UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – Praktična matematika (VSŠ)

Andreja Dokl

Orodje za risanje GIMP in skriptni jezik Scheme

Diplomska naloga

Ljubljana, 2010

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Zahvala

Za strokovno vodenje, nasvete, pomoč in vestno pregledovanje diplomske naloge se lepo zahvaljujem mentorju mag. Matiju Lokarju.

In sploh hvala vsem: družini, fantu, vsem prijateljem in sošolcem, ki so mi v času študija in pisanja diplomske naloge stali ob strani in me spodbujali.

Program dela

V sklopu diplomske naloge opišite uporabo programa za risanje in obdelavo slik GIMP. Pri tem posebno pozornost posvetite skriptnemu jeziku Scheme. Na praktičnih primerih demonstrirajte uporabo tega skriptnega jezika.

Osnovna literatura:

- Carey Bunks, Grokking the GIMP, New Riders Publishing, 2000, dostopno na http://GIMP-savvy.com/BOOK/index.html
- Dov Grobgeld, A Scheme Tutorial for GIMP Users, 2002, dostopno na http://imagic.weizmann.ac.il/~dov/GIMP/scheme-tut.html

mentor

mag. Matija Lokar

Povzetek

V diplomski nalogi je opisan program za risanje in obdelavo slik GIMP. Opisane so osnove skriptnega jezika Scheme, ki je podprt v programu GIMP.

V prvem poglavju je opisana kratka zgodovina programa GIMP, kje ga dobimo, čemu je namenjen in osnovna predstavitev delovnega okolja. V drugem poglavju so predstavljeni trije različni primeri uporabe programa GIMP za obdelavo fotografij. V tretjem poglavju so opisane osnove skriptnega jezika Scheme. Razloženo je, zakaj govorimo ravno o skriptnem jeziku Scheme, kaj je skriptni jezik in kako je videti skripta. Predstavljene so tudi osnove za delo s skriptnim jezikom Scheme. V četrtem poglavju se posvetimo primerom iz drugega poglavja. Tokrat isto delo opravimo s skriptami. Za vsak primer predstavimo skripto in njen rezultat. V petem poglavju združimo vse skripte iz četrtega poglavja in tako ustvarimo eno skripto, ki opravi delo vseh posameznih skript.

Abstract

In my thesis I have described GIMP program for drawing and image manipulation, as well as the basics of Scheme scripting language, which is supported in the program.

In the first chapter short history of GIMP program is outlined. It is explained where we can obtain the program, what it is designed for and what is its appearance like. In the second chapter, three different examples of program use for image manipulation are presented. The third chapter describes the basics of Scheme scripting language. It explains why the subject is precisely Scheme scripting language, what a scripting language is and it also describes the appearance of a script. Furthermore, the basics of working with Scheme scripting language are presented. In the forth chapter we devote our attention to examples from the second chapter. This time, the same work is done with the help of scripts. We present a script and its result. In the fifth chapter we combine all the scripts from the forth chapter, so that we create one script that does the work of all scripts.

Math. Subj. Class. (2010): 00A05, 03-04, 68N15, 68N19, 68N20, 97P40, 97R60

Computing Review Class. System (1998): D.1.5, D.3.1, D.3.2, D.3.3, I.3.4

Ključne besede: GIMP, obdelava slik, vektorska grafika, rastrska grafika, skriptni jezik, skriptni jezik Scheme

Keywords: GIMP, picture manipulation, vector graphics, raster graphics, script language, script language Scheme

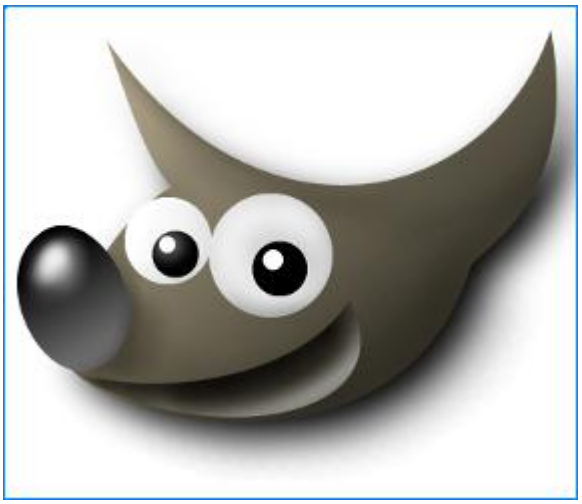
Kazalo

1 Kaj je GIMP in čemu je namenjen?

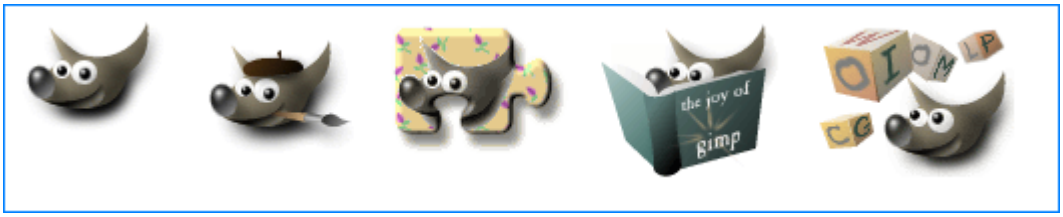
1.1 Kratka zgodovina programa GIMP

Projekt GIMP, katerega namen je bil ustvariti program za delo z rastrsko grafiko, sta začela leta 1995 dva študenta kalifornijske univerze Berkeley v ZDA: Spencer Kimball in Peter Mattis. V izvirniku je kratica GIMP pomenila »General Image Manipulation Program«. Prva javno dostopna različica programa (različica 0.54) je bila objavljena januarja 1996. Leta 1997 je GIMP postal del projekta GNU in ga vzdržuje skupina prostovoljcev. Takrat je pomen kratice postal »GNU Image Manipulation Program«. Od takrat pa vse do danes je nastalo mnogo različic GIMPa. Najnovejša različica v času pisanja diplomske naloge (junij 2010) je različica 2.6.8.

Kot zanimivost naj omenim, da je uradno maskoto skupine GIMP, poimenovano Wilber, Tuomas Kuosmanen ustvaril 25. septembra 1997. Od takrat do danes se maskota ni kaj dosti spremenila.



Slika 1: Wilber



Slika 2: Wilber z različnimi dodatki (vir: http://sl.wikipedia.org/wiki/GIMP)

1.2 Kje dobimo GIMP?

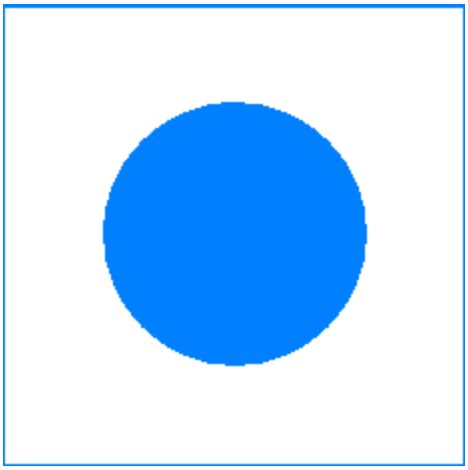
GIMP je prosto dostopen program, ki ga najlažje dobimo na uradni spletni strani http://www.GIMP.org/. Na voljo je tudi slovenska različica, kar precej pripomore k lažji uporabi tudi pri nas. Na spletu se nahaja tudi priročnik: http://docs.GIMP.org/2.6/en/. Žal poleg angleške različice obstajajo različice v kar nekaj tujih jezikih, v slovenščini pa ga ni.

1.3 Kaj je GIMP, čemu je namenjen in kako je videti?

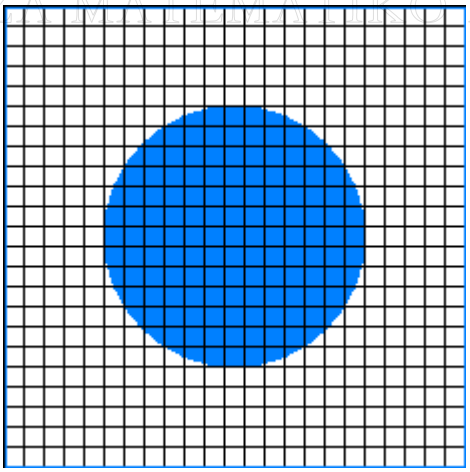
Program GIMP je prosto dostopen odprtokodni program za urejanje rastrske grafike. Ponuja tudi nekaj možnosti za delo z vektorsko grafiko. Kot smo omenili, je GIMP kratica za »GNU Image Manipulation Program« (»GNU program za obdelavo slik«). GIMP se uporablja predvsem za obdelavo digitalne grafike in fotografij. Z njim največkrat ustvarjamo grafične podloge in logotipe, spreminjamo velikosti fotografije, retuširamo fotografije, pretvarjamo med različnimi slikovnimi formati … Deluje na operacijskih sistemih Linux, MAC OSx in Windows.

1.3.1 Rastrska grafika

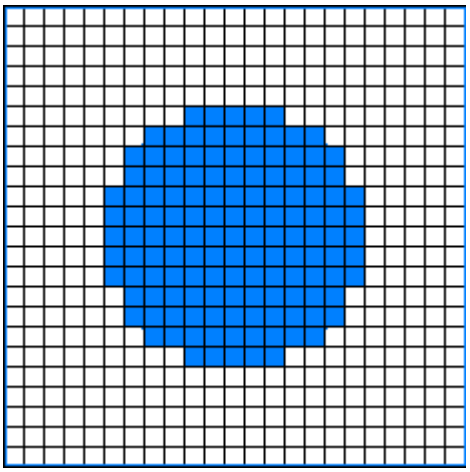
Rastrska grafika je v računalništvu način shranjevanja slik z množico slikovnih pik v obliki dvodimenzionalne matrike. Slika je v pomnilniku shranjena v obliki matrike, ki vsebuje podatke kot sta barva in intenziteta. Podatki v matriki so shranjeni za vsako slikovno točko ali piksel posebej. Tako si za grafiko kroga zapomnimo podatek o obarvanosti za vsako točko v mreži. Če določena točka pri tem, ko postavimo mrežo (raster) ni povsem obarvana (ali pa vsebuje različne barve), se moramo odločiti za skupen podatek te točke.



Slika 3: Krog



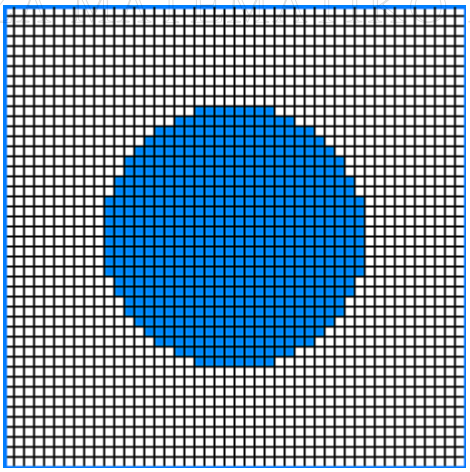
Slika 4: Krog z dodano mrežo (rastrom)



Slika 5: Rasterizacija kroga

Rastrski grafiki rečemo tudi bitna grafika. Beseda bitna nas spomni na bit – osnovno in hkrati najmanjšo enoto za zapis informacije v računalništvu in informatiki. Gre za to, da si za vsako slikovno točko zapomnimo število, ki označuje barvo te točke. Število bitov, ki jih potrebujemo za zapis teh števil, pa določa bitnost. Pri bitni grafiki si računalnik barvo vsake točke v bitni sliki zapomni s pomočjo bitov. Več kot ima bitov, več barv oziroma barvnih odtenkov bo zmožno uporabiti na sliki. Najpreprostejša slika je črno-bela, kjer imamo opraviti z 1-bitno grafiko. Točke so le črne ali bele barve. Slika, v kateri imamo 256 barv, je 8-bitna, slika s 65536 barvami 16-bitna, pri 24-bitnih slikah pa imamo možnost uporabe kar 16,7 milijonov barv

Slika, zapisana v rastrski grafiki, je torej sestavljena iz množice slikovnih točk, razporejenih v mrežo. Več kot je slikovnih točk, boljša je ločljivost slike. Če bi torej v primeru na sliki Slika 5 uporabili gostejšo mrežo, bi dobili več slikovnih točk.



Slika 6: Krog z gostejšo mrežo

Ločljivost merimo v slikovnih točkah na mersko enoto:

ločljivost = število slikovnih točk/dolžinska merska enota.

Običajno je merska enota inča. Ločljivost je torej običajno podana kot število pikslov na eno inčo (DPI – Dots Per Inch). Ločljivost pravzaprav ni povezana s fizično velikostjo slike in z velikostjo mreže (rastra) slike. Pomembno je kako gosta je mreža (raster). Odvisna je torej od izhodne enote. Če ima izhodna enota npr. zaslon visoko ločljivost, bo slika prikazana lepše, saj jo bodo prikazovale manjše slikovne točke. Na zaslonu z majhno ločljivostjo bo ista slika videti precej slabše. Med izhodno enoto s slabo ločljivostjo in izhodno enoto z dobro ločljivostjo je podobna razlika kot razlika med sliko Slika 5 in sliko Slika 6.

Prednost rastrske grafike je v tem, da slike s pomočjo izhodnih naprav z veliko ločljivostjo spominjajo na fotografije. Prav tako so je večina izhodnih naprav, med drugim tudi zaslon, narejena tako, da prikazuje slikovne točke. Zato pri prikazu rastrske slike na zaslonu praviloma ni potrebna dodatna obdelava ali pa ta ni preveč zahtevna. Preprosto je tudi tiskanje rastrskih slik, saj zmore računalnik brez težav sporočati tiskalniku, kako naj natisne posamezne točke. Slaba stran rastrske grafike pa je v tem, da slike, ki imajo visoko ločljivost, zavzamejo veliko prostora na disku. Prihaja pa tudi do problemov pri spreminjanju rastrskih slik. Pri povečavi bitne slike prihaja do nezaželenih popačenj. Težava je v tem, da se z rasterizacijo osnovna informacija izgubi. Recimo, da dvakrat povečamo sliko Slika 6. Sedaj, se bomo morali za vsako slikovno točko na sliki odločiti, kako jo bomo predstavili. Vsaka točka je sedaj sestavljena iz štirih novih točk. V notranjosti kroga, kjer so vse slikovne točke obarvane modro, ni težav. Na robu pa se moramo odločiti, katera od teh štirih bo sedaj obarvana z modro in katera ne. Težava je tudi v tem, da ko dodajamo grafične objekte (npr. krog), potem le-ti izgubijo lastnost objekta in so le še množica točk. Če potem želimo krog npr. pobrisati, moramo brisati točko za točko. Odstranjevanje kroga, kot geometrijske konstrukcije ni mogoče. Ko je objekt (v našem primeru krog) dodan, izgubi lastnost objekta. Ostane le še množica slikovnih točk. Enako velja, če bi želeli krogu spremeniti barvo z modre na zeleno. Potem ko smo krog dodali na sliko, mu kot geometrijski konstrukciji ne moremo več spremeniti barve. Lahko spremenimo barvo le množici slikovnih točk, ki so na zaslonu razporejene v krog.

1.3.2 Vektorska grafika

V računalniku je slika, narejena z vektorsko grafiko, shranjena v obliki geometrijskih formul. Rečemo ji tudi objektno orientirana grafika, saj uporablja objekte – osnovne geometrijske oblike kot so točke, premice, krivulje in like, ki temeljijo na matematičnih enačbah. Vektorsko grafiko običajno uporabljajo grafični programi in programi za risanje.

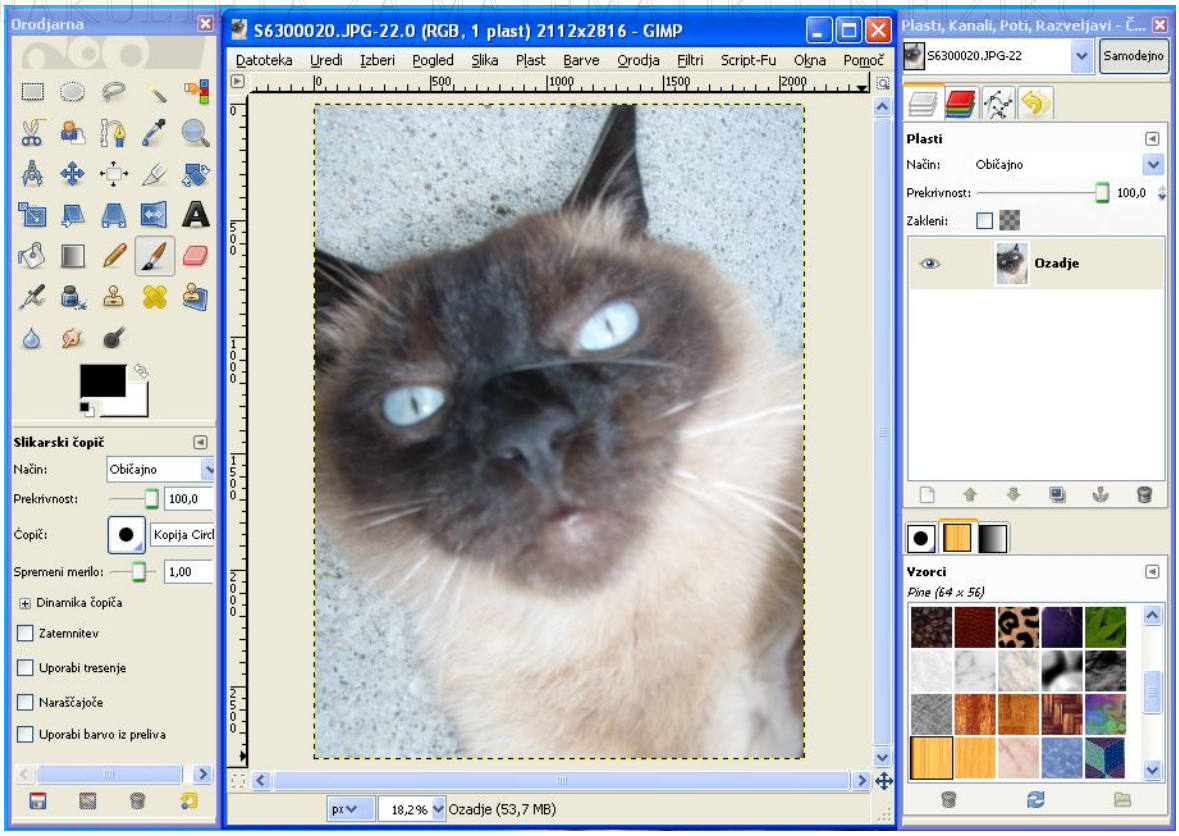
Večina izhodnih naprav (večina tiskalnikov, zasloni, večina risalnikov …) pričakuje informacije v obliki slikovnih točk (rečemo lahko, da so izhodne naprave »rastrske«). Formule, ki jih uporablja vektorska grafika, prevedemo v slikovne točke. Tako ustvarimo rastrsko sliko, primerno za večino izhodnih naprav.

Prednost vektorske grafike je v tem, da tako pripravljene slike po želji transformiramo (povečamo, vrtimo, raztegnemo …). Pri tem se kakovost slike ne spremeni. Prav tako se kakovost slike ohranja pri povečavi ali pomanjšavi. Ne glede na to, v kolikšni meri bomo povečali vektorsko sliko, bo ta še naprej gladka in tudi na zaslonu ali kateri drugi izhodni napravi bo videti ostra. Izbrati in transformirati je mogoče tudi vsako posamezno komponento vektorsko zapisane slike posebej, saj je vsaka komponenta v računalniškem pomnilniku definirana posebej. Težave z vektorsko grafiko se lahko pojavijo pri tiskanju. Pri tiskalnikih, ki sprejemajo rastrsko grafiko, je potrebno sliko pred tiskanjem rasterizirati. Rasterizacija je lahko časovno precej zahtevna in lahko potrebuje več pomnilnika. Tiskalniki, ki so sposobni sprejemati vektorsko grafiko (praviloma mora biti ta zapisana v postScriptu – kjer dejansko uporabljamo vektorski opis), sami opravljajo proces rasterizacije. Pri takih tiskalnikih se lahko zgodi, da tiskalnik ne razume vseh ukazov. Zato se lahko odtis precej razlikuje od pričakovanega.

Slika 7: Rastrska grafika proti vektorski grafiki (vir: http://www.provektor.si/vektorraster.html)

1.3.3 Videz programa GIMP

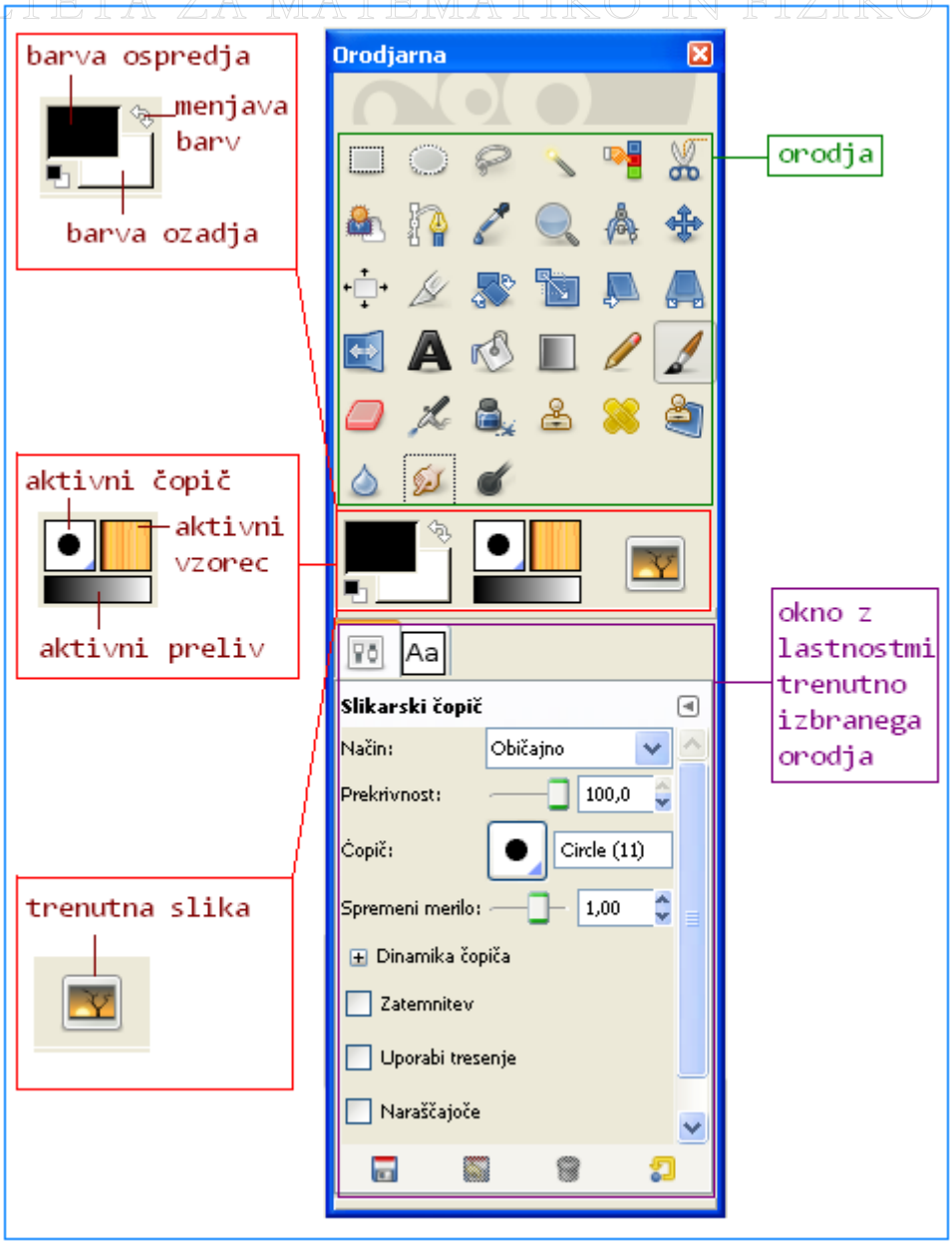
Okolje programa GIMP je sestavljeno iz treh glavnih oken: iz okna imenovanega orodjarna, iz okna z imenom plasti in iz okna obdelovane slike. Okna so ločena in jih lahko poljubno premikamo po namizju.



Slika 8: Videz programa GIMP

- **Okno orodjarna**

  Okno orodjarna (na sliki Slika 9) je sestavljeno iz treh glavnih delov. V prvem delu, ki je označen zeleno, so gumbi, ki aktivirajo orodja, ki se nahajajo v meniju `orodja`. Orodja so lahko orodja za izbiranje (pravokotni izbor, eliptični izbor, prosto izbiranje, izbor barve …), premikanje, povečavo ....



Slika 9: Okno orodjarna

V drugem delu, ki je označen rdeče, se nahajajo tri podokna. V prvem podoknu nastavljamo barvo ospredja in barvo ozadja. Če hočemo spremeniti barvo ozadja, to storimo s klikom na površino, ki označuje barvo ozadja. Odpre se meni s paleto barv, kjer izberemo želeno barvo. Enako spremenimo barvo ospredja. Kadar kliknemo na puščico za menjavo barv, barva ozadja postane barva ospredja in obratno. Drugo podokno kaže aktivni čopič, aktivni vzorec in aktivni preliv. V okencu aktivni čopič je prikazano tisto orodje, s katerim trenutno »rišemo« na sliko. Sem spadajo vsi čopiči in tudi radirka. Aktivni vzorec kaže trenutni vzorec, s katerim želimo zapolniti izbrana območja na sliki. Na voljo je mnogo različnih vzorcev in tekstur. Aktivni preliv prikazuje, kateri preliv uporabljamo za prelivanje. Prelivi omogočajo barvanje določenih površin slike ali cele slike. Omogočajo prelivanje barv iz ene barve v drugo. S klikom na katerikoli gumb se v tretjem oknu, imenovanem okno plasti, odpre meni, v katerem lahko izberemo aktivni čopič, aktivni vzorec in aktivni preliv.

Tretje podokno prikazuje trenutno aktivno sliko. V GIMPu namreč lahko delamo z več slikami hkrati. Določena orodja oziroma opravila pa zahtevajo, da v določenem trenutku delamo samo z eno sliko. Podokno prikazuje sliko, s katero delamo v tem trenutku. S klikom na trenutno sliko pa se odpre meni vseh slik, ki so v uporabi pri tem projektu.

V tretjem delu, označenem na Slika 10 z vijolično barvo, je okno, ki prikazuje možnosti orodja, ki ga trenutno uporabljamo.

- **Okno obdelovane slike**

Slika 10: Okno obdelovane slike

Na sliki Slika 10 rdeče označbe pomenijo naslednje:

Številka **1** označuje **naslovno vrstico**. Ta prikazuje ime trenutno odprte slike in nekaj osnovnih podatkov o sliki.

Takoj pod naslovno vrstico se nahaja **slikovni meni**, ki ga na sliki označuje številka **2**. V njem so na voljo vse operacije, ki jih s pomočjo GIMPa lahko izvedemo na sliki.

S klikom na številko **3** se odpre **slikovni meni**, ki je na sliki označen s številko 2. Meni se odpre v obliki stolpca.

Številka **4** označuje **merilo**. Z njim lahko določimo koordinate poljubne točke v oknu obdelovane slike. Privzeta enota za merjenje v GIMPu je piksel. Merilo uporabljamo tudi

za risanje pomožnih črt. Te niso del slike, ampak jih uporabimo, kadar želimo določen element umestiti na točno določen del slike. Pomožne črte preprosto odstranimo tako, da jih povlečemo preko roba slike.

Številka **5** označuje gumb za **hitro masko izbire**. S hitro masko izbire lahko poljubno spreminjamo del slike. Označimo del, ki ga ne želimo spremeniti, na preostalem delu pa vklopimo hitro masko izbire. Neoznačeno območje postane prosojno rdeče barve. To območje zdaj lahko s poljubnimi filtri spreminjamo. Na koncu hitro masko izklopimo. Na tistem delu slike, ki ga nismo označili, spremembe ostanejo. Del slike, ki je bil označen, pa po koncu dela z masko ostane nespremenjen.

Če se z miško postavimo nekje znotraj okna obdelovane slike, lahko v okencu, označenem s številko **6**, vidimo **koordinate miške**.

Številka **7** označuje **meni za enote**. Privzeto GIMP za mersko enoto uporablja piksle. S klikom na meni za enote lahko mersko enoto spremenimo.

S klikom na **gumb za povečavo**, ki se skriva pod številko **8**, v odstotkih velikosti celotne slike določimo povečavo.

Pri časovno zahtevnih operacijah (kot je recimo izvajanje obsežnih skript, ki zahtevajo veliko časa), se na mestu, označenem z **9,** pojavi **gumb prekliči**. S klikom nanj prekinemo izvajanje operacije. Včasih lahko prekinitev določene operacije pusti nepopravljive posledice na slikah.

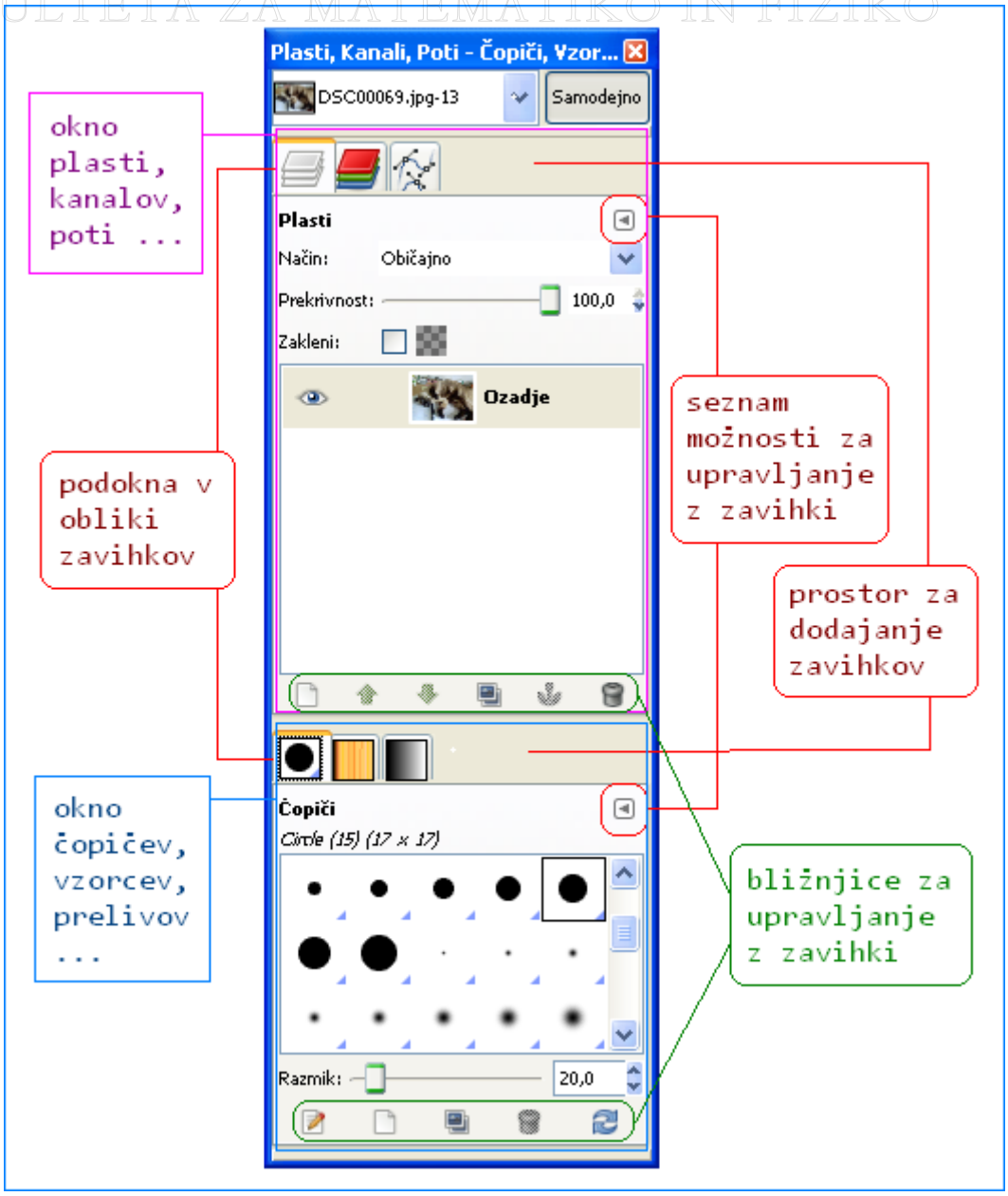
Gumb za krmarjenje se nahaja na mestu označenem s številom **10**. Če zadržimo klik nanj, opazimo, da se na mestu gumba pokaže pomanjšava slike, ki jo obdelujemo. Tej sličici rečemo navigacijski predogled slike. S tem se lahko hitro premaknemo na drugi konec slike, pa čeprav je obdelovana slika zelo velika. Če v oknu obdelovane slike ni odprte slike, potem gumba ni.

