UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO
Matematika – praktična matematika (VSŠ)

Petra Fajdiga

PRIMERJAVA RAZLIČNIH RAZVOJNIH ORODIJ ZA PROGRAMIRANJE V PYTHONU

Diplomska naloga

Ljubljana, 2014

Zahvala

Posebna zahvala gre mentorju, ki mi je v času študija, med prakso in ob pisanju diplomske naloge nesebično razdajal svoje znanje ter pri tem pokazal veliko mero potrpežljivosti.
Zahvala gre tudi staršem in starim staršem, ki so me vzgojili v osebo, kakršna sem zdaj, in me pri vsaki moji zamisli moralno in finančno podpirali.
Zahvalila bi se tudi bratu, njegovi ženi in vsem prijateljem, ki so mi pomagali.

KAZALO

Program dela

Proučite različna razvojna orodja, ki jih lahko uporabljamo za pisanje programov v Pythonu. Primerjajte njihovo uporabnost za pisanje različnih tipov programov, kot so npr. enostavni programi, programi za delo z datotekami, programi z grafičnimi uporabniškimi vmesniki, programi za delo z bazami podatkov ...

Primerjajte pomoč, ki jo nudijo pri pisanju, možnosti pri iskanju napak, enostavnost uporabe, dostopnost in podobne lastnosti. Za nekaj najuporabnejših pripravite tudi osnovni uporabniški priročnik.

Mentor: mag. Matija Lokar

POVZETEK

V diplomski nalogi sem predstavila osnovna razvojna orodja za pisanje programov. Delovanje vsakega razvojnega orodja sem slikovno podprla. Poleg razvojnih orodij sem tri razvojna okolja za pisanje programov v programskem jeziku Python tudi podrobno predstavila. Ta tri razvojna okolja so IDLE, Eclipse in PyCharm. Za vsako razvojno okolje sem opisala, kaj vse je potrebno storiti, preden lahko napišemo prvi program in kako ga zaženemo. Najbolj pa sem v diplomski nalogi poudarila, kakšna orodja ima posamezno razvojno okolje.
Del diplomske naloge so še opisi nekaterih drugih razvojnih okolij, ki podpirajo programiranje v jeziku Python. Za ta razvojna okolja sem navedla uradno spletno stran za prenos, podala informacijo, za kateri operacijski sistem so namenjena, navedla, katera osnovna orodja imajo in orodja, ki se razlikujejo od že opisanih, tudi predstavila.

Ključne besede: Python, razvojno okolje, razvojno orodje, IDLE, Eclipse, PyCharm, programiranje, programski jezik.

ABSTRACT

In my thesis I introduced basic development tools for writing programs. Working of each development tool is presented with a picture. I wrote also a detailed presentation of three development environments for writing programs in program language Python. Presented development environments are IDLE, Eclipse and PyCharm. For each of them I described what must be done before we can write the first program and how to start that program. The main focus of the thesis is on the development tools types for each of the presented development environment.
In my thesis you will also find some other development environments, which support programing in Python. For these development environments I added official web page for download, operating systems, basic tools and I also presented some particularities of these basic tools.

Keywords: Python, development evironment, devolopment tool, IDLE, Eclipse, PyCharm, programmig, programming language.

Math. Subj. Class. (2010): 68N15, 86N20
ComputingReview Class.System (1998): D.2.3, D.2.5, D.2.6

1 UVOD

Hitro in učinkovito programiranje je odvisno od znanja programiranja in izbire pravega razvojnega okolja. Razlike med razvojnimi okolji se kažejo v uporabniškem vmesniku, ceni, predvsem pa so si različna po uporabnosti. Če hočemo izbrati najboljše razvojno okolje, moramo vedeti, kaj nam posamezno orodje ponuja. Zato si bomo pogledali zmogljivosti nekaterih razvojnih okolij za programiranje v Pythonu, ki so trenutno na tržišču. Za vrsto orodij bomo našteli le njihove osnovne značilnosti, tri razvojna okolja pa si bomo pogledali natančneje.
Za vsako od teh treh okolij bomo preverili, kje ga lahko dobimo, kako ga pripravimo za delo s programskim jezikom Python in s kakšnimi razvojnimi orodji lahko razpolagamo.
Prvo od okolij je IDLE. IDLE dobimo skupaj s programskim paketom Python, tako da je to okolje prva in logična izbira za vsakega novega programerja v tem programskem jeziku.
Okolju IDLE bo sledil ogled razvojnega okolja Eclipse. Eclipse je nam že dobro poznano razvojno okolje, saj smo ga uporabljali med študijem za pisanje programov v programskem jeziku Java. Poleg tega je tudi brezplačno, kar je še dodaten razlog za ogled.
Ogledu Eclipse bo sledil PyCharm, ki je v tem ožjem izboru edino plačljivo razvojno okolje. Videli bomo, ali zaradi cene ponuja kaj več, čeprav je na prvi pogled zelo podoben okolju Eclipse.

