



Gimnazija Kranj

# PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

Računalništvo

Raziskovalna naloga



**Avtor:** Marko ZUPAN

**Mentorica:** mag. Zdenka VRBINC

Tržič, 2020



## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

### **Naslov naloge:**

PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

### **Področje:**

Računalništvo

### **Avtor:**

Marko Zupan

### **Mentorica:**

mag. Zdenka Vrbinc

### **Lektorica:**

Bernarda Lenaršič

### **Leto izdelave:**

2020

### **Šola:**

Gimnazija Kranj

## KAZALO VSEBINE

<b>1</b>	<b>UVOD</b> .....	<b>3</b>
1.1	CILJI.....	3
<b>2</b>	<b>HIPOTEZE</b> .....	<b>3</b>
<b>3</b>	<b>TEORETIČNE PREDPOSTAVKE</b> .....	<b>4</b>
3.1	PYTHON.....	4
3.1.1	Moduli.....	4
3.1.2	Tkinter.....	4
3.1.3	Math.....	5
3.1.4	Matplotlib.....	5
3.1.5	SymPy.....	5
3.1.6	NumPy.....	5
3.1.7	PyInstaller.....	5
3.2	ATOM.....	6
<b>4</b>	<b>EMPIRIČNI DEL</b> .....	<b>6</b>
4.1	NAMESTITEV PROGRAMSKEGA JEZIKA PYTHON.....	7
4.2	NAMESTITEV INTEGRIRANEGA RAZVOJNEGA ORODJA ATOM.....	8
4.3	PROGRAMIRANJE KALKULATORJA.....	9
4.3.1	Izdelava okna.....	9
4.3.1.1	Vnosni zaslon.....	10
4.3.1.2	Gumbi za vnos števil in gumbi za osnovne operacije.....	10
4.3.1.3	Gumbi za izvajanje znanstvenih operacij.....	11
4.3.1.4	Napisi za aktivne funkcije kalkulatorja.....	11
4.3.2	Nastavljanje osnovnih parametrov.....	13
4.3.3	Programiranje vnosa znakov na zaslon.....	13
4.3.4	Programiranje vnašanja znakov s pomočjo tipkovnice.....	13
4.3.5	Programiranje brisanja znakov iz zaslona.....	14
4.3.6	Funkcija za izračun računa v kalkulatorju.....	14
4.3.7	Programiranje znanstvenih funkcij.....	16
4.3.7.1	Omogočanje funkcij in shranjevanje rezultata.....	16
4.3.7.2	Programiranje znanstvenih operacij.....	17
4.3.7.3	Programiranje dvojnih funkcij.....	17
4.3.7.4	Programiranje kotnih funkcij.....	18
4.3.8	Programiranje znanstvenega zapisa rezultata.....	19
4.3.9	Programiranje funkcij za pretvorbo števil v druge številske sisteme.....	20
4.3.10	Programiranje abstraktnih znanstvenih funkcij.....	20
4.3.10.1	Programiranje načina za urejanje matematičnih funkcij.....	20
4.3.10.2	Programiranje funkcije za limitiranje.....	21
4.3.10.3	Programiranje funkcije za računanje odvoda.....	22
4.3.11	Programiranje funkcije za pretvarjanje kotnega zapisa.....	23
4.3.12	Reševanje polinomov.....	24
4.3.12.1	Programiranje funkcije za iskanje realnih ničel polinoma.....	24
4.3.12.2	Risanje grafa polinoma.....	25
4.3.13	Izpis postopka.....	27
4.3.14	Zapis z ulomki.....	27



## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

<b>4.4</b>	<b>Testiranje kalkulatorja .....</b>	<b>29</b>
<b>4.5</b>	<b>Priprava programa za distribucijo .....</b>	<b>29</b>
4.5.1	Namestitev PyInstallerja .....	30
4.5.2	Priprava aplikacije za platformo MacOS .....	30
4.5.3	Priprava aplikacije za platformo Windows .....	31
4.5.3.1	Poprava grafične podobe kalkulatorja .....	31
4.5.3.2	Izgradnja aplikacije na platformi Windows .....	31
4.5.4	Priprava kalkulatorja za uporabo na mobilnih napravah .....	33
	<b>ANKETIRANJE SOVRSTNIKOV .....</b>	<b>34</b>
<b>4.6</b>	<b>34</b>	
<b>5</b>	<b>ZAKLJUČEK .....</b>	<b>35</b>
5.1	IZZIVI ZA NADALJEVANAJE .....	35
<b>6</b>	<b>VIRI IN LITERAURA .....</b>	<b>36</b>
<b>7</b>	<b>PRILOGA .....</b>	<b>37</b>

## KAZALO ZASLONSKIH SLIK

Slika 1: IDE Atom.....	6
Slika 2: Namestitev Pythona .....	7
Slika 3: Uvozimo Tkinter .....	9
Slika 4: Kreiranje zaslonskega okna .....	9
Slika 5: Kreiranje vnosnega zaslona.....	10
Slika 6: Primer ene vrstice osnovnih vnosnih gumbov .....	10
Slika 7: Primer ene vrstice funkcijskih tipk.....	11
Slika 8: Kreiranje napisa za funkcijo kosinus.....	11
Slika 9: Grafična podoba kalkulatorja po dodajanju gradnikov .....	12
Slika 10: Nastavljanje osnovnih parametrov.....	13
Slika 11: Funkcija za vnos na vnosni zaslon .....	13
Slika 12: Funkcija za vnos na seznam postopek .....	13
Slika 13: Primer kode, ki ob pritisku tipke 1 na tipkovnici vnese število 1 v kalkulator .....	13
Slika 14: Funkciji za brisanje .....	14
Slika 15: Funkcija za računanje računa .....	15
Slika 16: Omogočanje funkcij in shranjevanje rezultata .....	16
Slika 17: Izsek iz funkcije za omogočanje funkcijskih gumbov .....	16
Slika 18: Primer funkcije, ki zahteva dvojni vnos .....	17
Slika 19: Primer dvojne funkcije.....	18
Slika 20: Koda za opisovanje izbrane vrste kota .....	18
Slika 21: Primer kotne funkcije sinusa .....	19
Slika 22: Števec znanstvenega zapisa .....	19
Slika 23: Funkciji za znanstveni in neznanstveni zapis .....	20
Slika 24: Funkcija za pretvorbo števila v binarni sistem .....	20
Slika 25: Programiranje funkcije za način pisanja matematične funkcije.....	21
Slika 26: Primer spremenjene funkcije ob vklopljenem načinu za pisanje matematičnih funkcij .....	21
Slika 27: Funkcija za limitiranje.....	22
Slika 28: Funkcija za odvajanje .....	23
Slika 29: Funkcija za pretvarjanje kotnega zapisa .....	23
Slika 30: Funkciji za iskanje realnih ničel polinoma .....	24
Slika 31: Prenašanje koeficientov v globalno spremenljivko .....	25
Slika 32: Primer izrisa grafa za kvadratno enačbo .....	25
Slika 33: Koda za izris grafa polinoma.....	26
Slika 34: Kreiranje zaslona (napisa) za izpis postopka.....	27
Slika 35: Primer vključevanja simbolov v postopek .....	27
Slika 36: Sledenje spremembam v postopku.....	27
Slika 37: Pretvorba decimalke v števec in imenovalec .....	27
Slika 38: Celotna funkcija za pretvorbo ulomka .....	28
Slika 39: Primer pretvorbe ulomka - pred pretvorbo .....	29
Slika 40: Primer pretvorbe ulomka - po pretvorbi .....	29
Slika 41: Ikona .....	29
Slika 42: Priprava potrebnih datotek.....	30
Slika 43: Izgrajena aplikacija Kalkulator.app .....	30
Slika 44: Priprava datotek .....	31
Slika 45: Izgrajena aplikacija Kalkulator.exe.....	32
Slika 46: Izgled aplikacije v sistemu Windows .....	32
Slika 47: Izgled aplikacije na sistemu Android.....	33



## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

### POVZETEK

Naslov naloge: PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA  
Področje: računalništvo  
Avtor: Marko Zupan  
Šola: Gimnazija Kranj  
Mentorica: mag. Zdenka Vrbinc

V okviru raziskovalne naloge sem v programskem jeziku Python sprogramiral lasten znanstveni kalkulator. Po opravljenih testih sem programsko kodo prevedel v program. Program sem nato s pomočjo modula Pyinstaller pretvoril v zagonljivo aplikacijo za operacijska sistema MacOS in Windows. Za večjo prepoznavnost aplikacije sem oblikoval tudi ikono. Aplikacija je prilagojena tudi za uporabo na pametnem telefonu z operacijskim sistemom Android.

Ključne besede: kalkulator, programiranje, Python, aplikacija, Windows, Android, MacOS, program

### ABSTRACT

Assignment title: PROGRAMMING MY OWN SCIENTIFIC CALCULATOR  
Field: Computer science  
Author: Marko Zupan  
School: Gimnazija Kranj  
Mentor: Msc. Zdenka Vrbinc

In the frame of a research project I programmed my own scientific calculator in Python programming language. After testing the code I compiled it into a executable file. I converted this file into an launchable application for Mac and Windows operating system using a Pyinstaller module. I also designed an icon. Then the app is applied also for the use on Android smart phone.

Keywords: calculator, programming, Python, application, Windows, Android, MacOS, software



Gimnazija Kranj

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

### ZAHVALA

Iskreno se zahvaljujem mentorici gospe Zdenki Vrbinc za nasvete pri izvedbi moje raziskovalne naloge in pomoč pri premagovanju izzivov, s katerimi sem se soočal pri programiranju kalkulatorja.

Zahvaljujem se tudi gospe Bernardi Lenaršič za lekturo raziskovalne naloge.

## 1 UVOD

---

Mladi se že od zgodnjih let srečujemo z informacijsko tehnologijo, saj je postala del našega vsakdana. Spremljajo nas različne tehnološke naprave – od mobilnih telefonov, računalnikov, pametnih ur do kalkulatorjev in drugih, na prvi pogled že zastarelih, a vendarle izredno pomembnih pripomočkov. Na poti izobraževanja nas spremljajo najrazličnejši pripomočki, vendar lahko priznamo, da je vsaj pri naravoslovcih posebno nepogrešljiv en pripomoček, kalkulator. Uporabljamo ga pri matematiki, fiziki, kemiji in še kje drugje. Vemo, da imajo obstoječi znanstveni kalkulatorji že ogromno funkcij, vendar sem kot dijak med učnim procesom določene na šolskem kalkulatorju pogršil. Zato me je zanimalo, ali bi pomanjkljivosti lahko kako odpravil oziroma že obstoječe funkcije nadgradil. Ali sem kot dijak gimnazijskega izobraževalnega programa, ki ne ponuja možnosti učenja programiranja, sposoben sprogramirati svoj kalkulator s funkcijami, ki sem jih pri standardnih kalkulatorjih pogršal? Ali bo moj kalkulator dovolj uporaben, da bo vsaj v določenih primerih zamenjal fizični kalkulator ter mi prihranil nekaj časa in energije pri računanju? Ker sem si želel odgovoriti na ta vprašanja in hkrati olajšati muke pri ročnem računanju, sem se odločil, da poskusim z izdelavo lastnega kalkulatorja.

Za uporabo programskega jezika Python sem se odločil, ker sem imel zelo malo predznanja o programiranju, Python pa velja za lahko učljivi jezik z ogromno spletne podpore uporabniku.