Po številom **11** se skriva **gumb za prilagoditev velikosti slike**. S klikom na gumb omogočimo funkcijo, ki sorazmerno z zmanjševanjem (povečevanjem) okna obdelovane slike, zmanjša (poveča) tudi sliko (spremeni povečavo kot jo tudi gumb pod številko 8).

Z **12** je označeno **polje, kjer je prikazana odprta slika**. Rečemo mu tudi »platno«.

Številka **13** prikazuje **statusno okence**. Statusno okence prikazuje aktivni del slike. Recimo, da trenutno urejamo plast z imenom ozadje. Tedaj v statusnem okencu piše, da urejamo plast ozadje. Če v orodjarni izberemo določeno orodje, se v statusni vrstici prikaže navodilo za uporabo izbranega orodja.

- **Okno plasti**



Slika 11: Okno plasti

Okno plasti je okno sestavljeno iz dveh glavnih oken. Obe okni sta sestavljeni iz več zavihkov. Zavihke dodajamo in odstranjujemo sami. Tako si lahko program GIMP prikrojimo tako, da nam kar najbolj ustreza.

V zgornjem pogovornem oknu (na sliki Slika 11) se običajno (ni pa nujno!) nahaja zavihek `plasti`. Če si sliko predstavljamo kot knjigo, potem si plast lahko predstavljamo kot en sam list v tej knjigi. Predpostavimo, da so listi v knjigi prosojni. Šele, ko pogledamo skozi vse liste v knjigi, vidimo celotno sliko. Plasti so nujno potrebne za risanje. Vsaka slika, ki jo narišemo v GIMPu, je sestavljena iz več plasti. Tista plast, ki je na seznamu plasti (v zavihku plasti) na prvem mestu, je v sliki najbolj pri vrhu. Ko plasti združimo, dobimo celotno sliko. Ker našo sliko razslojimo, je popravljanje določenih elementov slike enostavnejše in varnejše. Popravimo samo plast, na kateri se nahaja element, ki ga želimo popraviti. Ostali elementi na drugih plasteh tako zagotovo ostanejo nespremenjeni.

Zavihke upravljamo s klikom na puščico, ki nam odpre seznam za upravljanje zavihkov. Ponuja možnosti dodajanja in odstranjevanja zavihkov, nastavimo lahko način, kako zavihke vidimo, njihovo velikost … Zavihke lahko odcepimo v samostojno okno ipd.

S klikom na jeziček zavihka prikažemo celoten zavihek. Na sliki Slika 11 je trenutno odprt zavihek z imenom plasti. V zavihku so prikazane plasti trenutno obdelovane slike Na dnu odprtega zavihka se nahajajo bližnjice. Na sliki Slika 11 je odprt zavihek plasti. Zato se na dnu zavihka nahajajo bližnjice za delo s plastmi trenutno odprte slike. Z bližnjicami lahko ustvarimo novo plast, plasti lahko brišemo, ustvarimo dvojnike že obstoječih plasti …

Okno čopičev, vzorcev, prelivov … je okno, ki lahko prikazuje podrobnosti o uporabi trenutno aktivnega čopiča, vzorca in preliva. Če v oknu orodjarna kliknemo na okence za izbiro aktivnega čopiča, se v oknu čopičev, vzorcev in prelivov prikaže celoten zavihek s paleto čopičev, ki jih ponuja GIMP. Podobno s klikom na okence za izbiro vzorca GIMP ponudi zavihek s paleto vzorcev. S klikom na okence za izbiro prelivov, GIMP odpre zavihek s paleto prelivov.

Vse tri zavihke (čopiči, prelivi, vzorci) lahko zapremo. V tem primeru (v oknu orodjarna) s klikom na okence za npr. izbiro preliva, prav tako odpremo okno s celotno paleto prelivov, le da se ne pojavi v obliki zavihka v oknu čopičev, vzorcev, prelivov … Pojavi se samostojno okno. Isto velja za čopiče in vzorce.

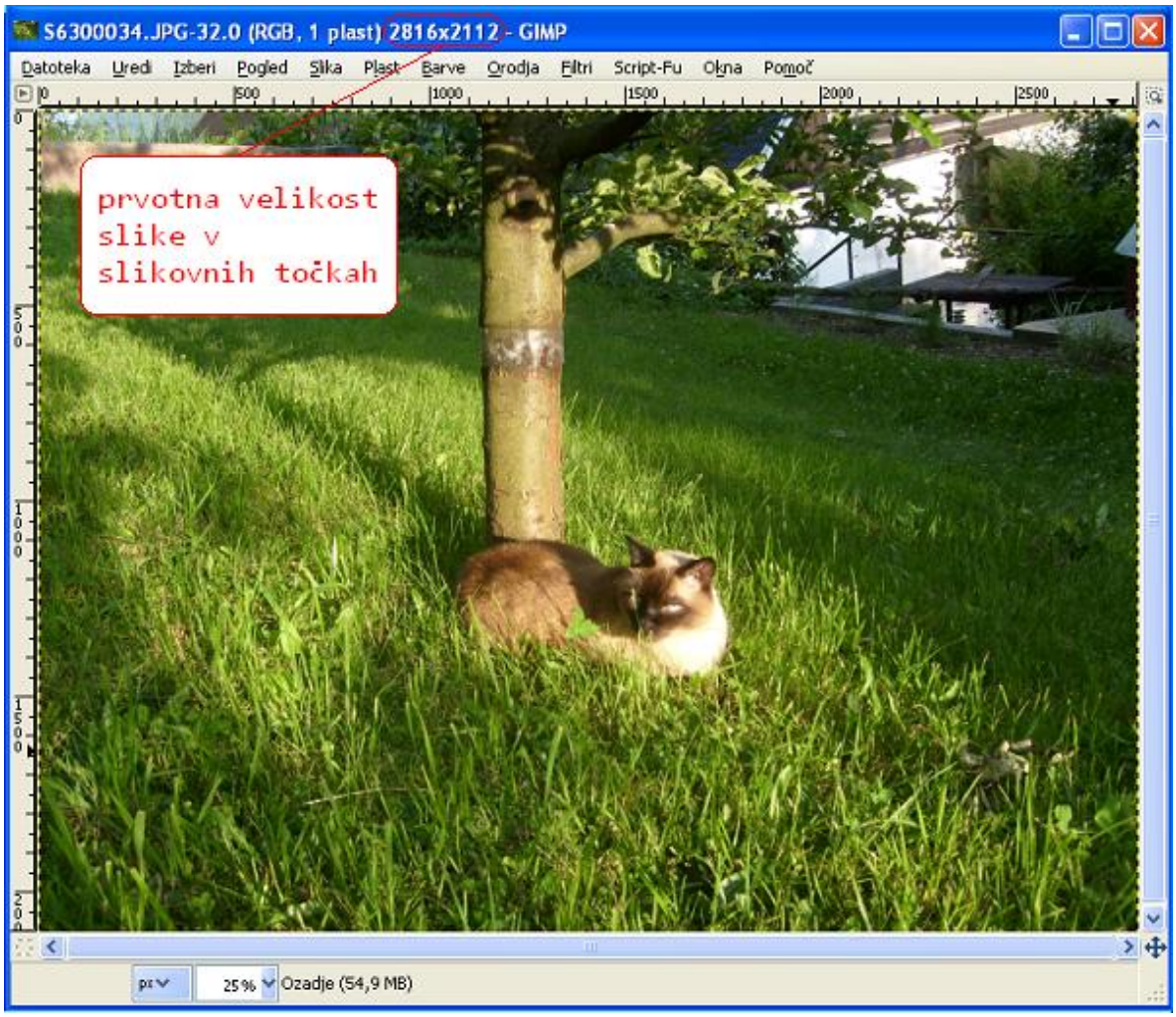
Zavihke v oknu čopičev, vzorcev, prelivov … lahko tudi poljubno dodajamo. To storimo s klikom na puščico za meni za upravljanje z zavihki (glej sliko Slika 11).

2 Primeri uporabe

2.1 Spreminjanje velikosti slike

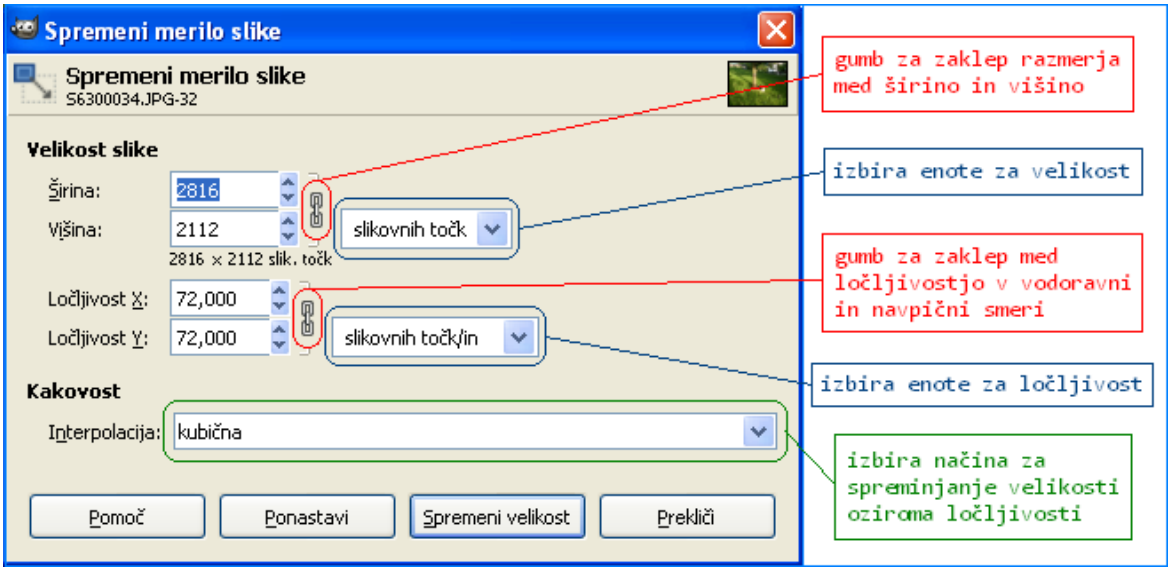
Pri velikosti slike seveda ne mislimo na njeno fizično velikost, saj je ta odvisna od ločljivosti in izhodne naprave, ampak na število slikovnih točk, ki jo sestavlja (raster). Slikovne točke ne moremo izmeriti npr. v milimetrih. Praktično to sicer lahko storimo, ampak se meritve na različnih npr. zaslonih ne bodo ujemale. Lahko imamo zaslon velikosti 1024 x 768 slikovnih točk, hkrati pa je lahko njegova fizična velikost (ki jo običajno merimo v inčih) 7″, 10″, 15,4″, 17″ … S podatkom o ločljivosti povemo, koliko slikovnih točk se nahaja v npr. enem centimetru. Če ločljivost povečujemo, se širina in višina slike sorazmerno povečujeta. Nasprotno, če ločljivost zmanjšujemo, se višina in širina sorazmerno povečata. Velikost v slikovnih točka ostaja enaka.

Najprej v programu GIMP odpremo poljubno sliko (Slika 12).



Slika 12: Prvotna slika

Za spreminjanje velikosti slike v slikovnem meniju (Slika 10, številka 2) izberemo meni `slika`. V meniju izberemo ukaz `spremeni merilo slike`. Odpre se pogovorno okno prikazano na sliki Slika 13.

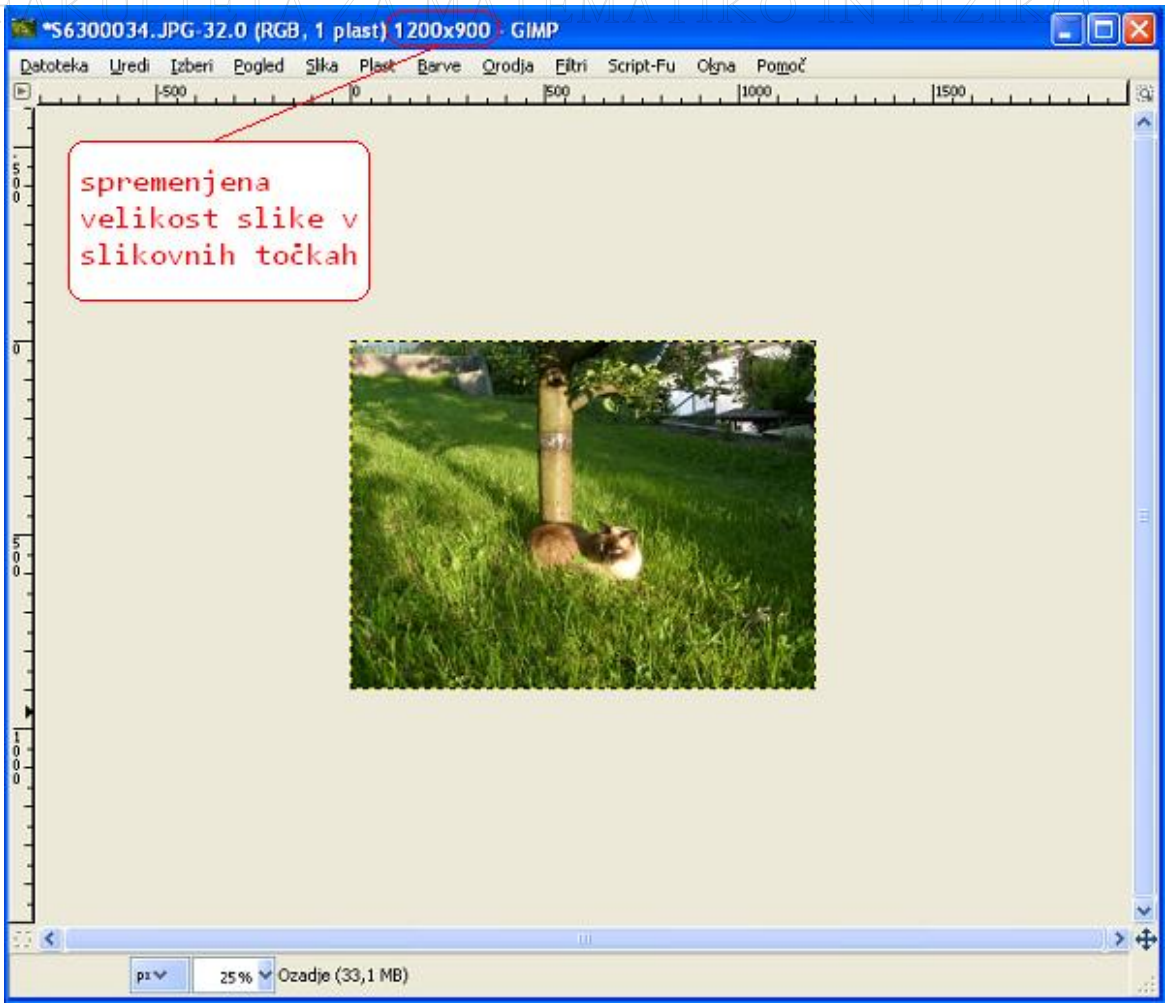


Slika 13: Pogovorno okno za spreminjanje velikosti slike

V našem primeru bomo sliko pomanjšali. V pogovornem oknu lahko nastavimo višino in širino. Najprej izberemo enoto za velikost. Na prvotni sliki (Slika 12) osnovno enoto predstavljajo slikovne točke. Dimenziji slike lahko preberemo v naslovni vrstici okna obdelovane slike. Ko izberemo osnovno enoto, lahko določimo novo širino in višino. Če hočemo ohraniti razmerje med širino in višino, kliknemo na gumb za zaklep razmerja med širino in višino. Zdaj lahko vnesemo samo širino in se višina sorazmerno prilagodi in obratno. Lahko vnesemo samo višino in se širina samodejno prilagodi.

Nazadnje izberemo še način spreminjanja velikosti slike. Rečemo mu interpolacija. Gre za to, kako določimo barvo vsake točke v novem rastru na podlagi skupine točk prvotnega rastra. Z izbiro interpolacije vplivamo na kakovost spremenjene slike. Kubična interpolacija je najboljša, a časovno tudi najbolj zahtevna. Zato ni primerna za zelo velike slike.

Po opravljenih nastavitvah le še kliknemo na gumb `Spremeni velikost` in v oknu obdelovane slike vidimo spremenjeno sliko (Slika 14). Spremenjeni dimenziji sta prikazani v naslovni vrstici okna obdelovane slike.



Slika 14: Spremenjena slika

Potrebno pa se je zavedati, da smo s spreminjanjem velikosti sliko trajno spremenili. Če bomo torej sliko najprej pomanjšali za 50%, potem pa jo za 100% povečali, ne bomo dobili originalne slike.

2.2 Ustvarjanje fotografije s sepija barvnim tonom

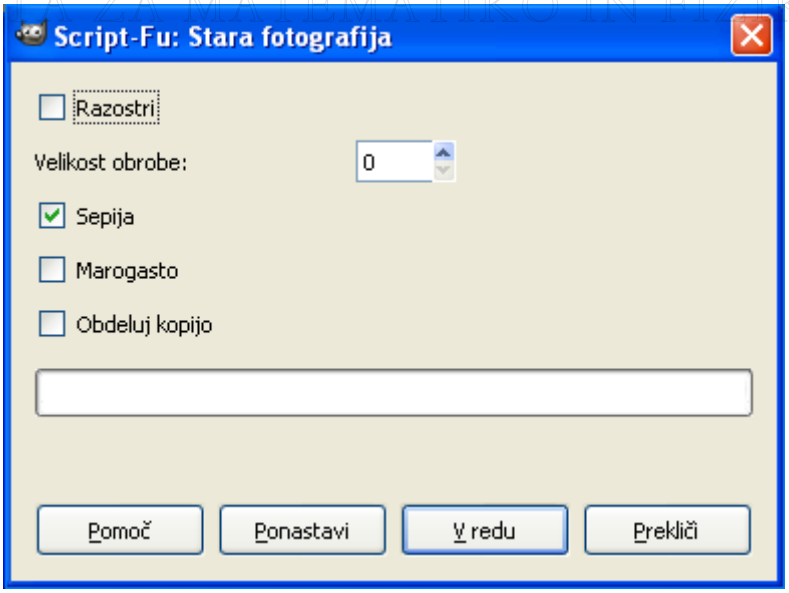
Fotografija s sepija barvnim tonom je podobna črno-beli fotografiji, le da ji dodamo še nekaj toplih rjavih tonov.

Učinek barvnega načina sepija lahko ustvarimo na več načinov. Vse primere bomo zaradi boljšega prikaza končnih razlik izvedli na isti sliki (Slika 15).

Slika 15: Prvotna slika

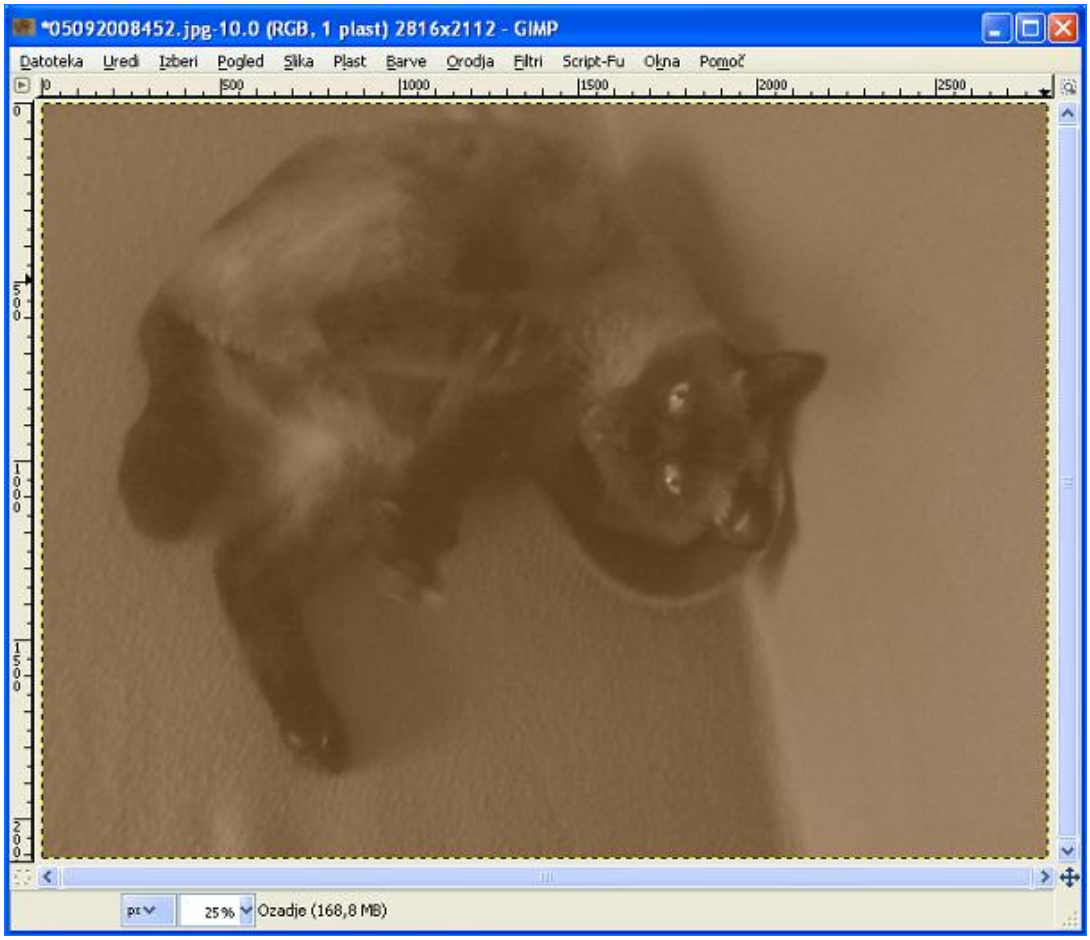
- **PRVI NAČIN**

  Najhitreje ustvarimo fotografijo s sepija barvnim tonom tako, da v slikovnem meniju okna obdelovane slike izberemo meni `filtri`, podmeni `okras` in nato ukaz `stara fotografija`. Dejansko s tem poženemo že vgrajeno skripto, torej izvajanje zaporedja ukazov. Kaj dejansko skripta je, si bomo ogledali v razdelku 3.1).
  Odpre se pogovorno okno (Slika 16).



Slika 16: Stara fotografija

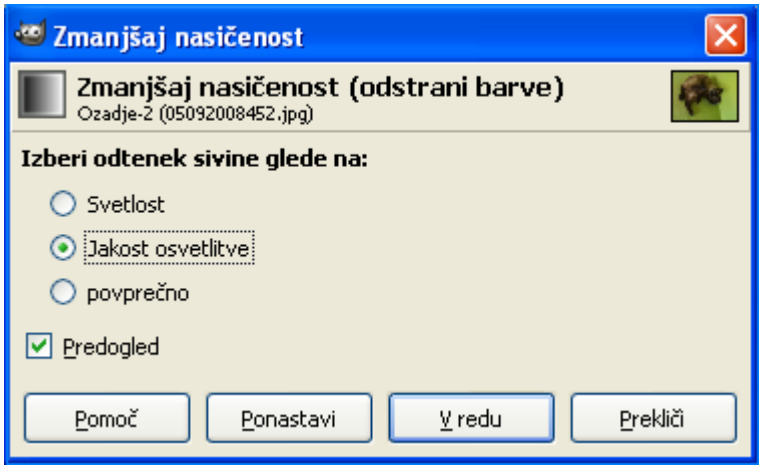
Izberemo samo možnost sepija. Velikost obrobe, ki predstavlja okvir slike, nastavimo na nič. Ostale možnosti so za naš primer nepomembne. S klikom na gumb `V redu` se v oknu obdelovane slike pojavi nova slika s sepija barvnim tonom (Slika 17).



Slika 17: Rezultat ukaza stara fotografija

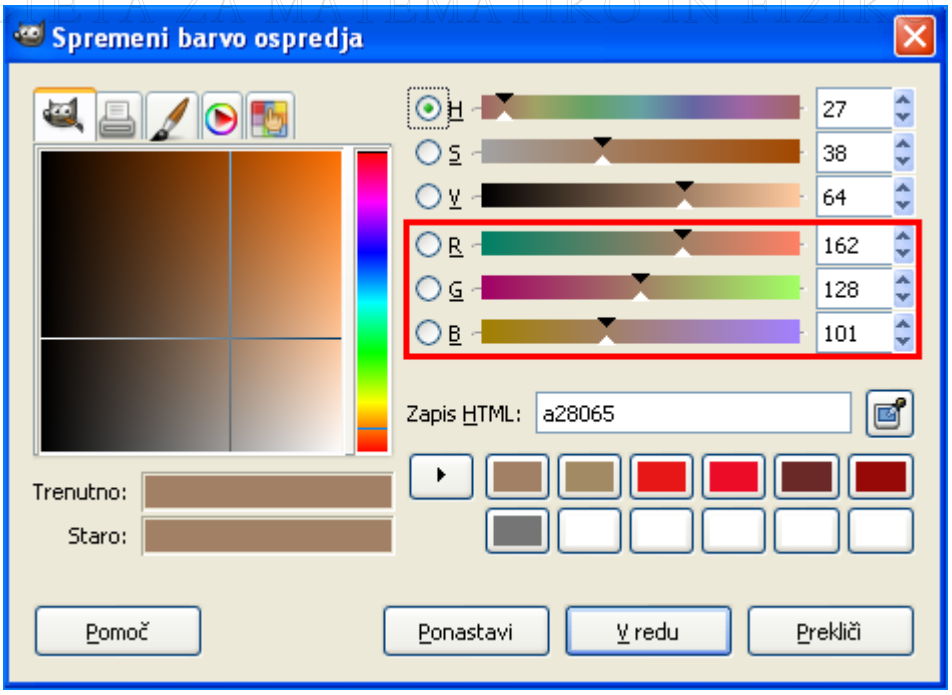
- **DRUGI NAČIN**

Pri tem načinu bomo ukaze izvedli ročno. S pomočjo mask in nastavljanja prosojnosti plasti bomo dosegli želeni učinek.
Najprej spremenimo plast z imenom `ozadje` v črno-belo. To storimo tako, da v slikovnem meniju izberemo meni `barve` in nato izberemo ukaz `zmanjšaj nasičenost`. Odpre se pogovorno okno (Slika 18). V pogovornem oknu opazimo, da lahko izberemo med tremi možnostmi za odtenek sivine. Odtenek sivine lahko izberemo glede na svetlost ali jakost osvetlitve, lahko pa izberemo povprečnega. V splošnem ne moremo reči, kateri način je najboljši, oziroma s katerim dobimo najlepšo črno-belo sliko. Najbolje je, da preizkusimo vse tri možnosti. Če v pogovornem oknu izberemo možnost predogled, vidimo končni rezultat vnaprej. Izberemo način, ki nam najbolj ustreza. V našem primero najlepšo sliko da izbor jakost osvetlitve. To je le ena izmed možnih poti, kako v GIMPu sliko spremeniti v črno-belo.



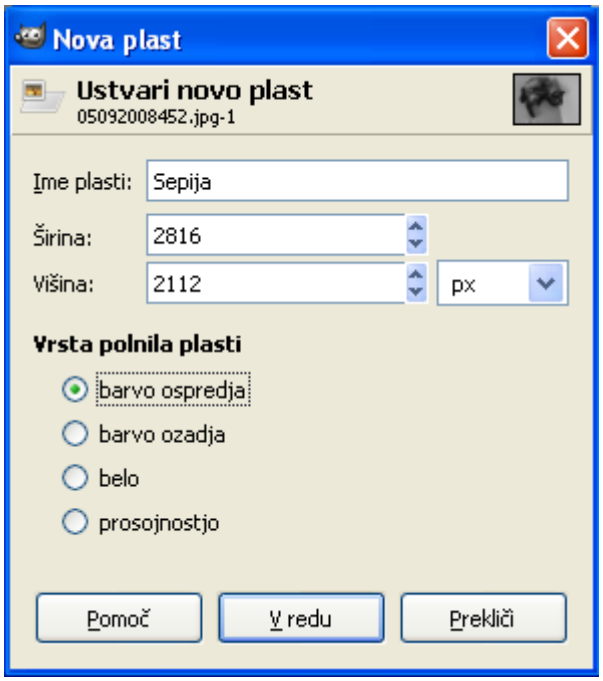
Slika 18: Pogovorno okno za zmanjšanje nasičenosti

Nato določimo barvo ospredja (opisano v razdelku 1.3.3). V pogovornem oknu za spreminjanje barve ospredja (Slika 19) nastavimo ustrezno barvo. Barvo lahko nastavimo z barvnim modelom RGB, z barvnim modelom HSV ali z zapisom HTML. Mi bomo uporabili barvni model RGB. Natančneje je barvni model RGB opisan v razdelku 3.3.1. Vrednosti R, G in B so za dosego videza sepije približno 162 za rdečo, 128 za zeleno in 101 za modro barvo. Za potrditev sprememb kliknemo na gumb `V redu`. Barva ospredja je tako spremenjena.



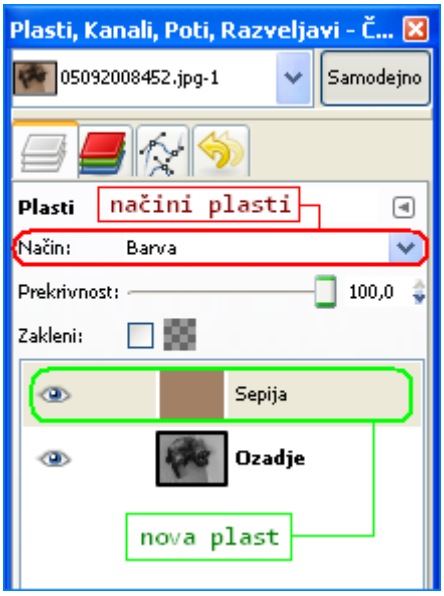
Slika 19: Pogovorno okno za spreminjanje barve ospredja

Zdaj ustvarimo novo plast. V oknu plasti izberemo zavihek `plast`. V spodnjem delu zavihka (kjer se nahajajo bližnjice za upravljanje z zavihki) kliknemo na bližnjico, ki ustvari novo plast (Slika 11). Odpre se pogovorno okno (Slika 20). Ime plasti naj bo `sepija`. Širina in višina nove plasti sta privzeto nastavljeni na širino in višino trenutno obdelovane slike. Višine in širine ne spreminjamo, saj hočemo, da je nova plast enakih dimenzij kot obdelovana slika. Določimo še vrsto polnila plasti. Ker želimo, da bo naša nova plast spremenila fotografijo v sepija barvni ton, izberemo barvo ospredja.



