjena kasnijim tekstom.

Prikazanim kodom definirana su dva atributa i jedna lokalna varijabla. Postavlja se pitanje kada je potrebno definirati nešto kao atribut ili kao lokalnu varijablu klase? Ukoliko je poznato da će varijabla biti potrebna u daljnjem kodu te da će biti korištena u nekim drugim metodama ili drugim funkcijama, ona se definira kao atribut klase, dok ako je potrebna samo u toj metodi ili funkciji, definira se kao lokalna varijabla. Lokalna varijabala može se koristiti samo u toj metodi ili funkciji i ne može se dohvatiti van klase iz objekta. Stvori li se za primjer instanca klase ili objekt iz klase Stanja_plotiranje, može se vidjeti razlika između atributa i lokalne varijable.

```
obj = Stanja_plotiranje(aha)

a = obj.objektica
print(type(a))
<class 'list'>

obj.stanja
AttributeError: 'Stanja_plotiranje' object has no attribute 'stanja'
```

Pozivanjem atributa objektica, koji je unutar klase definiran kao self.objektica, moguće je dohvaćanje atributa, no kada je ista radnja izvršena nad lokalnom varijablom stanja prikazuje se greška: „AttributeError: 'Stanja_plotiranje' object has no attribute 'stanja'“. Stoga, self je

vrlo koristan, pomoću njega se razlikuju lokalne varijable od varijabli instanci. Također, self dopušta metodama unutar klase pozivanje drugih metoda koje spadaju pod istu instancu.

Ukoliko bi se varijabla stanja definirala van metode kao u primjeru dolje, onda prethodno definirana lokalna varijabla stanja postaje klasni atribut.

```
class Stanja_plotiranje():
   stanja = 1
   def __init__(self, objektica, objektica1 = None):
      self.objektica=objektica
      self.objektica1 = objektica1

obj = Stanja_plotiranje(aha)
print(type(obj.objektica))
<class 'list'>

print(obj.stanja)
1
```

Trenutno stvorena klasa sadrži atribute i lokalnu varijablu, no nema interakcije između objekata. Interakcija objekata osigurava se metodama. Metoda je veoma slična funkcijama, stvorena je na isti način. Stvaranje metode ili funkcije započinje naredbom def, vlastitim nazivom metode ili funkcije i na kraju zagrade popraćene dvotočkom. Razlika metode i funkcije je u argumentu, taj argument je instanca klase, odnosno prethodno spomenuti self.

```
   def TS(self):
```

Metoda se može pozivati unutar klase, no onda ona djeluje kao funkcija. Sljedećim kodom prikazana je razlika metode i funkcije.

```
   def TS(self):
      print(type(self.TS))
<class 'method'>
```

```
    def plot():
        print(type(plot))
<class 'function'>
```

Funkcija plot ne uzima nikakve dodatne argumente, budući da nije metoda ne treba se upisati traženi argument self. Razlika funkcije i metode osim u inicijalnom self, vidljiva je tek nakon stvaranja objekta.

```
obj = Stanja_plotiranje(aha)
print(type(obj.TS))
<class 'method'>
print(type(obj.plot))
AttributeError: 'Stanja_plotiranje' object has no attribute 'plot'
```

U potpoglavlju 2.1.1. predstavilo se nasljeđivanje i njegova svrha. Svaka stvorena klasa direktno nasljeđuje atribute i metode iz klase object.

```
print(Stanja_plotiranje.__bases__)
(<class 'object'>,)
```

Naslijeđene metode i atributi mogu se provjeriti pozivanjem Pythonove ugrađene funkcije dir().

```
dir(Stanja_plotiranje)
['HS',
 'TS',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
```

```
'__format__',
'__ge__',
'__getattribute__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'pothladi',
'pothladi2',
'pregrijaj',
'pregrijaj2']
```

Prikazane su metode koje klasa sadrži. Ručno inicijalizirane metode su: TS, HS, pothladi, pothladi2, pregrijaj i pregrijaj2, a ostale metode je klasa Stanja_plotiranje naslijedila od object klase.

Korištenjem dvije podvlake ispred imena metode (eng. *name mangling*), Python mijenja ime metode u svrhu zaštite dotične od pristupa van klase. Podvlakama se ne postiže apsolutna

zaštita, nego djelomično sakrivanje metoda i atributa unutar klase. Njihovim se korištenjem stvaraju tzv. javne i privatne metode.

```
class Stanja_plotiranje():

   def __init__(self, objektica, objektica1 = None):
      self.objektica = objektica

   def pregrijaj(self,i):
      pregrijana_entrop = np.array(self.pregrijana[i][:,2], dtype = 'float64')

   def __privatna(self):
      print('ovo je privatna metoda')
```

Prikazanim primjerom objašnjava se razlika privatne i javne metode. Javna metoda je metoda pregrijaj, a privatna metoda sadrži dvije podvlake ispred svog imena, odnosno metoda __privatna. Metoda __privatna nije korištena nigdje unutar koda, već služi kao primjer razlike privatnih i javnih metoda.

```
obj = Stanja_plotiranje(aha)
obj.pregrijaj()

obj.__privatna ()
AttributeError: 'Stanja_plotiranje' object has no attribute '__privatna'
```

Prema prikazanom kodu opisuje se kako Python razlikuje privatnu i javnu metodu. Python ne dopušta pristup privatnoj metodi van klase iz stvorenog objekta, no uz već spomenuti *name mangling* moguće je zaobići zabranu.

```
obj._Stanja_plotiranje__privatna()
ovo je privatna metoda
```

Metoda __init__ ili konstruktor koristi se pri inicijalizaciji objekta. Ovo je prva metoda koja se poziva prilikom stvaranja instanca klase te programeru daje mogućnost postavljanja početnih vrijednosti atributa klase. Metoda __init__ ima dvije podvlake ispred svoga naziva i na prvu se doima kao privatna metoda, no ona nije stvarno privatna. Prilikom stvaranja instance klase njeno ime nije promijenjeno kao i kod metode __privatna. Iako je __init__ „privatna" metoda njezin pristup je omogućen. Dodavanjem podvlake ispred naziva metode sugerira se korisniku da ta metoda nije namijenjena za javnu upotrebu. Ne postoje potpuno privatne metode jer se svim metodama i atributima može pristupiti, iako se to ne preporučuje.

```
obj = Stanja_plotiranje(aha)
a = obj.objektica

obj.__init__(aha1)
b = obj.objektica

print(a == b)
False
```

U prošlom potpoglavlju opisao se polimorfizam i njegov značaj unutar objektnog programiranja. Rečeno je kako je nasljedstvo svojevrsni oblik polimorfizma. Na sljedećem kodu biti će prikazana njegova funkcionalnost.

```
class Trazi_tabl():
   att = 'ovo je atribut klase Trazi_tabl'

class Stanja_za_plotiranje(zd.Trazi_tabl):
   def __init__(self,sve):

      print(self. att)
      ovo je atribut klase Trazi_tabl
```

Unutar skripte definirana je klasa Trazi_tabl, unutar druge skripte spomenuta klasa korištena je kao roditeljska. Ovo nasljeđivanje nije bilo u svrhu iskorištavanja polimorfizma, već kao svojevrsno dijeljenje koda, no dobar je primjer za dublje shvaćanje njegovog značaja. Unutar klase Trazi_tabl definirao se klasni atribut att. Klasa Stanja_za_plotiranje nasljeđuje atribute i metode klase Trazi_tabl. Primjenom __init__ metode u klasi Stanja_za_plotiranje omogućen je pristup varijablama iz drugih klasa. To je definicija polimorfizma. Nadalje, primjer polimofrizma može se vidjeti na različitim metodama i funkcijama koje ugrađenih unutar Pythona. U Pythonu polimorfizam se koristi za preopterećenje operatora. Preopterećenje operatora vrši se preko metode __add__. Metoda __add__, unutar Pythona, koristi se za preopterećivanje '+' operatora. Polimorfizam omogućava '+' operatoru da se različito „ponaša“ ovisno o vrsti objekta kojeg zbraja. Npr., operator '+' može biti korišten pri zbrajanju brojeva, spajanju riječi ili lista.

```
plt.text(self.entropija_puna[h]-0.1,self.temperatura_puna[h]-10,
f'{h+tocke+1}'+'*',fontsize=15, fontweight='bold')
```

U danom kodu prikazana je naredba za postavljanje teksta na grafu. Možda se na prvu ne vidi, ali unutar nje sadržan je polimorfizam. Budući da su varijable h i tocke brojevi (eng. *integer*), Python ih zbraja i sprema u obliku riječi (eng. *string*), te novostvoreni *string* spaja se sa *stringom* znaka '*'. U primjeru se koristi isti operator '+' za zbrajanje brojeva i spajanje riječi, što je polimorfizam na djelu.

2.2. Grafičko korisničko sučelje

GUI (eng. *graphical user interface*) je grafičko korisničko sučelje s mogućnošću komunikacije čovjeka i računala putem ikona, menija i gumbova. GUI-a prenosi informacije bitne korisniku i radnje koje može učiniti. GUI se razvio iz tekstualnih korisničkih sučelja CUI. Sučelja CUI bila su zbunjujuća i neučinkovita, radnje su se morale pamtiti i ispravno upisivati svaki put. Godine 1981. Xerox je izradio prvo grafičko sučelje (GUI). Iako je bilo različito od današnjih sučelja označilo je prelazak s tekstualnih sučelja na grafička sučelja. Danas je teško zamisliti računala bez GUI-a. Unutar GUI-a vizualni elementi označavaju radnje koje korisnici mogu činiti, objekte kojima mogu upravljati i druge informacije bitne korisniku. Korisnicima je moguća interakcija s grafičkim sučeljem na nekoliko načina. Pritiskom miša se pomiče ili

klikne te se zatraži interakcija, GUI odgovara na interakciju s nekom od potvrda, npr. promjena boje ili veličine, a zatim izvršava zadanu radnju.

U ovom radu prikazat će se izrada GUI aplikacije pomoću Tkinter biblioteke. Biblioteka *widgeta* Tk potječe iz jezika naredbi alata Tcl. Tcl i Tk razvijeni su od strane John Oustermana, u svrhu lakše izrade računalnih alata korištenih na sveučilištu. Radi svoje jednostavnosti i brzine ubrzo je postao popularan. Njegova jednostavnost naspram drugih alata i dan danas je prednost. Tkinter je sučelje za Tk GUI biblioteku i dio je Python biblioteke od 1994. godine. Pri izradi GUI-a postoji nekoliko različitih alata na odabir, no Tkinter se često zaboravlja. Tkinter nije nova tehnologija, no prikladan je za veliki broj aplikacija te ima prednosti koje se ne trebaju zanemariti. Neke od prednosti Tkintera su:

- Tkinter se nalazi u standardnoj biblioteci. Gdje god je dostupan Python tamo je dostupan i Tkinter.
- Nije potrebno instalirati dodatne pakete niti virtualna okruženja.
- Tkinter je stabilan; napisani program će raditi nepromijenjen godinama ili desetljećima.
- Oslanja se na ostale Python biblioteke, te je lagan za primjenu GUI-a na već napravljeni Python kod.
- Jednostavan je, Tkinter je objektno orijentiran GUI dizajn. Nije potrebno učiti novi jezik niti klase *widgeta*.

Postoje i određene mane Tkintera. Jedan od nedostataka bio bi njegov arhaični izgled i nedostatak složenijih *widgeta*.

2.2.1. Stvaranje grafičko korisničkog sučelja

Prikazano je u potpoglavlju 2.1.1. kako je Python objektno orijentirani program. U potpoglavlju 2.1.2. za izrađivanje grafova korištene su klase, atributi i metode. Za izradu grafičkog sučelja, ništa se pretežno ne mijenja, i dalje se koriste klase, atributi i metode, no postoje i dodatne značajke pri njegovoj izradi. Te značajke nazivaju se *widgeti*. Pri izradi GUI aplikacije *widgeti* su poput osnovnih gradivnih blokova, oni predstavljaju svaki element s kojim korisnik može vršiti interakciju. *Widgeti* su sredstvo komunikacije između korisnika i programa. Mnogo je različitih vrsta, a neke od osnovnih su: gumbi, tekstualna polja, okviri, izbornici, itd. Za izradu korisničkog sučelja potrebno je učitati osnovne biblioteke.

```
import tkinter as tk
from tkinter import ttk
```

Naredbom import tkinter as tk učitana je Tkinter biblioteka i pohranjena u obliku modula unutar varijable tk. Naredbom from tkinter import ttk učitava se iz biblioteke tkinter tematska Tk *widget* biblioteka. Modul ttk nije nužan pri izradi korisničkog sučelja, ali je koristan, njegovom implementacijom dodaje se mnogo korisnih *widgeta* i mijenja ostarjeli izgled postojećih unutar modula tk. Nakon što se učitaju potrebne biblioteke i moduli, može se krenuti u izradu korisničkog sučelja.

```
class Procesi:
    def __init__(self, root):
        self.root = root
        self.root.title("Procesi")
        self.root.geometry("800x600")
```

Definira se klasa Procesi unutar koje će se dodavati metode, atributi, funkcije i *widgeti*. Prikazano je u prošlom poglavlju da svaka stvorena klasa nasljeđuje opću klasu object, tako i klasa Procesi.