### 1.1 CILJI

---

Moji cilji v tej raziskovalni nalogi so bili:

- sprogramirati lasten kalkulator v programskem jeziku Python
- kalkulator pripraviti za uporabo na operacijskih sistemih Windows in MacOS
- kalkulator pripraviti za uporabo na pametnih mobilnih telefonih Android.

## 2 HIPOTEZE

---

Postavil sem naslednje hipoteze.

- Dijak splošne gimnazije je sposoben samostojno izdelati znanstveni kalkulator in ga pripraviti za uporabo v različnih operacijskih sistemih.
- Kalkulator je primeren za splošno uporabo in olajša probleme, zaradi katerih je bila naloga zastavljena.
- Projekt je možno izpeljati v preprostem programskem jeziku, kot je Python.



## 3 TEORETIČNE PREDPOSTAVKE

---

### 3.1 PYTHON

---

Programski jezik Python je razvil Guido van Rossum leta 1990 na Nizozemskem. Poimenoval ga je po popularni britanski komediji *Monty Python's Flying Circus*. Van Rossum je programski jezik začel razvijati ljubiteljsko. Python je postal popularen programski jezik, ki je zaradi enostavne, skržene in intuitivne sintakse ter obsežne knjižnice pogosto uporabljen v industriji in izobraževanju.

Python je vsesplošno uporaben programski jezik. To pomeni, da se lahko uporablja za pisanje kode za katerokoli nalogo. V Pythonu je napisan tudi del iskalnika Google in transakcijskih procesov na newyorški borzi, prav tako pa ga uporablja tudi NASA.

Python je objektno usmerjen programski jezik. Podatki v njem so objekti, kreirani iz razredov. **Razred** je kategorija, ki definira posamezne vrste objektov s podatki in metodami.

Python je prosto dostopen programski jezik, razvija ga velika skupina prostovoljcev.

Najpogosteje uporabljeni izrazi v Pythonu so:

- **razred**: zaključen del programa, ki vsebuje podatke in metode;
- **objekt**: primer razreda z določenimi lastnostmi;
- **metoda**: funkcijski podprogram, ki pripada določenemu razredu in določa njegovo obnašanje;
- **funkcija**: metoda, ki vrne neko vrednost;
- **parameter**: vrednost, uporabljena v metodi ali funkciji.

(Liang, 2013)

---

#### 3.1.1 Moduli

---

Moduli so daljši programi, napisani v določenem programskem jeziku, ki so zaradi velikosti ločeni od osnovnega programa. V osnovni program jih kličemo z ukazom:

```
import ime_modula
```

(Gerrard, 2016)

---

#### 3.1.2 Tkinter

---

Tkinter je programska oprema znotraj programskega jezika Python. Gre za tanek, objektno usmerjen modul na vrhu paketa Tcl/Tk.

Tkinter vključuje orodja za programiranje grafičnega uporabniškega vmesnika (*ang. GuiProgramming*).

(EINinja, 2019)

---

### 3.1.3 Math

---

Math je programski modul, namenjen za programski jezik Python. Vključuje funkcije za kompleksne matematične funkcije, ki jih ni mogoče zapisati z enostavnimi operatorji.

---

### 3.1.4 Matplotlib

---

Matplotlib je programski modul, ki ga lahko uporabljamo v Pythonovi programski kodi, v lupinah Ipython, beležnici Jupyter, na spletnih aplikacijskih strežnikih in v orodjih uporabniškega vmesnika. Namenjen je risanju grafov v 2D-prostoru.

Z njim lahko rišemo histograme, spektre, stolpčne diagrame, diagrame napak itd.

(Matplotlib, 2020)

---

### 3.1.5 SymPy

---

SymPy je matematični programski modul za uporabo izključno v programskem jeziku Python. Uporablja se za operiranje z izrazi abstraktne matematike.

(SymPy, 2020)

---

### 3.1.6 NumPy

---

NumPy je matematični programski modul za uporabo izključno v programskem jeziku Python. Je temeljni paket za znanstvene operacije v Pythonu.

Vključuje:

- N-dimenzionalno objektno usmeritev;
- napredne matematične funkcije;
- orodja za integracijo C, C++ in Fortran programske kode;
- uporabno linearno algebro.

(NumPy, 2020)

---

### 3.1.7 PyInstaller

---

PyInstaller je programski modul, ki omogoča distribucijo Pythonovih programov in skriptov. PyInstaller deluje na vseh računalniških platformah ter uporablja dinamično nalaganje modulov in vtičnikov, kar zagotavlja popolno združljivost z vsemi moduli.

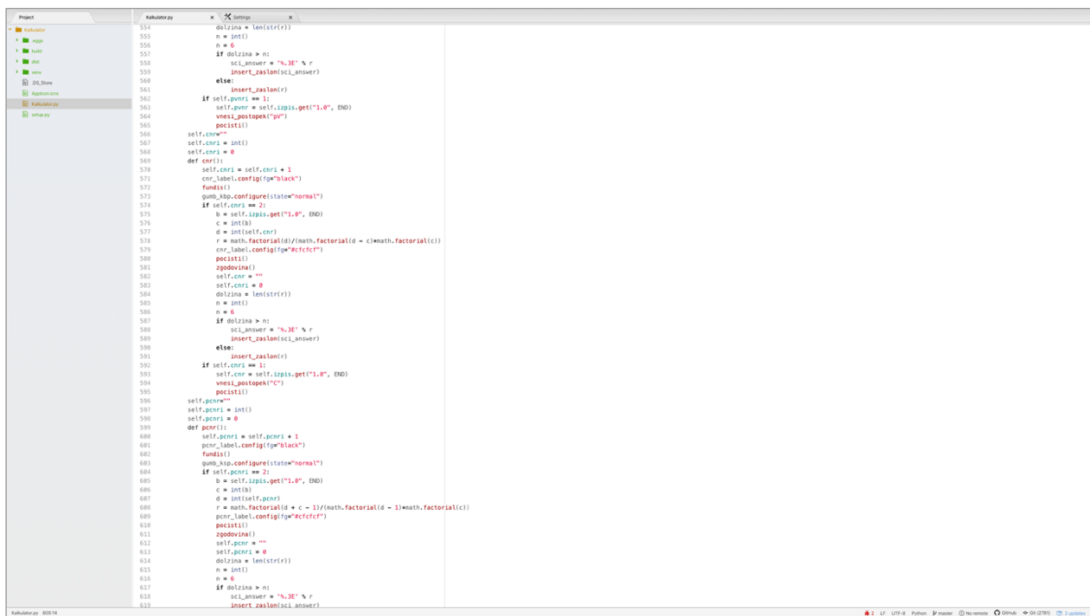
(PyInstaller, 2020)

### 3.2 ATOM

Atom je eden izmed integriranih razvojnih orodij (IDE – Integrated Development Environment). Programerjem pomaga pri razvoju programov in njihove programske kode. Razvojna orodja običajno vsebujejo urejevalnik izvorne kode, prevajalnik (*ang. compiler*) oziroma tolmač (*ang. interpreter*), orodje za avtomatizacijo izgradnje programa in razhroščevalnik. Danes večina orodij vsebuje tudi brskalnik razredov (*ang. class browser*), inšpektor objektov in diagram hierarhije programa. Tovrstna razvojna orodja podpirajo tudi programiranje v različnih programskih jezikih (Python, Java, CSS, C#...). Atom vse te funkcije podpira.

Posamezna integrirana razvojna orodja so prilagojena različnim ravnom izkušnosti uporabnika. Nekatera so namenjena začetnikom (npr. BlueJ, DrJava); ta niso zahtevna za uporabo, a temu primerno nudijo manj funkcij in profesionalnih orodij. Poznamo tudi naprednejša orodja, ki so namenjena za uporabo profesionalcem ter nudijo več funkcij in orodij, a so temu primerno težja za uporabo. Integrirana razvojna orodja so lahko prosto dostopna ali plačljiva. Atom je prosto dostopno orodje, zato sem ga izbral za urejanje svoje programske kode.

(Programski jeziki in orodja - IDE, 2019)



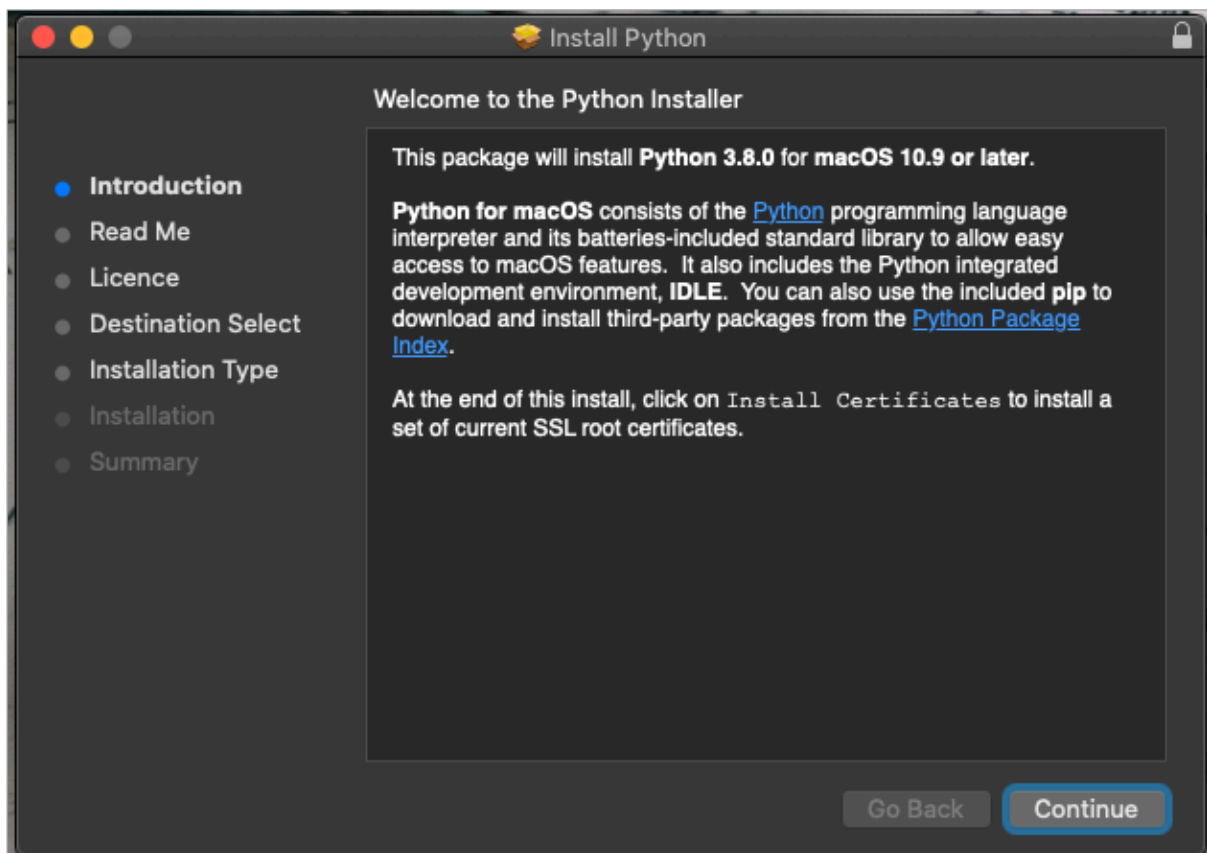
Slika 1: IDE Atom

## 4 EMPIRIČNI DEL

## 4.1 NAMESTITEV PROGRAMSKEGA JEZIKA PYTHON

Predem sem začel s programiranjem znanstvenega kalkulatorja, sem na računalnik namestil programski jezik. S spletne strani sem prenesel namestitveni paket, ki je na računalnik namestil knjižnice, tolmača in prevajalca programskega jezika Python. To so predispozicije, ki jih integrirano razvojno orodje potrebuje za prepoznavo in zagon programske kode.