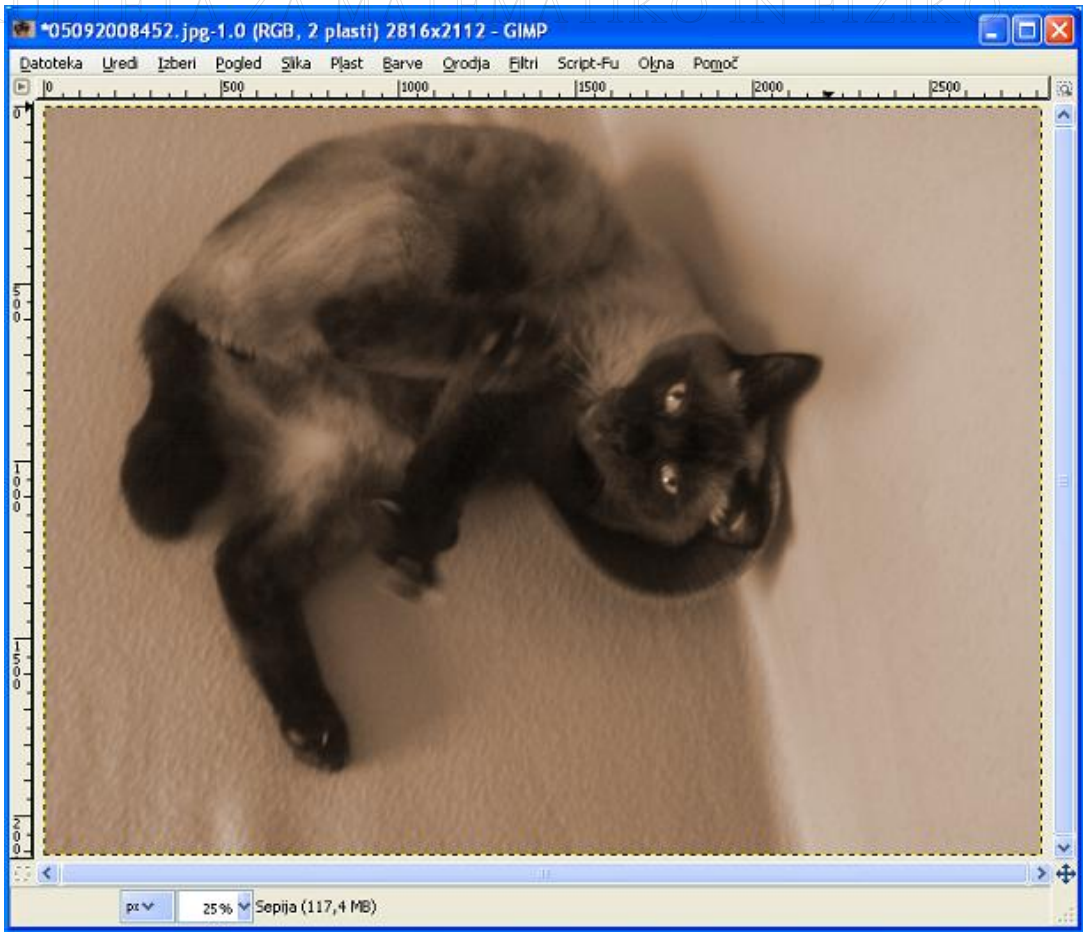
Slika 20: Pogovorno okno za izdelavo nove plasti

S tem se je nova plast z imenom `sepija` pojavila med plastmi v `oknu plasti`. Nastavimo ji način `barva`. Načinov plasti je veliko. Z njimi nastavimo na kakšen način naj obdelovana plast prekriva plast, ki leži pod njo. Z načinom tako določimo način združevanja ali mešanja barv pik gornje plasti z barvami pik, ki se nahajajo na spodnji plasti. Izbrani način pomeni, da zgornja plast (plast `sepija`) določa odtenek in nasičenost, spodnja plast (plast `ozadje`) pa svetlost. Kje se pojavi nova plast `sepija` in kje natančno nastavimo način plasti, prikazuje spodnja slika (Slika 21).



Slika 21: Plast sepija

Končni rezultat je na sliki (Slika 22).



Slika 22: Rezultat drugega načina

Če smo s to precej intenzivno sepija obarvanostjo zadovoljni, lahko z urejanjem končamo in sliko shranimo. Rezultat pa lahko tudi nekoliko izboljšamo. Več o tem v tretjem načinu.

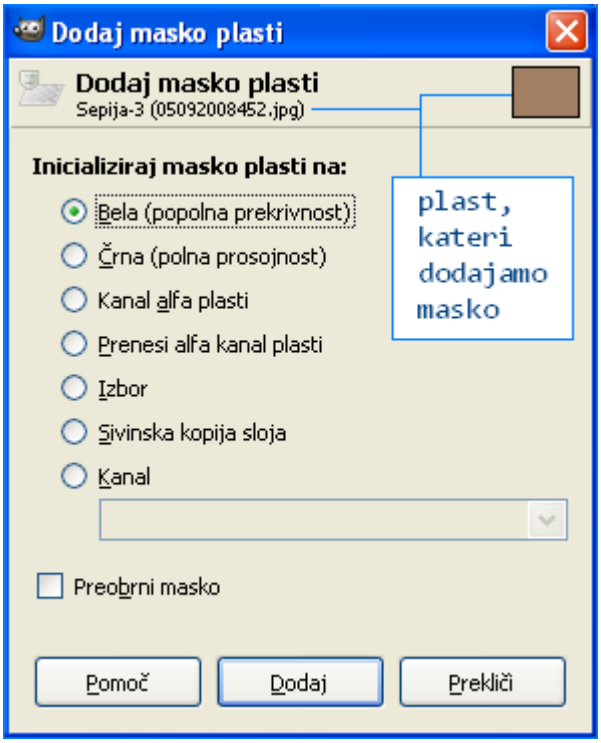
- **TRETJI NAČIN**

Tretji način je pravzaprav nadgradnja drugega načina. Ko zaključimo s postopkom, opisanim pri drugem načinu, se lotimo korakov, ki jih bom opisala v tretjem načinu.

Obstoječi plasti z imenom `sepija` bomo dodali masko plasti, s katero bomo nadzirali količino barve, ki jo sliki doda plast.

Maska plasti v GIMPu opisuje prosojnost plasti. Za opis uporablja črno in belo barvo. Črna barva pomeni, da je tam plast 100 odstotno prosojna, torej se skozi njo popolnoma vidi spodnjo plast. Bela barva pa pomeni, da je tam plast 100 odstotno prekrivna, torej to pomeni, da se skozi njo spodnje plasti ne vidi. Recimo, da smo zgornji plasti dodali belo neprosojno masko. Tam, kjer bo maska ostala bele barve, se skozi njo ne bo videlo spodnje plasti. Na tistem delu maske, kjer bomo masko obarvali s črno, tam bo plast popolnoma prosojna. Če bo maska npr. na nekem delu obarvana z odtenkom sive, ki je sestavljen iz 70 odstotkov črne in 30 odstotkov bele barve, bo tam prosojnost maske 70 odstotna.
Vse maske, ki jih GIMP ponuja, lahko vidimo na sliki Slika 23. Z izjemo omenjenih bele neprosojne maske, se z njimi ne bomo ukvarjali.

Najprej plasti `sepija` dodamo belo neprosojno masko. Z desnim miškinim gumbom kliknemo na plast `sepija` in iz padajočega menija izberemo ukaz `dodaj masko plasti`. Odpre se pogovorno okno (Slika 23). Izberemo belo (popolnoma prekrivno) masko.



Slika 23: Dodajanje maske plasti

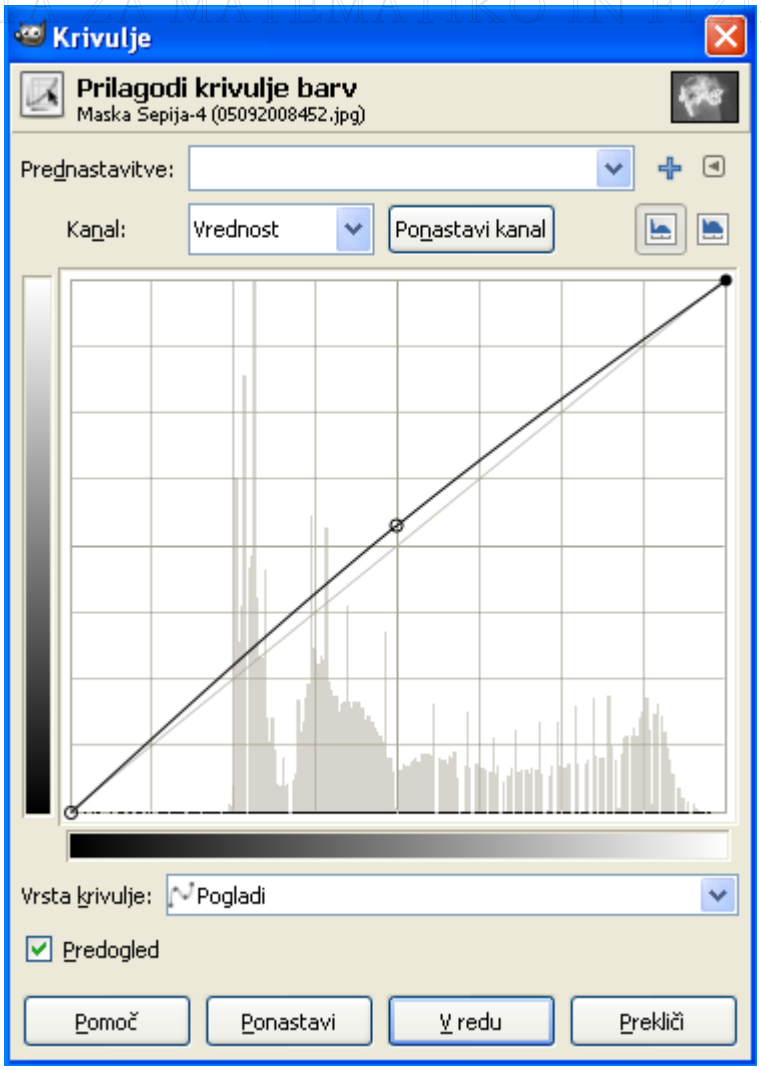
Poleg plasti `sepija` se na njeni desni strani pojavi še znak za belo masko. Nato kopijo plasti `ozadje` prilepimo beli neprosojni maski. S tem ustvarimo masko, s katero bomo dosegli obarvanje slika s sepija barvnim tonom. Do ukazov `kopiraj` in `prilepi` lahko dostopamo z bližnjicama Ctrl + C (kopiraj) in Ctrl + V (prilepi). Najdemo jih tudi v meniju `uredi`, v oknu obdelovane slike. V zavihku plasti se pojavi znak nova ikona, ki predstavlja prilepljeno plast. Imenujemo jo tudi plavajoči izbor. Nato z desnim klikom na plavajoči izbor odpremo padajoči meni in izberemo ukaz `zasidraj plast`. Z ukazom dokončno zlepimo skupaj kopijo plasti ozadje in masko. V masko smo dodali kopijo plasti ozadje. Ker je ozadje črno-belo, bo maska nekje bolj, nekje manj prosojna (odvisno od deleža bele in črne barve). Ker pa je maska dodana plasti sepija, bo barva sepije na svetlih delih fotografije manj intenzivno obarvala fotografijo, na temnih delih pa bolj.

V slikovnem meniju okna obdelovane slike sedaj izberemo meni `barve` in ukaz `preobrni`. S tem poskrbimo, da najbolj temni deli slike dobijo največ, srednje temni toni malo manj, najbolj svetli toni pa nič barve.

Na koncu se poigramo še s krivuljami barv. Te krivulje v GIMPu omogočajo nadzor nad jakostjo barv in osvetljenostjo v vsaki slikovni točki slike. So idealni pripomoček za spreminjanje barv in osvetljenosti slike. V slikovnem meniju izberemo meni `orodja`, podmeni `orodja barv` in nato ukaz `krivulje`. Odpre se pogovorno okno `krivulje` (Slika 24). Izberemo kanal z imenom `vrednost`. Z njim nastavljamo osvetljenost fotografije. Označimo predogled, da sproti vidimo, kaj se dogaja s sliko.

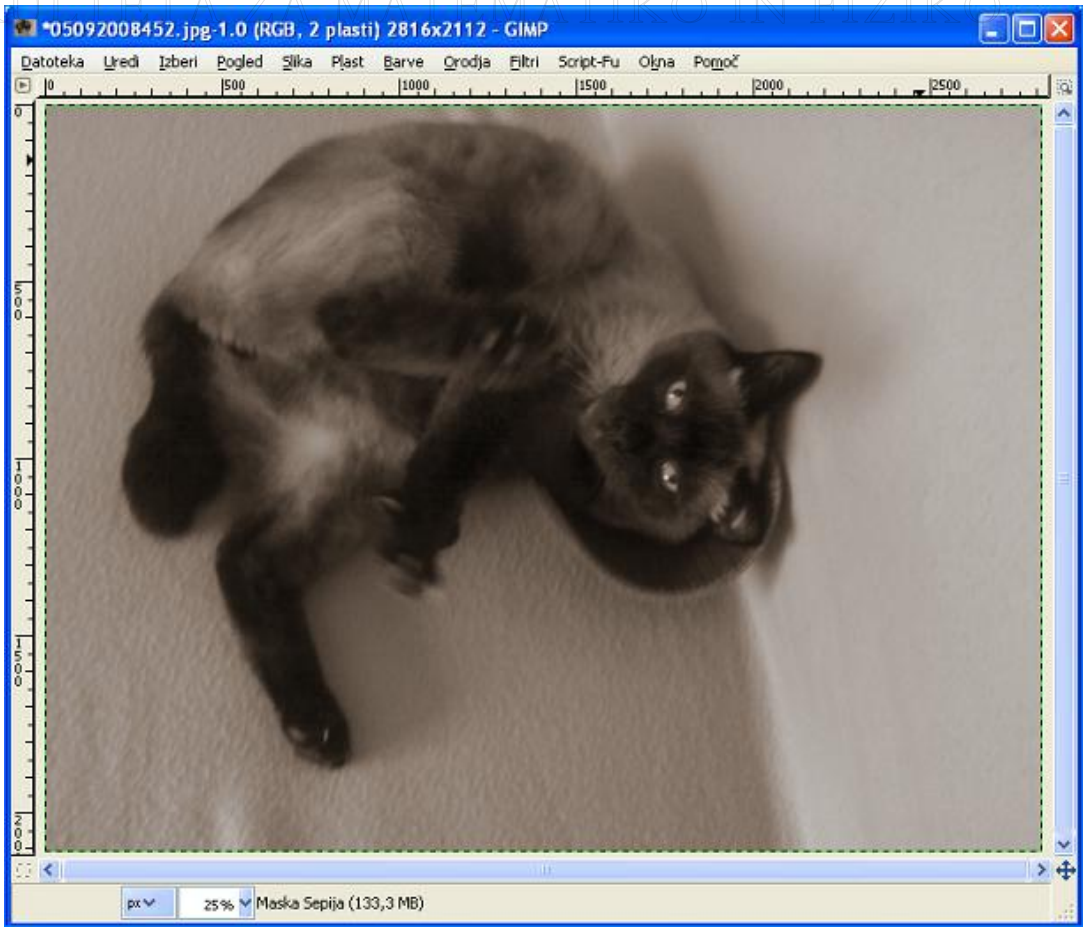
V osrednjem delu okna `krivulje` vidimo graf. Vodoravna os grafa predstavlja lestvico tonov, ki jo ima slika pred urejanjem. Navpična os predstavlja lestvico tonov, ki jo ima slika po urejanju. Od leve proti desni oziroma od spodaj navzgor si toni sledijo od 0 (črna barva) do 255 (bela barva). Graf pred urejanjem vedno poteka od spodnjega levega kota do zgornjega desnega kota.
Graf je pred urejanjem linearen, ker je vsak začetni ton (pred urejanjem) enak končnemu tonu (ker je slika še nespremenjena). Če kliknemo na graf, nanj postavimo točke. Privzeto sta na grafu postavljeni dve točki. Ena na začetku grafa (levi spodnji kot) in ena na koncu grafa (desni zgornji kot). Za osnovno urejanje slike je zadosti, če dodamo eno ali dve točki. Dodamo jih s klikom na želeno mesto na grafu. S premikanjem točk na krivulji spremenimo barve na sliki tako, da je v predogledu rezultat najboljši.
Če s postavitvijo točk na grafu nismo dosegli želenega učinka, lahko s klikom na gumb `ponastavi kanal`, izbrišemo vse točke. Krivulja zopet postane linearna in poteka od levega spodnjega kota do desnega zgornjega kota.

Poleg gumba `ponastavi kanal` se nahajata dva manjša gumba, s katerima izberemo vrsto histograma, ki ga vidimo pod grafom. Prvi gumb predstavlja `linearni histogram`, drugi pa `logaritemski histogram`. Linearni histogram prikazuje število pikslov z določenim tonom osvetljenosti. Na vodoravni osi so toni osvetljenosti, na navpični pa količina pikslov z določeno osvetljenostjo. Linearni histogram uporabljamo, kadar urejamo fotografije. Logaritemski histogram je primeren za urejanje slik z veliko količino pikslov enake barve. Za naš primer ni primeren.



Slika 24: Krivulje barv

Končni rezultat je na spodnji sliki (Slika 25).



Slika 25: Rezultat tretjega načina

2.3 Dodajanje besedila na sliko

Na sliko oziroma fotografijo bi radi dodali napis. Ukaz za dodajanje besedila se nahaja v slikovnem meniju in sicer v meniju `orodja`. Tam najdemo ukaz `besedilo`. Druga krajša pot do ukaza `besedilo` vodi preko okna `orodjarna`, kjer izberemo bližnjico `orodje za besedilo`. V spodnjem delu okna `orodjarna` se odpre zavihek `besedilo` (Slika 26). Besedilu lahko nastavimo vrsto pisave. To storimo s klikom na gumb za določanje vrste pisave. Določimo lahko velikost pisave in enoto za velikost.

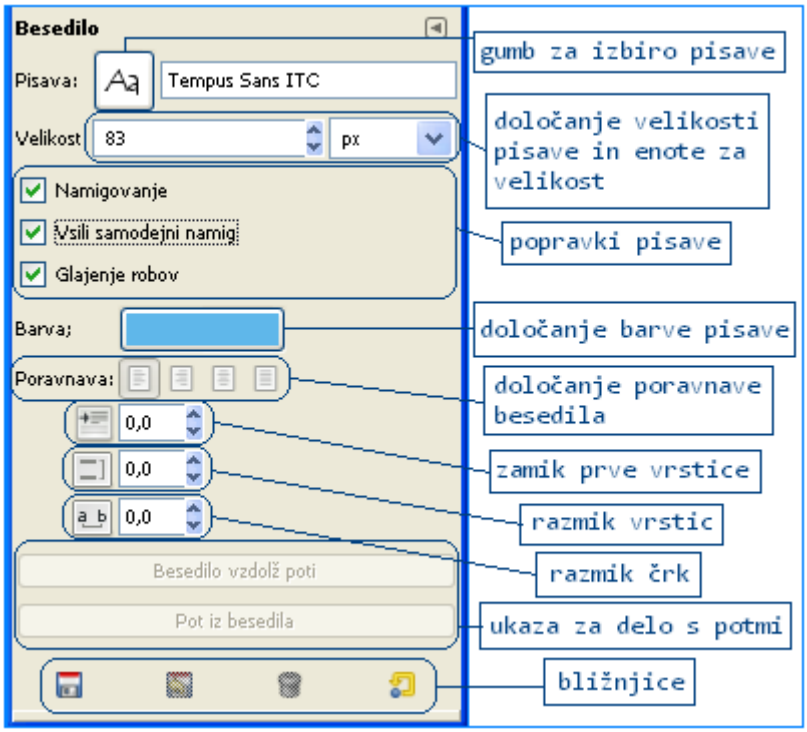
Popravki pisave nam omogočajo samodejno popravljanje obrisa pisave tako, da ostane ostra tudi pri majhnih velikostih. Popravki pisave so uporabni, če recimo želimo narediti zemljevid. Če na fotografiji pokrajine hočemo vpisati veliko mest, vasi, vrhov …, potrebujemo zelo majhno pisavo. Da je majhna pisava normalno berljiva, uporabimo popravke. Med popravki so namigovanje, samodejni namig in glajenje robov. Prvi algoritem programa skuša glede na postavitev črk spremeniti obliko črk tako, da bi bile črke čim bolj razločne. Funkcijo uporabljamo predvsem pri majhni velikosti pisave. Če vklopimo drugo možnost, dosežemo podobno kot pri prejšnjem algoritmu, le na drug način. Algoritem skuša izboljšati izpis znakov. Glajenje robov pa omogoča glajenje robov pisave. To pomeni, da funkcija preoblikuje znake z glajenjem robov in krivulj, pri čemer uporablja zamegljevanje in združevanje robov.

Besedilu lahko določimo tudi barvo pisave. S klikom na okence se prikaže paleta barv, kjer izberemo želeno barvo.

GIMP omogoča tudi poravnavo besedila na sliki (levo, desno, sredinsko in obojestransko). Nastavimo lahko tudi zamik prve vrstice besedila, razmik vrstic in razmik črk.

Skoraj na koncu zavihka se nahajata ukaza za delo s potmi. Ker nas delo s potmi v našem primeru ne zanima, se temu delu ne bomo posvečali.

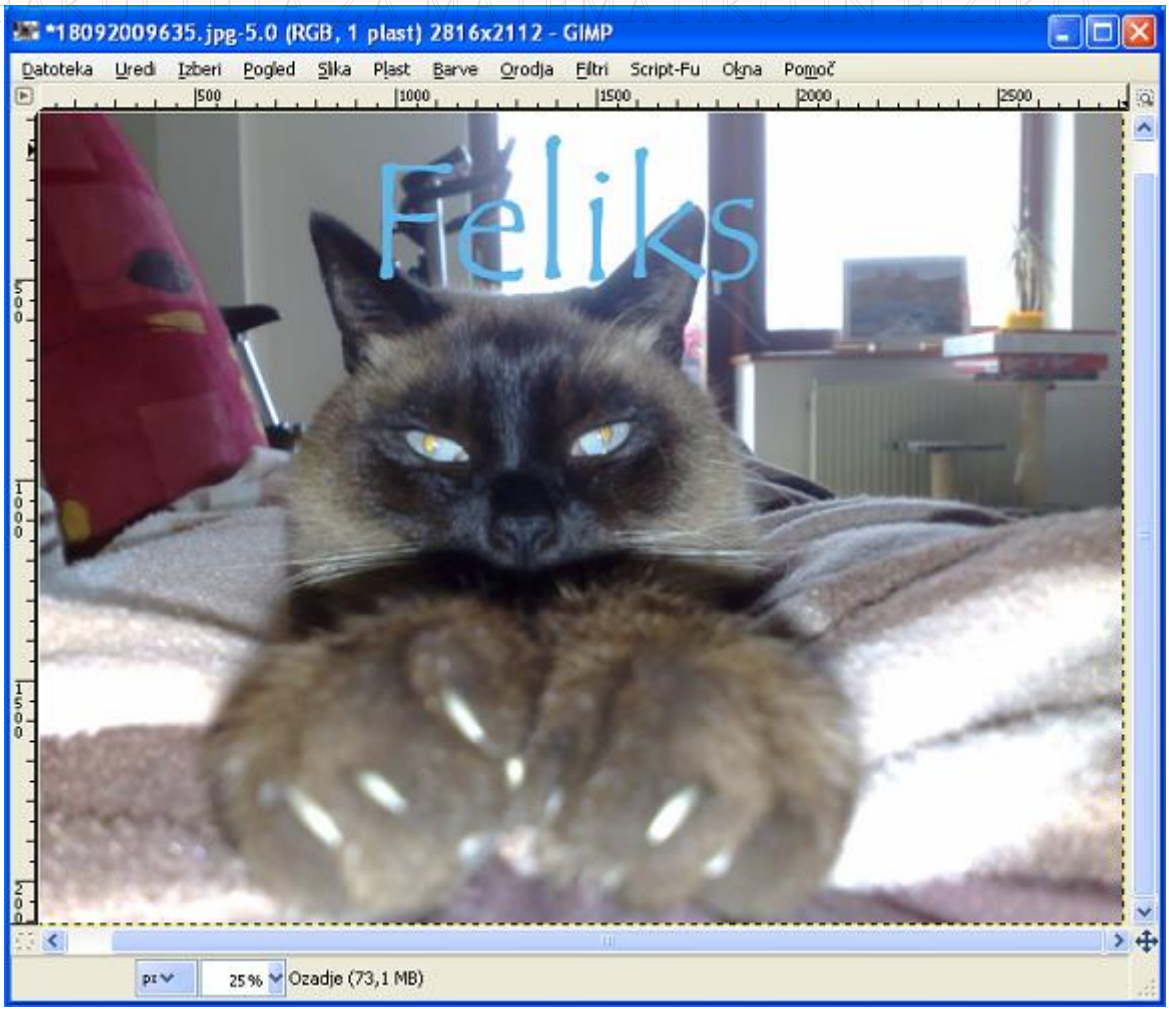
Na koncu zavihka opazimo bližnjice, kjer lahko shranimo trenutne nastavitve besedila. S tem ustvarimo slog, ki ga kasneje lahko uporabimo. Dostopamo do različnih shranjenih oblik besedila, določene oblike lahko izbrišemo. Zadnja bližnjica razveljavi trenutno oblikovanje besedila.



Slika 26: Urejanje besedila

Ko nastavimo pisavo, velikost besedila, poravnavo, barvo pisave …, kliknemo na sliko, odprto v oknu obdelovane slike. Prikaže se okno za vnos besedila. Nato napišemo želeno besedilo. Besedilo se na sliki pojavi v okvirju, ki ga z miško lahko večamo in manjšamo. Besedilo lahko poljubno prestavljamo po sliki. To storimo tako, da kliknemo na sredino okvirja za besedilo in klik zadržimo. Besedilo nato »odnesemo« na želeno mesto in spustimo miškin gumb. Vrstni red operacij lahko tudi obrnemo. Lahko najprej kliknemo na sliko, vpišemo želeno besedilo in ga šele nato oblikujemo. Ko končamo z urejanjem, sliko shranimo.

Končni rezultat našega oblikovanja je prikazan na spodnji sliki (Slika 27).



Slika 27: Besedilo na sliki

3 Osnove skriptnega jezika Scheme

3.1 Kaj je skriptni jezik? Kaj je skripta?

Poleg programskih jezikov kot so Java, C++, C, Fortran, Pascal … so se predvsem v zadnjem desetletju začeli pojavljati tudi skriptni jeziki. Java, C in podobni se od skriptnih jezikov v grobem razlikujejo po tem, da so namenjeni pisanju splošno namenskih programov in da običajno vsebujejo zelo enostavne osnovne ukaze.
Skriptni jeziki pa so namenjeni nekoliko drugačnim nalogam. Namenjeni so predvsem za rokovanje, prilagajanje in samodejno izvajanje določenih opravil v obstoječem sistemu, torej znotraj nekega obstoječega programa. Včasih skriptnemu jeziku rečemo kar »mehanizem za vodenje uporabnosti programa« (vir: http://www.s-sers.mb.edus.si/gradiva/w3/javascript/skript.html). Seveda pa se z razvojem tako prvonavedenih kot skriptnih jezikov te meje med obema skupinama brišejo. Tako v marsikaterem skriptnem programskem jeziku lahko napišemo tudi "splošno namenski program".

Praviloma se skripti jeziki razlikujejo od "pravih" programskih jezikov tudi v tem, da se običajno tolmačijo, medtem ko se drugi prevajajo. Zakaj gre?
Vsi programski jeziki imajo eno skupno značilnost: računalnik jih sam po sebi ne razume. Da bi jih računalnik razumel, jih moramo prevesti v strojno kodo. Pri določenih programskih jezikih za to poskrbi poseben program, ki mu pravimo prevajalnik. Ta prevede program v strojni jezik. Nekateri programski jeziki pa ne uporabljajo prevajalnika, ker jim pri spreminjanju ukazov v strojni jezik pomaga tolmač ali interpreter. Le ta vsako vrstico programske kode sproti prevaja v strojni jezik, jo izvede in nato isto ponovi z naslednjo vrstico. Prednost je v tem, da ne potrebujemo prevajalnika, program je takoj izvršljiv in tudi njegovo spreminjanje in popravljanje poteka hitro, saj vse napake javlja sproti. Uporaba tolmača pa ima tudi svoje slabosti. Izvajanje programa postane precej počasnejše, saj mora tolmač vsako vrstico znova prevesti, četudi jo ponavlja.

Slika 28: Razlika med tolmačem in prevajalnikom (Vir: http://lgm.fri.uni-lj.si/PA/SCRIPTING/HTML/index.html)

Obe vrsti programskih jezikov obstajata vzporedno. Zasledimo ju na skoraj vseh računalniških platformah že v šestdesetih letih. Vloga skriptnih jezikov se povečuje predvsem v zadnjem času, predvsem po razmahu uporabe grafičnih uporabniških vmesnikov in interneta.
Skriptni jezik omogoča pisanje skript. Skripta pomeni program, oziroma niz ukazov, ki jih uporabi oziroma izvede nek drug program.

Skriptni jeziki so uporabni predvsem:

- *pri grafičnih vmesnikih*: cilj skript je čim bolj učinkovito povezovanje elementov grafičnega uporabniškega vmesnika z internimi funkcijami raznih aplikacij.

- *Za pridobivanje informacij z medmrežja*: internet omogoča enostaven dostop do množice raznih podatkov in aplikacij. Idealno orodje, ki omogoča čim lažjo povezavo vseh obstoječih gradnikov kot so recimo spletne strani, strežniki … in s tem dostop do podatkov in aplikacij, je skriptni jezik.

- *Uporabi gradnikov*: gradnike, kot so na primer spletne strani, ki so že tako ali tako sestavljene iz kopice različnih datotek (HTML, CSS, skripte za odjemalca ...), predmeti v programskih strežnikih, podatki v različnih zbirkah, spletne storitve ... in jih pogosto ustvarimo s pomočjo sistemskih programskih jezikov, najlažje povežemo s pomočjo skriptnih orodij.

- *Znotraj programov za splošno rabo*: z množično uporabo računalnikov si je tudi vse več ljudi želelo poenostaviti izvedbo določenih opravil. Tudi vse več aplikacij je podpiralo vsaj neko obliko skriptnega programiranja. Tako si danes lahko skoraj vsakdo napiše skripto (čemur bi v okolju windows rekli makro) v najljubši preglednici ali urejevalniku besedil. S tem na primer na lažji način doseže, da se denimo v Wordu vse velike začetnice besed napišejo polkrepko.