Definiranjem konstruktora potreban je jedan ulazni podatak, uz postojeći self. Podatak koji se upisuje pri izradi instance klase pohranjuje se u varijablu root. Od lokalne varijable root naredbom self.root = root stvara se atribut klase. Nad atributom klase izvršene su određene značajke kao što su naziv i geometrija. Postavlja se pitanje, što je atribut sefl.root da mu se može dodijeliti geometrija i naziv? root ili self.root je glavni objekt aplikacije, on predstavlja primarni prozor najviše razine i izvršnu nit aplikacije.

```
if __name__ == "__main__":
    root = tk.Tk()
    app = Procesi(root)
    root.mainloop()
```

Kodom je prikazano kako je objekt root instanca klase tk.Tk(). Pri kreiranju objekta app iz klase Procesi dodaje se objekt root kao ulazni podatak unutar konstruktora te se prosljeđuje atributu self.root unutar klase Procesi.

Uz stvaranje objekata prikazane su dvije linije koda koje su vrlo bitne. Prvom linijom koda if __name__ == "__main__" omogućeno je izvršavanje koda jedino ako je program glavni program koji se izvršava. Njome se omogućava samostalno izvođenje programa i osigurava od automatskog izvođenja prilikom uvoza dotične kao modula. Posljednja linija koda root.mainloop() je zadnja linija koda koja započinje petlju događaja. Mainloop je ugrađena metoda unutar Tk klase, ona pokreće petlju koja obrađuje podatke (klik miša, pritisak gumba itd.) i biti će izvršavana dok se glavni prozor ne zatvori. Ova linija koda najčešće je zadnja pri izradi korisničkog sučelja pomoću Tkintera i bez nje se ne otvara glavni prozor.

Slika 2.1. Prikaz glavnog objekta zadane geometrije

Izvrši li se prikazani kod otvara se prozor, prema slici 2.1., zadanog naziva i upisanih dimenzija, no nema nikakve interakcije korisnika s programom. Interakciju korisnika i programa, kao što je već rečeno, osiguravaju *widgeti*. Prvi i najosnovniji *widget* bio bi gumb.

```
self.button_frame = tk.Frame(root)
self.button_frame.pack(anchor='nw')

self.buttons = [ttk.Button(self.button_frame, text=f"ProcesTr{i+1}", command=lambda i=i:
self.display_image(i)) for i in range(2)]
for button in self.buttons:
    button.pack(side=tk.LEFT, padx=10, pady=10)
```

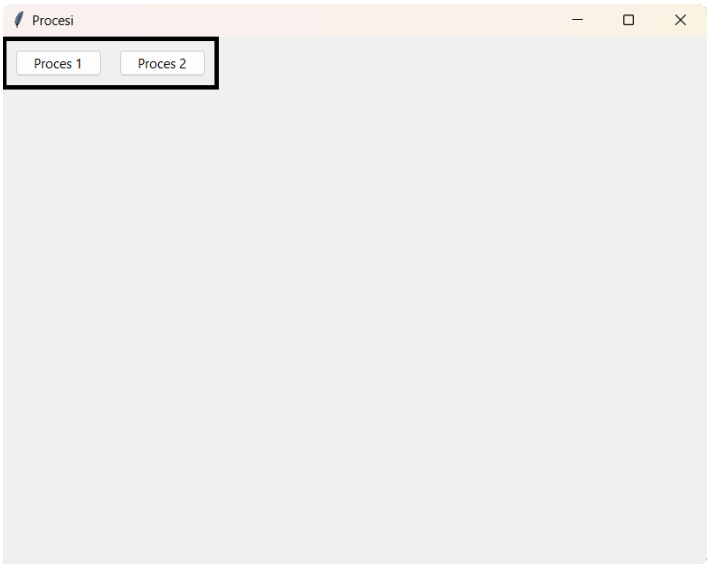
Unutar __init__ metode dodani su objekti klase Button (gumb). Naredbom ttk.Button stvara se spomenuti objekt. Prvi ulazni podatak unutar klase Button, ali i drugih *widgeta*, je *master*. *Master* predstavlja nadređeni prozor, odnosno prozor u kojem će se prikazati objekt. Taj prozor može biti i već stvoreni self.root, no zbog jednostavnosti i preglednosti prije stvaranja objekta specificira se okvir unutar prozora self.root. Stvaranje prozora prikazano je pomoću prve linije koda. Drugom linijom koda postavlja se stvoreni okvir na određenu lokaciju unutar prozora.

Postoje tri postavljača (metode) geometrije unutar Tkintera. Ugrađena Tkinterova metoda pack() najjednostavnija je od tri metode te postavlja prozor na okvirno mjesto koje se unosi unutar zagrada metode. Tablica 2.1. prikazuje ostale metode koje su korištene za upravljanje rasporeda.

Tablica 2.1 Metode geometrije

Metoda	Opis
grid()	Organizira *widgete* u strukturi poput tablice, jedna ćelija je jedan *widget*. Ne mora se misliti o veličini *widgeta*, jer se ona automatski prilagodi.
pack()	Metodom pack() Tkinter sam odredi gdje bi *widget* trebao stajati. Dovoljno je dati okvirnu lokaciju obzirom na ostale *widgete*.
place()	Omogućava upravljanje rasporeda s točno određenom pozicijom i veličinom jednog *widgeta* u odnosu na druge. Nudi više kontrole rasporeda nego pack() ili grid().

Nakon što je specificiran prozor u kojem će se objekt prikazati, drugi ulazni element je naziv gumba, koji se bira proizvoljno, te će on biti prikazan u aplikaciji. Trećim ulaznim elementom zadaje se funkcionalnost gumba, odnosno radnju koju će on izvršiti. Željena radnja najčešće se zadaje pomoću lambda funkcije, koja je pogodna za jednostavne funkcije bez naziva. U posljednjoj liniji koda je metoda pack() koja postavlja objekte na određenu lokaciju unutar stvorenog okvira. Izvršavanjem koda, stvaraju se dva gumba na određenim lokacijama i sa zadanim funkcionalnostima. Slika 2.2. prikazuje objekte, raspored i stvoreni okvir.



Slika 2.2. Prikaz glavnog objekta i dva objekta klase Button unutar zadanog okvira

Drugi vrlo koristan i često korišten *widget* je okvir za unos podataka (eng. *entry*).

```
self.maseni_protok_uk_entry = ttk.Entry(self.top_middle_frame, validate="key", validatecommand=(validate_numeric, '%P'))
self.maseni_protok_uk_entry.pack(side=tk.LEFT, padx=5)
```

Stvaranje polja za unos započinje naredbom ttk.Entry. Također, kao i kod gumba definira se prozor (okvir) u koji će se postaviti stvoreno polje za unos. Za dohvaćanje podatka koji je upisan unutar polja koristi se ugrađena metoda get().

```
self.maseni_protok_entry.get()
```

Pomoću metode get() dohvaća se tekst ili broj koji je upisan unutar self.maseni_protok_entry polja. Slika 2.3. prikazuje izgled *entry* objekta.

Slika 2.3. Prikaz objekta klase Entry

Postoje mnoge vrste *widgeta* unutar Tkinter biblioteke u Pythonu. Predstavljena su tri osnovna koja su najviše korištena za izradu GUI-a, kako se nebi previše duljilo ostale vrste *widgeta* prikazane su tablicom 2.2.

Tablica 2.2. Vrste widgeta i njihov opis

Widget	Opis
Button	Gumb se može pritisnuti i korisnik može provesti interakciju s programom.
Canvas	Prikazivanje kompleksnih slika i rasporeda.
CheckButton	Prikazuje broj mogućnosti između kojih korisnik može birati. Omogućava odabir više opcija u isto vrijeme.
Entry	Prikaz jednog polja za ulaz podataka.
Frame	Služi kao okvir za skladištenje i primanje ostalih *widgeta.*
Label	Za prikaz teksta ili slike.
Listbox	Korisnik može birati iz liste opcija.
Menu	Za davanje naredbi korisniku. Ovaj *widget* stvara vrste izbornika potrebnih u aplikaciji.
Menubutton	Koristi se za prikaz stavki iz izbornika korisniku.
Message	Prikazuje poruku korisniku unutar okvira. Prikazuje se više linijski tekst koji se ne može uređivati.
Radiobutton	Koristi se za prikaz više opcija korisniku. Može se izabrati samo jedna opcija.
Scale	Klizač koji omogućava odabir vrijednosti iz zadanog raspona.
Scrollbar	Za pomicanje prozora gore ili dolje.
Text	Prikazuje više linijski tekst koji korisnik može uređivati ili upisivati.
Toplevel	Koristi se za prikazivanje novog prozora.
SpinBox	Prikazuje ograničeni raspon brojeva, te se iz njega odabiru samo brojevi unutar raspona.
PanedWindow	Vrsta *widgeta* za okvire. Koristi se za upravljanje panelima, paneli se mogu rasporediti horizontalno ili vertikalno.
LabelFrame	*Widget* okvira koji se koristi za upravljanje složenih *widgeta*.
MessageBox	*Widget* za poruke. Koristi se prikaz poruke u desktop aplikacijama.

3. Dijagram toka i pravilno korištenje aplikacije

U ovom poglavlju predstaviti će se rad aplikacije i njezino pravilno korištenje. Kako bi se u potpunosti shvatio rad aplikacije izložit će se dijagram toka metode Kondenzacija i blokovski prikaz uvoza skripti.

3.1. *Back-end* i dijagram toka

Aplikacija koristi šest različitih skripti za pravilan rad. Slika 3.1. prikazuje uvoz naredno opisanih skripta za potpuni rad aplikacije. Prva skripta nazvana CSV, ona služi za učitavanje podataka izoploha pohranjenih oblikom CSV koda. Unutar te skripte podaci su raspoređeni u obliku rječnika, a svaki rječnik predstavlja po dva stanja koje H_2O može poprimiti. Unutar prvog rječnika nalaze se vrela kapljevina i suhozasićena para raspoređene po temperaturama, unutar drugog rječnika nalazi se isti sadržaj no raspoređen prema tlakovima te unutar trećeg rječnika nalaze se podaci o pregrijanoj pari i pothlađenoj kapljevini raspoređene po tlakovima i temperaturama. Ova skripta uvezena je u skoro sve ostale skripte.

Druga skripta nazvana je Računanje. Unutar te skripte stvorena je klasa Trazi_tabl koja služi za računanje, odnosno dohvaćanje podataka koji se nalaze u rječnicima. Klasa putem konstruktora prima po dva ulazna podatka pomoću kojih pretražuje rječnike i dohvaća traženo stanje vode. Klasa sadrži dvije metode koje se koriste za računanje. Prva metoda nazvana je interpol_tabl1, putem ove metode dohvaćaju se potrebni podaci iz prva dva rječnika. Druga metoda nazvana je interpol_tabl3, ovo metodom dobavlja se stanje pothlađene kapljevine ili pregrijane pare.