Ker je Python prosto dostopen programski jezik, sem namestitveni paket brezplačno prenesel s spletne strani <https://python.org>.



Slika 2: Namestitev Pythona

Za pisanje programske kode bi lahko uporabil razvojno orodje IDLE, ki je del namestitvenega paketa programskega jezika Python, vendar je slednje precej manj preprosto za uporabo. Zato sem v naslednjem koraku namestil integrirano razvojno orodje Atom, v katerem sem v nadaljevanju pisal svojo programsko kodo.

## 4.2 NAMESTITEV INTEGRIRANEGA RAZVOJNEGA ORODJA ATOM

---

V primerjavi s prednaloženim orodjem za urejanje izvorne kode IDLE ima razvojno okolje Atom kar nekaj prednosti, zaradi katerih sem ga izbral. Atom priročno obarva različne funkcije, metode in razrede, kar pri veliki količini izvorne kode izboljša preglednost. Prav tako lahko funkcije zapremo, da zavzamejo manj prostora, kar dodatno izboljša preglednost programske kode. Atom ima vgrajeno tudi podporo za git, kar pomeni, da lahko shranjujemo različne verzije programske kode, kasneje do nje dostopamo in pregledujemo spremembe v programski kodi. Te prednosti so me prepričale v izbor orodja Atom za programiranje kalkulatorja.

Atom je prosto dostopno orodje, zato sem ga brezplačno prenesel s spletne strani <https://atom.io>.

## 4.3 PROGRAMIRANJE KALKULATORJA

### 4.3.1 Izdelava okna

Na začetku sem uvozil knjižnico Tkinter, pri čemer je iz spodnje zaslonske slike razvidno, da sem najprej preveril, katero različico Pythona računalnik uporablja, in nato uvozil primerno knjižnico. Danes se namreč uporabljata dve različici programskega jezika Python, in sicer

- Python 2 in
- Python 3.

Ker različici programskega jezika Python (Python 2 in Python 3) uporabljata različno poimenovanje za knjižnico Tkinter, sem moral uvoziti obe, saj v času razvoja ne vem, katero od obeh različic Pythona bo uporabljal končni uporabnik.

```
# -*- coding: utf-8 -*-
import sys
if sys.version_info < (3, 0):
    #Python2
    from Tkinter import *
else:
    #Python3
    from tkinter import *
import matplotlib
```

Slika 3: Uvozimo Tkinter

Potem ko sem uvozil Tkinter, sem lahko začel z ustvarjanjem zaslonskega okna.

```
class Window(Frame):

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.init_window()

    #Definiramo, kreiramo okno
    def init_window(self):

        #Definiranje naslova okna
        self.master.title("Znanstveni kalkulator")

        #Dovoli da se zasede celotno root okno
        self.pack(fill=BOTH, expand=1)

        #Difiniramo frame za zaslon, da se ob spreminjanju velikosti pisave ne spremeni velikost okna
        zaslon_okno = Frame(self, width=330, height=200)
        zaslon_okno.grid(row=1, column=1)
        zaslon_okno.place(x = -4, y = -4)
        zaslon_okno.pack()
        zaslon_okno.grid_propagate(False)
```

```
root = Tk()

#size of the window
root.geometry("330x520")
root.resizable(0,0)

app = Window(root)
root.mainloop()
```

Slika 4: Kreiranje zaslonskega okna

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

Dobil sem prazno zaslonsko okno. Tako sem lahko nadaljeval z ustvarjanjem drugih gradnikov kalkulatorja.

### 4.3.1.1 Vnosni zaslon

Najprej sem na kalkulatorju potreboval polje, v katero sem kasneje izpisoval številke in znake za operacije. Za to sem uporabil modul *Text* iz knjižnice Tkinter in ga umestil na vrh zaslonskega okna.

```
#Zaslon
velika_pisava = ('Arial',50)
mala_pisava = ("Arial", 25)
self.izpis = Text(zaslon_okno, state='disabled',height = 2, width = 12, padx=8, background="#cfcfcf", foreground="#575757")
self.izpis.place(x= -4, y=-4)
self.izpis.config(font=velika_pisava, state="disabled")
```

Slika 5: Ustvarjanje vnosnega zaslona

### 4.3.1.2 Gumbi za vnos števil in gumbi za osnovne operacije

Gumbe za vnos števil od 1 do 9, vnos decimalne vejice, gumbe za osnovne računske operacije, izračun izraza, zamenjavo predznaka trenutno izpisanega števila in operiranje s spominskimi vrednostmi sem uvedel s pomočjo modula *Button* iz knjižnice Tkinter. Gumbe sem postavil na dno zaslonskega okna.

```
#Prva vrstica gumbov
gumb_sedem = Button(self, text="7", command = lambda: vnesi("7"))
gumb_sedem.place(x=15, y=visina)
gumb_sedem.config(height=1, width=5)

gumb_osem = Button(self, text="8", command = lambda: vnesi("8"))
gumb_osem.place(x=75, y=visina)
gumb_osem.config(height=1, width=5)

gumb_devet = Button(self, text="9", command = lambda: vnesi("9"))
gumb_devet.place(x=135, y=visina)
gumb_devet.config(height=1, width=5)

gumb_deljenje = Button(self, text="+", command = lambda: deljenje())
gumb_deljenje.place(x=195, y=visina)
gumb_deljenje.config(height=1, width=5)

gumb_vspomin = Button(self, text="x->M",fg="#638ccf", command = lambda: xtmem())
gumb_vspomin.place(x=255, y=visina)
gumb_vspomin.config(height=1, width=6)
```

Slika 6: Primer vrstice osnovnih vnosnih gumbov

### 4.3.1.3 Gumbi za izvajanje znanstvenih operacij

Sledila je postavitve gumbov, ki kličejo znanstvene funkcije. Ker je gumbov veliko, sem se odločil, da jih naredim manjše velikosti. Tudi za ustvarjanje teh gumbov sem uporabil modul *Button*, gumbe pa sem postavil med vnosni zaslon in prej narejene gumbe.

```
#Fukncijske tipke, prva vrsta
gumb_sin = Button(self, text="sin", fg="#d4a46a", command = lambda: sin(prikaz_int))
gumb_sin.place(x=15, y=vfun)
gumb_sin.config(height=1, width=4)

gumb_cos = Button(self, text="cos", fg="#d4a46a", command = lambda: cos(prikaz_int))
gumb_cos.place(x=65, y=vfun)
gumb_cos.config(height=1, width=4)

gumb_tan = Button(self, text="tan", fg="#d4a46a", command = lambda: tan(prikaz_int))
gumb_tan.place(x=115, y=vfun)
gumb_tan.config(height=1, width=4)

gumb_cot = Button(self, text="cot", fg="#d4a46a", command = lambda: cot(prikaz_int))
gumb_cot.place(x=165, y=vfun)
gumb_cot.config(height=1, width=4)

gumb_ce = Button(self, text="↵", fg="black")
gumb_ce.place(x=215, y=vfun)
gumb_ce.config(height=1, width=4, command=lambda: pobrisi_eno_mesto())

gumb_ce = Button(self, text="C", fg="black")
gumb_ce.place(x=265, y=vfun)
gumb_ce.config(height=1, width=4, command=lambda: pocisti_vse())
```

Slika 7: Primer vrstice funkcijskih tipk

### 4.3.1.4 Napisi za aktivne funkcije kalkulatorja

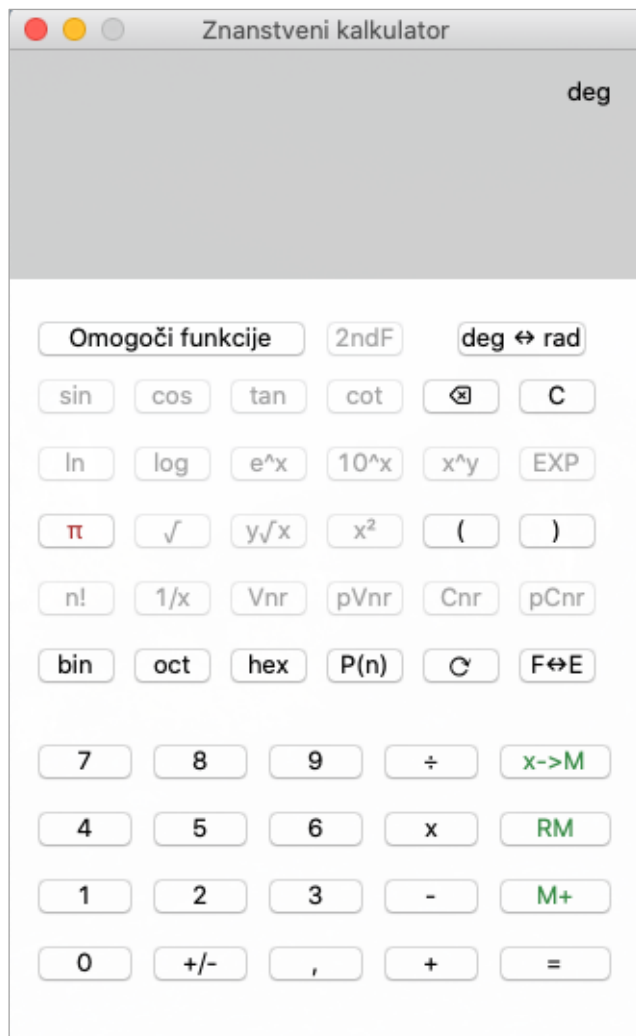
Napise, ki se ob aktivaciji določene funkcije pokažejo na zaslonu in označijo aktivnost te funkcije, sem sprogramiral z uporabo modula *Label* iz knjižnice Tkinter. Ko na primer izberemo funkcijo logaritem, ki potrebuje vnos dveh podatkov, se na zaslonu pojavi napis, ki označuje aktivno funkcijo logaritem. Napisom sem določil primarno barvo besedila tako, da se ta zlije z barvo vnosnega zaslona, kar skriva neaktivne napise.

```
gumb_cos = Button(self, text="cos", fg="#d4a46a", command = lambda: cos(prikaz_int))
gumb_cos.place(x=65, y=vfun)
gumb_cos.config(height=1, width=4)
```

Slika 8: Ustvarjanje napisa za funkcijo kosinus



## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA



Slika 9: Grafična podoba kalkulatorja po dodajanju gradnikov

---

### 4.3.2 Nastavljanje osnovnih parametrov

---

Pred programiranjem funkcij sem moral določiti vse globalne spremenljivke in sezname, v katere sem potem med uporabo kalkulatorja shranjeval vrednosti. Te so bile seznam vseh vnesenih števil v trenutni seji, zgodovina postopka, trenutna številka na zaslonu ipd.

```
self.zgodovina = ""
self.postopek = list()
self.disk = list()
self.koncni_rezultat = ""
self.sprotno = ""
```