Poznamo mnogo skriptnih jezikov. Med njimi so najbolj znani Perl, Php, Python, JavaScript, Tcl, Lisp, …

3.2 Zakaj Scheme?

Kot mnogo drugih programov, lahko tudi GIMP s skriptami prikrojimo za enostavnejše delo. To pomeni, da s skriptami lahko delo v GIMPu avtomatiziramo. Recimo, da smo se odločili, da bomo dvajsetim fotografijam spremenili velikost, jim dodali svoj podpis in jih spremenili v črno – bele. Namesto, da na vsaki sliki naredimo te tri stvari (vsak ukaz, ki je potreben pri teh treh opravilih, bomo morali torej izbrati in zagnati dvajsetkrat), napišemo skripto. Vanjo zložimo vse ukaze, ki jih je potrebno opraviti nad posamezno sliko. Nato le še pokličemo skripto nad vsako sliko. Ukazi se bodo sedaj izvedli avtomatsko. Lahko pa napišemo še eno skripto, ki denimo poskrbi, da se prva skripta izvede nad vsem slikami v določenem imeniku. Vse delo z obdelavo slike pripravimo le enkrat, nato pa le še pokličemo glavno skripto. Naprej je postopek avtomatiziran.

GIMP podpira več različnih skriptnih jezikov. Eden izmed njih je jezik Scheme. Izhaja iz družine jezikov, ki temeljijo na jeziku Lisp (List Processing). Lisp je programski jezik za funkcijsko programiranje in simbolično obdelavo podatkov, razvit za potrebe umetne inteligence. Njegovi začetki segajo v drugo polovico sedemdesetih let. Scheme je imenovan tudi »narečje« programskega jezika Lisp. Kot smo omenili, to ni edini skriptni jezik, s katerim lahko programiramo v GIMPu. Programiramo lahko tudi v jezikih Perl in Tcl. Vendar je Scheme edini skriptni jezik, ki je v programu GIMP že v osnovi podprt.

3.3 Skript - Fu skripte

V GIMPu skripte pisane v jeziku Scheme, imenujemo Skript – Fu skripte. Skript – Fu je pravzaprav dialekt jezika Scheme, torej posebna oblika jezika Scheme pripravljena posebej za GIMP.

Kot sem omenila v razdelku 3.1 (Kaj je skriptni jezik? Kaj je skripta?), so skripte uporabne na več področjih. Običajni uporabnik bo po njih posegel predvsem za avtomatizacijo stvari, ki jih v GIMPu počne pogosto. Prav tako si bo s skriptami olajšal izvedbo zapletenih akcij (npr. odstranjevanje skritih črt), katerih postopke si je težko zapomniti.

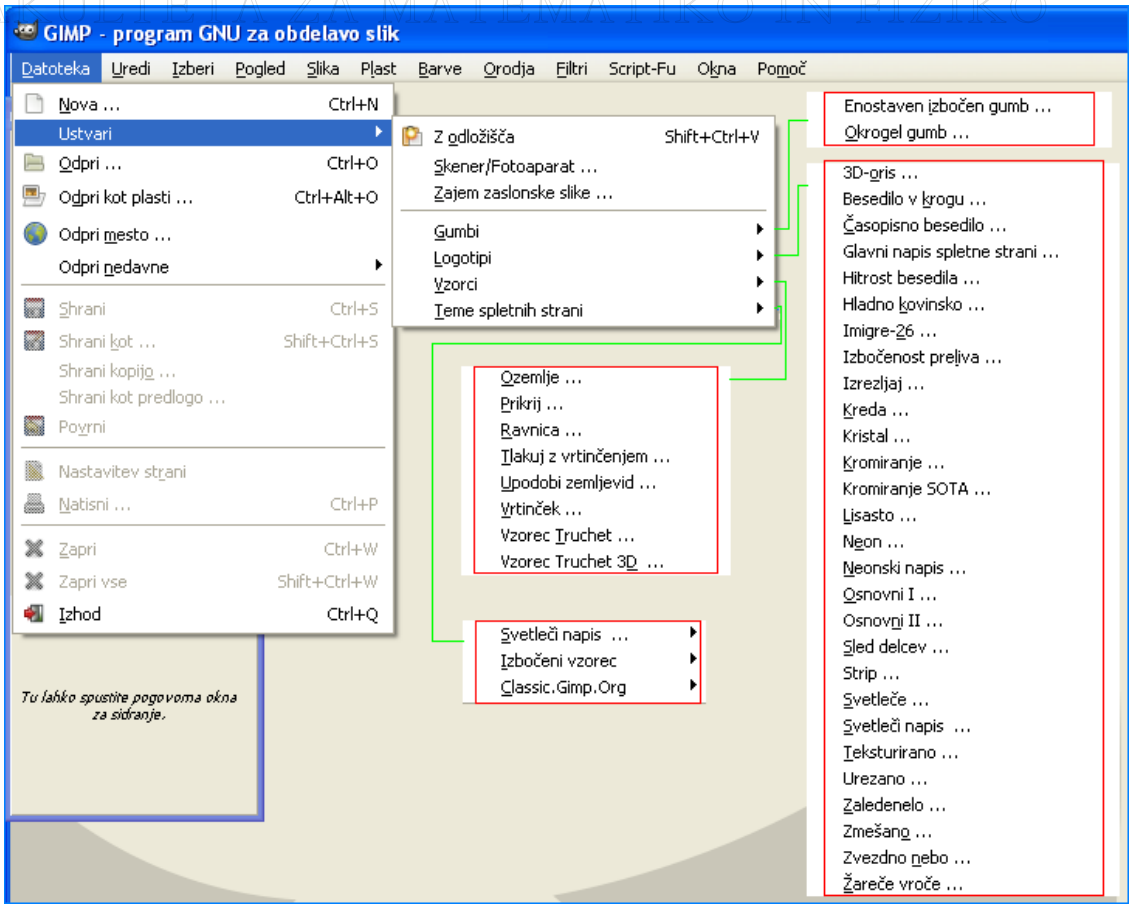
Veliko skript je v GIMP že vgrajenih. Posledica tega, da je GIMP odprtokoden program, je tudi ta, da ima precej veliko množico uporabnikov. Ti v skladu z odprtostjo delijo svoje skripte z drugimi. Zato lahko poleg vgrajenih skript še več teh prenesemo z raznih spletnih strani. Svoboda prenašanja z medmrežja pa ima tudi svoje slabosti. Ker lahko praktično vsak objavi svoje skripte na internetu, se pogosto srečujemo z nekvalitetnimi skriptami.
Poleg uporabe vgrajenih skript in skript, ki jih dobimo na medmrežju, se ponuja še tretja možnost. Skripte lahko napišemo tudi sami. Za to lahko uporabimo poljuben tekstovni urejevalnik.

Skripte v GIMPU so torej iz treh virov:

- Vgrajene Skript-FU skripre
- Pridobljene od drugih uporabnikov
- Naše lastne skripte

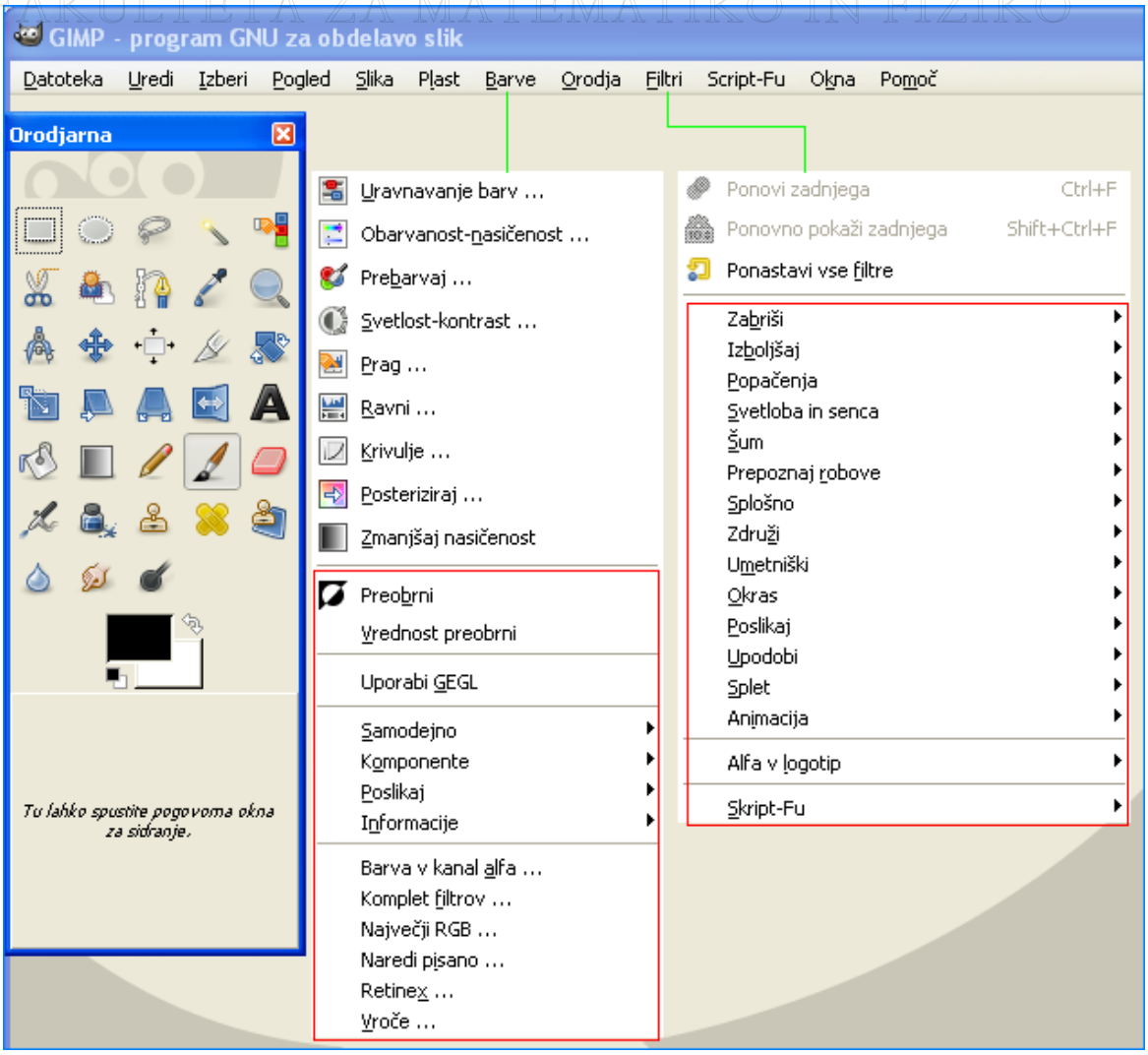
Vgrajene Skript – Fu skripte delimo na:

- Samostojne Skript - Fu skripte (Standalone Script – Fus) se nahajajo v meniju `datoteka`, v podmeniju `ustvari`. Med samostojne skripte štejemo tiste skripte, ki za svoje delovanje ne potrebujejo slik oziroma fotografij. To pomeni, da ukazi v skripti niso napisani z namenom, da bi spreminjali sliko, temveč zato, da z njimi ustvarjamo nove objekte. Take skripte so namenjene recimo ustvarjanju gumbov, logotipov, raznih vzorcev, podlag za spletne strani … Kje jih najdemo v programu GIMP, nam pove slika Slika 29.



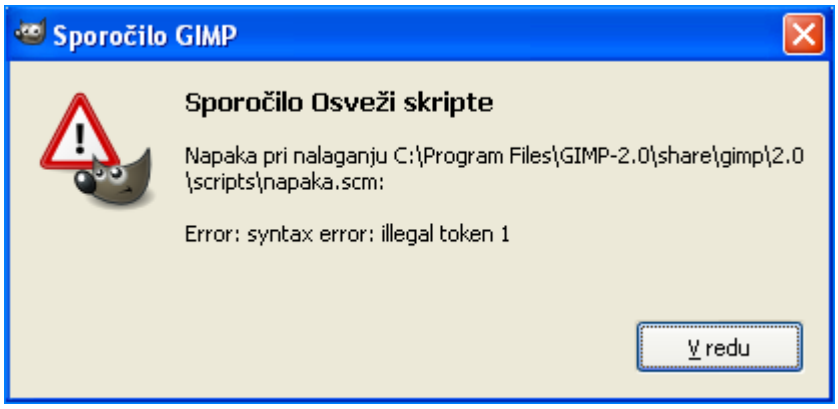
Slika 29: Samostojne Skript-Fu skripte

- Skript - Fu skripte delujoče na slikah (Image dependent Script – Fus) pa se nahajajo v menijih `barve` in `filtri`. Na sliki Slika 30 so označene z rdečim okvirjem. Te skripte pa se vedno nanašajo na določeno sliko oz. fotografijo. Napisane so z namenom, da spreminjajo lastnosti slike. Tako določeno sliko s tem, da zaženemo (izvedemo) skripto, lahko posvetlimo, jo spremenimo v črno – belo sliko, ji spremenimo barve …



Slika 30: Skript-Fu skripte delujoče na slikah

Skripte, ki smo jih pridobili na internetu, moramo pred uporabo shraniti v ustrezno mapo. Vse skripte moramo shraniti v mapo, kjer se že nahajajo privzete skripte. Skripte so sestavljene iz ene same datoteke. Sistemska pot do mape, v katero jih shranjujemo, je: `C<mapa, kjer je nameščen GIMP>\share\GIMP\2.0\scripts` (npr. `C:\Program Files\GIMP-2.0\share\GIMP\2.0\scripts`). Ko smo v ta imenik shranili novo skripto, moramo program GIMP osvežiti. Znotraj menija `filtri` se nahaja podmeni `Skript - Fu`, kjer najdemo ukaz `osveži skripte`. S tem bo GIMP samodejno zaznal nove skripte. Nato preveri njihovo sintakso. Tiste skripte, ki so brez sintaktičnih napak, se pojavijo v enem izmed menijev v GIMPu. Meni bo tisti, ki ga bomo določili v tako imenovanem registracijskem delu skripte - v `script-fu-menu-register`. Tam funkcija kot parameter sprejme ime glavne funkcije in lokacijo mape v kateri se nahaja skripta. Primeri so v poglavju 4 (Skripte za primere iz drugega poglavja).
Če določena skripta, ki smo jo dali v zgoraj omenjeno mapo, ni sintaktično pravilna, potem pri osveževanju GIMP javi napako. Če hočemo dodati več skript in ena izmed njih ni sintaktično pravilna, potem se tista skripta ne naloži v meni, kamor jo želimo dodati. Vse ostale skripte se naložijo normalno.
Dokler napake v skripti ne popravimo, oziroma dokler skripte ne odstranimo iz ustreznega datotečnega sistema, bo GIMP ob vsakem osveževanju javljal napako, da določena skripta ni sintaktično pravilna. Vse ostale skripte lahko nemoteno uporabljamo.



Slika 31: Obvestilo o sintaktični napaki

3.3.1 Sestava Skript – Fu skripte

Vsaka Skript – Fu skripta vsebuje vsaj eno funkcijo . To funkcijo imenujemo glavna funkcija skripte. Tudi kadar skripta vsebuje več funkcij, je glavna funkcija še vedno ena sama. Vse ostale funkcije v skripti so pomožne funkcije. Glavna funkcija za delovanje uporablja pomožne funkcije. Zapis vsake funkcije (tudi glavne) je v splošnem videti takole:

```
(define (script - fu - imeFunkcije parameter1 parameter2
parameter3 ...))
```

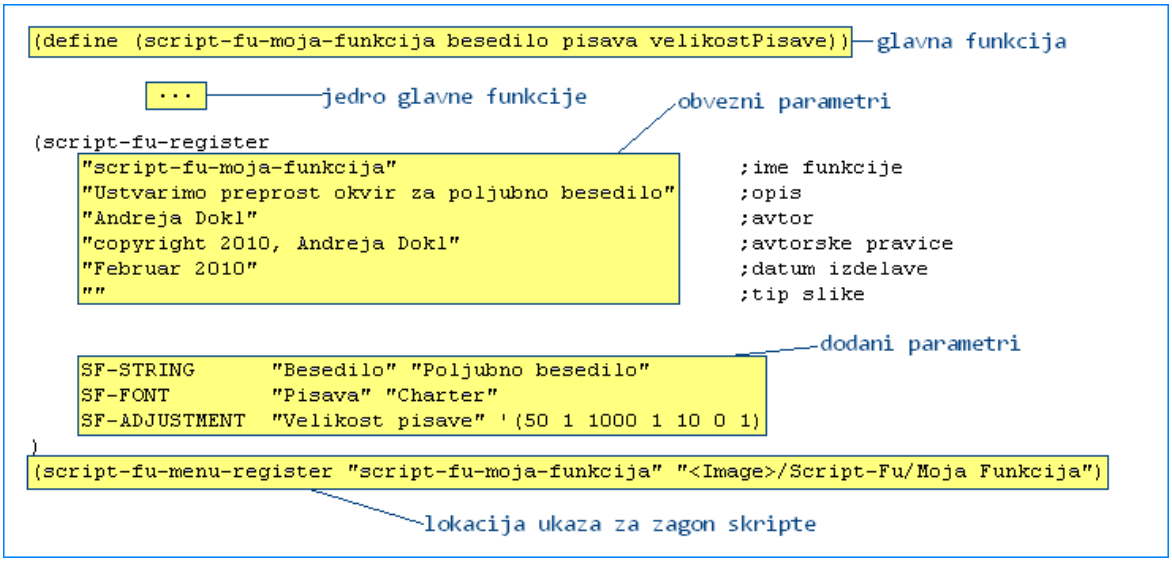
Pomemben del vsake Skript – Fu skripte je tudi registracija glavne funkcije, saj brez nje skripta ne deluje. Registriramo vedno samo glavno funkcijo. Registracijo glavne funkcije začnemo s klicem funkcije (več o tem je napisano v razdelku 3.4.2 (Osnovna sestava in klic funkcije)) za registracijo: `script - fu - register`. Nato napišemo sedem obveznih parametrov. Na koncu dodamo še parametre glavne funkcije z opisom in vrednostjo za vsak parameter posebej.

Obvezni parametri so:

- **IME FUNKCIJE** je ime glavne funkcije
- **OPIS SKRIPTE** je kratek opis, čemu je skripta namenjena. Opis skripte bo prikazan v brskalniku procedur. Ta se v GIMPu nahaja v meniju Pomoč. Če iz seznama na levi izberemo poljubno skripto, se v pogovornem oknu na desni izpiše opis skripte.
- **IME AVTORJA** je ime avtorja skripte.
- **INFORMACIJE O AVTORSKIH PRAVICAH** nam povedo, če je skripta avtorsko zaščitena.
- **DATUM IZDELAVE** je datum izdelave skripte oziroma datum zadnjega popravka.
- **TIP SLIKE** lahko določimo samo skriptam, ki jim, kot smo opisali v poglavju 3.3 (Skript –Fu skripte), rečemo skripte delujoče na slikah. Pri tako imenovanih samostojnih skriptah tipa ni mogoče določiti. Poznamo naslednje tipe: RGB, RGBA, GRAY, GRAYA,

INDEXED in INDEXEDA. Tip določimo glede na barvni model, ki ga v skripti uporabljamo pri obdelavi slike. Barvni modeli so na kratko opisani na koncu tega razdelka.

Na koncu s funkcijo `script-fu-menu-register` določimo, v katerem meniju bomo našli ukaz za zagon skripte.



Slika 32: Registracija skript

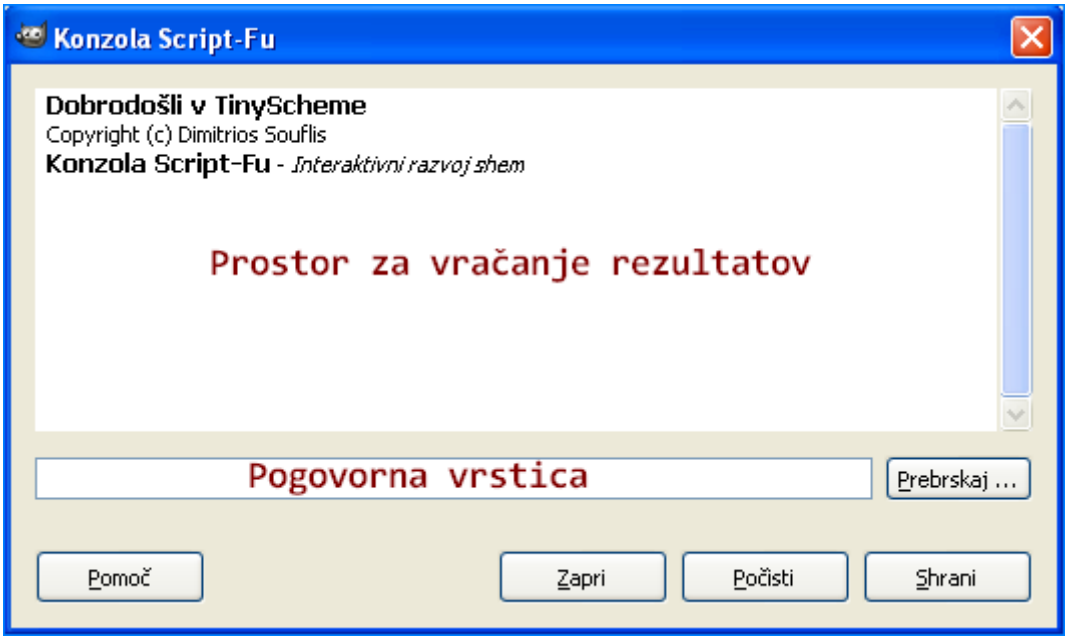
Bolj podrobno bom registracijo opisala v poglavju 0 (Skripte za primere iz drugega poglavja), kjer se bom posvetila trem različnim primerom skript.

Ker bomo v nadaljevanju pogosto govorili o različnih barvnih modelih, jih tukaj čisto na kratko opišimo.

- Pri barvnem modelu RGB barvo zapišemo z vrednostmi treh komponent: rdeče, zelene in modre. Princip je podoben zaznavanju, kot ga opravijo celice človeškega očesa. Ta princip mešanja barv uporabljajo tudi digitalni fotoaparati, ekrani …
- Model RGBA dobimo z dodanim kanalom alfa (prosojnost). Kanal alfa je maska, ki nosi podatke o prosojnosti (kaj je prosojnost, je razloženo v razdelku 2.2 (Ustvarjanje fotografije s sepija barvnim tonom)) slikovnih točk na plasti, kateri je bil alfa kanal dodan.
- Model GRAY uporabljamo za sivinske slike. Sivine so predstavljene z enim samim zlogom. Ker ima en zlog lahko vrednosti med 0 in 255, obstaja 256 različnih nivojev sivine.
- Model GRAYA dobimo, če modelu GRAY dodamo kanal alfa.
- Model INDEXED običajno uporabljamo za skice in risbe. Za fotografije ni primeren. Model INDEXED je omejen na 256 poljubnih barv. Barve so 24-bitne. Vsaka izmed teh barv je zapisana z modelom RGB. Vsaka slika v modelu INDEXED vsebuje tudi paleto vseh 256 izbranih barv. V vsaki slikovni točki (pikslu) se je podatek o barvi, ki jo slikovna točka vsebuje. Ta podatek pravzaprav pove, kje v paleti se barva slikovne točke nahaja. Torej pove indeks barve v paleti. Od tod ime INDEXED.
- Tudi model INDEXEDA dobimo tako, da modelu INDEXED dodamo kanal alfa.

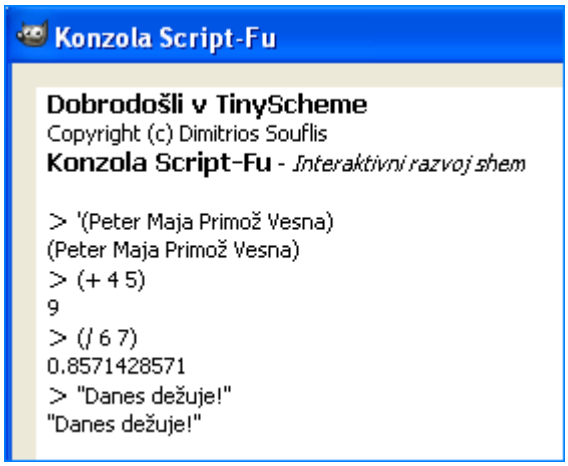
3.4 Osnove jezika Scheme

Kot pri učenju vsakega programskega jezika, tudi za učenje jezika Scheme velja, da vaja dela mojstra. Torej nas bo najprej zanimalo, kje svoje znanje lahko preizkusimo. GIMP nam ponuja interaktivno okno, ki mu rečemo tudi konzola. V meniju `filtri` izberemo podmeni `script – Fu`. V tem podmeniju najdemo ukaz `konzola`. S klikom na ukaz `konzola` se odpre okno s pogovorno vrstico in s prostorom za vračanje rezultatov.

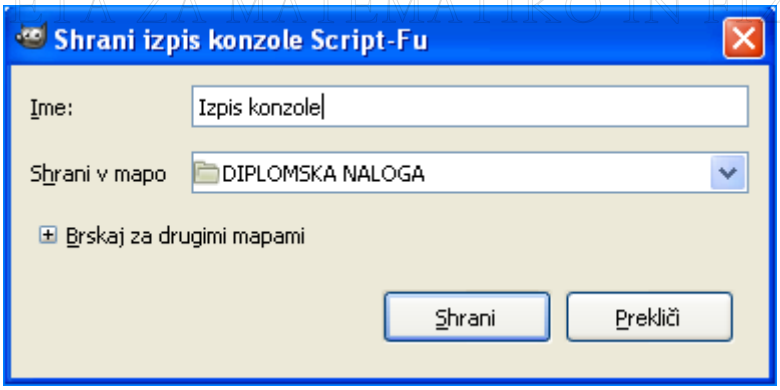


Slika 33: Pogovorno okno ali konzola

Poglejmo si gumbe na konzoli Script – Fu. V spodnjem desnem kotu se nahajajo trije gumbi: `Zapri`, `Počisti` in `Shrani`. S klikom na gumb `Zapri` se konzola zapre. Če kliknemo na gumb `Počisti`, se počisti vsebina v prostoru za vračanje rezultatov. S klikom na gumb `Shrani` pa na datoteko shranimo celoten zapis, ki se nahaja v prostoru za vračanje rezultatov.



Slika 34: Zapis v prostoru za vračanje rezultatov



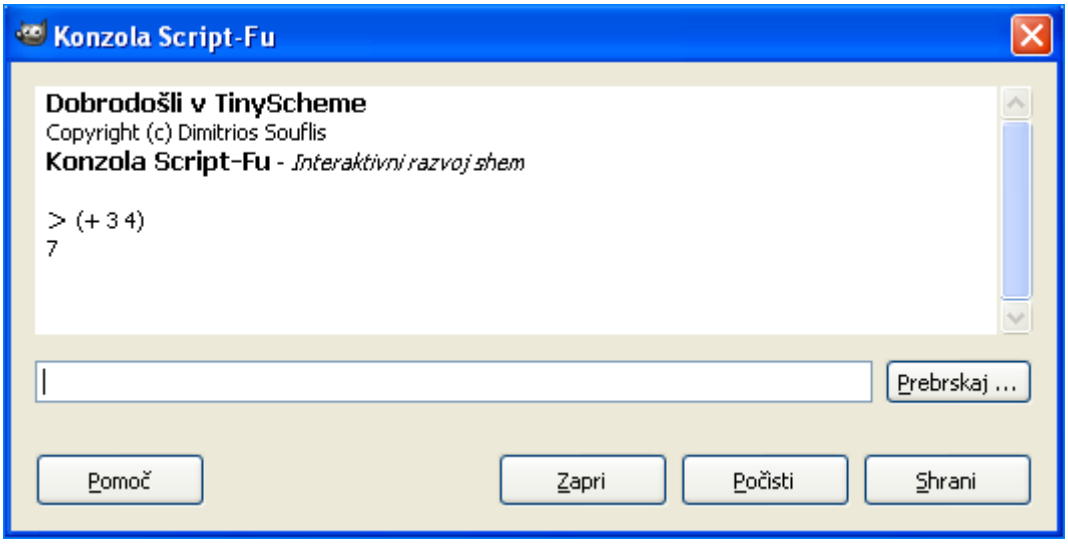
Slika 35: Okno za shranjevanje

```
Dobrodošli v TinyScheme
Copyright (c) Dimitrios Souflis
Konzola Script-Fu - Interaktivni razvoj shem

> '(Peter Maja Primož Vesna)
(Peter Maja Primož Vesna)
> (+ 4 5)
9
> (/ 6 7)
0.8571428571
> "Danes dežuje!"
"Danes dežuje!"
```

Slika 36: Shranjen izpis konzole, odprt v urejevalniku besedil NotePad

Levo spodaj v konzoli se nahaja gumb `pomoč`. S klikom nanj se povežemo na spletno stran http://docs.GIMP.org/2.6/C/GIMP-filters-script-fu.html#plug-in-script-fu-console, kjer je napisano vse o uporabi konzole. Pogovorna vrstica je namenjena vnosu ukazov.

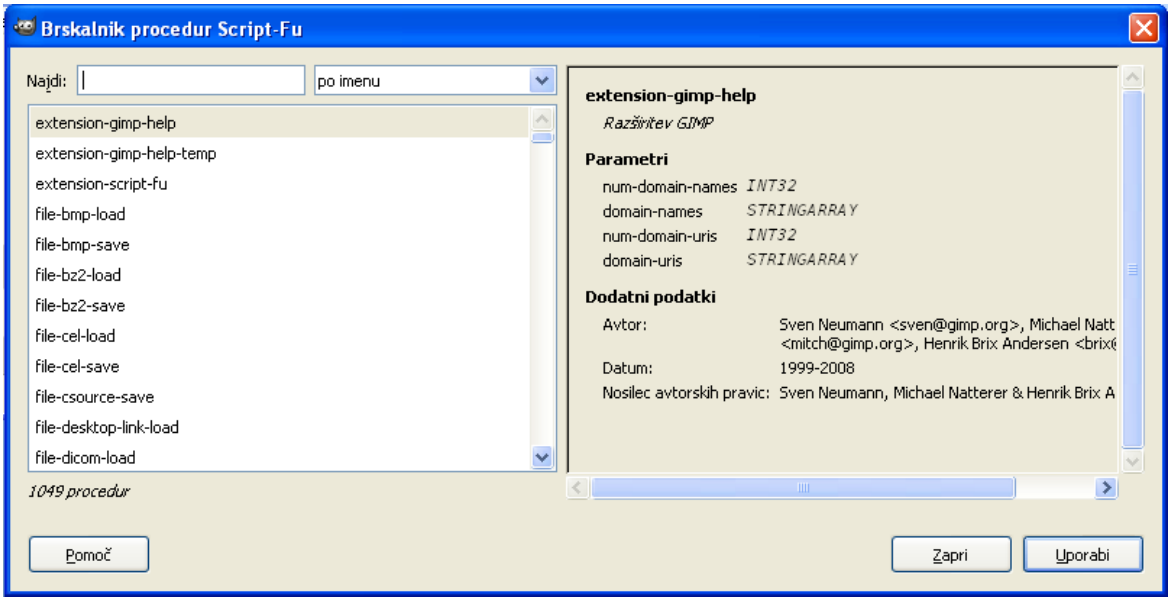
Oglejmo si enostaven zgled. V konzoli lahko računamo. Želimo sešteti števili 3 in 4. V pogovorno vrstico vtipkamo `(+ 3 4)`. S pritiskom na tipko Enter izvedemo ukaz.