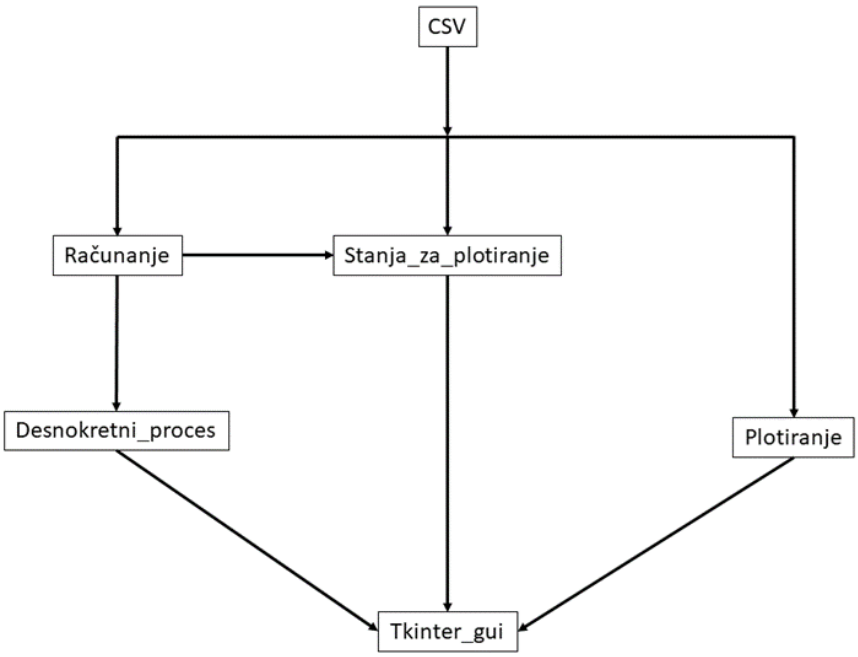
Treća skripta naziva se Desnokretni_proces. Skripta sadrži jednu klasu i mnoštva metoda kojima su opisani uređaji korišteni za provođenje desnokretnih kružnih procesa, poput: kotla, pumpe, turbine, međupregrijača, kondenzatora. Svaki od ovih uređaja predstavlja po jednu metodu unutar klase Desnokretni_proces. Slikama 3.2., 3.3., 3.4. i 3.5. prikazuje se dijagram toka metode Kondenzacija. Svaka metoda prima po tri argumenta uz poznati nam self. Prvi argument nazvan je broj i on predstavlja smještaj uređaja unutar procesa, također pomoću njega određeno je mjesto pohrane izračunatog stanja u obliku polja (eng. *array*). Drugi argument nazvan je snaga ili toplinski_tok, ovisno o metodi, ovim argumentom predstavlja se dovođeni ili odvođeni toplinski tok ili snaga uređaja. Zadnji argument je maseni_protok_pom, u slučaju da se toplinski tokovi i snage žele izračunati u mjernoj jedinici kW, ovaj argument je

obavezan. Nakon računanja svaka metoda vraća izračunato stanje, toplinski tok ili snagu i maseni protok koji je zadan ili izračunat.

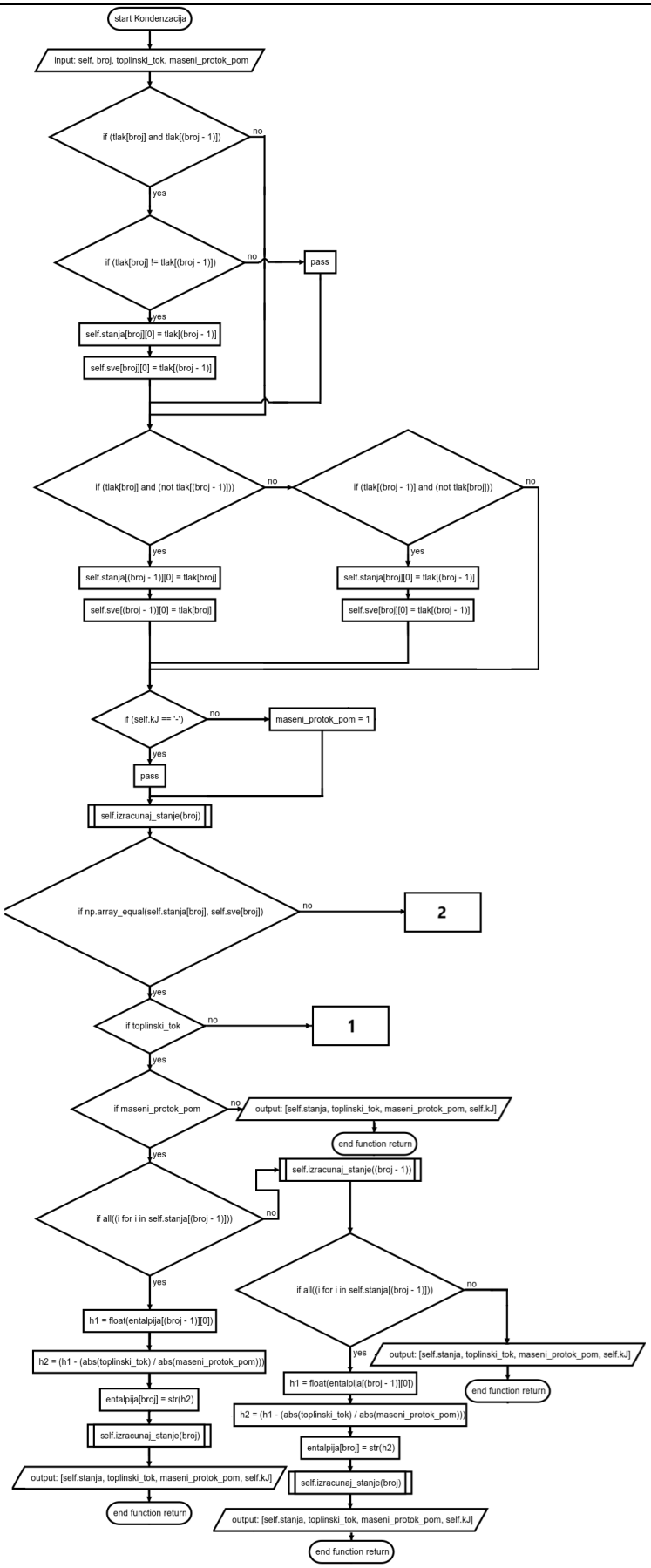
Četvrta skripta nazvana je Stanja_za_plotiranje. Skripta Stanja_za_plotiranje iz izračunatih podataka, odnosno izračunatih stanja, dohvaća sve podatke vezane za to stanje kako bi se stanje prikazalo unutar grafova. Pomoću ove skripte dohvaćaju se svi podaci pregrijane pare, suhozasićene pare, vrele kapljevine i pothlađene kapljevine za određeni tlak, kako bi se na dotičnoj izoplohi prikazalo stanje i promjena stanja. Podaci koje dohvaća spremaju se unutar polja.

Peta skripta nazvana je Plotiranje. Ova skripta sadrži jednu klasu i dvije metode hs i Ts. Pomoću ove dvije metode prikazuju se promjene stanja i traženi proces unutar *T,s*- i *h,s*-dijagrama.

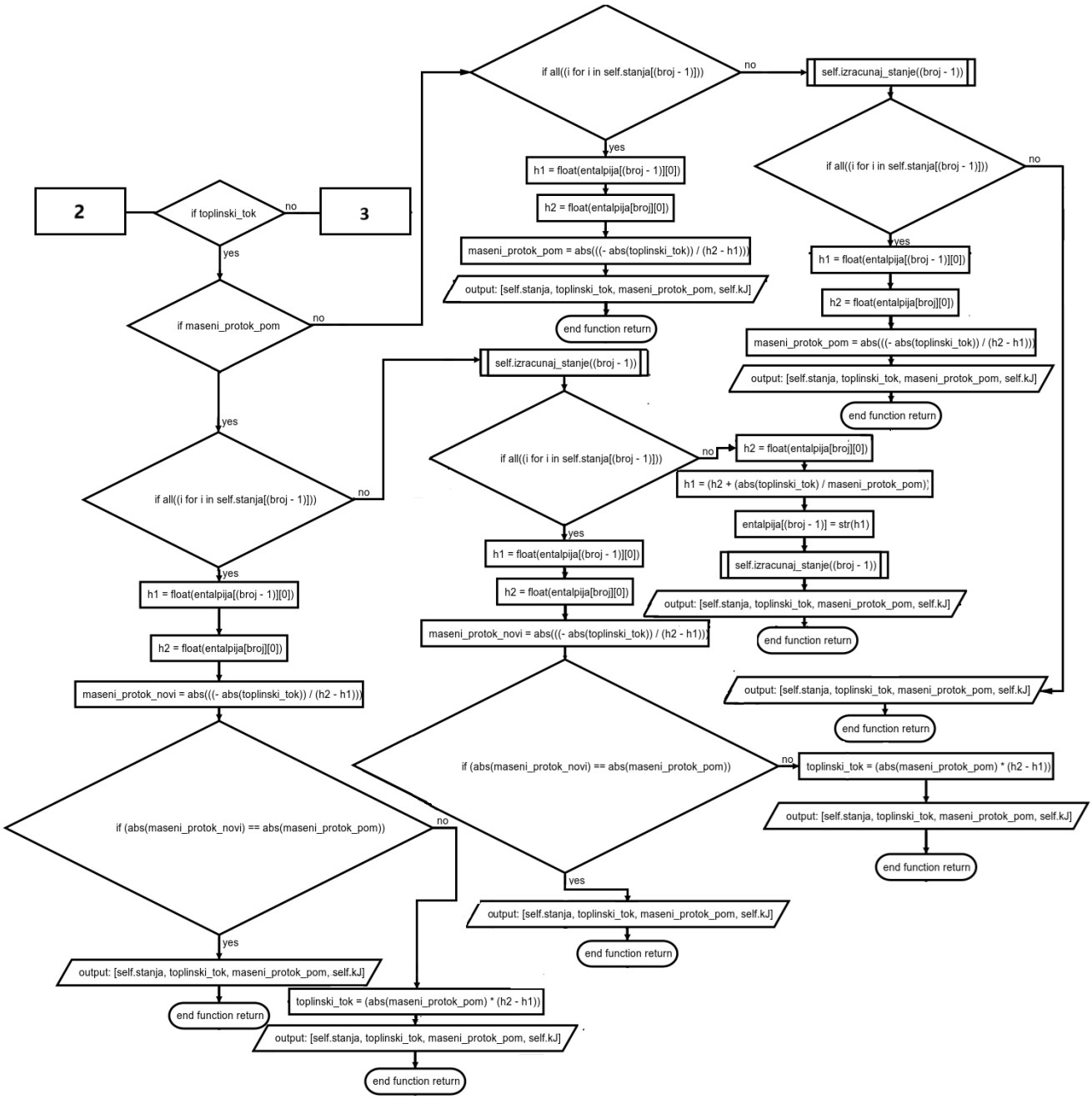
Zadnja skripta nazvana je Tkinter_gui. Sam naziv skripte govori za što je ona korištena. Unutar ove skripte napravljena je GUI aplikacija te je većina prethodno opisanih skripti uvezena unutar nje. Skripta se sastoji od klase nazvane Procesi i mnoštva metoda koje daju funkcionalnost i logiku za korištenje aplikacije.



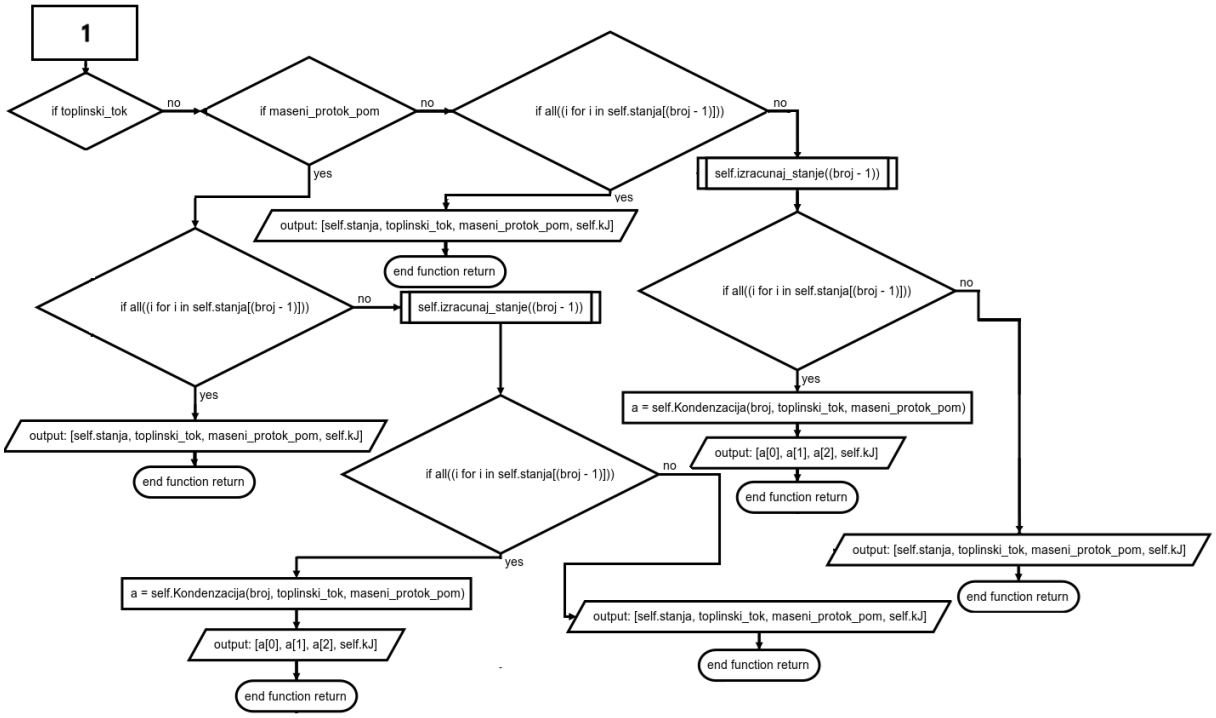
Slika 3.1. Blokovski prikaz uvoza skripti