Slika 10: Nastavljanje osnovnih parametrov

---

### 4.3.3 Programiranje vnosa znakov na zaslon

---

Po postavitvi vseh potrebnih gradnikov za kalkulator sem sprogramiral funkcijo, ki na vnosni zaslon vstavlja znake, določene v lastnosti funkcije (na zaslonski sliki vidno kot `value`).

```
def vnosi(value):
    self.postopek.append(value)
    insert_zaslon(value)
```

Slika 11: Funkcija za vnos na vnosni zaslon

Ustvaril sem tudi posebno funkcijo, ki znake vstavi samo na poseben seznam `self.postopek`.

```
def vnosi_postopek(value):
    self.postopek.append(value)
```

Slika 12: Funkcija za vnos na seznam postopek

---

### 4.3.4 Programiranje vnašanja znakov s pomočjo tipkovnice

---

Da uporabniku ni treba vseh operacij nadzirati samo z uporabo tabulatorja, sem dodal kodo, ki za pritiske določenih tipk na tipkovnici vnese določene znake v kalkulator (osnovne aritmetične operacije, uporaba oklepajev, izračun ob pritisku tipke enter, brisanje vnosa ipd.).

```
self.master.bind('1', lambda event: vnosi("1"))
```

Slika 13: Primer kode, ki ob pritisku tipke 1 na tipkovnici vnese število 1 v kalkulator

---

### 4.3.5 Programiranje brisanja znakov iz zaslona

---

Za brisanje celotnega seznama, odgovornega za računanje izrazov, ponastavitev (*ang. resetting*) kalkulatorja na začetno stanje in brisanje posameznih znakov na vnosnem zaslonu sem sprogramiral funkciji *pocisti\_vse()* in *pobrisi\_eno\_mesto()*.

```
def pobrisi_eno_mesto():
    izpis = self.izpis.get("1.0", END)
    l = len(izpis) - 1
    if l > 0: #Nic stevilck
        self.izpis.configure(state="normal")
        self.izpis.delete("end-2c")
        self.izpis.configure(state="disabled")
        del self.disk[-1]
        del self.postopek[-1]
        postopek_trace()

#Brisanje vsega - tukaj zaradi inita
def pocisti_vse():
    del self.disk[:]
    del self.postopek[:]
    resetfun()
    self.izpis.configure(state='normal')
    self.izpis.delete('1.0', END)
    self.racun=""
    self.koncni_rezultat=""
    self.sprotno=""
    self.postopek_trace=""
    postopek_trace()
    # Nastavi nastavitve ne privzeto
    self.izpis.config(font=velika_pisava, state="disabled", pady=2, height=2, width=12)
    self.fe = 0
    self.racuni = 0
    self.oklepaji = 0
    self.zaklepaji = 0
    oklepaj_label.config(fg="#cfcfcf")
    self.izpis.configure(state="disabled")
```

Slika 14: Funkciji za brisanje

---

### 4.3.6 Funkcija za izračun računa v kalkulatorju

---

Za preprosto računanje sem sprogramiral posebno funkcijo, ki lahko računa z osnovnimi računskimi operacijami. Za zahtevnejše operacije sem funkcije spisal posebej.

Ta funkcija uporablja ukaz *eval*, ki izračuna rezultat na podlagi niza številck in znakov v nizu.

Ob uspešnem rezultatu se kalkulator ponastavi (*ang. reset*), tako da je pripravljen na nadaljnje računanje.

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

Kodi sem dodal še odziv v primeru napak, pri čemer kalkulator namesto javljanja napake v lupino to javi na zaslon z izpisom besede »Err«.

```
def rezultat():
    try:

        izpis_postopka = ("".join(str(x) for x in self.postopek))
        print(izpis_postopka)
        #List pretvori v condensed string
        izpis_rezultata = ("".join(str(x) for x in self.disk))
        answer = str("%g" % (eval(izpis_rezultata)))
        #Preveri kako dolg je rezultat in ga pretvori ce je daljsi od n znakov
        dolzina = len(answer)
        n = int()
        n = 10
        if dolzina > n and (float(answer) >= 1000 or float(answer) <= 0.00000001):
            ans = float(answer)
            sci_answer = '%.3E' % ans
            pocisti_zaslon()
            insert_no_count(sci_answer, newline=True)
            self.izpis.configure(state="disabled")
            self.koncni_rezultat=sci_answer
        elif dolzina > n and (float(answer) < 1000 or float(answer) > 0.00000001):
            pocisti_zaslon()
            ans = float(answer)
            insert_no_count('{:.{}f}'.format(ans, 8 ))
            self.koncni_rezultat='{:.{}f}'.format(ans, 8 )
        else:
            pocisti_zaslon()
            insert_no_count(answer, newline=True)
            self.izpis.configure(state="disabled")
            self.koncni_rezultat=answer

        del self.postopek[:]
        self.postopek.append(answer)

        del self.disk[:]
        self.disk.append(answer)

    except TypeError:
        pocisti_vse()
        insert_zaslon("Err")
        del self.disk[:]
    except NameError:
        pocisti_vse()
        insert_zaslon("Err")
        del self.disk[:]
    except ValueError:
        pocisti_vse()
        insert_zaslon("Err")
        del self.disk[:]
```

Slika 15: Funkcija za računanje računa

### 4.3.7 Programiranje znanstvenih funkcij

#### 4.3.7.1 Omogočanje funkcij in shranjevanje rezultata

Po funkciji za računanje sem začel s programiranjem funkcij za znanstvene operacije. Ker večina teh operacij zahteva vpis več različnih števil (indeksi, stopnje ...), sem predvidel zahtevo, da uporabnik najprej klikne gumb za omogočanje funkcij. Ob tem se trenutni niz iz *self.disk* shrani v drugo spremenljivko, da je računanje znanstvene operacije povsem neodvisno od prejšnjega postopka. Ko izračunamo rezultat operacije, se ta v program vrne kot preprosta številka in se doda v *self.disk* hkrati s tem, da se vanj doda rezultat, ki smo ga prej shranili v novo spremenljivko.

```

self.funi = int()
self.funi = 0
def fun():
    self.funi = self.funi + 1
    if self.funi == 1:
        zaslon = self.izpis.get("1.0", END)
        l = len(zaslon) - 1
        if l != 0:
            del self.disk[-l:]
            del self.postopek[-l:]
            postopek_trace()
        pocisti_zaslon()
        trenutno = ("".join(str(x) for x in self.disk))
        self.zgodovina = trenutno
        fun_label.configure(fg="black")
        fun_button.configure(relief=RAISED, text="Omogoči funkcije")
        fun2_button.config(state="normal")
        opdis()
        funrm()

    else:
        self.zgodovina = ""
        self.funi = 0
        fun_label.configure(fg="#cfcfcf")
        fun_button.configure(relief=FLAT, text="Omogoči funkcije")
        fun2_button.config(state="disabled")
        opnrm()
        fundis()

```

Slika 16: Omogočanje funkcij in shranjevanje rezultata

Nato sem sprogramiral še, da se funkcijski gumbi ob kliku spremenijo iz neaktivnega (*ang. disabled*) stanje v aktivnega (*ang. enabled*), tako da operacije sploh lahko uporabljamo.

```

def funrm():
    #Omogoci funkcijske tipke med aktivno funkcio
    gumb_fakulteta.configure(state="normal")

```

Slika 17: Izsek iz funkcije za omogočanje funkcijskih gumbov

### 4.3.7.2 Programiranje znanstvenih operacij

Sledilo je programiranje znanstvenih operacij. Kadar je za operacijo potrebno vpisati dva podatka, sem problem rešil tako, da uporabnik najprej omogoči funkcije, potem vnese število in pritisne na funkcijsko tipko, s katero potrdi želeno operacijo. Število na zaslonu funkcija nato shrani kot lokalno spremenljivko. Ob izbiri funkcije se zaslon počisti in uporabnik vnese novo število ter ponovno klikne na funkcijsko tipko. Funkcija nato uporabi število na zaslonu in število, shranjeno v lokalni spremenljivki za izračun rezultata, to število pa nato vnese na prej počiščeni zaslon.

```

self.xnay=""
self.xnayi = int()
self.xnayi = 0
def xnay():
    self.xnayi = self.xnayi + 1
    xnay_label.config(fg="black")
    fundis()
    gumb_pow.configure(state="normal")
    if self.xnayi == 2:
        b = self.izpis.get("1.0", END)
        c = int(b)
        d = int(self.xnay)
        r = d**c
        xnay_label.config(fg="#cfcfcf")
        pocisti()
        zgodovina()
        self.xnay = ""
        self.xnayi = 0
        dolzina = len(str(r))
        n = int()
        n = 6
        if dolzina > n:
            sci_answer = '%.3E' % float(r)
            insert_zaslon(sci_answer)
        else:
            insert_zaslon(r)
    if self.xnayi == 1:
        self.xnay = self.izpis.get("1.0", END)
        vnese_postopek("")
        pocisti()
  
```

Slika 18: Primer funkcije, ki zahteva dvojni vnos

### 4.3.7.3 Programiranje dvojnih funkcij

Zaradi pomanjkanja prostora za funkcijske tipke na kalkulatorju sem podobne funkcije združil pod isto funkcijsko tipko, ki aktivira različni operaciji ob pritisku na gumb za aktivacijo sekundarnih funkcij (tipka »2ndF«).

To sem naredil tako, da sem ustvaril števec *self.fun2i*, ki ob vrednosti 1 izvede drugo funkcijo kot ob vrednosti 0.

```
def fakulteta(b):
    a = self.izpis.get("1.0", END)
    f = float(a)
    b = int(a)
    if self.fun2i == 0:
        r = math.factorial(b)
        vnesi_postopek("!")
    else:
        r = ((1+b)*b)/2
        vnesi_postopek("S")

    self.izpis.configure(state="normal")
    d = len(a)
    pocisti_zaslون()
    zgodovina()

    dolzina = len(str(r))
    n = int()
    n = 6
    if dolzina > n:
        sci_answer = '%.3E' % r
        insert_zaslون(sci_answer)
    else:
        insert_zaslون(r)
```

Slika 19: Primer dvojne funkcije

#### 4.3.7.4 Programiranje kotnih funkcij

Pri kotnih funkcijah sem moral sprogramirati funkcije za vrednost kota v radianih in stopinjah. Zato sem najprej dodal gumb za menjavo vrste kota in nastavil števec, ki opisuje stanje gumba.

```
#Izbira radiani oziroma stopinje
gumb_kot = Button(self, text="deg ↔ rad", command = lambda: izbira_kota())
gumb_kot.place(x=233, y=140)
gumb_kot.config(height="1", width="7")
label_kot = Label(self, text="deg", bg="#cfcfcf", fg="black")
label_kot.place(x=285, y = 10)
self.kot = int()
self.kot = 0
def izbira_kota():
    self.kot = self.kot + 1

#Izberemo radiane
if self.kot % 2 == 1:
    gumb_kot.config(text="rad ↔ deg")
    label_kot.config(text="rad")
#Izberemo stopinje
elif self.kot % 2 == 0:
    gumb_kot.config(text="deg ↔ rad")
    label_kot.config(text="deg")
else:
    print("Error")
```

Slika 20: Koda za opisovanje izbrane vrste kota

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

Pri programiranju operacij za kotne funkcije sem upošteval, da gre za dve funkciji (združena kotna in krožna funkcija) in da na to vpliva še izbrana vrsta kota (radiani oziroma kotne stopinje). Zato sem moral sprogramirati štiri različne možnosti računanja kotne funkcije.

```
def sin(b):
    a = self.izpis.get("1.0", END)
    b = float(a)
    if (self.kot % 2 == 0) or (self.kot == 0):
        if self.fun2i == 1:
            r = math.degrees(math.asin(b))
            vnesi_postopek("°asin")
        else:
            r = math.sin(math.radians(b))
            vnesi_postopek("°sin")
    if self.kot == 1:
        if self.fun2i == 1:
            r = math.asin(b)
            vnesi_postopek(" asin")
        else:
            r = math.sin(b)
            vnesi_postopek(" sin")
    self.izpis.configure(state="normal")
    pocisti_zaslون()
    zgodovina()
    insert_zaslون('{:.{}f}'.format(r, 8 ))
```

Slika 21: Primer kotne funkcije sinusa

---

### 4.3.8 Programiranje znanstvenega zapisa rezultata

---

Daljše številke so večkrat nepregledne, zato sem se odločil, da ta problem rešim z znanstvenim zapisom rezultata. Sprogramiral sem tudi števec, ki opisuje stanje gumba za menjavo vrste zapisa.

```
self.fe = 0
def FE():
    self.fe = self.fe + 1
    if self.fe % 2 == 1:
        znanstveni()
        pocisti_zaslون()
        insert_zaslون(self.znanstveni_rezultat)
    else:
        neznanstveni()
        pocisti_zaslون()
        insert_zaslون(self.neznanstveni_rezultat)
```

Slika 22: Števec znanstvenega zapisa

Nato sem števec uporabil v funkcijah za znanstveni zapis.



## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

```
#Pretvori zapis v znanstveni zapis
self.znanstveni_rezultat = ""
def znanstveni():
    trenutno = float(self.koncni_rezultat)
    self.znanstveni_rezultat = '%.3E' % trenutno

#Pretvori nazaj v neznanstvenega
self.neznanstveni_rezultat = ""
def neznanstveni():
    trenutno = float(self.koncni_rezultat)
    self.neznanstveni_rezultat = float(trenutno)
```

Slika 23: Funkciji za znanstveni in neznanstveni zapis

### 4.3.9 Programiranje funkcij za pretvorbo števil v druge številске sisteme

V kalkulator sem vgradil tudi pretvornik v binarni, šestnajstiški in osmiški številski sistem. Iz spodnje slike so razvidni koraki programiranja teh funkcij na primeru binarnega številskega sistema.

```
def fbin():
    self.bini+=1
    if self.bini==1:
        self.bin = ""
        trenutno = int(self.izpis.get("1.0", END))
        self.bin = bin(trenutno).replace("0b", "")
        pocisti()
        insert_zaslona(self.bin)
        del self.disk[:]
    else:
        self.bini=0
        trenutno=self.izpis.get("1.0", END)
        pocisti()
        insert_zaslona(int(trenutno,2))
        del self.disk[:]
```

Slika 24: Funkcija za pretvorbo števila v binarni sistem

### 4.3.10 Programiranje abstraktnih znanstvenih funkcij

Preden sem lahko začel s programiranjem teh funkcij, sem moral v kalkulator vpeljati nov način (*ang. mode*), da lahko v njem uporabnik zapiše izraz, s katerim želi operirati.

#### 4.3.10.1 Programiranje načina za urejanje matematičnih funkcij

Način sem sprogramiral tako, da sem ustvaril nov seznam, na katerega se dodajajo vpisani znaki na zaslonu, kadar je vklopljen način *editing\_x*. Nekatere funkcije sem moral nato na novo sprogramirati na logični pogoj za drugačno obnašanje ob vklopljenem načinu.

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

```

#OKOLJE ZA UREJANJE FUNKCIJE
self.editing_x=False
self.funkcija = list()
def editing_x():
    #Omogocimmo okolje za urejanje funkcij
    self.editing_x=True
    del self.funkcija[:]
    #Onemogocimo dolocene Funkcije
    gumb_fakulteta.configure(state="disabled")
    gumb_vsp.configure(state="disabled")
    gumb_kbp.configure(state="disabled")
    gumb_ksp.configure(state="disabled")
    gumb_10x.configure(state="disabled")
    gumb_EXP.configure(state="disabled")
    gumb_obrni.configure(state="disabled")

    gumb_mnozenje.configure(state="normal")
    gumb_deljenje.configure(state="normal")
    gumb_sestevanje.configure(state="normal")
    gumb_odstevanje.configure(state="normal")

    #Prikazemo opozorilo_funkcija
    opozorilo_funkcija.config(text="Urejanje funkcije", background="#ab3030")
  
```

Slika 25: Programiranje funkcije za način pisanja matematične funkcije

```

if self.editing_x==True:
    if self.fun2i==1:
        vnesi("asin")
    else:
        vnesi("sin")
else: =
  
```

Slika 26: Primer spremenjene funkcije ob vklopljenem načinu za pisanje matematičnih funkcij

### 4.3.10.2 Programiranje funkcije za limitiranje

Prva abstraktna matematična operacija, ki sem jo dodal v kalkulator, je bila limitiranje. Ko sem sprogramiral način za pisanje matematičnih funkcij, sem za zapis v seznamu, na katerega so se elementi dodajali med omogočenim stanjem prej omenjene funkcije, uvedel novo pravilo, ki je simbole, kot sta  $n$  in  $x$ , prepoznal kot dele matematičnega izraza.

Limitiranje poteka v več korakih. Najprej nas kalkulator vpraša po številu, kateremu se bo  $n$  bližal, nato omogoči način za pisanje matematičnih funkcij. V naslednjem koraku funkcijo zapišemo in ob tretjem pritisku funkcijske tipke program vrne rešitev za izraz.

```

self.limitai=self.limitai +1
meja=str()
if self.limitai == 3:
    simbol = sp.symbols("n")
    izraz = ("").join(str(x) for x in self.funkcija)
    postopek = ("").join(str(x) for x in self.postopek)
    inf = float("inf")
    try:
        r= sp.limit(izraz,simbol,float(self.meja))
        if str(r)=="oo":
            r="∞"
    except SympifyError:
        pocisti_vse()
        insert_zaslon("Err")
        vnesi_postopek("Napaka pri zapisu funkcije")
    except RuntimeError:
        r= sp.limit(izraz,simbol,float(self.meja))
    if "Inf" in self.meja:
        meja="n→ ∞"
    else:
        meja = "n→ " +self.meja

    pocisti_vse()
    zaslon_srednji()
    if len(r)>10:
        insert_zaslon("lim = "+ ('{:.{}f}'.format(r, 10)))
    else:
        insert_zaslon("lim = "+ str(r))
    vnesi_postopek("lim(" +postopek+")")
    limita.place(x=9, y=25)
    limita.config(text= meja,foreground="#575757", anchor=W)
    #vnesi_postopek("x -> ∞")
    fun2()
    fun()
    fundis()
    fun_button.config(text="Omogoči funkcije")
if self.limitai == 2:
    self.meja=self.izpis.get("1.0", END)
    postopek_pobrisi_trenutno()
    pocisti()
    editing_x()
    limita.place(x=9, y=30)
    if "Inf" in self.meja:
    else:
        limita.config(text=meja,foreground="#575757", anchor=W)
if self.limitai == 1:
    limita.place(x=9, y=30)
    limita.config(text="n-",foreground="#575757", anchor=W)
    zaslon_srednji()

```

Slika 27: Funkcija za limitiranje

#### 4.3.10.3 Programiranje funkcije za računanje odvoda

Za računanje oz. iskanje odvoda poljubne funkcije sem uporabil način za urejanje matematičnih funkcij in seznam *self.funkcija*, da sem sestavil matematični izraz, ki ga funkcija *diff* iz knjižnice *sympy* uporabi za izračun odvoda.

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

Pred tem sem  $x$  navedel kot možni simbol pojavljanja v izrazu. Če bi tako želel imeti funkcijo z več spremenljivkami (npr.  $n$  in  $m$ ), bi moral te prej določiti v programu, v nasprotnem primeru bi nam program javil napako (izpis »Err«).

```

else:
    if self.fun2i==0:
        xky_label.config(fg="black")
        fundis()
        gumb_pkoren.configure(state="normal")
        if self.xkyi == 2:
            if self.xkyi == 1:
                self.xky = self.izpis.get("1.0", END)
                pocisti()
                postopek_pobrisi_trenutno()
            else: #ODVOD
                if self.xkyi==1:
                    editing_x()
                    pocisti()
                    postopek_pobrisi_trenutno()

                if self.xkyi==2:
                    simbol = sp.symbols('x')
                    izraz = ("".join(str(x) for x in self.funkcija))
                    postopek = ("".join(str(x) for x in self.postopek))
                    r = sp.diff(izraz, simbol)
                    pocisti_vse()
                    zaslon_srednji()
                    insert_zaslon(str(r))
                    vnesi_postopek("(" + postopek + ")")
  
```

Slika 28: Funkcija za odvajanje

### 4.3.11 Programiranje funkcije za pretvarjanje kotnega zapisa

V kalkulator sem dodal še funkcijo, ki nam zapis kota iz stopinj spremeni v zapis s kotnimi minutami, sekundami *itd.* Kot deli po postopku, ki je prikazan na sliki 29. Rezultat izpiše v ustrezni obliki (kontne stopinje oziroma zapis s kotnimi minutami in sekundami).

```

def dms():
    kot = self.izpis.get("1.0", END)
    #Razcleni kot na posamezne dele
    stevilo, decimalke = kot.split(".", 1)
    d=int(float(kot))
    m=int((float(kot)-d)*60)
    s=int((((float(kot)-d-(m/60))*3600))*100)

    pocisti()
    zaslon_srednji()
    insert_no_count(str(d)+"°"+str(m)+"' "+str(s)+"'")
  
```

Slika 29: Funkcija za pretvarjanje kotnega zapisa

### 4.3.12 Reševanje polinomov

Za reševanje polinomov sem napisal funkcijo, ki išče realne in imaginarne ničle polinoma in dodal novo okno, ki polinom izriše na grafu.

#### 4.3.12.1 Programiranje funkcije za iskanje realnih ničel polinoma

Za izpis ničel polinoma sem potreboval ustrezen zapis polinoma s koeficienti za posamezne člene. Sprogramiral sem funkcijo, ki uporabnika najprej vpraša po stopnji polinoma, nato pa uporabnik drugega za drugim vnaša koeficiente tega polinoma. Za izračun ničel sem uporabil modul iz knjižnice *numpy poly1d*. Realne ničle se nato izpišejo na zaslon kalkulatorja.

```
self.polinom = list()
self.polinomi = int()
self.polinomii = int()
def polinom():
    self.polinomi = self.polinomi + 1
    if self.polinomi == 1:
        self.polinomii = int(self.izpis.get("1.0", END)) + 2
        del self.postopek[:]
        pocisti()
    elif self.polinomi > 1:
        trenutno = self.izpis.get("1.0", END)
        self.polinom.append(trenutno)
        global polinom
        polinom.append(float(trenutno))
        #Vnesi v postopek
        if sys.version_info < (3, 0):=
            else:=
        del self.disk[:]
        pocisti()
        postopek_trace()
def polinomje():
    pol = (poly1d(self.polinom)).r

    realni = pol[isreal(pol)]
    r = '\n'.join(str(x) for x in realni)
    del self.disk[:]
    pocisti()
    new_window(Window2)
    try:=
    self.polinomii=0
    self.polinomi=0

    gumb_polinom.config(text="P(n)")
```

Slika 30: Funkciji za iskanje realnih ničel polinoma

### 4.3.12.2 Risanje grafa polinoma

Za izris grafa polinoma sem ustvaril razred za novo okno, ki se odpre po izračunu ničel polinoma.