Slika 37: Vsota dveh števil

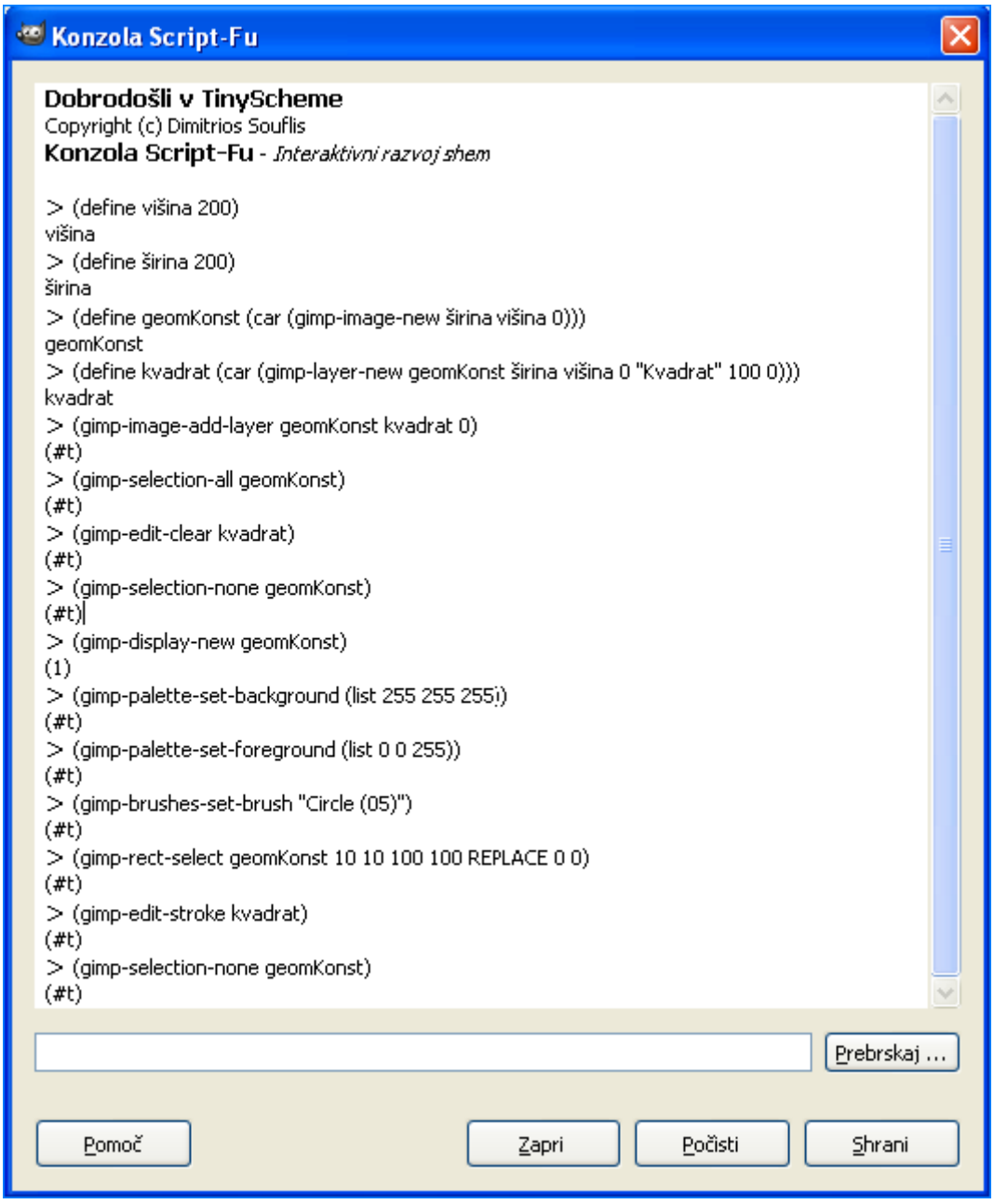
Kot vidimo, pri vnosu izrazov uporabljamo drugačen zapis, kot smo navajeni. Izraze namreč pišemo v prefiksni notaciji. Več o njej bom razložila v razdelku 3.4.1 (Osnovna pravila skriptnega jezika Scheme).

V pogovorni vrstici lahko kot ukaz uporabimo tudi klic poljubne skripte, ki je vgrajena v GIMP ali pa smo jo vanj dodali Ker se verjetno na pamet ne spomnimo vseh skript, si lahko pomagamo z gumbom `Prebrskaj`. S klikom nanj se odpre okno z vsemi funkcijami, do katerih lahko dostopamo preko konzole. Rečemo mu tudi brskalnik procedur. Najprej iz seznama na levi izberemo funkcijo, ki jo želimo uporabiti. Na desni vidimo njen opis. Nato kliknemo gumb `Uporabi` in funkcija se bo pokazala v pogovorni vrstici konzole.



Slika 38: Brskalnik procedur

Oglejmo si, kako z ukazi narišemo kvadrat. Da bomo to dosegli, bomo morali v konzolo vnesti kar 15 ukazov. Te si lahko ogledamo na Slika 39 (Risanje kvadrata).



Slika 39: Risanje kvadrata

Kot vidimo, zadeva sploh ni tako enostavna. Oglejmo si, kaj smo počeli:

Najprej smo ustvarili spremenljivke (več o tem v razdelku 3.4.2 (Spremenljivke in funkcije)), ki jih bomo potrebovali in jim priredili vrednosti. V spremenljivki `višina` in `širina` smo shranili vrednost 200, s katerima smo določili velikost okna, namenjenega risanju.

V ukazih, ki sledijo, smo si pomagali s funkcijami, ki so v GIMPu že vgrajene (`GIMP-image-add-layer`, `gimp-edit-stroke`, `gimp-palette-set-background` ...). To smo storili s klicem funkcije, ki je v splošnem videti takole: `(imeFunkcije parameter1, parameter2, parameter3 ...)`. Več je napisano v razdelku 3.4.2 (Spremenljivke in funkcije).

Po definiciji spremenljivk smo z ukazom `(define geomKonst (car (gimp-image-new sirina visina 0)))` ustvarili novo sliko in jo shranili v spremenljivko `geomKonst`. Sliko smo ustvarili s klicem funkcije `gimp-image-new`. Ta zahteva parametre. S parametroma `sirina` in `visina` smo povedali, koliko naj bo slika široka in koliko visoka, s tretjim parametrom z vrednostjo `0` pa, da bo slika uporabljala barvni model RGB.

Podobno smo s pomočjo funkcije `gimp-layer-new` ustvarili (in shranili v spremenljivko `kvadrat`) plast (layer). Kaj je plast, je opisano v razdelku 1.3.3 (Videz programa GIMP).

Sledi ukaz: `(define kvadrat (car (gimp-layer-new geomKonst širina višina 0 »Kvadrat« 100 0)))`. Z argumentom `geomKonst` smo povedali, da ta plast pripada sliki z imenom `geomKonst`. Njeni dimenziji višina in širina sta enaki dimenzijam slike. Z argumentom `0` smo povedali, da smo uporabili barvni model RGB, z imenom `»Kvadrat«` pa smo poimenovali novonastalo plast. Argument, ki sledi, pomeni prosojnost plasti in se meri v odstotkih. Ker smo želeli, da bo plast vidna, smo prosojnost nastavili na 100 %. Zadnji argument 0 pomeni način, kako vidimo plast. Uporabljena vrednost 0 pomeni, da plasti nastavimo način `barva` (glej razdelek 2.2 (Ustvarjanje fotografije s sepija barvnim tonom (drugi način))). … Pri ustvarjanju spremenljivk `geomKonst` in kvadrat smo uporabili še ukaz `car`. Ta omogoča, da iz seznama izločimo posamezne vrednosti. Ukaz `gimp-image-new` namreč kot rezultat vrne ID slike in smo ga prej lahko uporabili neposredno. Ukaz `gimp-layer-new` pa kot rezultat vrne seznam. Ta seznam ima sicer en sam element, vendar ukaz `define` pričakuje vrednost in ne seznam. Za dostop do elementov seznama zato uporabimo ukaz `car`. Več o tem si bomo ogledali v razdelku 3.4.2 (Urejeni pari in seznami).

Z ukazom `(gimp-image-add-layer geomKonst kvadrat 0)` smo dodali plast na sliko, z argumentom 0 pa smo povedali, kakšen je položaj plasti na sliki.Vrednost 0 pomeni, da dodajamo plast, ki nima dodanega alfa kanala.

Da smo počistili zgodovino risanja, smo uporabili ukaze: `(gimp-selection-all geomKonst)`, s katerim izberemo celotno področje slike, z ukazom `(gimp-edit-clear kvadrat)` počistimo plast in z `(gimp-selection-none geomKonst)` zaključimo označevanje. Zgodovino pobrišemo zato, ker bi na plasti `kvadrat` že lahko bilo kaj narisano. Če zgodovine risanja ne bi pobrisali, bi ob ponovni uporabi ukazov za risanje le dodali novo narisan predmet. Predmeti, ki smo jih narisali pred tem, bi ostali na sliki.

Nato z ukazom (`gimp-display-new slika`) prikažemo polje za risanje na zaslonu. V glavnem oknu programa se je ustvarilo polje (kvadrat) z dimenzijama višina in širina.

Nato smo določili še barvo ozadja in barvo za risanje kvadrata. Prvo storimo z `(gimo-palette-set background (list 255 255 255))`, drugo pa z `(gimp-palette-set foreground (list 0 0 255))`. Obe funkciji uporabljata kot argumenta seznam (od tod `»list«`) treh števil. To so ustrezne vrednosti rdeče, zelene in modre barve barvnega modela RGB. Barva ozadja je bela, barva kvadrata pa modra.

Za risanje smo uporabili še čopič: `(GIMP-brushes-set-brush (»Circle (05)«))`. Argument pove, da smo izbrali okrogel čopič debeline 5.

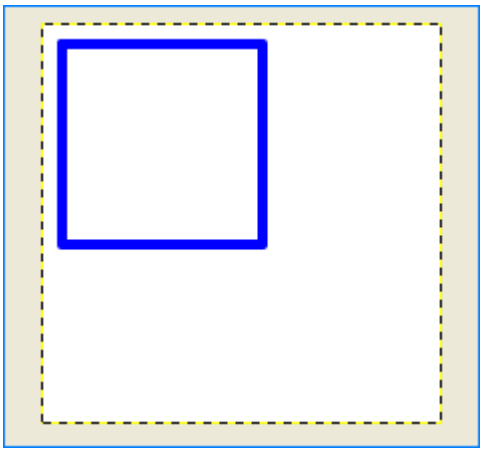
Nato smo uporabili ukaze za risanje. Z `(GIMP-rect-select geomKonst 10 10 100 100 REPLACE 0 0)` na sliki smo izbrali območje v obliki kvadrata. Drugi argument je koordinata x levega zgornjega kota kvadrata, tretji argument pa je koordinata y levega zgornjega kota kvadrata. Tretji in četrti argument sta širina in višina kvadrata. S petim argumentom pa

povemo, kaj želimo doseči. Tokrat hočemo območje, ki smo ga izbrali s predhodnimi štirimi argumenti, nadomestiti z našim izrisanim kvadratom.. Z zadnjima dvema argumentoma lahko kvadratu ustvarimo mehke oziroma zabrisane robove . Če želimo mehke robove, na mestu prvega argumenta napišemo 1, drugače napišemo 0. Z drugim argumentom povemo radij kroga, s katerim zabrišemo robove.

Ukaz `(gimp-edit-stroke)` nariše označeno območje. V našem primeru na tem mestu program nariše kvadrat na območju, ki smo ga s prejšnjim ukazom označili. Z `(gimp-selection-none)` pa to območje odstranimo. Ostane samo narisan kvadrat.

Vsi ukazi se izvajajo sproti. Ko v konzolo vpišemo sintaktično pravilen ukaz, se ta ukaz izvede. Če gre za ukaz, ki ne vrne vrednosti, ampak ima učinek (npr. risbo), GIMP vrne `(#t)`. To pomeni, da se je ukaz uspešno izvršil. Če ukaz sintaktično ni pravilen, potem GIMP namesto `(#t)` javi napako.

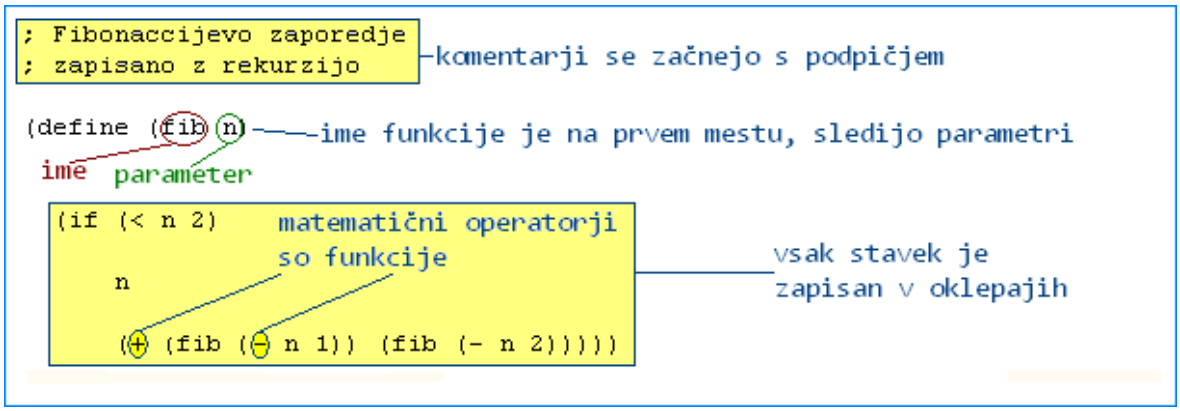
Kot rezultat program ustvari to, kar vidimo na sliki Slika 40:



Slika 40: Kvadrat

3.4.1 Osnovna pravila skriptnega jezika Scheme

Poglejmo si najprej del skripte, napisane v jeziku Scheme:



Slika 41: Uporaba pravil

Iz zgleda lahko razberemo:

- Vsak stavek v Scheme mora biti zapisan znotraj okroglih oklepajev
- Funkcijo definiramo s pomočjo ukaza `define`, ki mu za imenom v oklepajih sledi seznam parametrov. Pri klicu funkcije pa v oklepajih najprej zapišemo ime funkcije. Imenu sledijo parametri funkcije: `(imeFunkcije parameter1 parameter2 parameter 3 …)`
- Matematični operatorji so prav tako funkcije. Njihovo ime je kar operator sam. Uporabljamo torej prefiksno notacijo. Prefiksna (imenovana tudi poljska) notacija je način zapisa, katerega posebnost je, da operator vedno stoji levo od operanda. Primer zapisa je `(+ 7 4)`, ki vrne 11.
- Komentarji se začnejo s podpičjem in končajo na koncu vrstice. To pomeni, da moramo pri večvrstičnih komentarjih novo vrstico zopet začeti s podpičjem . Primer komentarja: `(+ 6 7) ; to je vsota števil 6 in 7.`

Verjetno najbolj nenavadna stran za nas je način pisanja stavkov – klicov funkcij, saj je uporabljena prefiksna notacija. Pri uporabi zapisa moramo paziti. S pomočjo oklepajev namreč kličemo funkcije, potrebujemo jih pa tudi, kadar pišemo sezname. Sezname lahko začnemo z ukazom list: `(list 22 33 44)` ali z opuščajem »'« pred oklepaji z element seznama. Če v primeru `'(22 33 44)` ne bi napisali opuščaja, bi Scheme iskal funkcijo z imenom 22 in s parametroma 33 in 44.

3.4.2 **Spremenljivke in funkcije**

Večina ukazov v GIMPu je izvedena v obliki klica funkcije. GIMP ima veliko funkcij že vgrajenih. Če jih želimo uporabljati, moramo vedeti , kako funkcijo pokličemo. Klic funkcije ima naslednjo obliko:

```
(ImeFunkcije parameter1 parameter2 parameter3 …)
```

Funkcije lahko sestavimo tudi sami. To storimo z ukazom:

```
(define (ImeFunkcije parameter1 parameter2 parameter3 …)
<ukazi>)
```

Tu je `ImeFunkcije` ime novo nastale funkcije, ki sprejme parametre oziroma argumente `parameter1, parameter2, parameter3 … <ukazi>`, pa pove, kaj funkcija naredi. Podrobneje si bomo sestavljanje lastnih funkcij ogledali v poglavju 0 (Skripte za primere iz drugega poglavja).

V Scheme poznamo globalne in lokalne spremenljivke. Globalno spremenljivko v splošnem deklariramo z

```
(define imeSpremenljivke vrednostSpremenljivke).
```

Pri tem z ukazom `define` ustvarimo prostor za novo spremenljivko. `imeSpremenljivke` je njeno ime, ki je lahko sestavljeno iz črk, številk in znakov `+ - . * / < = > ! ? :` `VrednostSpremenljivke` je začetna vrednost spremenljivke. Tako z ukazom

```
(define pozdrav »Zdravo!«)
```

definiramo spremenljivko tipa niz z imenom `pozdrav`. Priredili smo ji vrednost »`Zdravo!`«. Hitro opazimo bistveno razliko v primerjavi s programskim jezikom java. Pri deklaraciji spremenljivk ni potrebno nikjer določiti tipa spremenljivke. V jeziku Scheme se tip spremenljivke določi glede na vrednost spremenljivke. Če kasneje v spremenljivko shranimo podatek drugačne vrste (drugačnega tipa), se spremeni tudi tip spremenljivke.

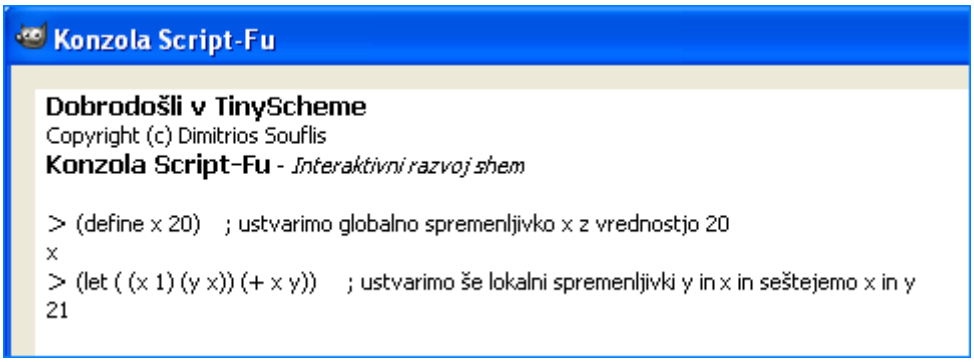
Globalne spremenljivke, ki jih definiramo v skripti, lahko uporabljamo skozi celo skripto. Ko je globalna spremenljivka enkrat ustvarjena, lahko do nje dostopamo v katerem koli delu naše skripte. Ko pa skripto izvedemo, globalnih spremenljivk ni več. Prav tako ji kadarkoli lahko spreminjamo vrednost in z njo tudi tip spremenljivke.

Kadar potrebujemo lokalne spremenljivke, torej take, ki obstajajo samo znotraj nekega bloka (dela) skripte, jih lahko ustvarimo na dva načina.

- Prvi je z ukazom `let`. Ta znotraj () določa območje, kjer so spremenljivke veljavne. Spomnimo se na programski jezik java, kjer področje veljavnosti spremenljivk določimo z bloki (torej z območjem znotraj {}). Splošna oblika ukaza je videti takole:

  ```
  (let (spremenljivke) izrazi).
  ```

  Spremenljivke, ki jih definiramo v `(spremenljivke)` torej obstajajo le znotraj tega stavka. Da bo stvar bolj jasna, si oglejmo primer.



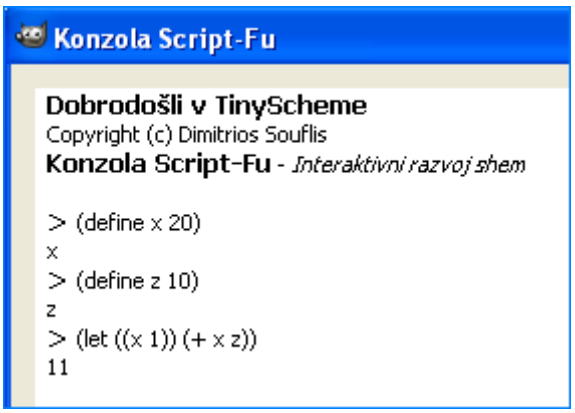
Slika 42: Ukaz let (primer 1)

Najprej smo ustvarili globalno spremenljivko x z vrednostjo 20. Nato smo z ukazom `let` povedali, da bomo definirali še lokalni spremenljivki `x` in `y`. Definirali smo lokalno spremenljivko `x` in ji priredili vrednost 1. Nato smo definirali še lokalno spremenljivko `y`. Spremenljivki `y` smo priredili vrednost globalne spremenljivke `x`, torej 20. Nato izračunamo izraz `x + y`. Vrednost izraza `x + y` je torej 21.

V primeru, če imamo tako lokalno kot tudi globalno spremenljivko z istim imenom, se globalna uporablja le pri definiciji lokalnih spremenljivk.

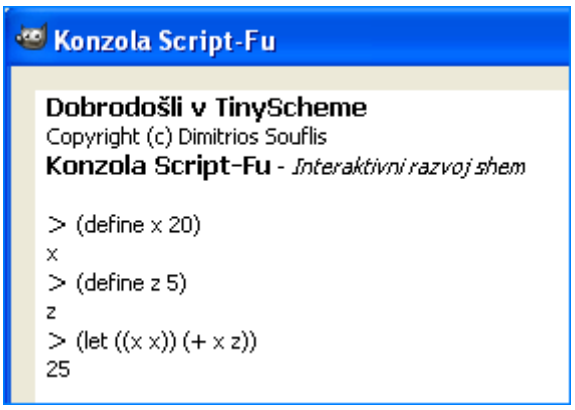
Da bomo to še bolje pokazali, si oglejmo še dva primera na sliki Slika 43 in na sliki Slika 44.

V prvem primeru (Slika 43) smo definirali globalni spremenljivki `x` in `z`. `x`-u smo priredili vrednost 20, `z`-ju pa vrednost 10. Nato smo definirali še lokalno spremenljivko `x` in ji priredili vrednost 1. Z `x + z` seštejemo lokalno spremenljivko `x` z vrednostjo 1 in globalno spremenljivko z vrednostjo 10. Dobimo rezultat 11.



Slika 43: Ukaz let (primer 2)

V drugem primeru (Slika 44) smo definirali globalni spremenljivki `x` in `z`. Spremenljivki `x` smo priredili vrednost 20, spremenljivki `z` pa vrednost 5. Nato smo definirali še lokalno spremenljivko `x`, ki ima vrednost globalne spremenljivke `x`. Seštevek `x`-a in `z`-ja tako da rezultat 25.

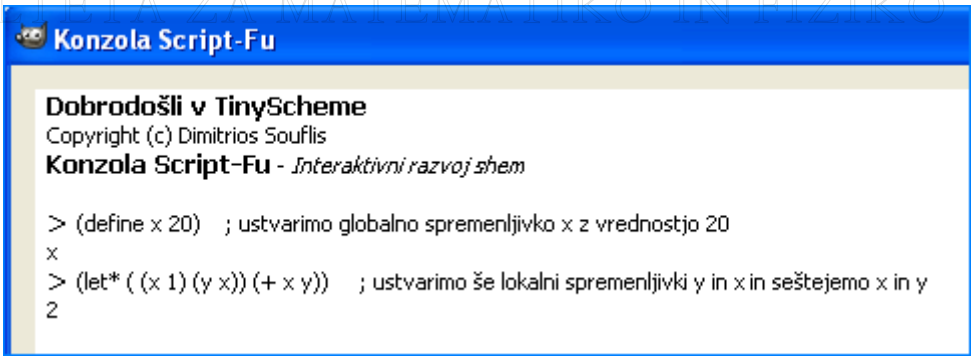


Slika 44: Ukaz let (primer 3)

- Seznanimo se še z ukazom `let*`. Tudi ta znotraj oklepajev določa območje, kjer so spremenljivke veljavne. Splošna oblika je enaka kot pri ukazu let:

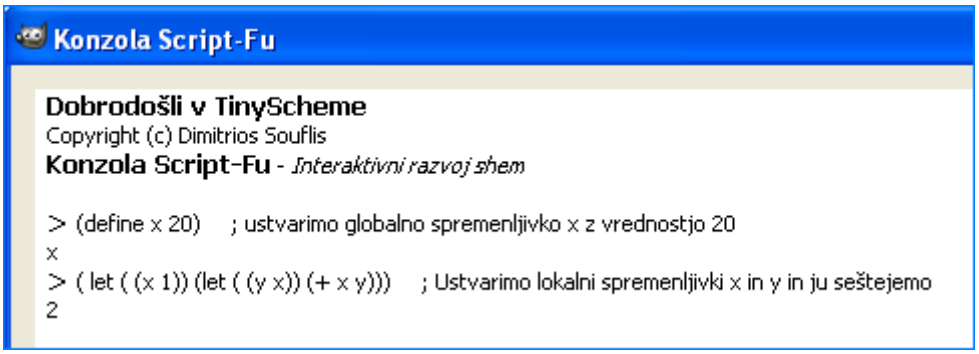
```
(let* (spremenljivke) izrazi)
```

Oglejmo si primer s slike Slika 42 še tukaj, le da uporabimo `let*`:



Slika 45: Ukaz let*

Kot prej najprej ustvarimo globalno spremenljivko `x` z vrednostjo 20. Nato smo definirali lokalni spremenljivki `x` in `y`. Spremenljivki `x` smo priredili vrednost 1, spremenljivki `y` pa smo priredili vrednost lokalne spremenljivke `x`, ki je enaka 1. Torej ima `y` vrednost 1. Scheme se torej v primeru uporabe let* ne sklicuje na globalno spremenljivko `x`, temveč na lokalno spremenljivko `x`. Seštevek je torej enak 2. Ekvivalenten rezultat dobimo, če zapišemo primer takole:



Slika 46: Drugačen zapis ukaza let*

Opazimo razliko med uporabo ukazov `let` in `let*`. Če si predstavljamo, da oklepaji (podobno kot v javi) določajo neke vrste nivoje, potem lahko rečemo, da je bistvena razlika med stavkom `let` in `let*` v tem, da prvi na določenem nivoju uporablja vrednosti spremenljivk določenih na prejšnjem nivoju, pri drugem pa lahko na istem nivoju definiramo spremenljivko in tudi uporabimo njeno vrednost.

Spoznali smo, na kakšen način lahko spremenljivke ustvarimo. Spoznajmo še, kako spremenljivkam spremenimo vrednost. To storimo z ukazom `set!`. V jeziku Scheme ima `set!` naslednjo obliko:

```
(set! imeSpremenljivke novaVrednostSpremenljivke)
```

Pri tem je `imeSpremenljivke` ime spremenljivke, katere vrednost spreminjamo, `novaVrednostSpremenljivke` pa je vrednost, ki jo bo spremenljivka zavzela. Ukaz ima torej učinek kot to storimo v javi s klasičnim prireditvenim stavkom `imeSpremenljivke = novaVrednost`. Če spremenljivka še ne obstaja, jo s tem ustvarimo. Na ta način nastala spremenljivka je globalna.

Oglejmo si zgled. Spremenljivki z imenom `oranžna` nastavimo vrednost oranžne barve v skladu z barvnim modelom RGB. V ta namen moramo v spremenljivko shraniti seznam treh števil. Nato s

funkcijo za spreminjanje barve ozadja nastavimo barvo na barvo, ki smo jo shranili v spremenljivko z imenom `oranžna`.

```
(set! oranžna '(255 127 0))

(gimp-set-background oranžna)
```

Slika 47: Ukaz set!

Kot sem že omenila, pri deklaraciji spremenljivk v jeziku Scheme ni potrebno določiti tipa spremenljivke. To pa še ne pomeni, da tipi spremenljivk za pisanje in razumevanje skript niso pomembni. Poznamo naslednje tipe spremenljivk:

- **LOGIČNE VREDNOSTI (boolean)**

  V Scheme za logično vrednost true uporabimo `#t` , `#f` pa za logično vrednost false. Funkcija `boolean?` preveri, če je argument tipa boolean. Funkcija `not` negira vrednost argumenta. Obe funkciji sta tipa boolean. To pomeni, da vračata rezultat tipa boolean.

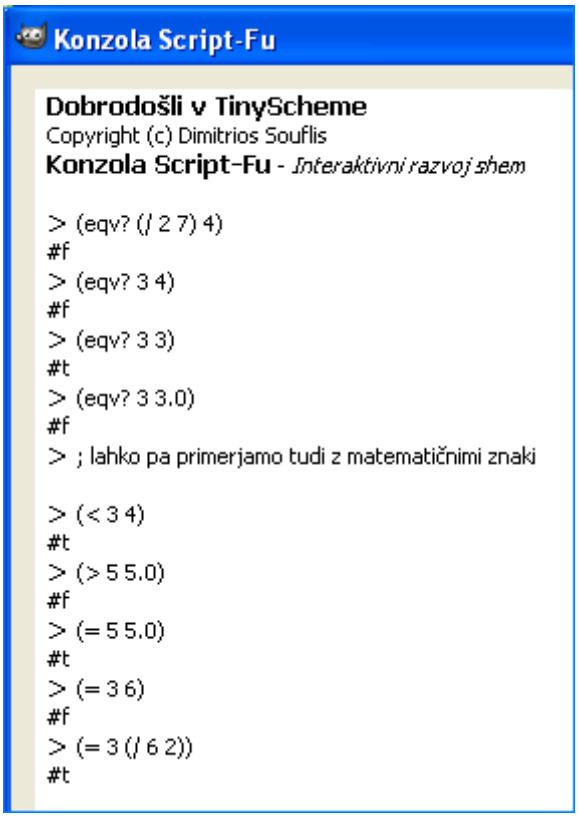
Slika 48: Logične vrednosti

- **ŠTEVILA (number)**

  Scheme pozna več tipov števil: cela števila (integer), ulomke (rational) in decimalna števila (real). Funkcije za preverjanje, ali je število določenega tipa, vračajo rezultate tipa boolean . Tako z `number?` preverimo, če je argument poljubno število, z `real?`, če je število decimalno, z `rational?`, če je število ulomek in z `integer?`, če je število celo.

  Števila med seboj lahko tudi primerjamo. Primerjamo jih lahko s funkcijo `eqv?`, pa tudi z matematičnimi operatorji `<`, `<=`, `>`, `>=`, `=`. Iz Slika 49 lahko razberemo, da `eqv?` upošteva tudi tip števila in ne le vrednost. Zato števili 3 in 3.0 nista enaki. Razlikujeta se v

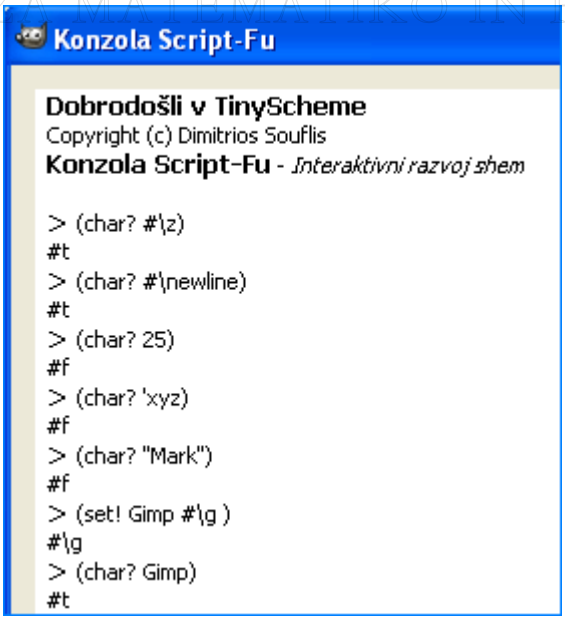
podatkovnem tipu. Pri primerjanju z operatorji `<`, `<=`, `>`, `>=`, `=` se upošteva le vrednosti števil.