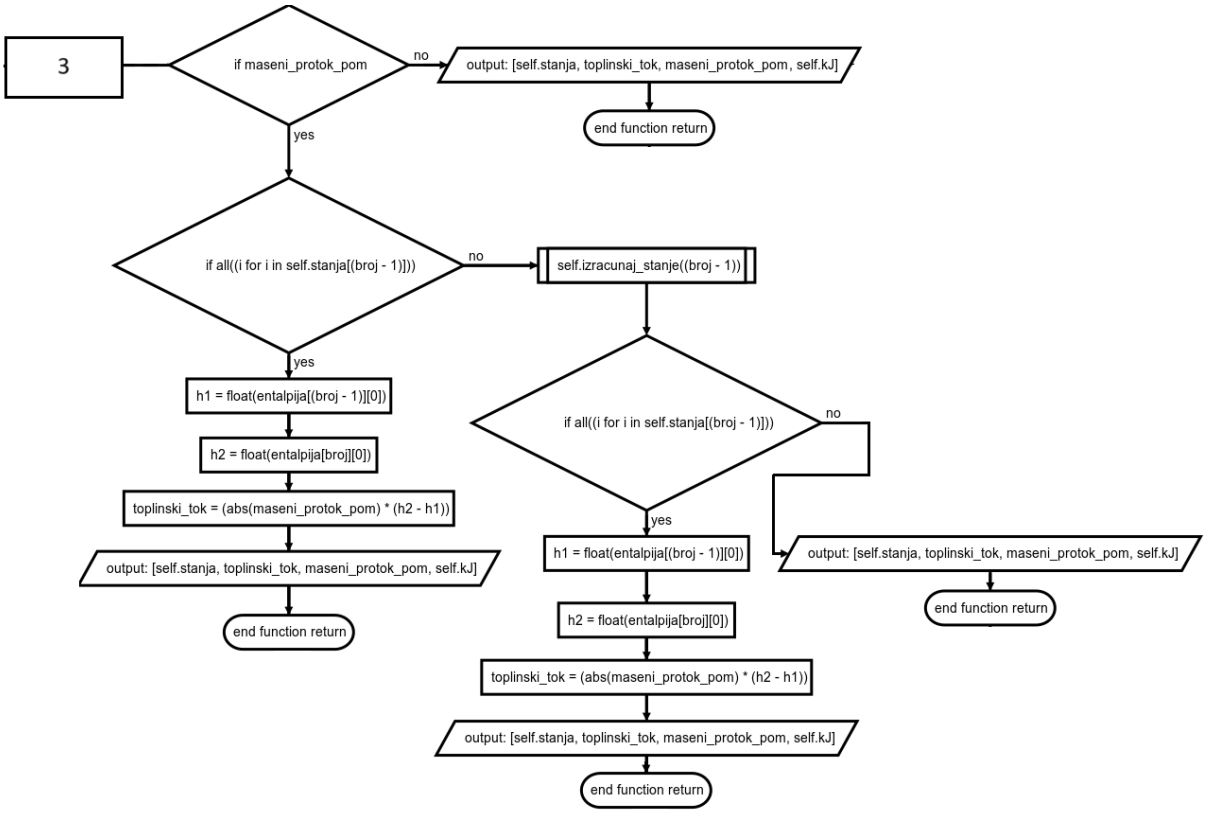
Slika 3.2. Dijagram toka metode Kondenzacija



Slika 3.3. Nastavak 1 dijagrama toka metode Kondenzacija



Slika 3.4. Nastavak 2 dijagrama toka metode Kondenzacija



Slika 3.5. Nastavak 3 dijagrama toka metode Kondenzacija

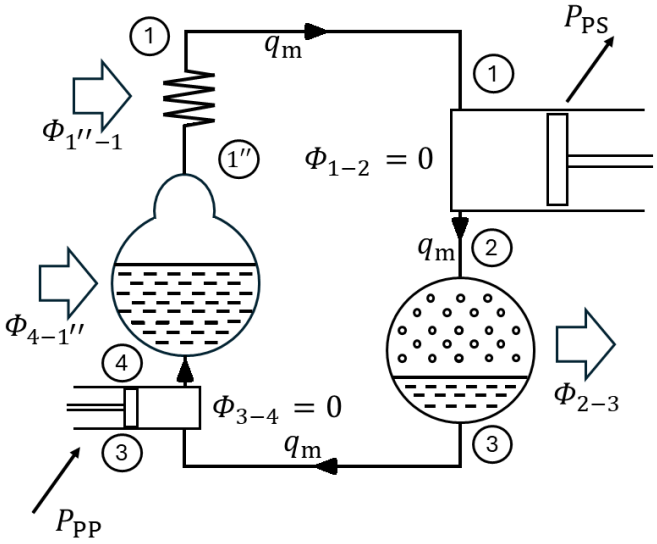
3.2. Pravilno korištenje aplikacije

U ovom poglavlju obrađuje se rad grafičkog korisničkog sučelja te će se prikazati osnovni procesi i jednadžbe korištene za njihovo računanje. Kao što je rečeno grafičko sučelje izrađeno je pomoću Tkinter biblioteke unutar programskog koda Python. Rad aplikacije prikazat će se na dva tehnička procesa iz termodinamike.

Odabrani tehnički procesi su iz područja desnokretnih kružnih procesa s vodenom parom kao radnom tvari. Desnokretni procesi koriste se u svrhu dobivanja mehaničke snage. Dobivena mehanička snaga i dovedeni toplinski tok kod desnokretnih procesa opisani su termičkim stupanjem djelovanja. Dva procesa na kojima će biti prikazan rad GUI-a su: Rankineov ciklus s pregrijanom parom i Rankineov ciklus s međupregrijačem pare. Rankineov ciklus u srži sastoji se od dviju izentropa i dviju izobara, no radi povećanja termičkog stupnja djelovanja moguća su određena poboljšanja procesa. Moguća poboljšanja procesa su pregrijavanje pare, međupregrijavanje pare i regenerativno predgrijavanje kondenzata.

U ovom radu obrađena su dva procesa s pregrijavanjem i međupregrijavanjem pare. Stoga, biti će objašnjena samo ta dva poboljšanja Rankineovog ciklusa. Narednim tekstom predstaviti će se sheme procesa i jednadžbe korištene za izračun odgovarajućih energijskih tokova preuzetih iz [4].

Jedan od načina povišenja srednje temperature dovođenja topline je pregrijavanje pare na ulasku u parnu turbinu, čime se povisuje termički stupanj djelovanja procesa. Slika 3.6. prikazuje shemu Rankineovog ciklusa s pregrijanom parom.



Slika 3.6. Desnokretni Rankineov ciklus s pregrijanom parom [4]

Kako bi se izračunale potrebne veličine energijskih tokova unutar aplikacije koristile su se sljedeće jednadžbe

$$P_{1-2} = q_{\mathrm{m}}(h_1 - h_2) \quad \textbf{(3.1.)}$$

$$\Phi_{2-3} = q_{\mathrm{m}}(h_3 - h_2) \quad \textbf{(3.2.)}$$

$$P_{3-4} = q_{\mathrm{m}} v_3 (p_3 - p_4) \quad \textbf{(3.3.)}$$

$$\Phi_{4-1} = q_{\mathrm{m}}(h_1 - h_4) \quad \textbf{(3.4.)}$$

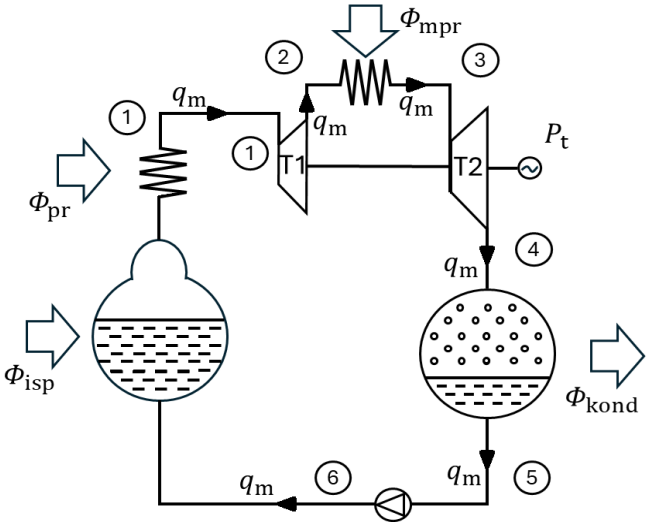
$$\eta = \frac{P_{1-2} + P_{3-4}}{\Phi_{4-1}} \quad \textbf{(3.5.)}$$

Također, potrebne veličine stanja, ako se nalaze u području mokre pare izračunate su prema sljedećim jednadžbama.

$$s = s' + x(s'' - s') \quad \textbf{(3.6.)}$$

$$x = \frac{s - s'}{s'' - s'} \quad \textbf{(3.7.)}$$

Drugi način poboljšanja termičkog stupnja djelovanja je međupregrijavanje pare. Slikom 3.7. prikazuje se shema Rankineovog ciklusa s međupregrijačem pare.



Slika 3.7. Desnokretni Rankineov ciklus s međupregrijačem pare [4]

Uz spomenute jednadžbe (3.6.) i (3.7.) korištene su sljedeće jednadžbe za računanje desnokretnog Renkineovog ciklusa s međupregrijačem pare.

$$\Phi_{dov} = q_m(h_1 - h_6 + h_3 - h_2) \quad \textbf{(3.8.)}$$

$$\Phi_{kond} = q_m(h_5 - h_4) \quad \textbf{(3.9.)}$$

$$P_t = P_{t1} + P_{t2} = q_m(h_1 - h_2) + q_m(h_3 - h_4) \quad \textbf{(3.10.)}$$

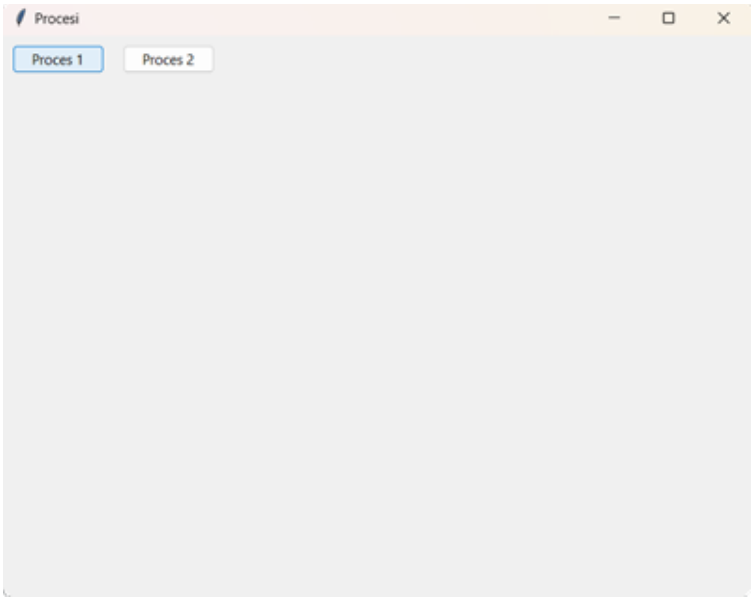
$$P_{pumpa} = q_m v_5(p_5 - p_6) \quad \textbf{(3.11.)}$$

$$\eta = \frac{P_{t1} + P_{t2} + P_{pumpa}}{\Phi_{dov}} \quad \textbf{(3.12.)}$$

Rezultati dobiveni prethodno spomenutim jednadžbama za izračunate procese prikazat će se pomoću *h,s*- i *T,s*-dijagrama. Prednost prikaza unutar *T,s*-dijagrama je u tome što se izmijenjena toplina može prikazati u obliku površine ispod linije ravnotežne promjene stanja. Procesni *T,s*-dijagram ima još jedno bitno svojstvo, iz tijeka linije promjena stanja može se zaključiti o predznaku izmijenjene topline. Pogodnosti *h,s*-dijagrama je u lakšem očitavanju razlika entalpija, čime se olakšava izračun odgovarajućih energetskih tokova.