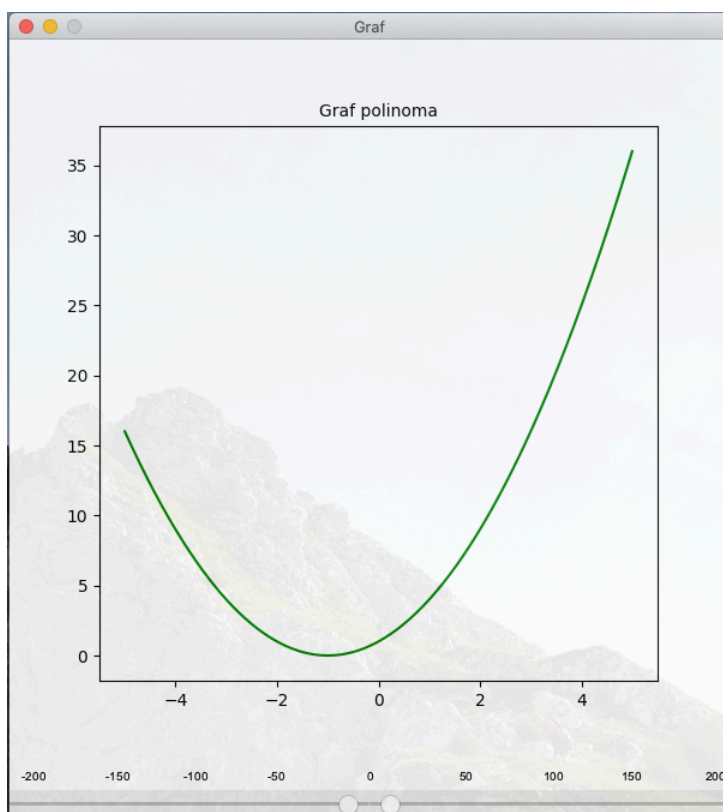
Uporabil sem knjižnico *matplotlib*, ki s pomočjo modula *plot* izriše graf. Ker pa sem koeficiente pridobil v drugem razredu programa, sem moral spisati kodo, ki jih prenese v globalno spremenljivko, do katere lahko dostopamo iz obeh razredov.

```
global polinom  
polinom.append(float(trenutno))
```

Slika 31: Prenašanje koeficientov v globalno spremenljivko

Oknu sem dodal gradnike za določanje zalogo vrednosti grafa. Uporabil sem drsnike z možno zalogo vrednosti  $-200 < x < 200$ .

Dodal sem kodo, ki koeficiente obrne – za risanje grafa so namreč potrebni v ravno obratnem vrstnem redu – in izriše graf glede na vse parametre (zaloga vrednosti, barva črte ...).



Slika 32: Primer izrisa grafa za kvadratno enačbo

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

```

#Definiranje naslova okna
self.master.title("Graf")
self.master.geometry("600x650+1100+400")
self.master.attributes("-alpha", 0.90)
self.master.resizable(0,0)

okno = Frame(self, width=600, height=600)
okno.grid(row=1, column=1)
okno.place(x = 0, y = 0)
okno.pack()
okno.grid_propagate(False)
#Dovoli da se zasede celotno root okno
self.pack(fill=BOTH, expand=1)

#Risanje grafa

def PolyCoefficients(x, coeffs):=

najmanjsa_pisava = ("Arial", 10)

#Definiraj sliderje za vnos obsega na x
labelmin = Label(self, text="-200", font=najmanjsa_pisava)
labelmin.place(x=5, y=605)
labell1 = Label(self, text="-100", font=najmanjsa_pisava)
labell1.place(x=140, y=605)
labell2 = Label(self, text="-150", font=najmanjsa_pisava)
labell2.place(x=75, y=605)
labell3 = Label(self, text="-50", font=najmanjsa_pisava)
labell3.place(x=210, y=605)
labelmax = Label(self, text="200", font=najmanjsa_pisava)
labelmax.place(x=575, y=605)
labeld1 = Label(self, text="50", font=najmanjsa_pisava)
labeld1.place(x=370, y=605)
labeld2 = Label(self, text="100", font=najmanjsa_pisava)
labeld2.place(x=440, y=605)
labeld3 = Label(self, text="150", font=najmanjsa_pisava)
labeld3.place(x=505, y=605)
labelcenter = Label(self, text="0", font=najmanjsa_pisava)
labelcenter.place(x=293, y=605)

slider1 = ttk.Scale(self, from_=(-200), to=0)
slider1.place(x=0, y=625)
slider1.config(value=-5, length=300)
slider2 = ttk.Scale(self, from_=0, to=200)
slider2.place(x=300, y=625)
slider2.config(value=5, length=300)

fig = Figure(figsize=(6,6))
a = fig.add_subplot(111)
x = np.linspace(-5, 5, 1000)
coeffs = polinom[:::-1]
print(polinom)
a.plot(x, PolyCoefficients(x, coeffs), color="green")

```

Slika 33: Koda za izris grafa polinoma

### 4.3.13 Izpis postopka

Za izpis postopka na zaslonu kalkulatorja sem uvedel novo funkcijo. Najprej sem ustvaril napis (*ang. label*) iz modula *Label* in ga postavil pod vnosni zaslon.

```
#Zaslon samo za postopek
self.postopek_trace = ""
self.zaslon_postopek = Label(zaslon_okno, text=self.postopek_trace, background="#cfcfcf", foreground="#575757", anchor=W)
self.zaslon_postopek.place(x=8, y=52)
self.zaslon_postopek.configure(width=80)
```

Slika 34: Ustvarjanje zaslona (napisa) za izpis postopka

Ker sem želel, da se postopek prikazuje sproti, sem spisal funkcijo, ki je odgovorna za sledenje spremembam na seznamu *self.postopek*.

S pomočjo funkcije *vnesi\_postopek()* sem sprogramiral, da znanstvene funkcije v postopek vključujejo svoje simbole (npr.  $3\sqrt{2}$ ,  $4!$  ipd.).

```
vnesi_postopek("²")
```

Slika 35: Primer vključevanja simbolov v postopek

```
def postopek_trace():
    #Sprotno prikazovanje postopka

    for (i, x) in enumerate(self.postopek):
        if x == "√":
            self.postopek[i] = "√"
        if x == "/":
            self.postopek[i] = "÷"
    self.postopek_trace = ("".join(str(x) for x in self.postopek))
    self.zaslon_postopek.configure(text=self.postopek_trace)
```

Slika 36: Sledenje spremembam v postopku

### 4.3.14 Zapis z ulomki

Za lažjo preglednost decimalnih števil sem sprogramiral funkcijo za pretvorbo decimalke v ulomke.

Za to sem uporabil funkcijo *fraction()*. Ta nam decimalno število pretvori v zapis s poševnico (npr.  $4/3$ ). Nato sem spisal kodo, ki nam besedilo s poševnico razbije na dve ločeni spremenljivki z besedilom – prva predstavlja števec in druga imenovalec.

```
r = str((Fraction(n)).limit_denominator(100000))
a,b = r.split("/", 1)
```

Slika 37: Pretvorba decimalke v števec in imenovalec

Uvedel sem kodo, ki števec izpiše v prvo vrsto zaslona, imenovalec pa vnese v drugo vrsto. Med njima nariše črto, katere dolžino dobimo s pomočjo štetja znakov daljše številke v ulomku in množenjem tega števila z dolžinsko enoto ene črke.

Na koncu ulomek sredinsko poravnamo.



## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

```

#Funkcija za pretvarjanje razlicnih zapisov
self.ulomeki = 0
self.predulomkom = float()
def ulomek():
    trenutno = self.izpis.get("1.0", END)
    zaslon_srednji()

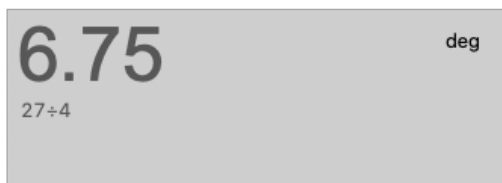
    self.ulomeki = self.ulomeki + 1

    if self.ulomeki % 2 == 1:
        if self.ulomeki == 1:
            pocisti()
            del self.postopek[:]
            n = Decimal(trenutno)
            r = str((Fraction(n)).limit_denominator(100000))
            a,b = r.split("/", 1)
            presledek=" "
            self.izpis.configure(state='normal')

        #Reguliranje poravnave ulomka
        if len(a)>len(b):
            l= len(a)
            razdalja=(len(a)-len(b))/2
            r1=int(razdalja)
            if razdalja-r1 == 0:
                else:
                    print(razdalja)
            elif len(b)>len(a):
                l=len(b)
                razdalja=(len(b)-len(a))/2
                r1=int(razdalja)
                if razdalja-r1 == 0:
                    a = presledek*r1 + a
                else:
                    a = presledek*r1 + " " + a
                print(razdalja)
            else:
                l=len(a)
            self.izpis.insert(END, a)
            self.izpis.insert(END, "\n" + b )
            self.izpis.configure(state='disabled')
            #Za risanje ulomka ustvarimo risalno površino
            crta.config( bg="#575757", width=l*14)
            crta.place(x=8, y=30)
        else:
            r = self.predulomkom
            pocisti()
            insert_no_count(r)
            zaslon_maximize()
            crta.config(bg="#cfcfcf")
            crta.place(x=0, y=0)

```

Slika 38: Celotna funkcija za pretvorbo ulomka



Slika 39: Primer pretvorbe ulomka – pred pretvorbo



Slika 40: Primer pretvorbe ulomka – po pretvorbi

#### 4.4 Testiranje kalkulatorja

---

Končanemu programiranju je sledilo testiranje kalkulatorja. Najprej sem preveril pravilnost izračunov z uporabo osnovnih aritmetičnih operacij in oklepajev. Ko sem potrdil pravilnost izračunov z zadostnim številom primerov, sem nadaljeval s testiranjem preprostejših funkcij, kot so kvadriranje, korenjenje, logaritmiranje ipd. Na koncu sem preveril še najkompleksnejše funkcije, kot so reševanje polinomov in pravilnost izrisov grafov, limitiranje funkcij in odvajanje funkcij. Preveril sem tudi pravilno delovanje zaokroževanja rezultatov in pretvorbe v ulomke iz decimalnega števila. Po končanem testiranju je bil kalkulator pripravljen za distribucijo.

#### 4.5 Priprava programa za distribucijo

---

Po končanem programiranju in uspešnem testiranju kalkulatorja sem se začel ukvarjati z distribucijo programa. Želel sem narediti aplikacijo, ki jo bo mogoče uporabljati na vseh operacijskih sistemih.

Na trgu obstaja velika ponudba aplikacij in vtičnikov, ki nam pomagajo »zamrzniti« (ang. freeze) aplikacijo, torej vključiti interpreter Python in vse datoteke, ki jih program potrebuje za izvajanje. Za razliko od ostalih je PyInstaller celovito orodje, ki deluje na vseh operacijskih sistemih.