Slika 49: Primerjanje števil

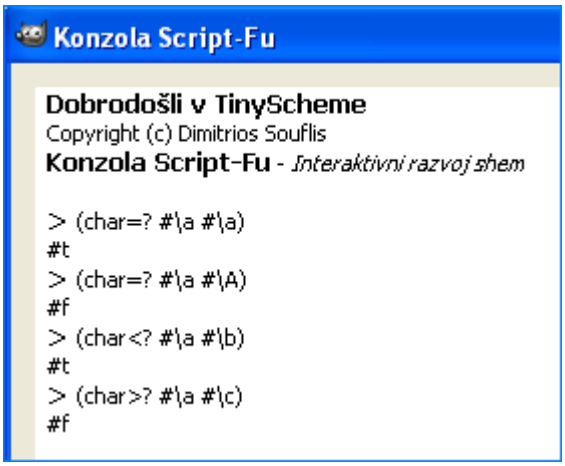
- **ZNAKI (character)**

  Znak je predstavljen s predpono `#\`. Torej zapis `#\c` pomeni znak c. Ukaz `#\newline` pomeni skok v novo vrstico, `#\ ` in `#\space` pomenita presledek. S funkcijo `char?` lahko preverimo, če je nekaj res znak.



Slika 50: Znaki

Znake med seboj lahko primerjamo in ugotovimo, če sta dva znaka enaka ali ne. To storimo s `char=?`. S `char<?` in s `char>?` pa preverimo, če je prvi znak leksikografsko pred drugim in obratno.



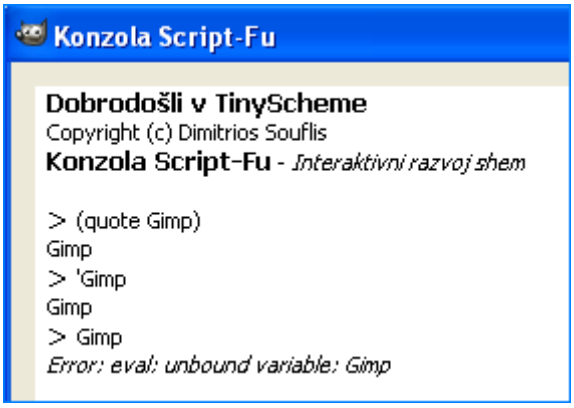
Slika 51: Primerjava znakov

Logične vrednosti, števila, simboli in znaki so enostavni podatkovni tipi (simple data types). Ne moremo jih tvoriti iz vrednosti ostalih podatkovnih tipov. Velja, da če v pogovorno vrstico konzole vpišemo `#t`, program vrne `#t`. Podobno, če vpišemo število 44, vrne število 44. Če vpišemo znak `#\a`, vrne `#\a`. S simboli pa je malo drugače.

- **SIMBOLI (symbol)**

Simbole uporabljamo za tako imenovane označevalce (angl. identifiers) (kot so besede `do, else, if, let, quote, for, let*, begin, case, define` ...) in za poimenovanje spremenljivk. Besed, ki so rezervirane za označevalce, ne smemo uporabljati kot imena spremenljivk. Da določimo simbol (ne, da bi ga Scheme zaznal kot spremenljivko), zapišemo: `(quote imeSimbola)` ali pa

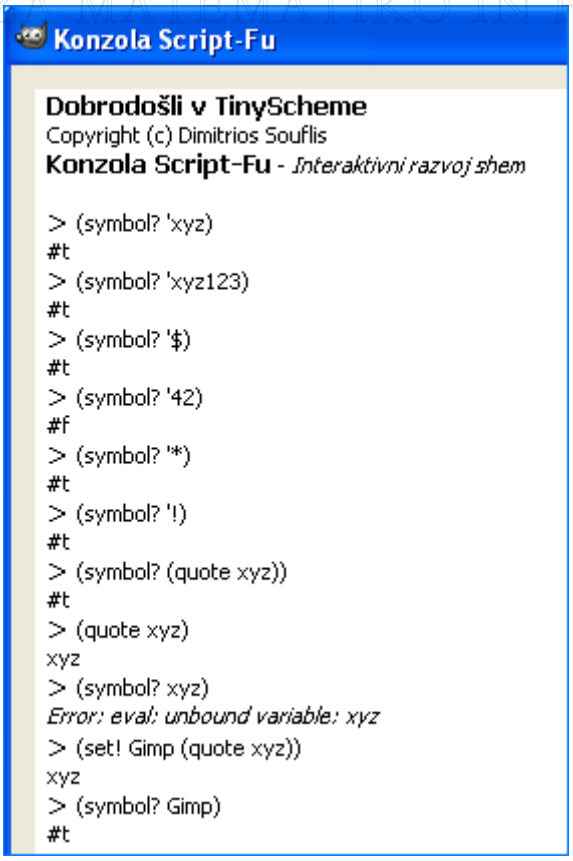
`'imeSimbola`. Ker smo na Slika 52: Zapis simbola v tretjem ukazu zapisali samo `Gimp` (in ne `(quote Gimp)` oziroma `'Gimp`), program izpiše obvestilo, da spremenljivka z imenom `Gimp` ne obstaja. Program je torej ukaz `Gimp` zaznal kot spremenljivko in ne kot simbol (Slika 52).



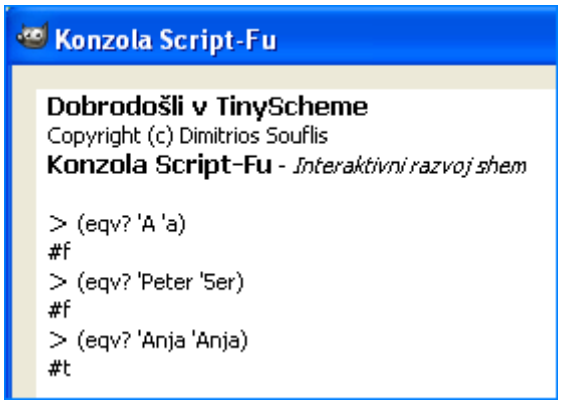
Slika 52: Zapis simbola

S funkcijo (ki vrača rezultate tipa boolean) `symbol?` preverimo, če je določen zapis tipa simbol (Slika 53). Kot pri številih, tudi pri simbolih lahko za primerjanje dveh simbolov uporabimo `eqv?`. Če sta primerjana simbola enaka, funkcija vrne rezultat `#t`, če nista enaka, pa `#f`. Iz primera na sliki Slika 54 je videti, da Scheme v imenih simbolov loči med malimi in velikimi črkami.

Tudi simboli spadajo v skupino enostavnih podatkovnih tipov, le da delujejo malo drugače kot logične vrednosti, števila in znaki. Ker so namenjeni poimenovanju spremenljivk, vračajo vrednosti spremenljivk, ki jih označujejo. Recimo, da obstaja spremenljivka z imenom `Gimp`. Če v konzolo vpišemo `Gimp`, program vrne vrednost spremenljivke `Gimp`. Če spremenljivka ne obstaja, potem program vrne obvestilo, da spremenljivka z imenom `Gimp` ne obstaja (primer 3 na sliki Slika 53).



Slika 53: Simboli

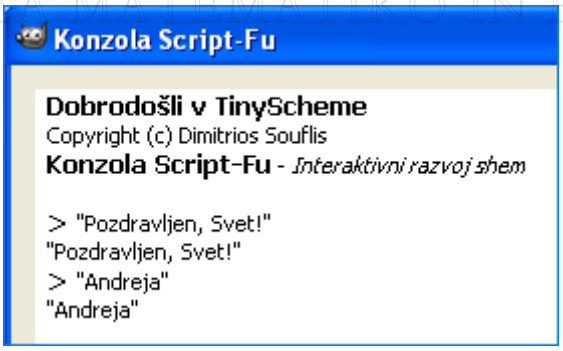


Slika 54: Primerjava simbolov

Podatkovni tipi kot so: nizi, vektorji, urejeni pari in seznami se imenujejo sestavljeni podatkovni tipi (compound data types). Sestavljeni podatkovni tipi so sestavljeni s kombiniranjem poljubnih podatkovnih tipov. Tako recimo niz lahko sestavimo iz znakov in števil, vektor pa iz števil, nizov in znakov.

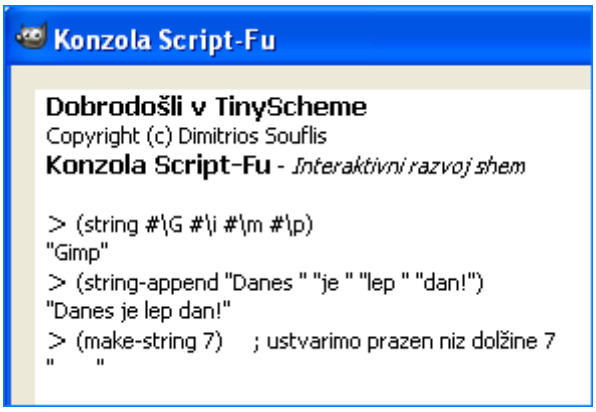
- **NIZI (string)**

  Niz je definiran kot zaporedje znakov, ki jih pišemo znotraj dvojnih narekovajev.



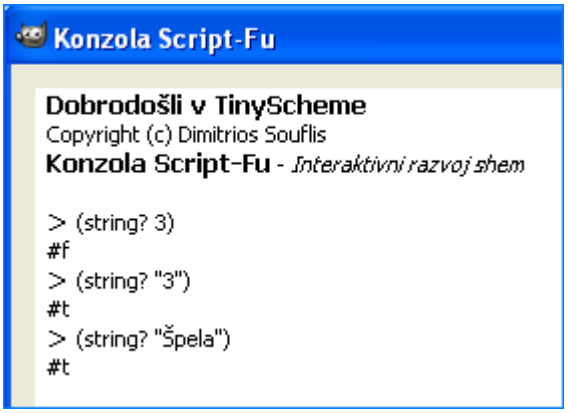
Slika 55: Zapis nizov

Metoda `string` sprejme posamične znake in vrne iz znakov sestavljen niz. Niz lahko sestavimo tudi iz več nizov. To storimo z `string-append`. Prazen niz dolžine n ustvarimo z `(make-string n)`.



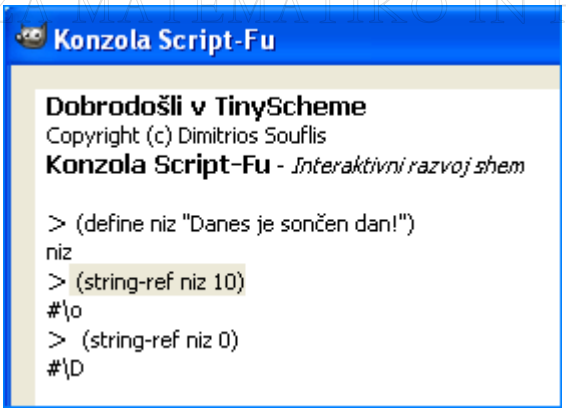
Slika 56: Tvorba nizov

S `string?` (tipa boolean) preverimo, če je določen argument niz.



Slika 57: Nizi

S `(string-ref poljubenNiz mesto)` lahko dostopamo do posameznega znaka v nizu. `poljubenNiz` je ime niza, `mesto` pa je mesto v nizu, do katerega želimo dostopati. `(string-ref poljubenNiz mesto)` vrne znak na želenem mestu.

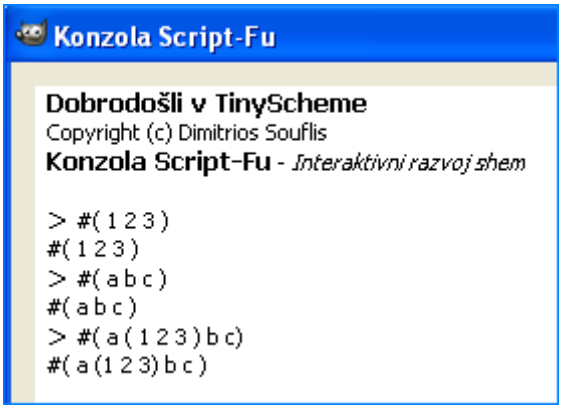


Slika 58: Dostop do znaka v nizu

Kot vidimo, tudi v Scheme mesta znakov štejemo od 0 dalje.

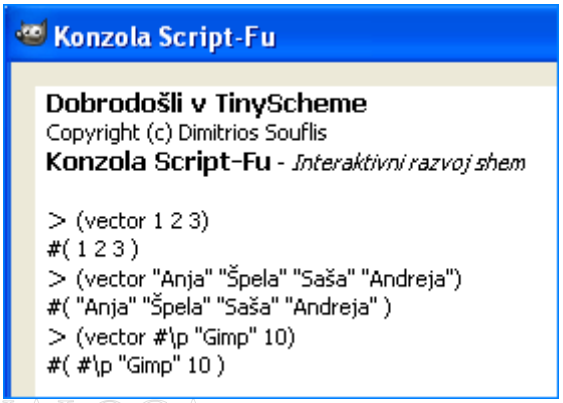
- **VEKTORJI (vector)**

Vektor je, podobno kot niz, definiran kot zaporedje, le da so elementi zaporedja lahko poljubnega tipa. Vektor sestavimo z `#(element1 element2 element3 …)`. Če vektor vsebuje elemente tipa vektor, potem dobimo večrazsežen vektor.

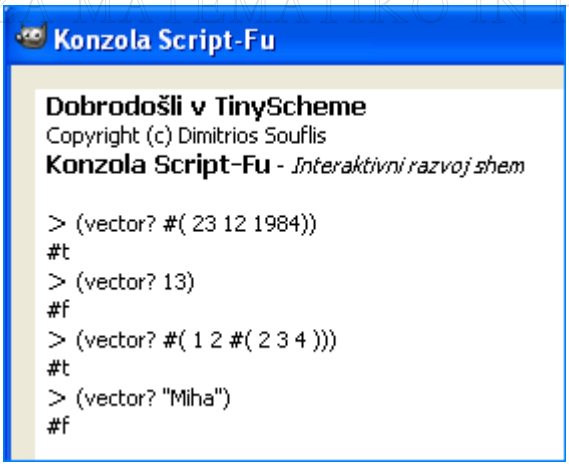


Slika 59: Zapis vektorjev

S funkcijo `vector` ustvarimo nov vektor, s funkcijo `vector?` pa preverimo, če je določen argument tipa vektor.



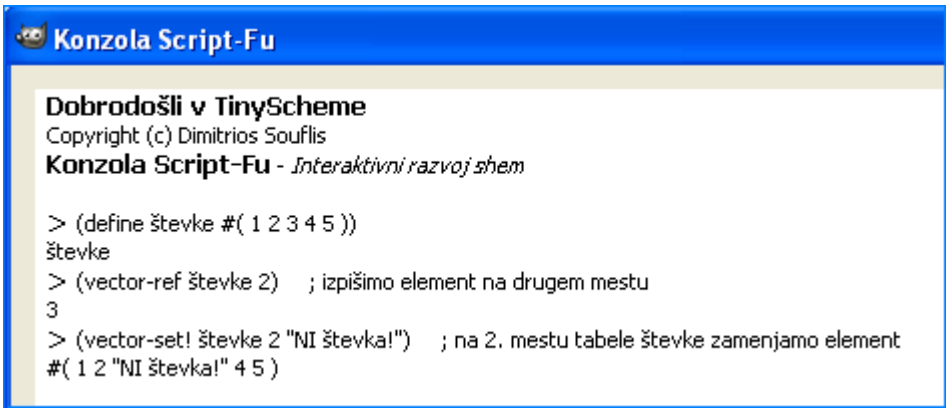
Slika 60: Tvorba vektorjev



Slika 61: Vektorji

Za vektorje prav tako velja, da lahko dostopamo do poljubnega njihovega elementa. Tako z `(vector-ref imeVektorja mesto)`, podobno kot pri nizih, dostopamo do elementa na določenem mestu.

Elemente vektorja pa lahko tudi zamenjamo. V ta namen uporabimo `(vector-set! imeVektorja mesto novElement)`. Na določenem mestu zamenjamo element z novim elementom.

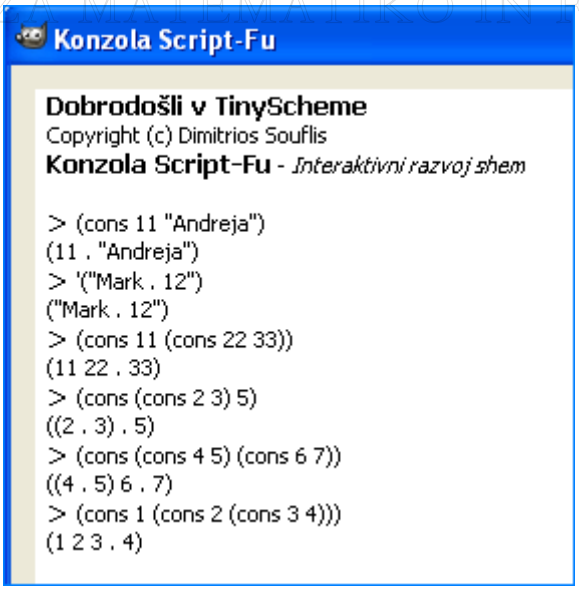


Slika 62: Dostop in zamenjava elementa v tabeli

- **UREJENI PARI (dotted pairs) IN SEZNAMI (list)**

Urejen par (x, y) je dvojica elementov poljubnega tipa, v katerem je x prvi element, y pa drugi element. Vrstni red je pomemben. Torej par (x, y) ni enak paru (y, x). Urejen par v Scheme ustvarimo s `(cons prviElement drugiElement)` ali pa z `'(prviElement . drugiElement)`. Pri zadnji obliki je pika zelo pomembna, saj, če jo spustimo, dobimo seznam.

Kadar je eden (ali oba) od elementov urejenega par spet urejen par, potem takemu urejenemu paru rečemo ugnezden urejen par. Ne glede na tip elementov, ki jih urejen par vsebuje, Scheme s piko loči prvi element od drugega.



Slika 63: Tvorba urejenih parov

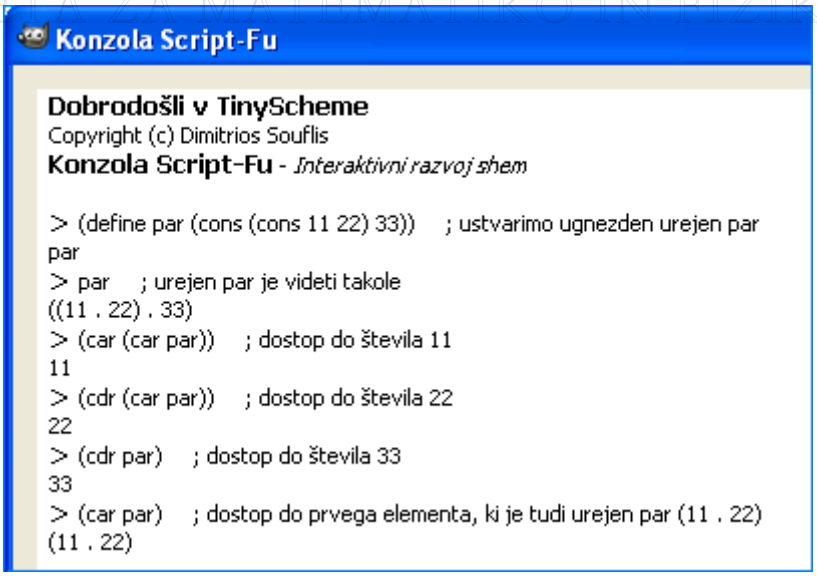
S prvim ukazom na sliki Slika 63 ustvarimo urejen par s številom 11 in z nizom »Andreja«. Podobno ustvarimo drugi urejeni par. Naslednji trije primeri pa prikazujejo različne načine gnezdenja:

- `(cons 11 (cons 22 33))` je urejen par, ki ima za prvi element število 11, drugi element pa je urejen par (22 . 33). To zapišemo kot (11 . (22 . 33)). Scheme v primeru gnezdenja znotraj drugega elementa uporablja za izpis krajši zapis, ki je v tem primeru videti takole: (11 22 . 33).
- `(cons (cons 2 3) 5)` je urejen par, ki ima za prvi element urejen par (2 . 3), za drugi element pa številko 5. Na dolgo zapišemo: ((2 . 3) . 5). V primeru gnezdenja znotraj prvega elementa tudi Scheme pri izpisu uporablja daljši zapis.
- `(cons (cons 4 5) (cons 6 7))` je urejen par, kjer sta prvi in drugi element urejena para. Prvi element je urejen par (4 . 5), drugi element pa urejen par (6 . 7). Na dolgo zapišemo: ((4 . 5) . (6 . 7)). Iz primera lahko vidimo, da pri gnezdenju znotraj prvega elementa Scheme uporabi daljši zapis. Če pa uporabimo gnezdenju znotraj drugega elementa, pa Scheme uporabi krajši zapis s piko.

Zaradi razlike med izpisom pri gnezdenju znotraj prvega in drugega elementa, vedno točno vemo, kje uporabljamo gnezdenje.

V zadnjem primeru na sliki Slika 63 imamo primer, kjer smo dvakrat gnezdili znotraj drugega elementa. Prvi element je številka 1. Drugi element je urejen par, katerega prvi element je številka 2, drugi element pa je urejen par (3 . 4). Poglejmo bolj podrobno samo drugi element, torej: (cons 2 (cons 3 4)). Primer je pravzaprav enak primeru `(cons 11 (cons 22 33))`. Torej dobimo par (2 . (3 . 4)) oziroma na kratko: (2 3 . 4). Zdaj združimo prvi element 1 in drugi element (2 . (3 . 4)) oz. (2 3 . 4). N dolgo zapisano dobimo: (1 . (2 . (3 . 4))). Ker smo zopet gnezdili znotraj drugega elementa, je krajši zapis celotnega urejenega para videti takole: (1 2 3 . 4).

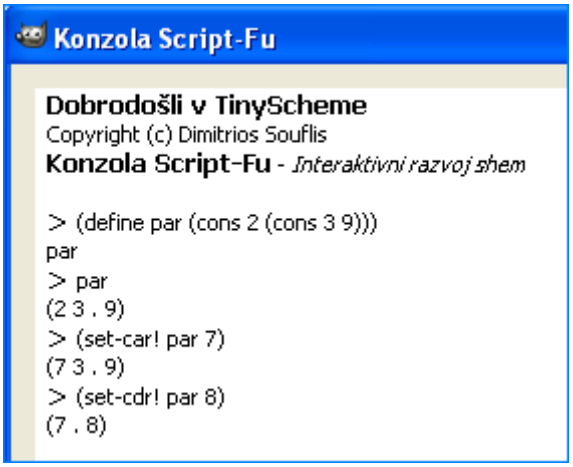
Z ukazom `car` dostopamo do prvega elementa v urejenem paru, z ukazom `cdr` pa do drugega elementa. Nekoliko bolj zapleteno je dostopati do elementov, če so elementi urejenega para urejeni pari. Primer je prikazan na sliki Slika 64.



Slika 64: Ukaza cdr in car

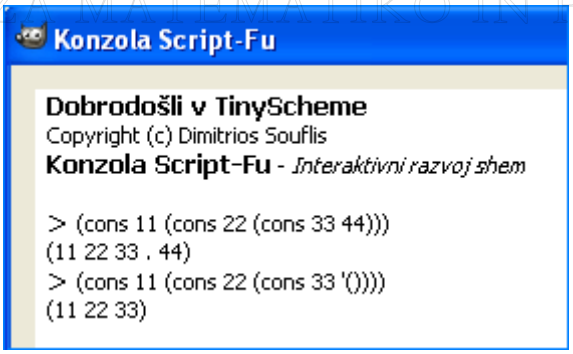
Ukaz `car` v zgornjem primeru vrne prvi element urejenega para - urejen par (11 . 22). Ukaz `cdr` pa vrne drugi element urejenega para – število 33. Če iščemo število 11, potem iščemo prvi element urejenega para (11 . 22). Ta je obenem prvi element urejenega para ((11 . 22) . 33. Ker iščemo prvi element prvega elementa, napišemo `(car (car par))`. Če pa iščemo število 22, iščemo drugi element urejenega para (11 . 22). Ta par pa je hkrati tudi prvi element para ((11 . 22) . 33). Napišemo torej `(cdr (car par))`.

Kot vektorjemh tudi urejenim parom lahko zamenjamo element. Za zamenjavo prvega elementa uporabimo `(set-car! imePara starElement novElement)`, za zamenjavo drugega elementa pa `(set-cdr! imePara starElement novElement)`.



Slika 65: Zamenjava elementov v urejenem paru

Poseben primer ugnezdenega urejenega para je par, katerega drugi element najgloblje ugnezdenega urejenega para je ničen. Primer je na sliki Slika 66. V primeru `(cons 11 (cons 22 (cons 33 44)))` je najgloblje ugnezden par (33 . 44). Njegov drugi element je število 44. Če to število zamenjamo za prazen seznam, potem takemu urejenemu paru rečemo kar seznam.

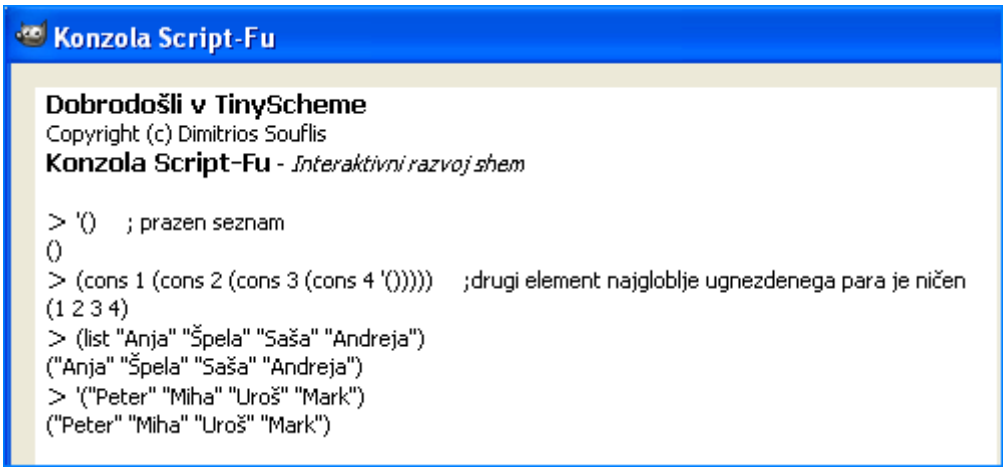


Slika 66: Ugnezden par postane seznam

Seznami so torej v bistvu urejeni pari, ki imajo kot drugi elemet prazen seznam. Prazen seznam je v Scheme predstavljen z (). Definiramo ga z uporabo '.

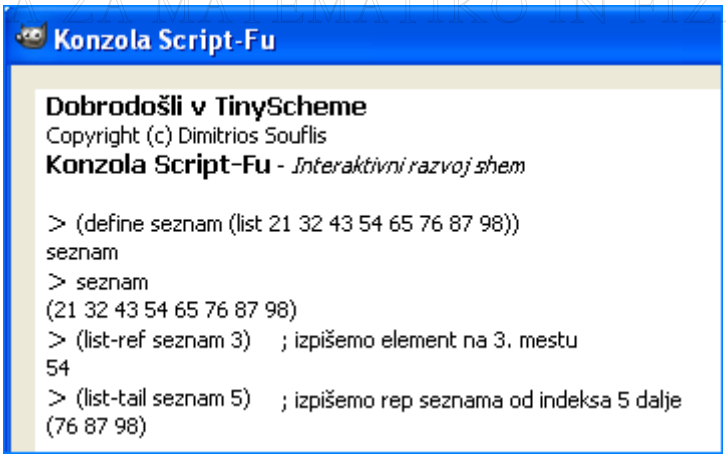
Sezname pa lahko ustvarimo tudi drugače. To lahko storimo z ukazom `(list element1 element2 element3 …)` ali pa z `'(element1 element2 element3 … )`.

Kot sem omenila v razdelku 3.4.1 (Osnovna pravila skriptnega jezika Scheme), vse funkcije in sezname pišemo v oklepajih. Recimo, da želimo ustvariti seznam števil 1, 2 in 3. Če zapišemo samo `(1 2 3)`, GIMP išče funkcijo z imenom 1 in s parametroma 2 in 3. Zato s `quote` oziroma z `'` GIMPu sporočimo, da ne gre za funkcijo, temveč za seznam.



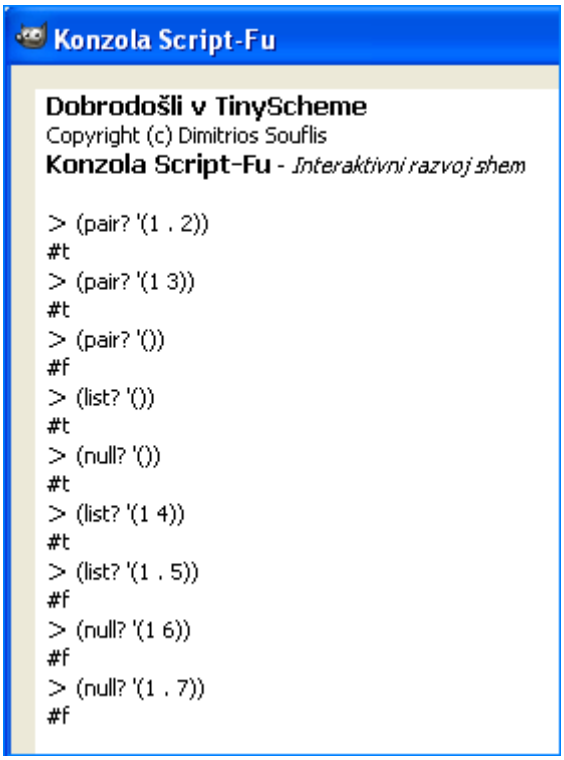
Slika 67: Tvorba seznamov

Tudi v seznamih lahko dostopamo do poljubnega elementa. To storimo z ukazom `(list-ref imeSeznama mesto)`. Pri tem `mesto` pomeni mesto v seznamu, katerega vrednost želimo izpisati. Z `(list-tail imeSeznama mesto)` pa dobimo del seznama od indeksa `mesto` (vključno z njim) do konca. Dobimo tako imenovani rep seznama.



Slika 68: Dostop do elementov in ukaz tail

S `pair?`, `list?` in `null?` preverimo, če je argument teh funkcij urejen par, seznam ali prazen seznam.



Slika 69: Urejeni pari, nizi in prazni nizi

3.4.3 Stavek if in zanka while

Oblika stavka if v Scheme je naslednja:

```
(if pogoj
  (ukaz1)
  (ukaz2)
)
```

Pogoj je tipa boolean. Če ima pogoj vrednost true, se izvrši `ukaz1`, če pa ima pogoj vrednost false, potem se izvrši `ukaz2`.

```
(if (< x 0)
        (print "x je negativen")
        (print "x je nenegativen")
        )
```

Slika 70: Primer if stavka

Koda na Slika 70 najprej preveri, če je vrednost v spremenljivki `x` negativna. Če je, izvrši `ukaz1`. To je `(print "x je negativen")`, ki izpiše »`x je negativen`«. Če pa pogoj ne velja, torej če v x ni negativna vrednost, potem izvrši `ukaz2`. Ta izpiše »x je nenegativen«.