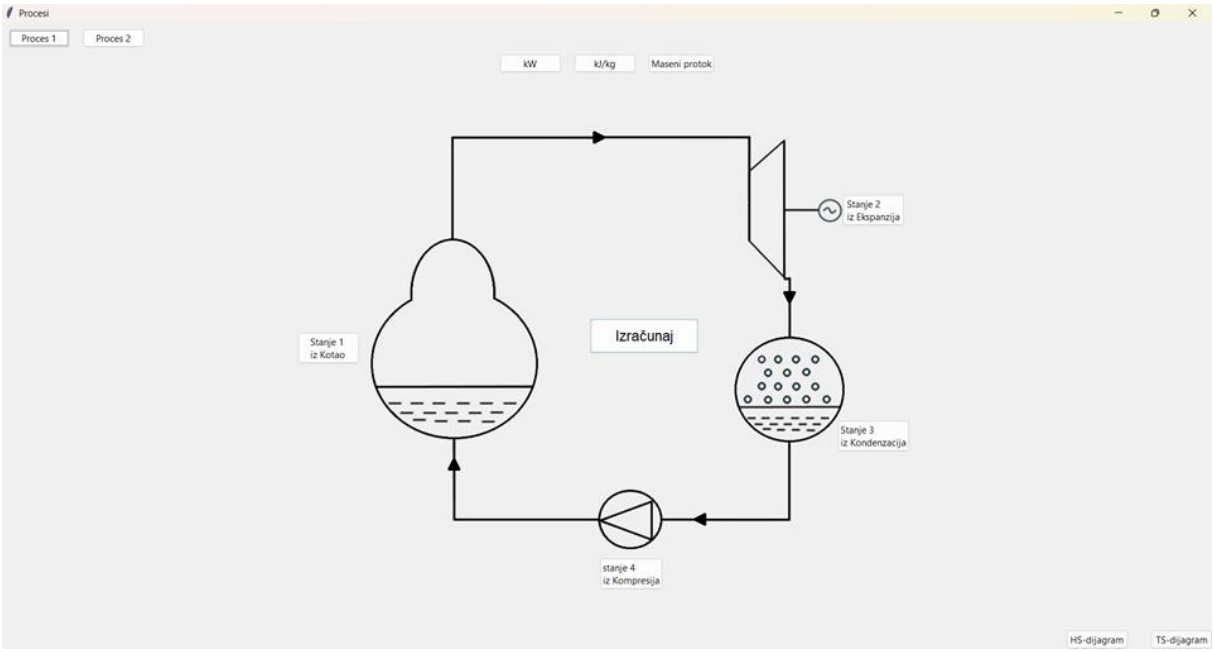
Programski kod i određeni dijelovi aplikacije bili su prikazani u prethodnim poglavljima, a sada će se predstaviti sam rad aplikacije. Pokretanjem programskog koda otvara

se objekt klase tk.Tk() ili glavni prozor. Unutar glavnog prozora, raspoređeni u okviru, su dvije instance klase Button. Objektima (gumbima) dodijeljeni su nazivi Proces 1 i Proces 2. Proces 1 označava Rankineov ciklus s pregrijanom parom, Proces 2 označava Rankineov ciklus s međupregrijačem pare. Slika 3.8. prikazuje početni zaslon i instancu klase tk.Tk() s dva objekta klase Button.



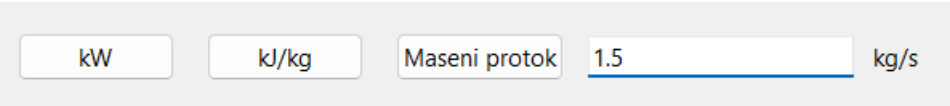
Slika 3.8. Prikaz glavnog objekta i dva objekta klase Button

Pritiskom na prvi objekt pokreće se metoda display_image. Metoda display_image učitava sliku procesa, a unutar nje poziva se metoda procesa, koja učitava potrebne *widgete* i postavlja ih na upisana mjesta.



Slika 3.9. Početni zaslon objekta Proces 1

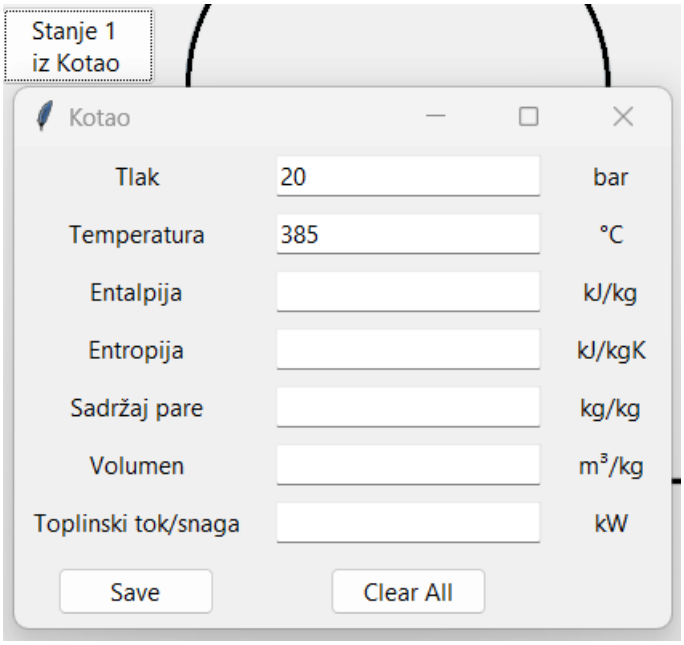
Slika 3.9. prikazuje proces i mnoštvo objekata iz klase Button. U lijevom kutu ostala su dva inicijalna objekta za procese. Na sredini se nalaze tri objekta (gumba) pod nazivima kW, kJ/kg i Maseni protok. Objekt kW pri inicijalizaciji je automatski aktivan te njegovim pritiskom stvara se objekt Maseni protok. Pritiskom na objekt Maseni protok otvara se *entry* polje za unos brojčane vrijednosti masenog protoka, kao što je prikazano na slici 3.10. U slučaju da su željene vrijednosti toplinskih tokova u mjernoj jedinici kW potrebno je aktivirati ovaj objekt. Na mjesto *entry* polja moguće je upisati brojčanu vrijednost, no ako sustav može izračunati vrijednost masenog protoka iz upisanog željenog toplinskog toka ili snage, vrijednost masenog protoka automatski se prikazuje u *entry* polju. Pritiskom na objekt kJ/kg uklanja se objekt Maseni protok i stvoreno *entry* polje, kao što je prikazano na slici 3.11.,te će sve vrijednosti toplinskih tokova ili snaga biti iskazane kao specifične.



Slika 3.10. Objekti klase Button i objekt klase Entry

Slika 3.11. Aktiviranje objekta kJ/kg

Nadalje, oko slike procesa nalaze se objekti klase Button. Objektima su dodijeljeni nazivi izlaznih stanja iz uređaja, npr. „stanje 1 iz Kotao“. Pritiskom na objekt otvara se podprozor s nekoliko polja za unos brojčanih vrijednosti. Opisana funkcionalnost prikazana je prema slici 3.12.

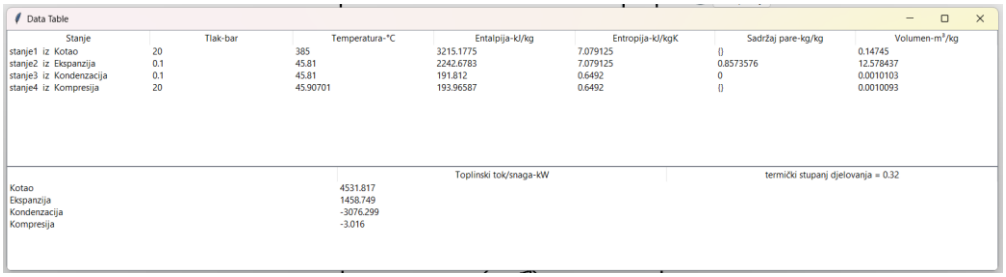


Slika 3.12. Otvoreni podprozor s poljima za unos numeričkih vrijednosti

Nakon otvaranja podprozora moguće je upisati željeno stanje, unutar polja automatski su upisani brojevi kao inicijalni primjer, no moguće je pritiskom objekta Clear all ukloniti sve napisane brojeve i unijeti druge. Nakon što se upiše željeno stanje pritiskom objekta Save pohranjuju se željeni podaci. Zatvaranjem podprozora i njegovim ponovnim otvaranjem podaci koji su zadnje upisani biti će ponovno prikazani. Za svaki ostali objekt, koji opisuje stanje, vrijede ista pravila i otvara se slični prozor, jedina razlika je u njihovom nazivu[2].

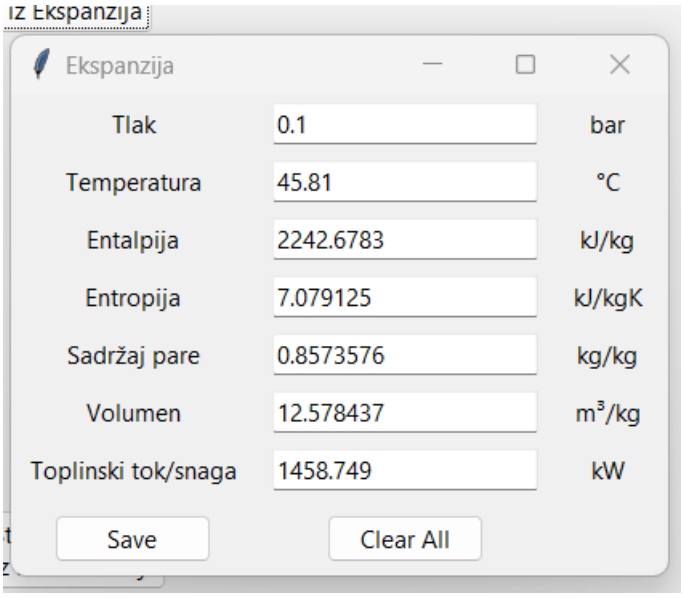
[2] kao što se može vidjeti otvoreni prozor nazvan je Kotao, a za objekt „stanje 2 iz Ekspanzija“ pisala bi Ekspanzija

Unutar slike procesa nalazi se objekt klase Button pod nazivom Izračunaj. U slučaju da su sve vrijednosti koje su vezane za proces valjano unesene, ovaj objekt pokreće metodu izracunaj. Metoda izracunaj iz ostalih skripta računa željena stanja procesa. Ako je sva stanja moguće izračunati otvara se podprozor naziva Data Table, kao što je prikazano prema slici 3.13. U ovom prozoru sva izračunata stanja prikazana su tablicom, te su upisane vrijednosti toplinskih tokova ili snaga. Na samom dnu tablice nalazi se ćelija naziva „termički stupanj djelovanja“ u kojem je upisana izračunata vrijednost termičkog stupnja djelovanja provedenog procesa.



Slika 3.13. Prikaz podprozora Data Table

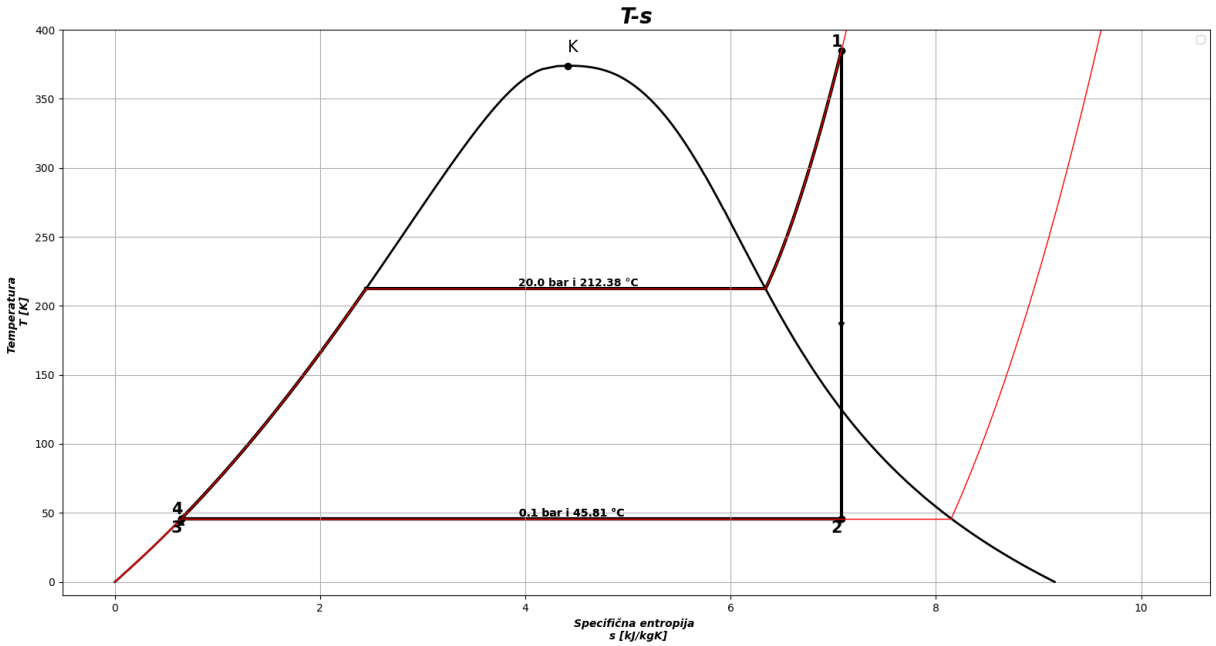
Ako je stanje pravilno izračunato, unutar podprozora vezanih za stanja biti će prikazane numeričke vrijednosti izračunate za to stanje, kao što prikazuje slika 3.14.