Najprej sem kodo prilagodil posameznim operacijskim sistemom, saj je bil zaradi različne interpretacije dolžinskih enot uporabniški vmesnik drugačen kot na platformi, na kateri sem kalkulator razvijal (MacOS).

V brezplačnem programu GIMP sem izdelal ikono za svoj program.



Slika 41: Ikona

### 4.5.1 Namestitev PyInstallerja

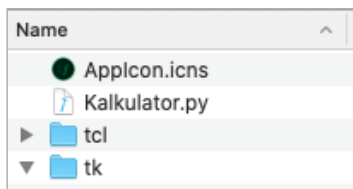
PyInstaller sem namestil z ukazom v terminalu.

```
$ py3 -m pip install pyinstaller
```

Potrebno je upoštevati, da distribucijo opravimo na tistem operacijskem sistemu, za katerega aplikacijo pripravljamo, tako npr. za MacOS aplikacijo naredimo samo na MacOS. To pomeni, da sem ta ukaz moral uporabiti še v okolju Windows.

### 4.5.2 Priprava aplikacije za platformo MacOS

Na namizju sem ustvaril novo mapo z imenom aplikacije in vanjo prenesel vse datoteke, potrebne za izgradnjo aplikacije.



Slika 42: Priprava potrebnih datotek

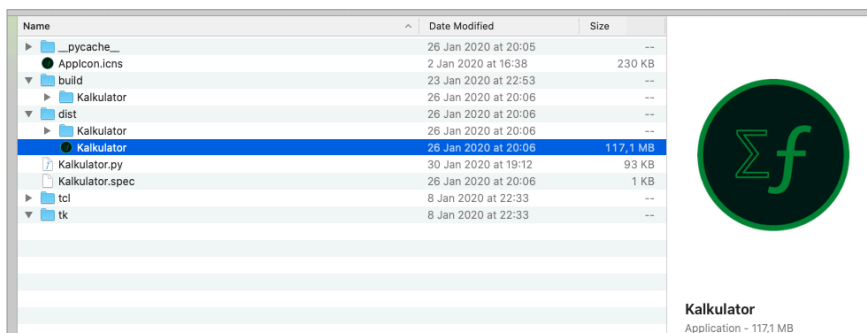
Te so vključevale ikono, datoteko s programsko kodo in dve mapi, ki vsebujeta datoteki za pravilno izvajanje Tkinterja. S spodnjim ukazom sem določil mapo za izvajanje ukazov:

```
$ cd /Users/pibrnik/Desktop/Kalkulator
```

Nato sem pognal ukaz, ki uporabi modul PyInstaller za izgradnjo aplikacije:

```
$ pyinstaller --windowed --icon="Applcon.icns" Kalkulator.py
```

*Windowed* pomeni, da se aplikacija odpira brez terminala, torej se odpira kot zaslonsko okno. Z ukazom *icon* sem določil ikonsko datoteko programa. Zadnja v ukazu je navedena datoteka s programsko kodo.



Slika 43: Izgrajena aplikacija Kalkulator.app

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

Zaradi napake v modulu PyInstaller aplikacije ni mogoče takoj zagnati. Na platformi MacOS PyInstaller v datoteke aplikacije ne kopira kode za **tk** in **tcl**, ki sta glavna gradnika modula Tkinter. Modula, ki sem ju prej prenesel v mapo z datotekami, sem kopiral med programske datoteke novonastalega programa.

### 4.5.3 Priprava aplikacije za platformo Windows

Zaradi razvijanja osnovne programske kode na operacijskem sistemu MacOS sem moral kodo za operacijski sistem Windows zaradi drugačnega videza uporabniškega vmesnika nekoliko spremeniti.

#### 4.5.3.1 Poprava grafične podobe kalkulatorja



Osnovni gumbi Tkinter z operacijskim sistemom Windows niso bili estetsko usklajeni, zato sem uporabil **ttk**, verzijo Tkinterja, v kateri imamo več nadzora nad gradniki. Primer spremembe programske kode za spremenjen videz gumbov.

```
gumb_koren = Button(text='Koren') -> gumb_koren = ttk.Button(text='Koren')
```

Spremenil sem razdalje in velikosti, ki zaradi različnih stilov gradnikov v operacijskem sistemu Windows niso bile ustrezne, kot so bile v operacijskem sistemu MacOS.

#### 4.5.3.2 Izgradnja aplikacije na platformi Windows

Postopek za izgradnjo aplikacije je bil skoraj enak kot na operacijskem sistemu MacOS. Prav tako sem vse datoteke (tokrat brez map **tk** in **tcl**) prenesel v novo mapo za namizju.

Ime	Datum spremembe	Vrsta	Velikost
 Kalkulator	21. 01. 2020 22:00	Python File	89 KB
 Applcon	2. 01. 2020 20:25	Ikona	1.117 KB

Slika 44: Priprava datotek

Izvršil sem ukaz, ki je določil mesto izvanjanja ukazov:

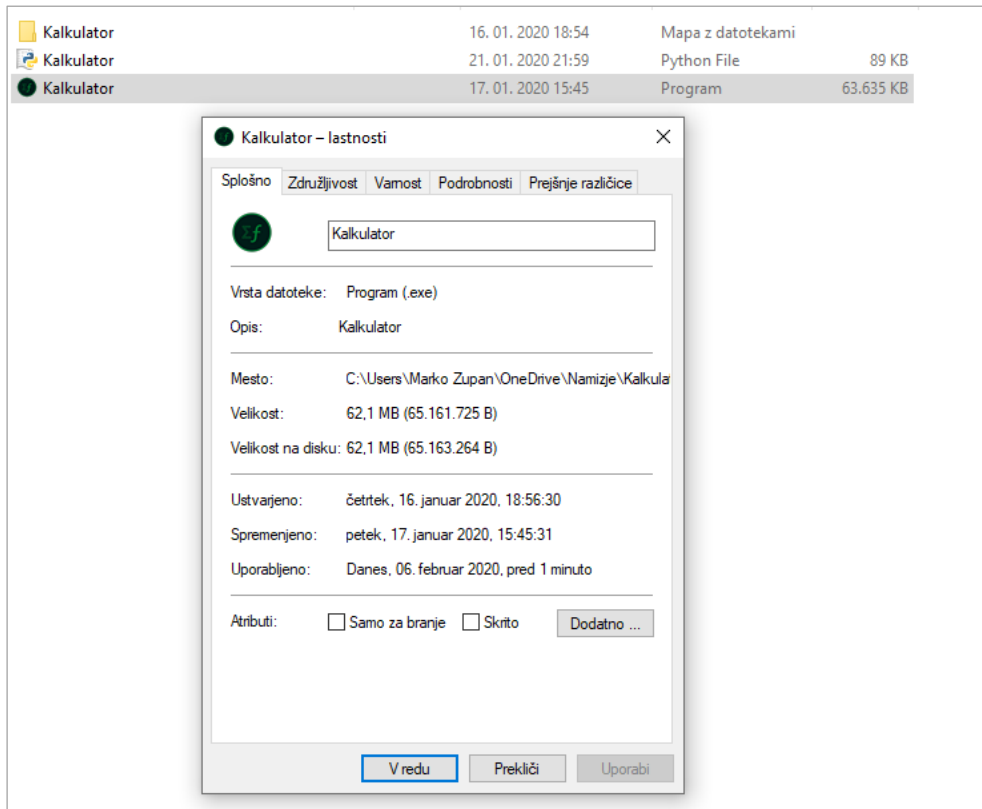
```
$ cd C:/Users/MarkoZupan/Desktop/Kalkulator
```

Nato sem izvršil ukaz za izgradnjo aplikacije:

```
$ pyinstaller --onefile --noconsole --icon="Applcon.ico" Kalkulator.py
```

Ukaz **--onefile** naredi eno samo datoteko in ne celotne mape posameznih datotek. Ukaz **--no-console** pove, da se aplikacija odpira brez terminala. Skupaj ti možnosti tvorita ukaz, enakovredem ukazu **--windowed** na platformi MacOS.

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA



Slika 45: Izgrajena aplikacija Kalkulator.exe

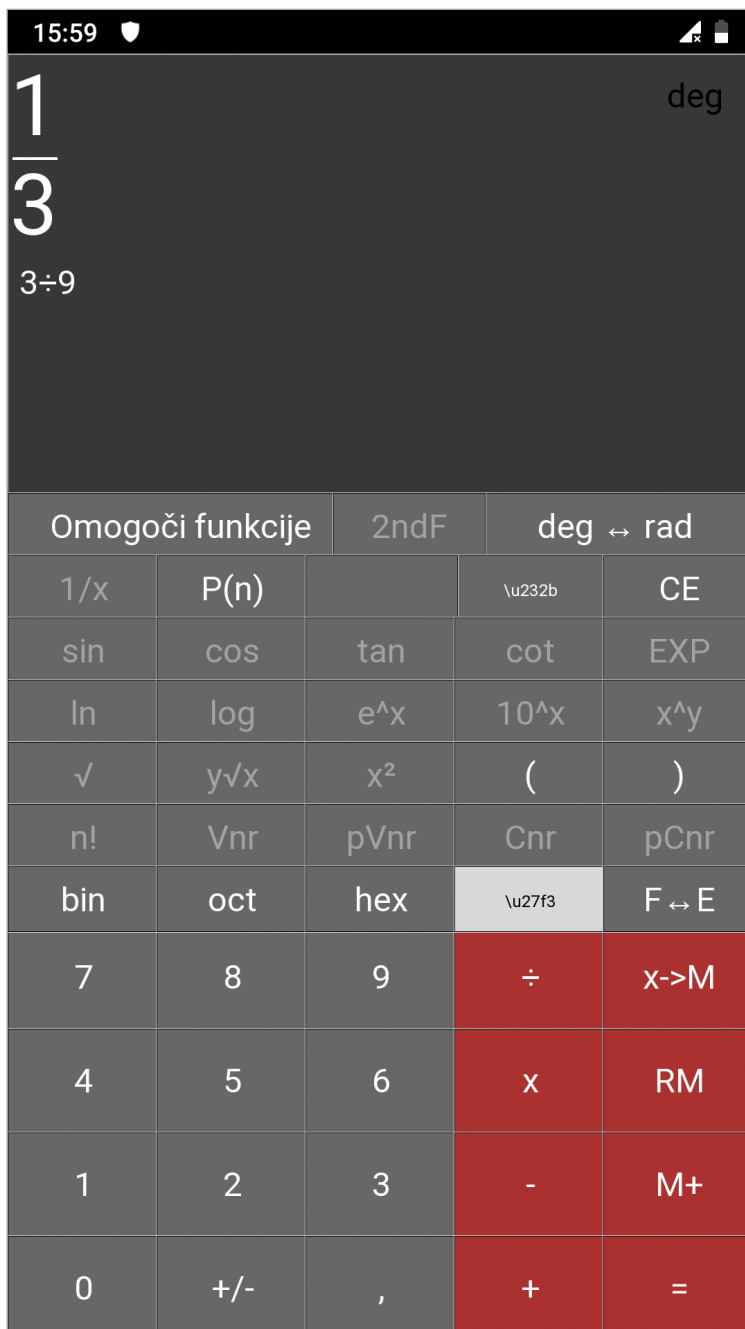


Slika 46: Videz aplikacije v sistemu Windows

#### 4.5.4 Priprava kalkulatorja za uporabo na mobilnih napravah

Pri distribuciji za mobilne naprave sem naletel na oviro, saj za aplikacije Tkinter ne obstaja noben modul, ki bi omogočil pretvorbo programske kode v aplikacijo; Tkinter na mobilnih napravah namreč ni podprt. Zato sem uporabil aplikacijo *Py3Droid*, s katero lahko zaženem svojo programsko kodo.