Recimo, da bi v našem primeru poleg ustreznega izpisa radi nastavili še vrednost spremenljivke `predznak`. Če bo pogoj resničen, bo torej program izpisal, da je `x` negativno število in hkrati nastavil vrednost spremenljivke `predznak` na -1. Če pa pogoj ne bo resničen, potem bo program izpisal, da je `x` pozitivno število nastavil še vrednost spremenljivke `predznak` na 1. S `set!` smo hkrati ustvarili globalno spremenljivko.

```
(if (< x 0)
        (begin
                (print "x je negativen")
                (set! predznak -1)
        )
        (begin
                (print "x je nenegativen")
                (set! predznak 1)
        )
)
```

Slika 71: Primer razširjenega if stavka

Opazimo, da smo uporabili ukaz `(begin (ukazA) (ukazB) …)`. Ta omogoča, da `ukaz1` in `ukaz2` sestavimo iz več ukazov. Primer lepo ponazarja Slika 71.

Zanka while je edina zanka, ki obstaja v skriptnem jeziku Script – Fu. Njena oblika je videti takole:

```
(while (pogoj)
  (ukaz1)
  (ukaz2)
  (ukaz3)
     …
 )
```

Na Slika 72 je primer, ki po celi širini slike nariše vodoravne črte z debelino 10 pikslov. Med črtami je 16 pikslov prostora.

```
(set! y 0)
(while (< y visinaSlike)
        (gimp-rect-select fotografija 0 y sirinaSlike 10 REPLACE 0 0)
        (gimp-edit-fill črte BG-IMAGE-FILL)
        (set! y (+ y 26))
)
```

Slika 72: Primer while zanke

Predpostavimo, da urejamo sliko `fotografija`, na kateri je plast z imenom `črte`. V spremenljivki z imenom `visinaSlike` je shranjena višina slike, v spremenljivki `sirinaSlike` pa širina slike.

Ideja postopka je ta, da bomo na sliko zaporedoma dodajali črte. Črte dobimo tako, da narišemo pravokotno območje z višino 10 (če bodo črte debeline 10) in širino `sirinaSlike`, ki je hkrati širina slike. Črte rišemo na način, ki smo ga opisali na začetku v primeru risanja kvadrata v razdelku 3.4 (Osnove jezika Scheme – primer risanja kvadrata).

Števec `y` pove, kje v sliki se nahajamo. Najprej smo števec `y` postavili na nič, nato pa se premikamo s korakom 26 (10 za debelino in 16 za razmik).

Z zanko while poskrbimo, da bomo risali črte po celotnem območju slike.

V zanki najprej z `gimp-rect-select` izberemo pravokotno območje na sliki `fotografija`. Ta kot parametre sprejme: sliko (v našem primeru je to slika z imenom `fotografija`), x in y koordinato levega zgornjega kota izbranega pravokotnika (v našem primeru je x vedno nič, y se spreminja), širino in višino pravokotnega območja (širina pravokotnega območja je hkrati širina slike, shranjena v spremenljivki `sirinaSlike`) in še tri parametre, o katerih sem govorila v primeru v razdelku 3.4 (Osnove jezika Scheme – primer risanja kvadrata). Nato z `gimp-edit-fill` pobarvamo izbrano območje. `gimp-edit-fill` sprejme dva parametra. Prvi parameter je plast (v našem primeru je to plast z imenom `črte`), na kateri bodo črte narisane. Drugi parameter pove, da bomo za barvanje območja uporabili trenutno barvo ozadja. (`BG-IMAGE-FILL` pomeni background image fill).

Opisani postopek ima pomanjkljivost. Do težav lahko pride pri risanju zadnje črte, kadar je y še manjši od višine slike in je hkrati razlika med višino slike in y-om manjša kot 10 točk. V tem primeru je pogoj `y < visinaSlike` izpolnjen, in ukazi v zanki while se začnejo izvajati. Ko z `gimp-rect-select` želimo izbrati območje z višino 10 (debelina črte), se zgodi, da nam »zmanjka slike«. Črte ne moremo narisati, ker imamo v višino na voljo manj kot 10 pikslov. Skripta v tem primeru javi napako.

4 Skripte za primere iz drugega poglavja

4.1 Uvod

V poglavju 2 (Primeri uporabe) smo opisali tri primere uporabe programa GIMP. Prvi razdelek 2.1 (Spreminjanje velikosti slike) opisuje, kako poljubni sliki spremenimo velikost. V drugem razdelku 2.2 (Ustvarjanje fotografije s sepija barvnim tonom) smo opisali dodajanje sepija barvnega tona poljubni sliki. V tretjem razdelku, 2.3 (Dodajanje besedila na sliko) smo razložili, kako poljubni sliki dodamo besedilo. Vse to dosežemo postopoma, z uporabo različnih orodij, ki jih GIMP ponuja. Če bi te postopke morali opraviti večkrat, je smiselno, da raje napišemo ustrezne skripte,

V razdelkih 4.2 (Skripta za sepijo), 0 (Skripta za dodajanje besedila na sliko) in 0 (Skripta za spreminjaje velikosti slike) si bomo ogledali, kako to storimo .

Vse tri skripte , ki jih bomo napisali, spadajo v skupino skript, delujočih na slikah. Torej vsaka od teh skript deluje izključno takrat, kadar imamo v oknu obdelovane slike odprto neko sliko. Če v oknu obdelovane slike ni odprte nobene slike, GIMP pri poskusu zagona tovrstnih skript ponudi pogovorno okno, v katerem lahko izberemo sliko, na kateri bo skripta delovala. Če še tu ne izberemo nobene slike, potem GIMP javi napako. Sporoči, da skripta poskuša delovati na sliki, ki ne obstaja več (torej ni odprta v programu GIMP).

Naše tri skripte bomo napisali tako, da jih bomo potem našli v slikovnem meniju programa GIMP in sicer v meniju Skript-Fu. Skripta za dodajanje besedila na sliko bo v podmeniju `besedilo`, zagnali jo bomo z ukazom `dodaj besedilo`. Skripta za spreminjanje velikosti slike bo v podmeniju `velikost`. Uporabili jo bomo s klikom na ukaz `spremeni velikost`. Skripta za sepijo bo v podmeniju `sepija`. Izvedli jo bomo s klikom na ukaz `dodaj sepija barvni ton`.

Kot pove že ime, skripta za dodajanje sepija barvnega tona poljubni sliki doda videz starinske črno-bele slike s pridihom toplih rjavih tonov. Skripta za spreminjanje velikosti slike spremeni velikost slike. Skripta za dodajanje besedila pa poljubni sliki doda poljubno besedilo s poljubno pisavo, velikostjo in postavitvijo besedila.

4.2 Skripta za sepijo

Ta skripta, kot tudi preostali dve, za svoje delovanje ne bo potrebovala definicije pomožnih funkcij. Zato bo sestavljena le iz glavne funkcije. To bomo poimenovali `script-fu-Sepija-barvni-ton`.

Sprejme tri parametre. Prvi parameter je parameter z imenom `fotografija`, drugi parameter je `ozadje`, tretji parameter pa je `barva`. Kadar skripta deluje na odprti sliki, potrebujemo parameter, ki označuje odprto sliko. V našem primeru je ta parameter `fotografija`. Z istim razlogom potrebujemo tudi drugi parameter. Parameter `ozadje` označuje aktivno plast slike `fotografija`. Aktivna plast je tista plast, na kateri imamo dejansko fotografijo. To je tista plast, ki se pojavi v oknu plasti takoj, ko v GIMPu odpremo poljubno fotografijo. V slovenski

verziji GIMPa je poimenovana `ozadje`. Tretji parameter `barva` potrebujemo, ker želimo, da ima tisti, ki kliče skripto, možnost nastavljanja barve, torej tega, kar bo uporabljeno kot barvni ton za sepijo.

Ker mogoče s končnim rezultatom skripte ne bomo zadovoljni, bomo poskrbeli, da bi učinek delovanja skripte lahko razveljavili. Če bomo torej skripto uporabili na neki fotografiji in nam rezultat ne bo všeč, bomo potem lahko v glavnem meniju uporabili ukaz `razveljavi`, ki razveljavi zadnjo izvedeno akcijo. To storimo tako, da v jedro glavne funkcije vključimo »blok za razveljavitev«. Začnemo ga z `(GIMP-image-undo-group-start fotografija)`, končamo pa z `(GIMP-image-undo-group-end fotografija)`. V obeh ukazih je edini parameter `slika`, na kateri bo ukaz razveljavi izvršen. Če tega ne bi storili, bi potem z ukazom `razveljavi` razveljavili le zadnji ukaz, ki bi se v skripti izvedel. Seveda bi lahko z vztrajnim klikanjem na `razveljavi` počasi razveljavili vse ukaze, ki jih je izvedla skripta, vendar tega ne želimo. Hočemo z enim klikom razveljaviti vse, kar je skripta naredila.

Ideja, kako dodati sliki sepija barvni ton, je popolnoma enaka, kot pri ročnem dodajanju sepija barvnega tona. Kako smo ročno spremenili fotografijo v fotografijo s sepija barvnim tonom, smo opisali v razdelku 2.2 (Ustvarjanje fotografije s sepija barvnim tonom).

Postopek ima štiri korake:

- Najprej bomo ustvarili plast `sepijaPlast`, ki je kopija plasti ozadje. Pri ročnem obdelovanju spreminjamo kar plast `ozadje` in ne njene kopije. Nato spremenimo plast `sepijaPlast` v črno-belo fotografijo. Pri ročni obdelavi v črno-belo fotografijo spremenimo kar plast `ozadje`.
- Nato ustvarimo plast `plastMaske`. Pri ročni obdelavi smo jo poimenovali `Sepija`. Plast zapolnimo z barvo ospredja in ji dodamo belo neprosojno masko.
- Nato kopijo plasti `sepijaPlast` prilepimo v masko, ki jo v skripti imenujemo `maska`. Pri ročni obdelavi maska nima posebnega imena.
- Masko na koncu preobrnemo, da dobimo ustrezne tone za sepija barvni ton.

Začnemo z definicijo lokalnih spremenljivk. Definicijo začnemo z ukazom `let*`, o katerem smo govorili v razdelku 3.4.2 (Spremenljivke in funkcije). Nato sledijo spremenljivke z imeni `sepijaPlast`, `plastMaske` in `maska`. V spremenljivki `sepijaPlast` hranimo plast, ki bo na koncu spremenila barvo slike fotografija. Spremenljivko `plastMaske` bomo potrebovali, da bomo na njej lahko ustvarili masko, ki je potrebna za dober končni rezultat sepija barvnega tona. V spremenljivko `maska` pa bomo shranili belo neprosojno masko.

S `(set! sepijaPlast (car (gimp-layer-copy ozadje FALSE)))` nastavimo spremenljivko `sepijaPlast`. `(gimp-layer-copy ozadje FALSE)` naredi kopijo plasti `ozadje`. To pove prvi parameter. Drugi parameter pove, če kopiji plasti dodamo alfa kanal ali ne. S `car` dostopamo do vrnjenih vrednosti funkcij, ki so v GIMPu že vgrajene `(gimp-layer-copy vrne plast)`. V spremenljivki `sepijaPlast` torej hranimo kopijo plasti ozadje. Pri ročnem obdelovanju v razdelku 2.2 (Ustvarjanje fotografije s sepija barvnim tonom (drugi način)) smo se stvari lotili malo drugače. Namesto kopije plasti ozadje smo v črno-belo spremenili kar plast ozadje samo. Zdaj plasti `sepijaPlast` nastavimo ime: `(gimp-layer-set-name sepijaPlast "Sepija plast")`. Nato sliki `fotografija` dodamo plast `sepijaPlast`: `(GIMP-image-add-layer fotografija sepijaPlast -1)`. Tretji parameter -1 pove, da plast dodamo nad trenutno aktivno plast slike fotografija. Pri ročni obdelavi smo plast z imenom `sepija` ustvarili

tako, da smo v oknu plasti izbrali zavihek `plasti` in izbrali bližnjico `ustvari novo plast`. Hkrati smo ji nastavili tudi ime.

Nato spremenimo plast `sepijaPlast` v črno-belo plast. To storimo z ukazom: `(gimp-desaturate sepijaPlast)`. Z ročnim obdelovanjem smo isto dosegli z ukazom `zmanjšaj nasičenost` v meniju `barve`.

V naslednjem koraku nastavimo barvo ospredja: `(GIMP-context-set-foreground barva)`. Parameter `barva` je barva, ki določa sepija barvni ton. Že pri ročnem spreminjanju barve smo ugotovili, da so smiselne nastavitve (glej razdelek 2.2 (Ustvarjanje fotografije s sepija barvnim tonom)) za sepija barvni ton (162 138 101). Seveda pa ima uporabnik možnost, da nastavi sepija barvni ton povsem po svoje.

Nato ustvarimo novo plast imenovano `plastMaske`. Ustvarimo jo z `gimp-layer-new`. Prvi parameter `fotografija` je slika, kateri bomo dodali novo plast `plastMaske`. Drugi in tretji parameter sta dimenziji nove plasti. Želimo, da dimenziji nove plasti sovpadata z dimenzijami slike `fotografija`, zato za dimenziji plasti `plastMaske` uporabimo dimenziji slike `fotografija`. Tako dosežemo, da bo nova plast velika toliko kot je velika slika `fotografija`. Torej s `(car (GIMP-image-width fotografija))` dostopamo do širine slike `fotografija`, s `(car (GIMP-image-height fotografija))`, pa do višine slike. Naslednji (četrti) parameter je tip plasti, ki pove kateri barvni model plast uporablja. Izberemo RGB barvni model, zato zapišemo 0. Sledi peti parameter, s katerim določimo ime plasti. Šesti parameter je prosojnost plasti. Vrednost za prosojnost je med 0 in 100. Zadnji parameter določa način plasti. Določimo način `COLOR-MODE`. Več o načinih plasti sem razložila v razdelku 2.2 (Urejanje fotografije s sepija barvnim tonom).

Ko sliko urejamo ročno, ustvarimo novo plast s klikom na bližnjico `ustvari novo plast` v zavihku plasti. Ker imamo sliko (ki jo spreminjamo) odprto v oknu obdelovane slike, se nova plast doda tej sliki. Ko pa pišemo skripto, ni očitno, kateri sliki dodajamo plast. To moramo storiti posebej. Plast `plastMaske` dodamo na sliko `fotografija`. To storimo z : `(GIMP-image-add-layer fotografija plastMaske -1)`. Zadnji parameter -1 pomeni, da želimo plast dodati nad trenutno aktivno plast obdelovane slike.

Nato plast `plastMaske` zapolnimo z barvo ospredja: `(GIMP-drawable-fill plastMaske 0)`. To smo pri ročni obdelavi storili že v oknu, ki se odpre s klikom na ukaz `dodaj masko`, kjer smo izbrali barvo ospredja kot vrsto polnila. Postopek je podrobno opisan v drugem primeru razdelka 2.2 (Ustvarjanje fotografije s sepija barvnim tonom).

Ustvarimo masko z imenom `maska`: `(set! (car (GIMP-layer-create-mask plastMaske 0)))`. Z `(GIMP-layer-create-mask plastMaske 0)` ustvarimo masko na plasti `plastMaske`. 0 pomeni belo neprosojno masko. Kaj je maska, sem opisala v razdelku 2.2 (Urejanje fotografije s sepija barvnim tonom), v tretjem načinu dodajanja sepija barvnega tona. S `car` zopet dostopamo do maske, ki jo `gimp-layer-create-mask` vrne kot rezultat. Pri ročnem postopku smo videli, da moramo plasti dodati masko. Ker smo kliknili na ustrezno plast, se je točno vedelo, za katero plast gre. Prti pisanju skripte pa moramo plast določiti preko imena plasti. Zato Uporabimo ukaz `(GIMP-layer-add-mask plastMaske maska)`, s katerim povemo, da masko `maska` dodajamo plasti `plastMaske`.

Potem z `(gimp-layer-resize-to-image-size sepijaPlast)` plasti `sepijaPlast` spremenimo dimenziji tako, da ustrezajo dimenzijam obdelovane slike. To smo pri ročni obdelavi fotografije storili že ob ustvarjanju plasti z imenom `sepija`.

Skopiramo plast `sepijaPlast`: `(gimp-edit-copy sepijaPlast)`. Nato jo prilepimo v masko plasti `plastMaske`: `(GIMP-edit-paste maska 0)`. S `(selection (car (GIMP-edit-paste maska 0)))`, ustvarimo plavajoči izbor in ga zasidramo plavajoči izbor: `(gimp-floating-sel-anchor selection))`. Več smo razložili pri ročni obdelavi v 2.2 (Urejanje fotografije s sepija barvnim tonom)). Postopek je popolnoma enak.

Z `(gimp-invert maska)` preobrnemo masko, da dobimo ustrezne barve za sepija ton. Pri ročni obdelavi smo to dosegli z ukazom `preobrni` znotraj menija `barve`.

Na koncu spojimo še plast `plastMaske` s sliko `fotografija`: `(GIMP-image-merge-down fotografija plastMaske 0)`. Pri ročni obdelavi nam tega ni poterebno storiti, ker GIMP to stori sam.

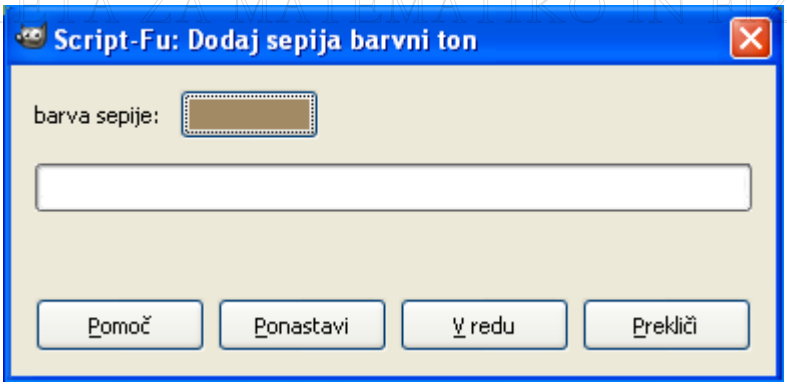
Preden zaključimo glavno funkcijo, zaključimo še blok »razveljavi«.

Kot smo omenili v razdelku 3.3.1 (Sestava Skript – Fu skripte), moramo pokrbeti za registracijo glavne funkcije. Vsako registracijo funkcije začnemo z script-fu-register. Sledijo ji obvezni parametri, katerih pomen je razložen v razdelku 3.3.1 (Sestava Skript – Fu skripte).

Dodani parametri so parametri, ki jih glavna funkcija sprejme. V našem primeru so to slika z imenom `fotografija`, aktivna plast z imenom `ozadje` in `barva` s katero določimo barvo sepije. Vsakemu od njih moramo določiti tip parametra, opis parametra in njegovo privzeto vrednost. Prvi dodani parameter je slika in je zato vrste `SF-IMAGE`. Opišemo ga z »obdelovana slika z imenom fotografija«, njegova privzeta vrednost pa je 0. Drugi dodani parameter je plast in zato vrste `SF-DRAWABLE`. Njegov opis je »aktivna plast slike z imenom ozadje«, njegova privzeta vrednost je 0. Tretji dodani parameter je barva in je vrste `SF-COLOR`. Njegov opis je: »barva sepije«. Njegova privzeta vrednost je barva, ki je v barvnem modelu RGB videti takole: '(162 138 101).

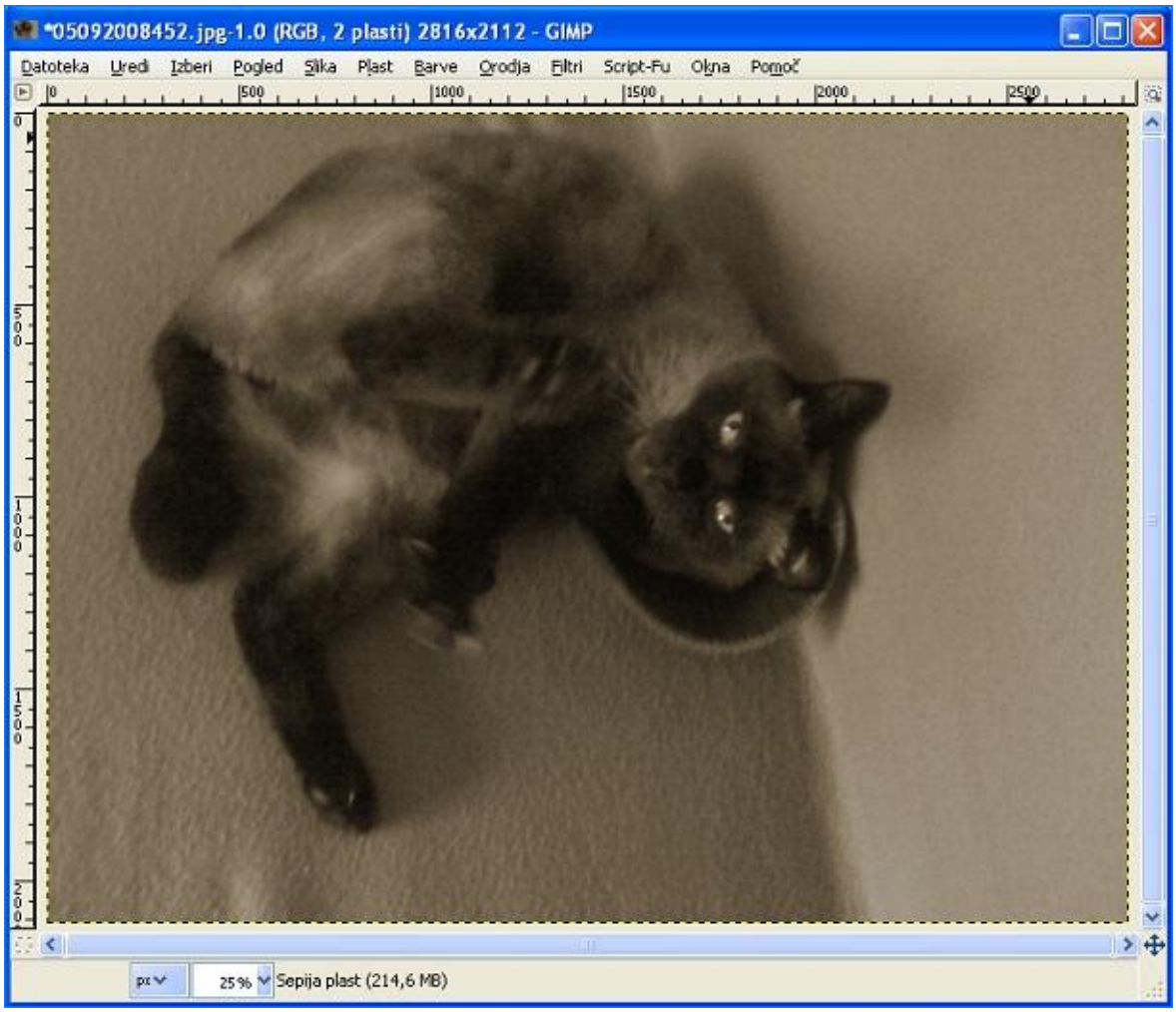
Nazadnje z `(script-fu-menu-register "script-fu-Sepija-barvni-ton" "<Image>/Script-Fu/Sepija")` povemo, da se bo skripta v GIMPu pojavila v meniju Script-Fu, v mapi Sepija.

Ko zaženemo skripto, se pojavi pogovorno okno, kjer lahko spremenimo privzeto barvo, ki določa barvo sepije. Barvo spremenimo s klikom na obarvano okence. Ko izberemo želeno barvo, kliknemo na gumb `V redu`. S klikom na gumb `ponastavi`, Gimp zopet ponudi privzeto barvo, ki smo jo določili v registraciji skripte.



Slika 73: Nastavitev sepija barve

Skripta kot rezultat vrne sliko, obarvano s sepija barvnim tonom (Slika 74).



Slika 74: Rezultat skripte za sepijo

Skripta za dodajanje sepija barvnega tona:

```
(define ; definicija glavne funkcije z imenom Sepija barvni ton
    (script-fu-Dodaj-sepija-barvni-ton
        fotografija ; obdelovana fotografija
        ozadje      ; privzeta plast slike fotografija
        barva       ; sepija barva
    )

    (gimp-image-undo-group-start fotografija) ; vse, kar je zapisano v tem
                                              ; "bloku" lahko razveljavimo

    (let*
        (
            (sepijaPlast 0) ; definiramo lokalno spremenljivko sepijaPlast
            (plastMaske 0)  ; definiramo lokalno spremenljivko plastMaske
            (maska 0)       ; definiramo lokalno spremenljivko maska
        )

      (set! sepijaPlast ; sepijaPlast je kopija privzete plasti
            (car
                (gimp-layer-copy
                    ozadje
                    FALSE
                )
            )
        )

        (gimp-layer-set-name sepijaPlast "Sepija plast")
                                    ; sepijaPlast poimenujemo Sepija plast

        (gimp-image-add-layer fotografija sepijaPlast -1) ; sliki fotografija
                                              ; dodamo plast sepijaPlast
                                              ; -1 pove, da dodamo plast
                                              ; nad trenutno aktivno plast

        (gimp-desaturate sepijaPlast)
                               ; plast sepijaPlast spremenimo v črno-belo

        (gimp-context-set-foreground barva) ; nastavimo barvo ospredja

        (set! plastMaske ; ustvarimo novo plast plastMaske, ki ji dodamo masko
            (car
                (gimp-layer-new
                    fotografija
                           ; slika, kateri bomo novo plast PlastMaske dodali

                   (car (gimp-image-width fotografija))  ; širina slike
                   (car (gimp-image-height fotografija)) ; višina slike
                    0                                    ; tip slike fotografija
                    "Plast z masko"                      ; ime plasti plastMaske
                    100                                     ; prekrivnost plasti
                    COLOR-MODE                              ; način plasti
                )
            )
        )
    )
```

```
        (gimp-image-add-layer fotografija plastMaske -1)
                                                ; plast plastMaske dodamo
                                                     ; sliki fotografija
                                                     ; -1 pomeni, da novo
                                                     ; plast dodamo
                                                     ; nad trenutno aktivno
                                                     ; plast

        (gimp-drawable-fill plastMaske 0) ; plast plastMaske zapolnimo z barvo
                                          ; ospredja

        (set! maska ; ustvarimo masko na plasti plastMaske
            (car
                (gimp-layer-create-mask plastMaske 0) ; 0 je tip maske
            )
        )

        (gimp-layer-add-mask plastMaske maska)
                                           ; masko dodamo plasti Plast maske

        (gimp-layer-resize-to-image-size sepijaPlast) ; plasti sepijaPlast
                                                      ; spremenimo
                                           ; dimenziji tako, da ustrezajo
                                           ; dimenzijam obdelovane slike

        (gimp-edit-copy sepijaPlast) ; skopiramo plast sepijaPlast

        (let ; skopirano sepijaPlast prilepimo v masko
                 (
                     (selection
                         (car
                             (gimp-edit-paste maska 0)
                         )
                     )
                 )
            (gimp-floating-sel-anchor selection) ; zasidramo plavajoči izbor
        )

        (gimp-invert maska)
                    ; preobrnemo masko, da dobimo ustrezne barve za sepijo

        (gimp-image-merge-down ; spojimo plast plastMaske s sliko fotografija
             fotografija
             plastMaske
             0                 ; način spajanja
        )

    )

    (gimp-image-undo-group-end fotografija) ; konec "razveljavi" bloka

)


; registracija glavne funkcije
(script-fu-register "script-fu-Dodaj-sepija-barvni-ton"
```

```
    ; obvezni parametri
      "Dodaj sepija barvni ton"              ; ime glavne funkcije
      "Poljubno fotografijo spremenimo v     ; opis glavne funkcije
       fotografijo s sepija barvnim tonom"
      "Andreja Dokl"                         ; avtor
      ""                                     ; avtorske pravice
        "03.2010"                             ; datum izdelave
        "RGB"                                 ; tip slike

    ; dodani parametri
      SF-IMAGE      "obdelovana slika z imenom fotografija"      0
      SF-DRAWABLE   "aktivna plast slike z imenom ozadje"        0
      SF-COLOR      "barva sepije"                   '(162 138 101)
)

(script-fu-menu-register          ; meni, kjer se v GIMPu pojavi skripta
    "script-fu-Dodaj-sepija-barvni-ton"
    "<Image>/Script-Fu/Sepija"
)
```

4.3 Skripta za dodajanje besedila na sliko

Skripta začnemo z definicijo glavne funkcije z imenom `script-fu-Besedilo`. Glavna funkcija sprejme tri parametre. Ker skripta deluje na odprti sliki, potrebujemo parameter `fotografija`, parameter `ozadje` in parameter `besedilo`. Parameter `fotografija` označuje sliko na kateri skripta deluje, parameter `ozadje` označuje aktivno plast slike `fotografija`, parameter `besedilo`, pa predstavlja besedilo, ki ga želimo dodati na sliko.

Kot v razdelku 4.2 (Skripta za sepijo) tudi tukaj poskrbimo, da bomo učinek skripte lahko razveljavili. Začnemo z ukazom: `(gimp-image-undo-group-start fotografija)`, končamo pa z `(gimp-image-undo-group-end fotografija)`.

Glavna ideja je v tem, da z ukazom na sliko dodamo besedilo. Pri ročni obdelavi dosežemo enak učinek s klikom na `orodje za dodajanje besedila` v oknu orodjarna. Razlika je v tem, da pri ročnem dodajanju besedila lahko nastavimo tudi barvo besedila, lahko vklopimo namigovanje in vsiljeno namigovanje. Podrobnosti so razložene v razdelku 2.3 (Dodajanje besedila na sliko). Barvo besedila v skripti nastavimo z ukazom `gimp-text-layer-set-color`, namigovanje in vsiljeno namigovanje pa v skripti ne omogočimo.

Definiramo spremenljivko z imenom `napis`, ki predstavlja plast. Pri ročnem dodajanju besedila na sliko, GIMP na sliki samodejno ustvari plast, na kateri je besedilo. Tu pa plast z imenom `napis` ustvarimo z: `gimp-text-fontname`, ki sprejme deset parametrov. Prvi parameter je slika z imenom `fotografija`, na kateri skripta deluje. Drugi parameter je aktivna plast slike `fotografija`, ki ji pripnemo plast z besedilom. Tretji parameter je odmik besedila od levega zgornjega kota slike v vodoravni smeri, četrti parameter pa je odmik besedila od levega zgornjega kota slike v navpični smeri. Peti parameter je besedilo, ki ga želimo prikazati na sliki. Šesti parameter je velikost okvirja okoli besedila. Ker okvirja ne želimo, zapišemo 0. Sedmi parameter predstavlja glajenje robov pisave (pri zelo majhnih pisavah). Več o glajenju robov je zapisano v razdelku 2.3 (Dodajanje besedila na sliko). Osmi parameter je velikost pisave, deveti pa merilo za velikost pisave. Če želimo velikost pisave meriti v slikovnih točkah, napišemo na mesto osmega parametra 0. Zadnji parameter je vrsta pisave. Parameter podamo kot niz, v katerem je zapisana želena pisava. S `car` dostopamo do plasti besedila, ki ga `gimp-text-fontname` vrne kot rezultat.