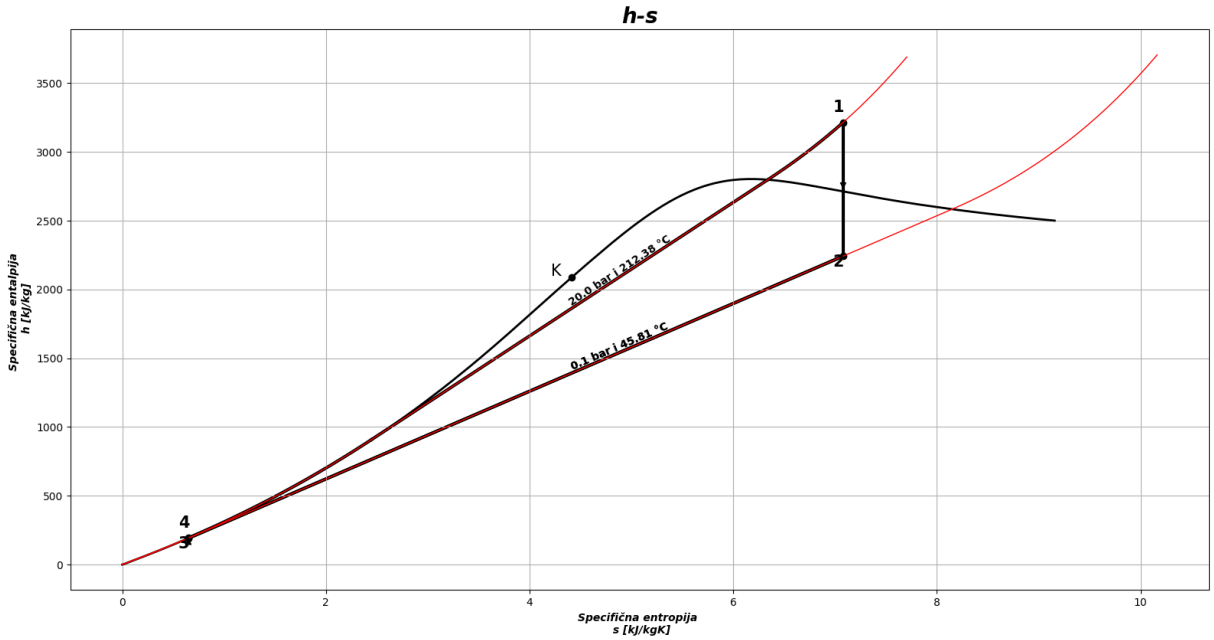
Slika 3.14. Podprozor Ekspanzija s upisanim izračunatim stanjem

Ostala su još dva objekta koja nisu opisana, ta dva objekta također su instance klase Button i nalaze se u donjem desnom kutu aplikacije. Objekti su nazvani HS-dijagram i TS-

dijagram. Prilikom aktivacije pokreće se skripta Plotiranje koja stvara grafičke prikaze procesa unutar *h,s*- i *T,s*-dijagrama. Grafički prikaz dotičnog procesa i projekcija njegovih izoploha prikazani su slikama 3.15. i 3.16.



Slika 3.15. Prikaz *T,s*-dijagrama unutar objekta Proces 1

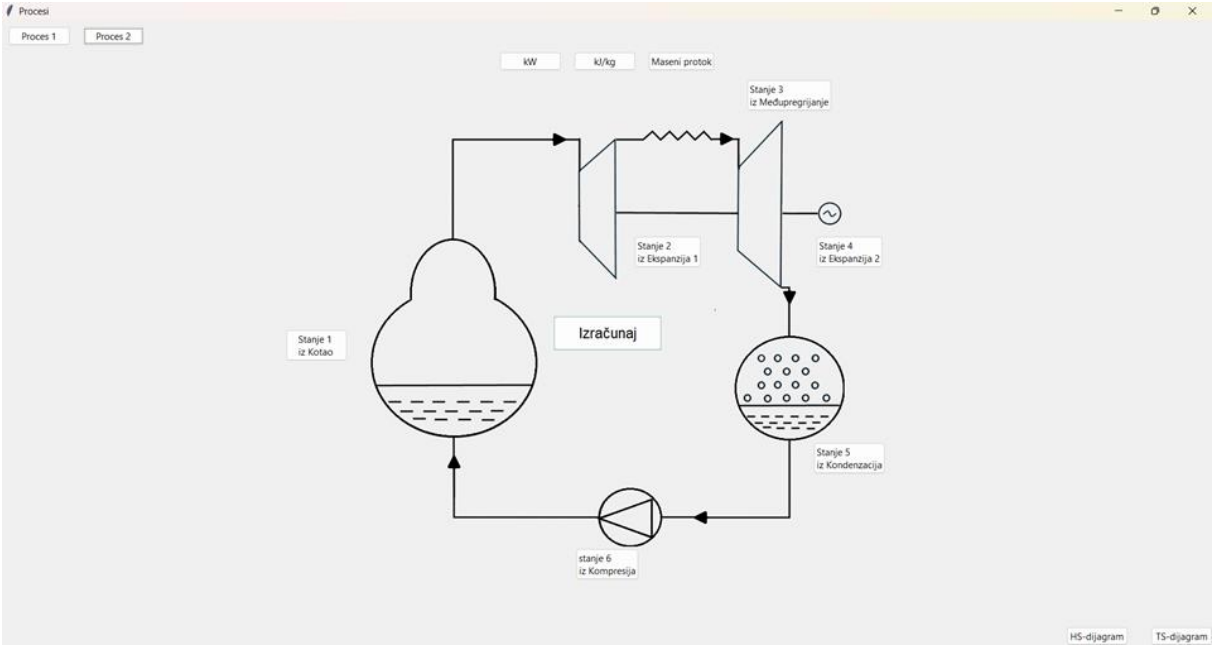


Slika 3.16. Prikaz *h,s*-dijagrama unutar objekta Proces 1

S grafa je teško uočljivo, no stanje 3 je stanje vrele kapljevine, a stanje 4 je stanje pothlađene kapljevine. Razlog tome je odnos bliskosti izobara u području pothlađene kapljevine

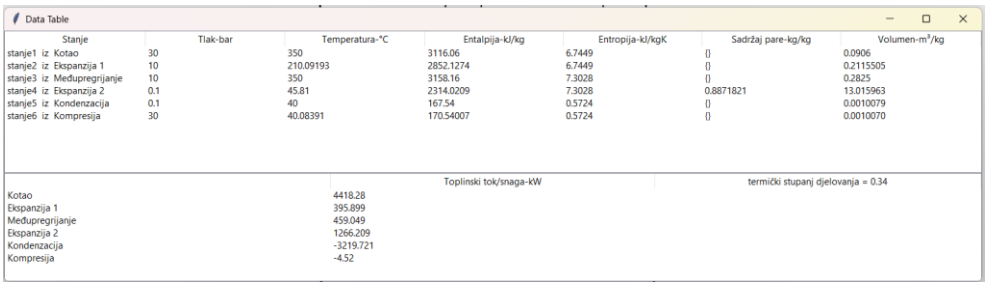
i mjerila prikaza. Nije lako uočiti razliku između pothlađenog stanja i stanja vrele kapljevine kada su blizu linije zasićenja. To znači da točke koje prikazuju stanja pothlađene kapljevine i vrele kapljevine mogu izgledati gotovo identično na dijagramu, iako termodinamički nisu potpuno iste. Stoga, čak i ako se točka stvarno nalazi u pothlađenom području, ona će u dijagramima izgledati kao da je vrlo blizu ili čak na liniji zasićenja, kao što je prikazano na slikama 3.15. i 3.16.

Proces 2, kao što je već rečeno, je Rankineov ciklus s međupregrijačem pare. Za Proces 2 vrijede sva ista pravila i logika upravljanja objektima kao i za Proces 1. Slika 3.17. prikazuje Proces 2.



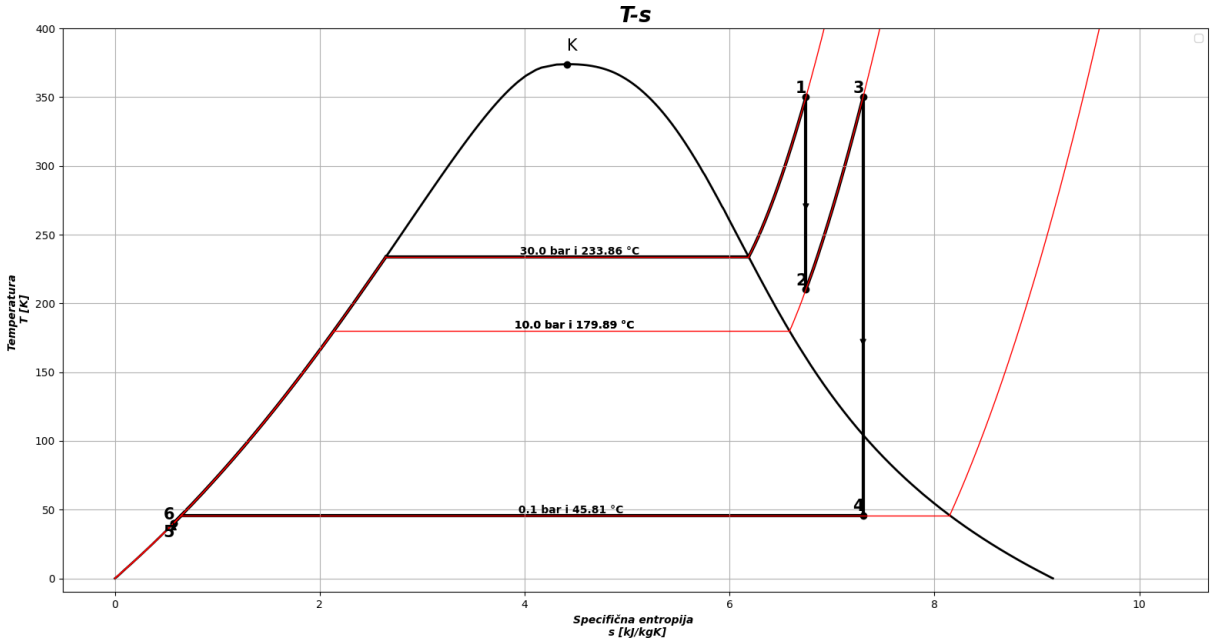
Slika 3.17. Početni zaslon objekta Proces 2

Pritiskom na objekt Izračunaj prikazuju se izračunata stanja, kao što je prikazano na slici 3.18.