Zaradi velikih razlik v dolžinskih enotah je bil poseg v kodo obsežen. Spremenil sem vse velikosti objektov, vse odmike objektov in razmike med njimi ter odstranil risanje grafa, saj na mobilnem telefonu lahko naenkrat prikazujemo le eno okno. Spremenil sem tudi barvno podobo aplikacije.



Slika 47: Videz aplikacije v sistemu Android

## 4.6 ANKETIRANJE SOVRSTNIKOV

---

Zaradi potrebe po mnenju sovrstnikov o kalkulatorju in njegovi uporabnosti sem sestavil kratko anketo, v kateri so mi vrstniki odgovorili na vprašanja o uporabnosti in predlagali izboljšave. Vključil sem tudi vprašanje, ali uporabljajo aplikacijo PhotoMath, saj gre za aplikacijo, ki se je dijaki običajno poslužujejo kot alternative klasičnemu kalkulatorju. Anketiral sem 18 vrstnikov.

Anketo sem objavil na spletni strani <https://www.1ka.si>.

**Anketa z analizo in grafi je v prilogi.**

## 5 ZAKLJUČEK

---

Z raziskovalno nalogo sem uspel razviti kalkulator, ki je odpravil pomanjkljivosti šolskih kalkulatorjev.

Ugotovite

**Hipoteza 1:** Dijak splošne gimnazije je sposoben samostojno izdelati znanstveni kalkulator in ga pripraviti za uporabo na različnih operacijskih sistemih.

Drži!

**Hipoteza 2:** Kalkulator je primeren za splošno uporabo in olajša probleme, zaradi katerih je bila naloga zastavljena.

Drži, saj je kalkulator zmožen tako operiranja z osnovnimi funkcijami, ki jih potrebujemo za splošno uporabo, in je pri tem zanesljiv, prav tako pa nam ponuja dodatne funkcije, ki jih nismo vajeni iz tradicionalnih fizičnih kalkulatorjev. Te nam olajšajo računanje zahtevnejših matematičnih izrazov, kot je na primer iskanje racionalnih ničel polinoma višje stopnje.

**Hipoteza 3:** Projekt je možno izpeljati v preprostem programskem jeziku, kot je Python.

Vsekakor drži, saj mi je uspelo sprogramirati popolnoma funkcionalen kalkulator in to povsem v programskem jeziku Python. To je možno predvsem zato, ker na spletu obstaja globalna mreža ljudi in primerov, ki nam pomagajo pri reševanju težav, s katerimi se soočamo pri programiranju. S prebiranjem njihovih izkušenj, odgovorov in predlogov sem lahko odpravil marsikateri problem, na katerega sem naletel pri izvedbi raziskovalne naloge.

Vse tri hipoteze so bile potrjene, torej raziskovalno nalogo lahko označim kot zelo uspešno načrtovano in izvedeno.

### 5.1 IZZIVI ZA NADALJEVANAJE

---

V anketi, ki so jo rešili vrstniki, sem dobil odziv tudi na to, katere funkcije bi bilo smiselno še vpeljati v kalkulator. Predlagano je bilo npr. integriranje. Izziv je tudi izboljšanje enostavnosti uporabe, ki se je izkazala za kalkulatorjev največji problem. Prav tako bi rad našel boljše rešitve za uporabo kalkulatorja na mobilni platformi Android. Torej imam še veliko možnosti za širjenje funkcionalnosti kalkulatorja, s čimer se bom zanesljivo še ukvarjal v prihodnosti.



## 6 VIRI IN LITERAURA

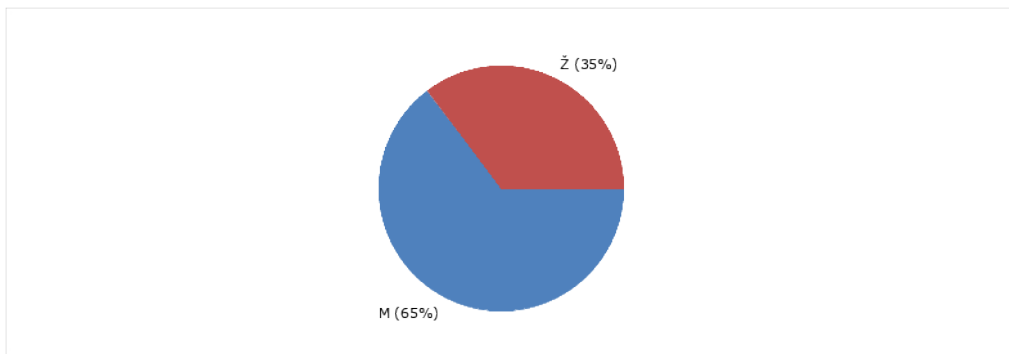
---

- BDM. (2018). Tkinter module. V BDM, *The Python Manual*.
- ElNinja. (6. 12. 2019). *Tkinter*. Pridobljeno s <https://wiki.python.org/moin/Tkinter>.
- Gerrard, P. (2016). *Learn Python*. Berkshire: Apress.
- Liang, Y. D. (2013). *Introduction to programming using Python*. Pearson.
- Matplotlib. (9. 2. 2020). *Homepage*. Pridobljeno s <https://matplotlib.org/>.
- NumPy. (28. 2. 2020). *Homepage*. Pridobljeno s <https://numpy.org/>.
- Programski jeziki in orodja - IDE*. (13. 12. 2019). Pridobljeno s [http://colos.fri.uni-lj.si/ERI/RACUNALNISTVO/PROG\\_JEZIKI\\_ORODJA/prog\\_ide.html](http://colos.fri.uni-lj.si/ERI/RACUNALNISTVO/PROG_JEZIKI_ORODJA/prog_ide.html).
- PyInstaller. (28. 2. 2020). *Homepage*. Pridobljeno s <https://www.pyinstaller.org/>.
- SymPy. (28. 2. 2020). *Homepage*. Pridobljeno s <https://www.sympy.org/en/index.html>.

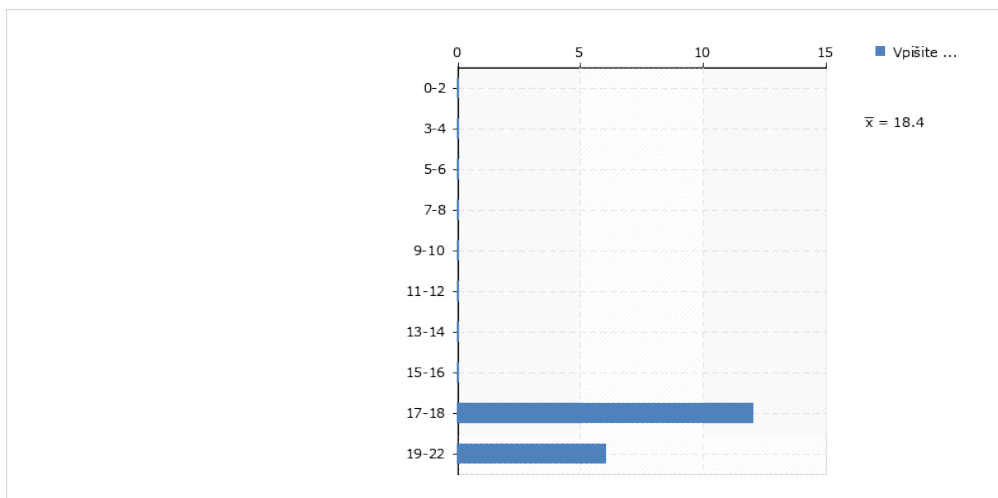
## 7 PRILOGA

Anketni vprašalnik z rezultati, ki sem jih pridobil na <https://www.1ka.si>.

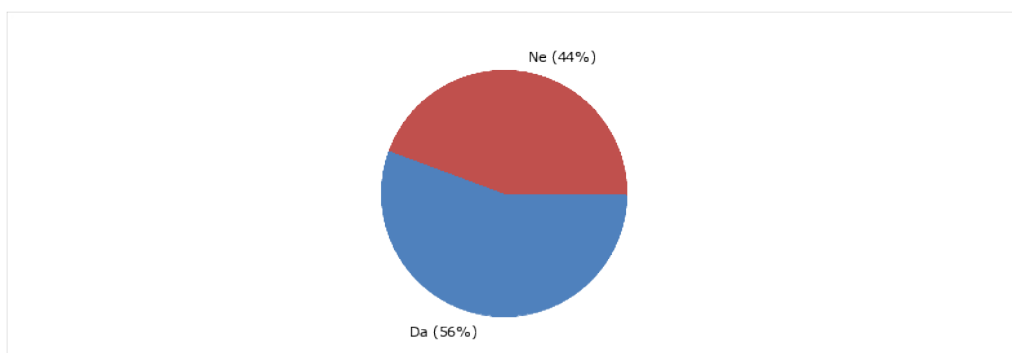
**Spol:** (n = 17)



**Starost:** (n = 18)



**Ali si pri reševanju matematičnih nalog pomagata s programom PhotoMath?** (n = 18)

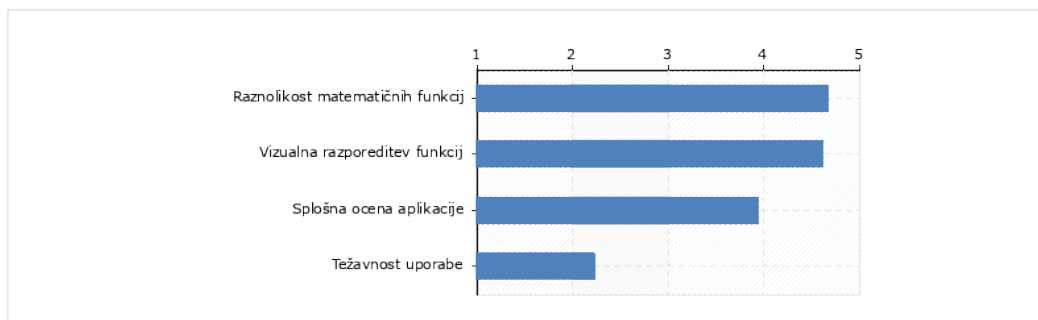


## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

### Če da, katere možnosti pogrešate?

/
računanje zaporedij
reševanje sistema več enačb z več neznankami. risanje funkcij s spreminjanjem parametrov v živo (desmos)
ne pogrešam ničesar
nobene

### Ocenite uporabniško izkušnjo z mojim kalkulatorjem: (n = 18)



### Katere funkcije pogrešate v kalkulatorju?

prikaz postopka reševanja
nobenega
integrale
/
postopek reševanja
nič.
navodila za uporabo naprednejših funkcij kalkulatorja in razlaga delovanja tipk. računanje nedoločenih in določenih integralov. risanje funkcij
reševanje funkcij
integriranje
računanje zaporedij, integralov in višjih odvodov
lažja uporaba z manj pritiski
integrale, reševanje ekstremov

## PROGRAMIRANJE LASTNEGA ZNANSTVENEGA KALKULATORJA

**Ali bi kalkulator uporabljali, če bi bil na voljo kot mobilna aplikacija? (n = 17)**