Barvo besedila moramo v skripti nastaviti posebej: `(gimp-text-layer-set-color napis (list 0 0 0))`.

Z `(GIMP-floating-sel-anchor napis)` zasidramo plavajoči izbor, ki smo ga ustvarili z `gimp-text-fontname`. Tega nam pri ročnem urejevanju ni potrebno storiti.

Nato zaključimo »blok razveljavi«: `(gimp-image-undo-group-end fotografija)`.

In na koncu še osvežimo zaslon: `GIMP-displays-flush`.

Sledi registracija glavne funkcije. Registracija je videti skoraj enako kot pri skripti za sepija barvni ton. Razlika je v tem, da glavna funkcija skript za dodajanje besedila na sliko sprejme parameter `besedilo`, ne sprejme pa parametra `barva`. Zato pri registraciji dodamo tri parametre: sliko z imenom `fotografija`, njeno privzeto plast `ozadje` in parameter `besedilo`. Parameter `besedilo` je tipa SF-STRING.

S `(script-fu-menu-register "script-fu-Dodaj-besedilo" "<Image>/Script-Fu/Besedilo")` povemo, kje v GIMPu se nahaja skripta z imenom `script-fu-Dodaj-besedilo`.

Skripta kot rezultat vrne sliko z besedilom (Slika 75).

Slika 75: Rezultat skripte za dodajanje besedila

Skripta za dodajanje besedila poljubni sliki:

```
(define
    (script-fu-Dodaj-besedilo
        fotografija
        ozadje
        besedilo
    )

    (gimp-image-undo-group-start fotografija) ; začetek razveljavi bloka

    (define napis ; ustvarimo spremenljivko v kateri je
                  ; shranjena plast z besedilom
        (car
            (gimp-text-fontname
                fotografija           ; ime slike
                ozadje                ; besedilo je pripeto plasti
                                      ; ozadje kot plavajoči izbor
                1100                  ; odmik v slikovnih točkah od levega
                                      ; zgornjega kota slike v vodoravni smeri
                180                   ; odmik v slikovnih točkah od levega
                                      ; zgornjega kota slike v navpični smeri
                "Feliks"              ; besedilo, ki se prikaže na sliki
                0                     ; velikost okvirja okoli besedila
                                      ; okvirja ne želimo, zato zapišemo 0
                1                     ; glajenje robov pisave (true=1, false=0)
                120                   ; velikost pisave
                0                     ; enota za velikost so slikovne točke
                "Ravie Semi-Expanded" ; vrsta pisave
            )
        )
    )

    (gimp-text-layer-set-color napis (list 0 0 0)) ; barva besedila napis

    (gimp-floating-sel-anchor napis) ; zasidramo plavajoči izbor napis

    (gimp-image-undo-group-end fotografija) ; zaključimo "blok razveljavi"

    (gimp-displays-flush) ; osvežimo zaslon
)

; registracija glavne funkcije
(script-fu-register "script-fu-Dodaj-besedilo"

    ; obvezni parametri
        "Dodaj besedilo"                          ; ime glavne funkcije
        "Doda besedilo na sliko"                  ; opis glavne funkcije
        "Andreja Dokl"                            ; avtor
        ""                                        ; avtorske pravice
        "03.2010"                                 ; datum izdelave
        "RGB, GRAY"                               ; tip slike

    ; dodani parametri
        SF-IMAGE     "obdelovana slika z imenom fotografija" 0
        SF-DRAWABLE  "aktivna plast slike fotografija" 0
```

```
        SF-STRING    "besedilo, ki ga želimo dodati sliki" "besedilo"

)

(script-fu-menu-register          ; meni, kjer se v GIMPu pojavi skripta
    "script-fu-Dodaj-besedilo"
    "<Image>/Script-Fu/Besedilo"
)
```

4.4 Skripta za spreminjanje velikosti slike

Pri tej skripti bomo predpostavili, da potrebujemo tako, ki bo poljubni fotografiji spremenila velikost na 1408 x 1106. Nova velikost slike bo torej vedno enaka in ne bo parameter skripte.

Tudi skripto za spreminjanje velikosti slike začnemo z definicijo glavne funkcije z imenom `script-fu-Spremeni-velikost`. Funkcija sprejme dva parametra. Prvi parameter `fotografija` je slika, na kateri skripta deluje. Drugi parameter `ozadje` označuje aktivno plast slike `fotografija`.

Uporabimo »blok za razveljavitev«. Začnemo ga z `(gimp-image-undo-group-start fotografija)`. Parameter `fotografija` je slika, na kateri bo ukaz `razveljavi` izvršen.

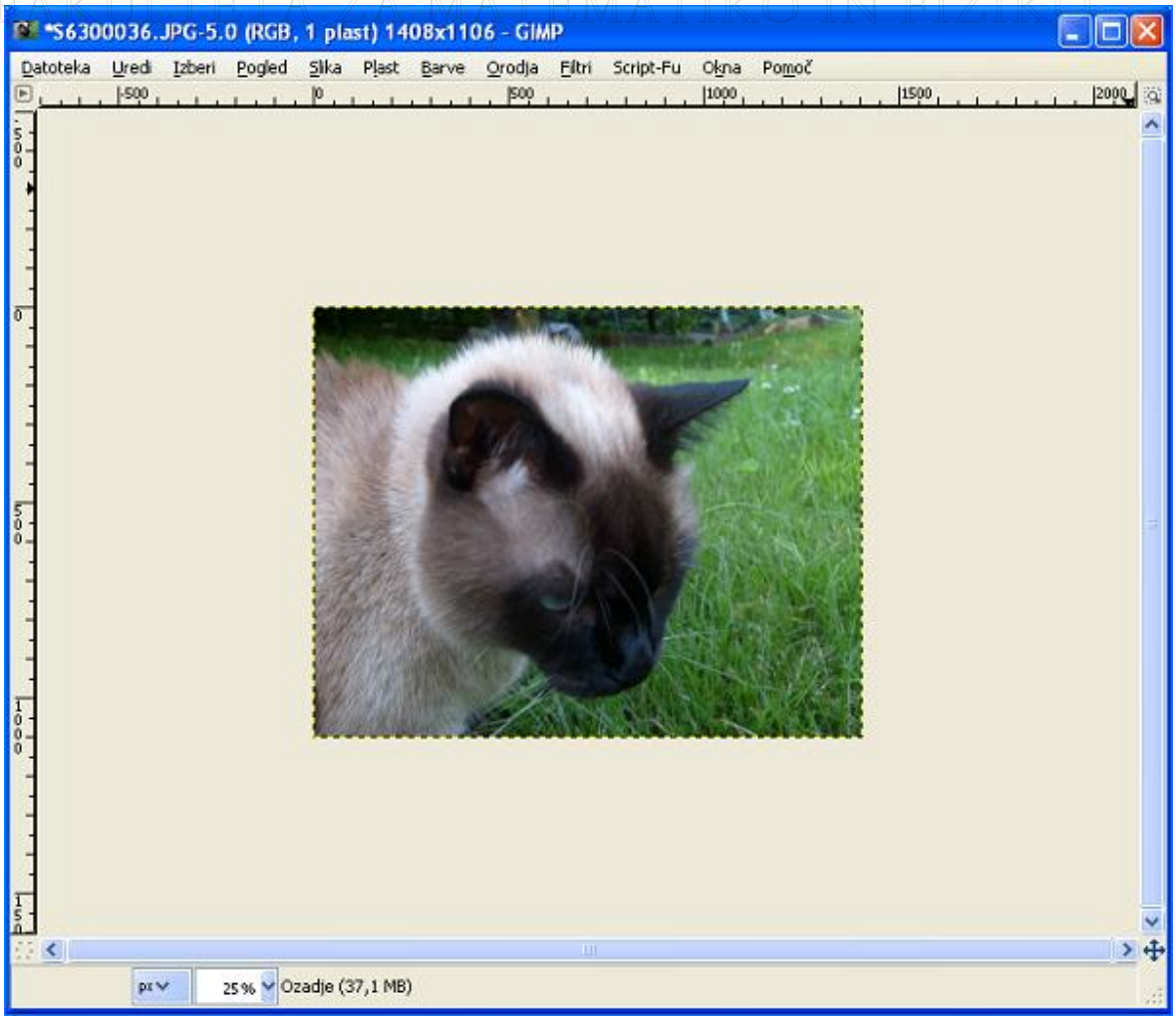
Glavni del skripte izvedemo z `(gimp-image-scale fotografija 1408 1106)`. Prvi parameter fotografija je slika, kateri spreminjamo velikost. Drugi in tretji parameter sta novi dimenziji slike fotografija. Drugi parameter je širina, tretji pa višina. S tem sliko povečamo ali pomanjšamo tako, da jo »skaliramo«. Ukaz `gimp-image-scale` privzeto uporablja za spreminjanje merila slike kubično interpolacijsko metodo. Nalogo ukaza `gimp-imag-scale` pri ročnem spreminjanju velikosti opravi ukaz `spremeni merilo slike` v meniju `slika`, v katerem lahko nastavimo dimenziji slike. Pri ročnem spreminjanju velikosti lahko izberemo tudi vrsto interpolacije. Ukaz `gimp-image-scale` privzeto uporablja kubično interpolacijsko metodo.

Zaključimo »blok za razveljavitev«: `(gimp-image-undo-group-end fotografija)`.

Sledi registracija glavne funkcije z imenom `script-fu-Spremeni-velikost`. Registracija je enaka registraciji skripte za dodajanje besedila na sliko. Opisana je v razdelku 0 (Skripta za dodajanje besedila na sliko).

Skripto zaključimo z `(skript-fu-menu-register "script-fu-Spremeni-velikost" "<Image>/Script-Fu/Velikost")` in tako povemo kje v GIMPu se skripta nahaja.

Skripta kot rezultat vrne sliko (Slika 76).



Slika 76: Rezultat skripte za spreminjanje velikosti slike

Skripta za spreminjanje velikosti:

```
(define
    (script-fu-Spremeni-velikost
        fotografija
        ozadje
    )

    (gimp-image-undo-group-start fotografija) ; razveljavi blok

    (gimp-image-scale fotografija 1408 1106) ; velikost slike fotografija
                                             ; spremenimo na velikost z
                                              ; širino 1408 in višino 1106

    (gimp-image-undo-group-end fotografija) ; zaključimo "blok razveljavi"
)

; registracija glavne funkcije
(script-fu-register "script-fu-Spremeni-velikost"

 ; obvezni parametri
        "Spremeni velikost"                       ; ime glavne funkcije
        "Spremeni velikost slike"                 ; opis glavne funkcije
        "Andreja Dokl"                            ; avtor
        ""                                        ; avtorske pravice
        "03.2010"                                 ; datum izdelave
        "RGB, GRAY"                               ; tip slike

    ; dodani parametri
        SF-IMAGE        "obdelovana slika z imenom fotografija"  0
        SF-DRAWABLE     "aktivna plast slike fotografija"        0

)

(script-fu-menu-register          ; meni, kjer se v GIMPu pojavi skripta
    "script-fu-Spremeni-velikost"
    "<Image>/Script-Fu/Velikost"
)
```

5 Združitev skript

Skripta z imenom `Zdruzitev` je skripta, v kateri združimo vse tri skripte iz razdelkov 4.2 (Skripta za sepijo), 0 (Skripta za dodajanje besedila na sliko) in 0 (Skripta za spreminjaje velikosti slike). Naredimo torej eno skripto.

Skripto začnemo z definicijo glavne funkcije `skript-fu-Zdruzitev-skripta`, ki sprejme štiri parametre. Prvi in drugi parameter sta enaka kot v vseh treh posamičnih skriptah. To sta parametra `fotografija` in `ozadje`. Prvi parameter je slika na kateri skripta deluje, drugi parameter pa je aktivna plast slike na kateri skripta deluje. Ker sta nujna v vseh funkcijah, ki delujejo na slikah, jih moramo uporabiti tudi v skripti `zdruzitev`. Tretji parameter je parameter `besedilo`, četrti parameter pa `barva`. Tretji parameter nosi podatek o besedilu, ki ga dodajamo na sliko, četrti parameter pa je parameter, ki določa barvo za sepija barvni ton.

Ker obstaja možnost, da ne bomo zadovoljni s končnim rezultatom skripte, se poslužimo uporabe »razveljavi bloka«. Kot v posameznih skriptah, tudi tukaj »blok razveljavi« začnemo z: `(gimp-image-undo-group-start fotografija)` in končamo z: `(gimp-image-undo-group-end fotografija)`. »Blok razveljavi« končamo tik pred registracijo.

Glavni del skripte predstavljajo ukazi, s katerimi pokličemo že napisane skripte oziroma njihove glavne funkcije.
Najprej s `(script-fu-Dodaj-besedilo fotografija ozadje besedilo)` pokličemo funkcijo, s katero dodamo besedilo na fotografijo. Nato sledi klic funkcije za spreminjanje velikosti slike: `(script-fu-Spremeni-velikost fotografija ozadje)`. Kot zadnjo pokličemo še funkcijo za dodajanje sepija barvnega tona: `(script-fu-Dodaj-sepija-barvni-ton fotografija ozadje barva)`.

Registracija glavne funkcije z imenom `skript-fu-Zdruzitev` je skoraj enaka registraciji funkcije `skript-fu-Sepija-barvni-ton` skripte za dodajanje sepija barvnega tona. Skripta `zdruzitev` ima (v nasprotju s skripto za dodajanje sepija barvnega tona) še četrti parameter `besedilo`. Spremenimo le parameter za opis glavne funkcije.

S `(script-fu-menu-register "script-fu-Zdruzitev-skript" "<Image>/Script-Fu/Zdruzitev")` povemo kje se funkcija nahaja v programu GIMP.

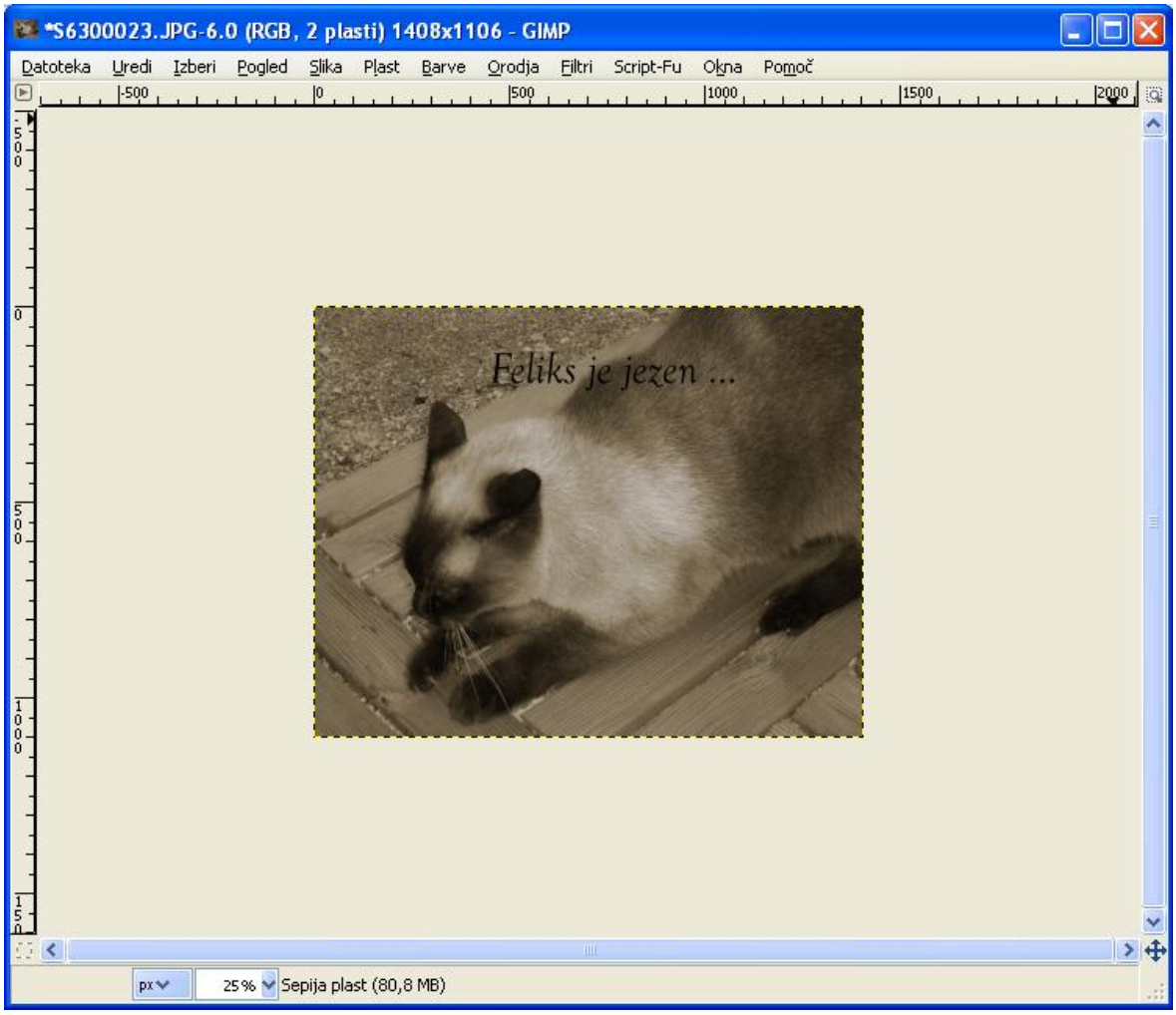
Dobimo končni rezultat našega dela.

Skripta, ki združuje programsko kodo vseh treh skript, je videti takole:

```
(define
    (script-fu-Zdruzitev
        fotografija ; obdelovana fotografija
        ozadje      ; privzeta plast slike fotografija
        besedilo    ; poljubno besedilo
        barva       ; sepija barva
    )

    (gimp-image-undo-group-start fotografija) ; vse, kar je zapisano v tem
                                      ; "bloku" lahko razveljavimo

    (script-fu-Dodaj-besedilo fotografija ozadje napis)
    (script-fu-Spremeni-velikost fotografija ozadje)
    (script-fu-Dodaj-sepija-barvni-ton fotografija ozadje barva)

    (gimp-image-undo-group-end fotografija) ; konec "razveljavi" bloka

)

; registracija glavne funkcije
(script-fu-register "script-fu-Zdruzitev"

    ; obvezni parametri
      "Zdruzitev skripta"                   ; ime glavne funkcije
      "Poljubno fotografijo spremenimo v    ; opis glavne funkcije
       fotografijo s sepija barvnim tonom,
       ji spremenimo velikost in dodamo
       napis"
      "Andreja Dokl"                        ; avtor
      ""                                    ; avtorske pravice
       "03.2010"                            ; datum izdelave
       "RGB"                                ; tip slike

    ; dodani parametri
      SF-IMAGE         "obdelovana slika z imenom fotografija"      0
      SF-DRAWABLE      "aktivna plast slike z imenom ozadje"        0
      SF-STRING        "poljubno besedilo"                   "besedilo"
      SF-COLOR         "barva sepije"                 '(162 138 101)

)

(script-fu-menu-register          ; meni, kjer se v Gimpu pojavi skripta
    "script-fu-Zdruzitev "
    "<Image>/Script-Fu/Zdruzitev"
)
```

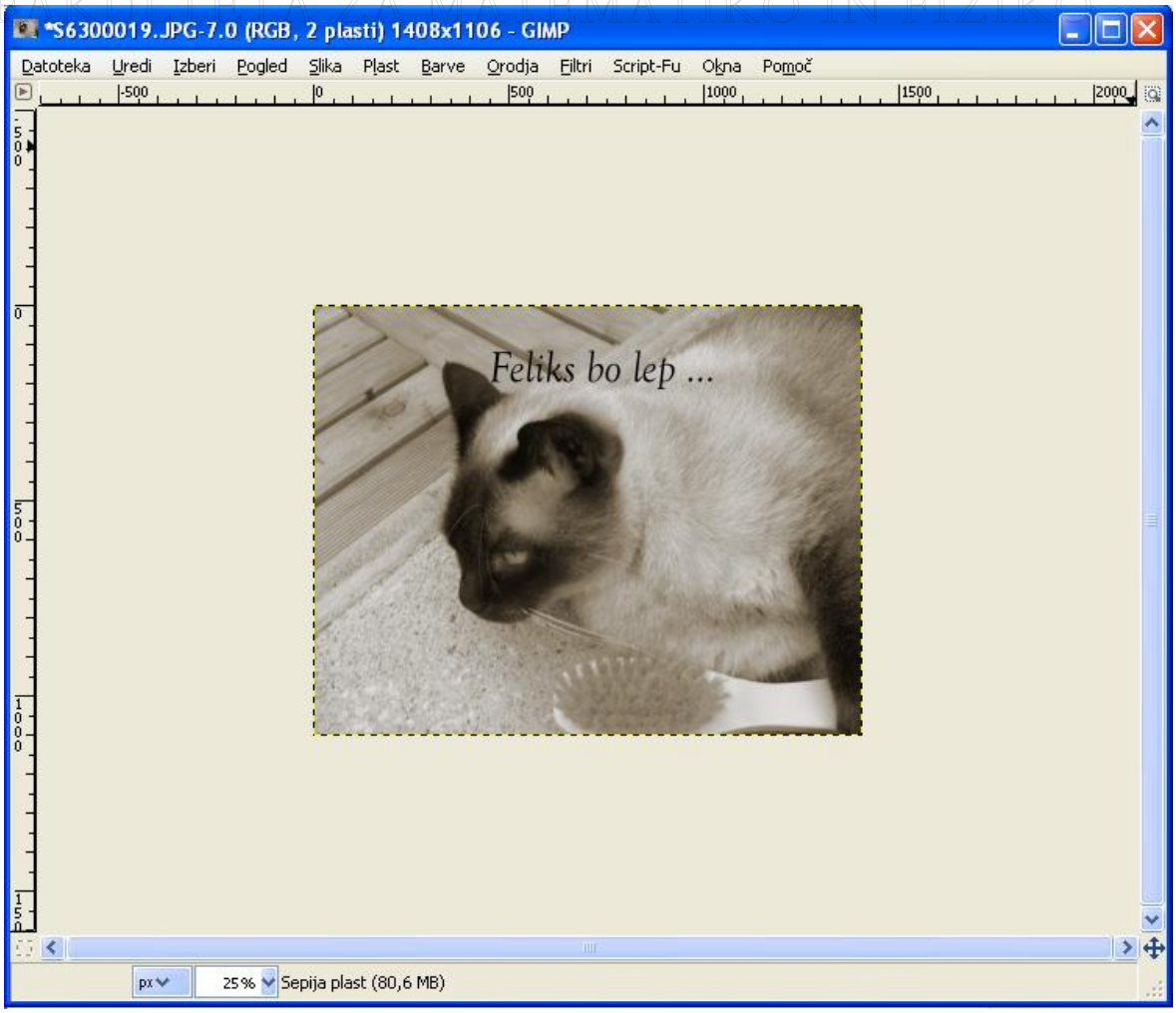
6 Primeri združitve

Prvi primer uporabe skripte, ki združuje ukaze iz vseh treh posamičnih skript iz poglavja 4 (Skripte za primere iz drugega poglavja), lahko vidimo na sliki Slika 77. Sliko smo najprej pomanjšali na velikost 1408 x 1106 slikovnih točk. Dodali smo ji besedilo »Feliks je jezen …«. Levi zgornji kot besedila je od levega zgornjega kota slike oddaljen 900 pikslov v vodoravni smeri in 180 pikslov v navpični smeri. Velikost pisave je 220 pikslov, vrsta pisave pa je `Goudy Old Style Italic`. Barva pisave je nastavljena na črno barvo.



Slika 77: Prvi primer združitve skript

V drugem primeru uporabe skripte smo sliko najprej pomanjšali na velikost 1408 x 1106 slikovnih točk. Dodali smo napis »Feliks bo lep …«. Levi zgornji kot besedila je od levega zgornjega kota slike oddaljen 900 slikovnih točk v vodoravni smeri in 180 slikovnih točk v navpični smeri. Začetna barva pisave (pred uporabo ukazov za sepija barvni ton) je nastavljena na črno barvo. Velikost pisave je 180 slikovnih točk, vrsta pisave pa je `Goudy Old Style Italic`. Končni rezultat vidimo na sliki Slika 78Slika 77.



Slika 78: Drugi primer združitve skript

V tretjem primeru uporabe skripte smo sliki najprej spremenili velikost na 1408 x 1106 slikovnih točk. Napis »Feliks …« je od levega zgornjega kota slike oddaljen 900 slikovnih točk v vodoravni smeri in 180 slikovnih točk v navpični smeri. Barva pisave (pred uporabo ukazov za sepija barvni ton) je črna. Velikost pisave je 180 slikovnih točk, vrsta pisave pa je `Goudy Old Style Italic`. Končni rezultat vidimo na sliki Slika 79.

Slika 79: Tretji primer združitve skript

7 Zaključek

V diplomski nalogi sem spoznala program GIMP, ki je namenjen risanju in obdelavi slik in fotografij.

Posvetila sem se trem različnim primerom obdelave fotografij. Pokazala smo, kako spremeniti velikost slike, kako na več načinov ustvariti fotografijo s sepija barvnim tonom in kako dodati poljubno besedilo na poljuben del slike. Postopke sem razložila korak za korakom in korake tudi utemeljila. Vse to sem opisala v poglavju 2 (Primeri uporabe).

Nato sem razložila osnove skriptnega jezika Scheme. Pojasnila sem, kako ustvariti spremenljivke, funkcije, zanke … in kako jih uporabljati. Povedala smo, kaj je skripta in kako jo napisati. Povedala smo kaj je glavna funkcije skripte in razložila pojem registracije.

Skripti za sepija barvni ton in za dodajanje besedila na sliko sta nastali s pomočjo skript, ki sem jih našla na spletu, skripta za spreminjanje velikosti slike pa je nastala brez tovrstne pomoči. V skriptah sem uporabljala ukaze, s katerimi sem dosegla enak ali vsaj podoben učinek kot v poglavju 2 (Primeri uporabe), kjer sem slike obdelala »ročno«. Nato sem vse tri skripte združila v eno samo. Na eni sami sliki, ki je odprta v oknu obdelovane slike, se tako izvedejo ukaze iz treh različnih skript. Najprej sem sliki spremenila, ji dodala besedilo in jo spremenila v sliko z dodanim sepija barvnim tonom

Ugotovili sem, da je GIMP program, po katerem lahko poseže tudi precej zahteven uporabnik. Ponuja veliko možnosti risanja in obdelave fotografij in omogoča pisanje lastnih skript.

8 Literatura

1. Bunks C. *Grokking the GIMP, New Riders Publishing, 2000*, pridobljeno iz http://GIMP-savvy.com/BOOK/index.html (zadnji dostop: 9.6.2010)
2. Dybvig R. K. *The Scheme Programming Language, Second Edition*, pridobljeno iz http://www.scheme.com/tspl2d/index.html (zadnji dostop: 28.2.2010)
3. Grobgeld D. *A Scheme Tutorial for GIMP Users, 2002*,pridobljeno iz http://imagic.weizmann.ac.il/~dov/GIMP/scheme-tut.html (zadnji dostop: 9.6.2010)
4. Krek J. *Prostorski modelirnik z BREP in CSG predstavitvijo*. Ljubljana: Fakulteta za strojništvo, pridobljeno iz http://www.lecad.uni-lj.si/~krek/yasm/html/node119.html (zadnji dostop: 4.6.2010)
5. Ousterhout J. K. *Scripting: Higher Level Programming for the 21st Century*, pridobljeno iz http://home.pacbell.net/ouster/scripting.html (zadnji dostop: 9.6.2010)
6. Community Scheme Wiki, Scheme frequently asked questions, pridobljeno iz http://community.schemewiki.org/?scheme-faq-general (zadnji dostop: 29.10.2009)
7. e – gradiva, Skriptni jeziki, pridobljeno iz http://www.s-sers.mb.edus.si/gradiva/w3/javascript/skript.html (zadnji dostop: 4.6.2010)
8. Everything2, Teach Yourself Scheme, pridobljeno iz http://everything2.com/title/Teach+Yourself+Scheme (zadnji dostop: 28.2.2010)
9. Generating Images with Script – Fu, pridobljeno iz http://www.cs.grinnell.edu/~rebelsky/Glimmer/ScriptFu/Labs/script-fu.html (zadnji dostop: 28.2.2010)
10. Gimp, pridobljeno iz http://www.gimp.si/ (zadnji dostop: 24.5.2010)
11. GIMP – Basic Scheme, pridobljeno iz http://www.gimp.org/tutorials/Basic_Scheme/ (zadnji dostop: 4.6.2010)
12. GIMP Library Reference Manual, pridobljeno iz http://developer.gimp.org/api/2.0/libgimp/index.html (zadnji dostop: 6.6.2010)
13. Gimp user manual v0.7, Writing a Script – Fu, pridobljeno iz http://www.ic.al.lg.ua/~ksv/gimpdoc-html/write_sc.html (zadnji dostop: 4.6.2010)
14. GimpTalk, pridobljeno iz http://www.gimptalk.com/forum/topic/An-Introduction-To-Script-fu-10447-1.html (zadnji dostop: 28.2.2010)
15. GNU image manipulation program, user manual, pridobljeno iz: http://docs.GIMP.org/2.6/en/(zadnji dostop: 16.6.2010)
16. Monitor, Groovy, pridobljeno iz http://www.monitor.si/clanek/groovy/ (zadnji dostop: 4.6.2010)
17. Monitor, Odraščanje gumpca Wilberja, pridobljeno iz http://www.monitor.si/clanek/odrascanje-gumpca-wilberja/ (zadnji dostop: 9.6.2010)
18. NEKAJ MALEGA O GIMPU, pridobljeno iz http://www.kiberpipa.org/~delavnice/gimp/NMOG21052007.pdf (zadnji dostop: 10.6.2010)
19. René Nyffenegger's collection of things on the web, pridobljeno iz http://www.adp-gmbh.ch/misc/tools/script_fu/index.html#examples (zadnji dostop: 28.2.2010)
20. Script – Fu and plug – ins for The GIMP, pridobljeno iz http://www.gimp.org/docs/scheme_plugin/index.html (zadnji dostop: 28.2.2010)
21. Skriptno programiranje: visokonivojsko programiranje za 21. stoletje, pridobljeno iz http://lgm.fri.uni-lj.si/PA/SCRIPTING/HTML/index.html (zadnji dostop: 9.6..2010)
22. Vrste programskih jezikov, prevajanje in tolmačenje, pridobljeno iz http://colos.fri.uni-lj.si/eri/RACUNALNISTVO/PROG_JEZIKI_ORODJA/vrste_progJezikov_prevajanje_tolmacenje.html (zadnji dostop: 4.6.2010)

23. Wikipedia, GIMP, pridobljeno iz http://sl.wikipedia.org/wiki/GIMP (zadnji dostop: 28.10.2010)
24. Wikipedia, Programski jezik Scheme, pridobljeno iz http://sl.wikipedia.org/wiki/Programski_jezik_Scheme (zadnji dostop: 12.1.2010)
25. Wikipedia, Scheme, pridobljeno iz http://en.wikipedia.org/wiki/Scheme (programming_language) (zadnji dostop: 5.2.2010)
26. Wikipedia, System programming language, pridobljeno iz http://en.wikipedia.org/wiki/System_programming_language (zadnji dostop: 22.2.2010)