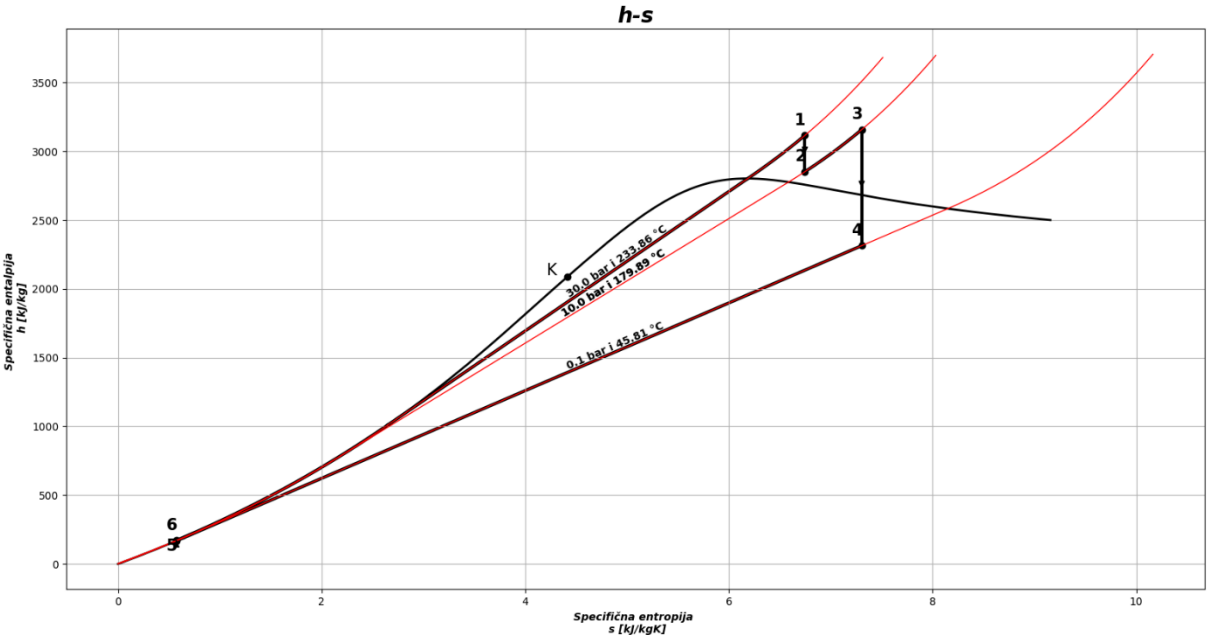


Slika 3.18. Izračunata stanja u podprozoru Data Table

Kao i kod objekta Proces 1 pritiskom na objekte HS-dijagram i TS-dijagram dobivaju se prikazi izračunatog procesa, kao prema slikama 3.19. i 3.20.



Slika 3.19. Prikaz *T-s* dijagrama unutar objekta Proces 2



Slika 3.20. Prikaz *h-s* dijagrama unutar objekta Proces 2

4. SciPy

Prijašnjim poglavljem opisao se rad aplikacije, predstavila su se dva procesa i prikazale numeričke i grafičke vrijednosti, točnost grafičkih prikaza i njezina uređenost treba se prepisati Pythonovoj ugrađenoj biblioteci matplotlib. Prednosti ove biblioteke su:

- Jednostavno kreiranje dijagrama s malom količinom koda, što je čini popularnom u znanstvenom programiranju.
- Prilagodljivost; korisnici mogu sami prilagoditi izgleda dijagrama.
- Moguća pohrana dijagrama u različite formate datoteka, kao što su PNG, JPG, PDF i SVG.
- Integracija s drugim alatima; matplotlib dobro se integrira s alatima poput NumPy, SciPy, Pandas itd., što olakšava rad s velikim bazama podataka.

Kako bi se izoplohe uspjele prikazati sa što većom točnosti koriste se različite biblioteke poput SciPy-a. SciPy je Python biblioteka koja se koristi za tehničke izračune, ova biblioteka se nadovezuje na biblioteku NumPy i dodaje funkcionalnosti za rad s numerikom. Funkcionalnosti unutar ScyPy su: optimizacija, integracija, interpolacija, linearna algebra, itd. U sklopu ovog poglavlja predstaviti će se različiti oblici interpolacije u svrhu dobivanja što veće točnosti pri interpolaciji podataka izoploha, te obrazložiti korištenje odabrane metode interpolacije.

Prije početka potrebno je razjasniti što je to interpolacija. Interpolacija opisuje metodu stvaranja novih podataka unutar ograničenog raspona poznatih podataka. Uz poznate podatke, novim vrijednostima linija interpolacije mora prolaziti!

4.1. Interp1d

Klasa interp1d nalazi se u sklopu SciPy biblioteke. Ova klasa koristi se za linearnu interpolaciju zadanog skupa podataka, no prima određene argumente kao što su 'quadratic' i 'cubic'. Ovim argumentima umjesto inicijalne linearne interpolacije koristi se polinomska interpolacija[3]. Interp1d prima ulazne vrijednosti iz diskretne baze podataka te ih koristi kako bi aproksimirala funkciju oblika y = f(x). Izlazni parametar ove klase je funkcija, čija metoda

[3] linija koja interpolira podatke je oblika polinoma trećeg ili četvrtog reda

koristi zadani oblik interpolacije kako bi našla vrijednosti novih točaka. Sljedećim kodom prikazat će se korištenje klase interp1d.

```
f1 = interpolate.interp1d(entropija1, temperatura1, kind = 'linear')
f3 = interpolate.interp1d(entropija1, temperatura1, kind = 'cubic')
f5 = interpolate.interp1d(entropija1, temperatura1, kind = 'quadratic')
```

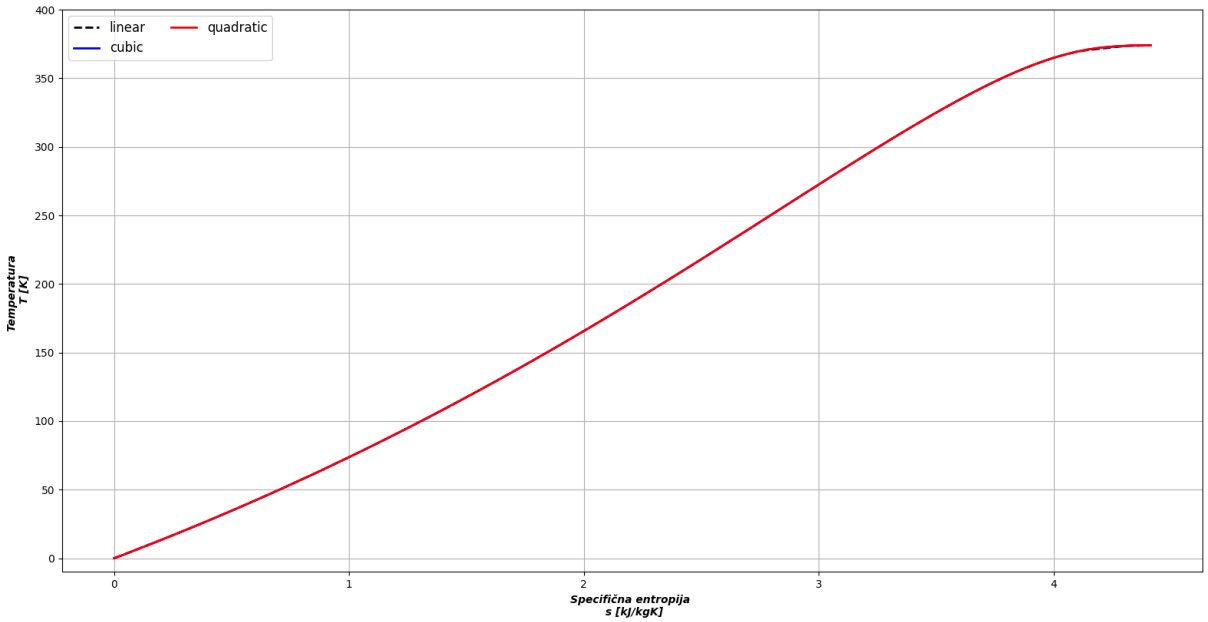
Prije same uporabe klase interp1d učitala se SciPy biblioteka i dotični modul interpolate naredbom from scipy import interpolate, U klasu interp1d upisuju se atributi. Prva dva atributa su poznati podaci koje treba interpolirati, te kao treći atribut upisuje se vrsta krivulje kojom se interpolira diskretni skup podataka. Klasa interp1d vraća aproksimiranu funkciju, npr. funkcija f1.

```
entropija_nova1 = np.linspace(entropija1[0], entropija1[-1],200)
```

Nakon inicijalizacije aproksimirajuće funkcije, proširuje se baza podataka. Proširenje baze podataka prikazano je naredbom np.linspace. U diskretni skup pohranilo se 200 točaka. Točnost krivulje raste povećanjem broja podataka, no time se zauzima memorija obrade podataka. Nakon što se proširio skup podataka iz aproksimirajuće funkcije dobivaju se vrijednosti novih točaka.

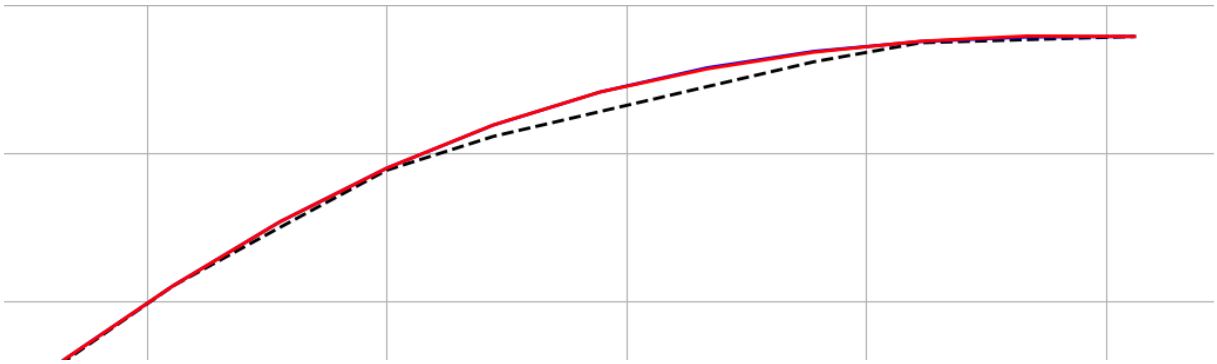
```
temperatura_nova1 = f1(entropija_nova1)
```

Slikom 4.1. prikazuje se razlika između linearne i interpolacije polinomom.



Slika 4.1. Prikaz interp1d interpolacije

Sa slike 4.1. djeluje kao da su krivulje jednako interpolirale zadani skup podataka, no uvećavanjem će se vidjeti njihova razlika.



Slika 4.2. Usporedba 'quadratic', 'cubic' i 'linear' interpolacije

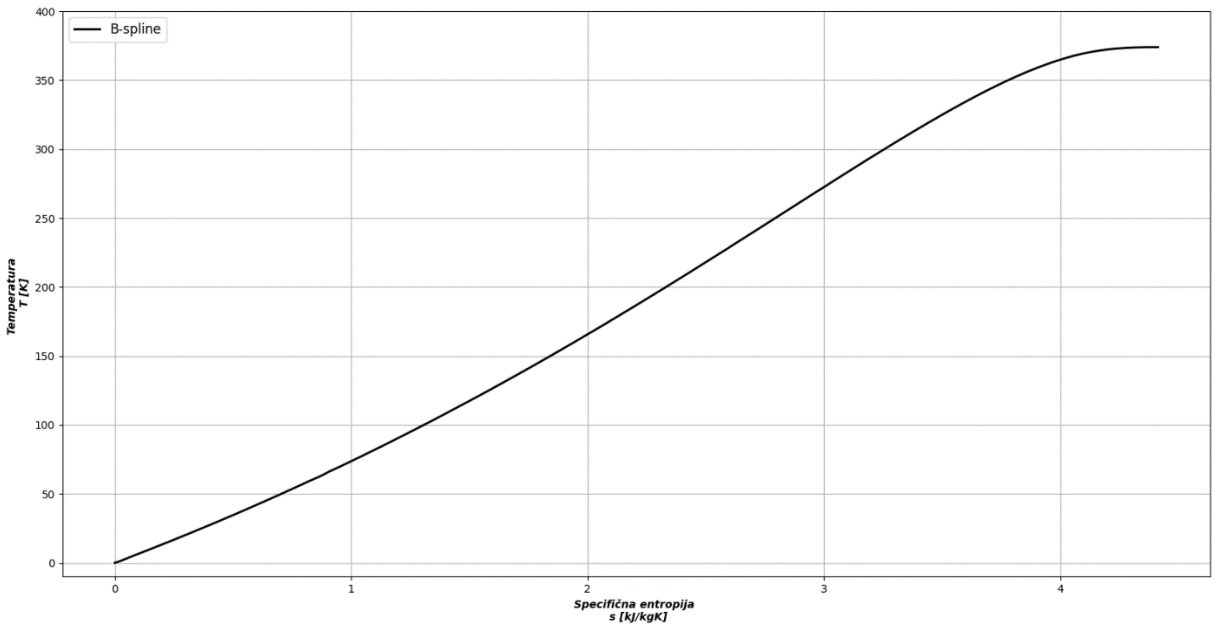
Slika 4.2. prikazuje razliku interpolacije polinoma i pravca. Kod linearne interpolacije zapravo vidimo pravac između dvije točke, dok su polinomi trećeg i četvrtog reda praktički jedna na drugoj, te imaju mekši prijelaz između točaka.

4.2. Splrep i Splev

Klasa splrep unutar biblioteke SciPy, koristi diskretizirane podatke kako bi pronašla 1D *B-spline* (eng. *Basis spline*) krivulju. Za određeni skup podataka aproksimira splajn krivulju stupnja k na određenom intervalu. Klasa kao ulazni podatak prima x i y podatke, a vraća *tuple* koji sadrži vektor čvorova, koeficijente *B-spline* i stupanj splajna.

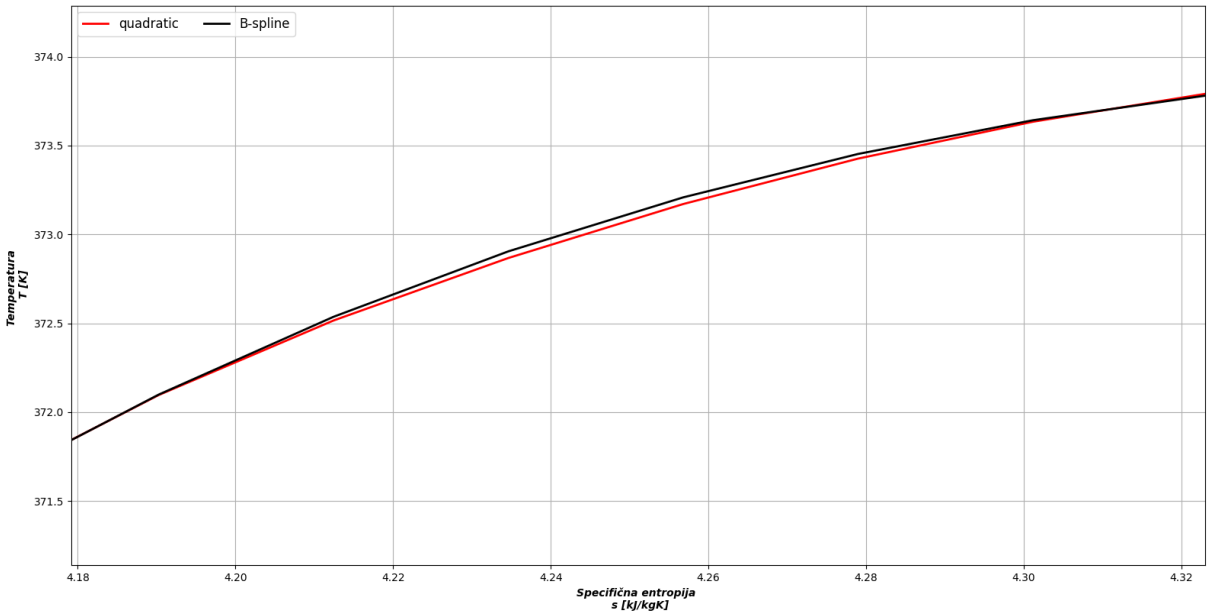
Splev je klasa unutar SciPy biblioteke. Klasa splev za dane čvorove i koeficijente *B-spline* krivulje, procjenjuje stupanj polinoma i njegove derivacije. Kao ulazni podaci daju se podaci s osi apscisa i *tuple* koji vraća splrep. Splev klasa vraća polje vrijednosti točaka s osi apscisa. Ako je vraćen i *tuple* onda je to lista polja koja predstavlja zakrivljenost u N-dimenzionalnom prostoru. Na sljedećem kodu biti će prikazano korištenje splev i splrep klasa.

```
f6 = interpolate.splrep(entropija1, temperatura1)

entropija_nova1 = np.linspace(entropija1[0], entropija1[-1],200)

temperatura_nova1= interpolate.splev(entropija_nova1, f6)
```



Slika 4.3. Prikaz B-spline krivulje

Slika 4.3. prikazuje izgled *B-spline* krivulje, no da bi se vidjela razlika potrebno je usporediti s 'quadratic' krivuljom iz interp1d klase.



Slika 4.4. Usporedba B-spline i interp1d 'quadratic' interpolacije

Slika 4.4. prikazuje usporedbu interpolacije pomoću *B-spline* i polinoma četvrtog reda. Sa slike se vidi da krivulja koja je interpolirana polinomom četvrtog reda ima veće pregibe na točkama nego *B-spline*, te da ima određena odstupanja. Zato je *B-spline* krivulja pogodnija pri interpolaciji višedimenzionalnih podataka i korištena pri interpolaciji unutar aplikacije.

5. ZAKLJUČAK

Ovaj rad je demonstrirao kako se Python, kao objektno orijentirani programski jezik, može uspješno koristiti za izradu grafičkih korisničkih sučelja u edukacijske i znanstvene svrhe. Kroz razvoj sučelja za vizualizaciju izoploha u N-dimenzionalnom prostoru, omogućena je interaktivna vizualizacija tehničkih procesa, čime se pruža pojednostavljenje repetitivnih zadataka i jednostavniji rad s podacima i njihova analiza.

Primjena biblioteka Tkinter, Numpy, Scipy i Matplotlib osigurala je funkcionalnost potrebnu za rad s podacima, interpolaciju te vizualni prikaz rezultata. Različite metode interpolacije koje su ugrađene u Numpy i Scipy testirane su i uspoređene, čime je omogućeno razumijevanje njihovih primjena u tehničkim procesima. Osim toga, objektno orijentirani pristup izradi sučelja olakšao je modularnost i proširivost aplikacije, što je korisno za moguće nadogradnje.

Osim edukacije, druga prednost ovog rada je integracija dviju različitih grana inženjerstva – računalnog i tehničkog. Korištenjem računalnih alata otvara se mogućnost učinkovitijeg rješavanja inženjerskih problema. Primjenom interdisciplinarnog pristupa mogu se razvijati naprednije aplikacije koje će služiti u raznim tehničkim i znanstvenim područjima.

Rad predstavlja osnovu za moguće razvijanje budućih aplikacija koje kombiniraju inženjerske proračune s interaktivnim grafičkim sučeljima. Preporuke za budući rad uključuju daljnju optimizaciju interpolacijskih metoda, proširenje baze podataka za unos fizikalnih veličina, kao što je uvoz biblioteke `CoolProp` te dodatne funkcionalnosti u GUI-ju, kao što su izrada svoga desnokretnog ili lijevokretnog procesa. Ova aplikacija može se dalje razvijati i prilagođavati specifičnim tehničkim potrebama.

Time je ovaj rad predstavio osnovu za daljnja istraživanja i napredak u području vizualizacije i simulacije tehničkih procesa korištenjem Python-a i njegovih biblioteka, prikazavši doprinos spajanja inženjerskih grana kroz aplikacije.

LITERATURA

[1] https://www.enciklopedija.hr/clanak/programiranje

[2] Dusty, P.: Python 3 Objected – oriented Programming, Packt Publishing Ltd., Birmingham, 2015.

[3] D. Moore, A.: Python GUI programming with Tkinter, Packt Publishing Ltd., Birmingham, 2018.

[4] Halasz, B.: Uvod u termodinamiku, Fakultet strojarstva i brodogradnje, Zagreb, 2021.

PRILOZI

I. Listing koda metode Kondenzacija

```
def Kondenzacija(self, broj, toplinski_tok = None, maseni_protok_pom = None):

    if tlak[broj] and tlak[broj-1]:
        if tlak[broj] != tlak[broj-1]:
            print('Različiti tlakovi za kondenzaciju uzimam tlak od stanja prije upisanog')
            self.stanja[broj][0] = tlak[broj-1]
            self.sve[broj][0] = tlak[broj-1]
        else:
            pass

    if tlak[broj] and not tlak[broj-1]:
        self.stanja[broj-1][0] = tlak[broj]
        self.sve[broj-1][0] = tlak[broj]

    elif tlak[broj-1] and not tlak[broj]:
        self.stanja[broj][0] = tlak[broj-1]
        self.sve[broj][0] = tlak[broj-1]

    if self. kJ == '-':
        pass
    else:
        maseni_protok_pom = 1

    self.izracunaj_stanje(broj)

    if np.array_equal(self.stanja[broj], self.sve[broj]):
        if toplinski_tok:
            if maseni_protok_pom:
                if all(i for i in self.stanja[broj-1]):
```

```
                h1 = float(entalpija[broj-1][0])
                h2 = h1 - (abs(toplinski_tok)/abs(maseni_protok_pom))
                entalpija[broj] = str(h2)
                self.izracunaj_stanje(broj)
                return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
            else:
                self.izracunaj_stanje(broj-1)
                if all(i for i in self.stanja[broj-1]):
                    h1 = float(entalpija[broj-1][0])
                    h2 = h1 - (abs(toplinski_tok)/abs(maseni_protok_pom))
                    entalpija[broj] = str(h2)
                    self.izracunaj_stanje(broj)
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
                else:
                    print('Neznam stanje od prije pa nemogu iračunati stanje!')
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
        else:
            print('Imam topl tok, ali nemam mas_protok i nemam stanje, nemogu izracunati!')
            return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
    else:
        if maseni_protok_pom:
            print('Neznam stanje niti toplinski tok, ali znam maseni protok')
            if all(i for i in self.stanja[broj-1]):
                return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
            else:
                self.izracunaj_stanje(broj-1)
                if all(i for i in self.stanja[broj-1]):
                    print('izračunao sam stanje prije, računam sadanšnje stanje')
                    a = self.Kondenzacija(broj, toplinski_tok, maseni_protok_pom)
                    return [a[0], a[1], a[2], self.kJ]
                else:
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
```

```
            else:
                print('Neznam stanje niti toplinski tok niti znam maseni protok')
                if all(i for i in self.stanja[broj-1]):
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
                else:
                    self.izracunaj_stanje(broj-1)
                    if all(i for i in self.stanja[broj-1]):
                        print('izračunao sam stanje prije, računam sadanšnje stanje')
                        a = self.Kondenzacija(broj, toplinski_tok, maseni_protok_pom)
                        return [a[0], a[1], a[2],self.kJ]
                    else:
                        return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
        else:
            if toplinski_tok:
                if maseni_protok_pom:
                    if all(i for i in self.stanja[broj-1]):
                        h1 = float(entalpija[broj-1][0])
                        h2 = float(entalpija[broj][0])
                        maseni_protok_novi = abs(-abs(toplinski_tok)/(h2-h1))
                        if abs(maseni_protok_novi) == abs(maseni_protok_pom):
                            print('Maseni protoci se podudaraju, sve štima')
                            return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
                        else:
                            print('maseni protoci se ne podudaraju, vjv krivo upisan topl tok')
                            toplinski_tok = abs(maseni_protok_pom)*(h2-h1)
                            return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
                    else:
                        self.izracunaj_stanje(broj-1)
                        if all(i for i in self.stanja[broj-1]):
                            h1 = float(entalpija[broj-1][0])
                            h2 = float(entalpija[broj][0])
                            maseni_protok_novi = abs(-abs(toplinski_tok)/(h2-h1))
```

```
                if abs(maseni_protok_novi) == abs(maseni_protok_pom):
                    print('Maseni protoci se podudaraju, sve štima')
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
                else:
                    print('maseni protoci se ne podudaraju, vjv krivo upisan topl tok')
                    toplinski_tok = abs(maseni_protok_pom)*(h2-h1)
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
            else:
                print('Neznam prošlo stanje ali ga mogu izračunati')
                h2 = float(entalpija[broj][0])
                h1 = h2 + (abs(toplinski_tok)/maseni_protok_pom)
                entalpija[broj-1] = str(h1)
                self.izracunaj_stanje(broj-1)
                return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
        else:
            if all(i for i in self.stanja[broj-1]):
                h1 = float(entalpija[broj-1][0])
                h2 = float(entalpija[broj][0])
                maseni_protok_pom = abs( -abs(toplinski_tok)/(h2-h1))
                return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
            else:
                self.izracunaj_stanje(broj-1)
                if all(i for i in self.stanja[broj-1]):
                    h1 = float(entalpija[broj-1][0])
                    h2 = float(entalpija[broj][0])
                    maseni_protok_pom = abs(-abs(toplinski_tok)/(h2-h1))
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
                else:
                    print('imam toplinski tok, nemam maseni protok i nemam stanje od prije')
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
    else:
        if maseni_protok_pom:
```

```
            if all(i for i in self.stanja[broj-1]):
                h1 = float(entalpija[broj-1][0])
                h2 = float(entalpija[broj][0])
                toplinski_tok = abs(maseni_protok_pom)*(h2-h1)
                return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
            else:
                self.izracunaj_stanje(broj-1)
                if all(i for i in self.stanja[broj-1]):
                    h1 = float(entalpija[broj-1][0])
                    h2 = float(entalpija[broj][0])
                    toplinski_tok = abs(maseni_protok_pom)*(h2-h1)
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
                else:
                    print('Neznam stanje od prije pa nemogu iračunati stanje!')
                    return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
        else:
            print('Znam stanje, ali nemam maseni protok niti toplinski tok')
            return [self.stanja, toplinski_tok, maseni_protok_pom, self.kJ]
```