2 SLOVAR

PROGRAMSKI PAKET

Programski paket je paket, ki vsebuje tolmač za Python, ustrezne programske knjižnice in še določene druge programe, ki jih potrebujemo za pisanje programov v programskem jeziku Python.

ODPRTOKODNI PROGRAM

Programom, katerih izvorna koda je dostopna (vidna) vsem, pravimo odprtokodni programi.

BLOKI

Bloki so deli kode, ki sestavljajo vsebino (telo) določenih stavkov – npr. vsebina zank »`for`« in »`while`«, vsebina metode, vsebina »`if`« stavka.

NAPAKE

Poznamo dve vrsti napak.

- Sintaktične napake, ko koda ni napisana v skladu s sintakso programskega jezika. Nanje običajno opozarja pojavno okno.

- Semantične napake, ko se koda obnaša drugače, kot je predvideno:

  - program se uspešno izvede do konca, a je rezultat drugačen od predvidenega. Te napake ugotovimo s preučevanjem kode (pri tem si pomagamo z razhroščevalnikom);

  - med izvajanjem kode povzročijo napako, zaradi katere se program ne more več izvajati naprej. Nanje ponavadi opozarja sporočilo v izvajalni lupini, ki pove, kakšna napaka/izjema je povzročila prekinitev izvajanja programa.

IDE

IDE je kratica za »`integrated development environment`«. Pomeni, da imamo v sklopu istega programa vsaj urejevalnik kode, tolmač kode in razhroščevalnik.

MODUL

Modul je dokument, ki vsebuje metode, razrede in spremenljivke, napisane v Pythonu. Tak dokument spoznamo po končnici »`.py`«. Modul lahko uvozimo v drug modul (za to uporabimo »`import`« in dodamo ime želenega modula).

PAKETI

Paketi so imeniki, v katere shranjujemo module, ki se med seboj dopolnjujejo, ter tako tvorijo funkcionalno celoto. Da lahko govorimo o paketu, mora imenik vsebovati modul »*_init_.py*«. Vsak paket ima lahko poljubno število podpaketov.

PROJEKT

Projekt je zbirka vsega potrebnega za razvoj programa (aplikacije) in njegovo delovanje.
V zbirki so lahko različne knjižnice, nastavitvene datoteke, podatkovne datoteke, testni programi, datoteke, kjer je določen grafični uporabniški vmesnik (GUV), metapodatki, ki opisujejo uporabljeni tolmač.

PREKINITVENA TOČKA

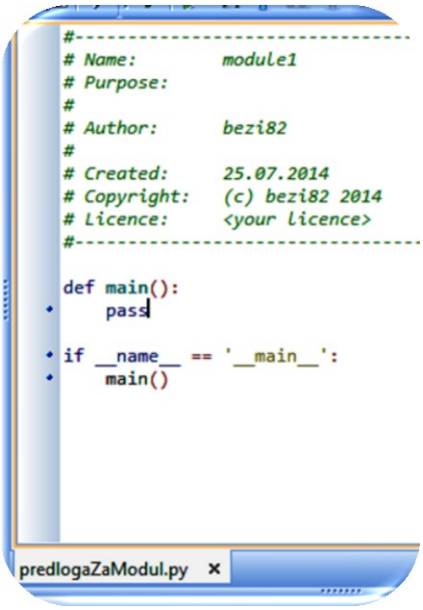
Prekinitvena točka je oznaka, postavljena na vrstico izvorne kode, ki pove razhroščevalniku, naj se ustavi, ko pride do te vrstice. Razhroščevalnik izvede vsako vrstico, dokler ne naleti na prekinitveno točko. Tam si običajno ogledamo vrednosti spremenljivk in korak za korakom sledimo delovanju programa od prekinitvene točke dalje. Z uporabo prekinitvene točke lahko hitro preletimo del, za katerega vemo, da deluje pravilno in se osredotočimo na del, ki povzroča težave.

3 RAZVOJNA ORODJA

Pogledali si bomo vsako izmed osnovnih razvojnih orodij, ki jih bomo skozi diplomo opazovali. Navedli bomo, kaj je naloga orodja, kako nam tako orodje lahko pomaga pri pisanju kode in vse slikovno podprli.

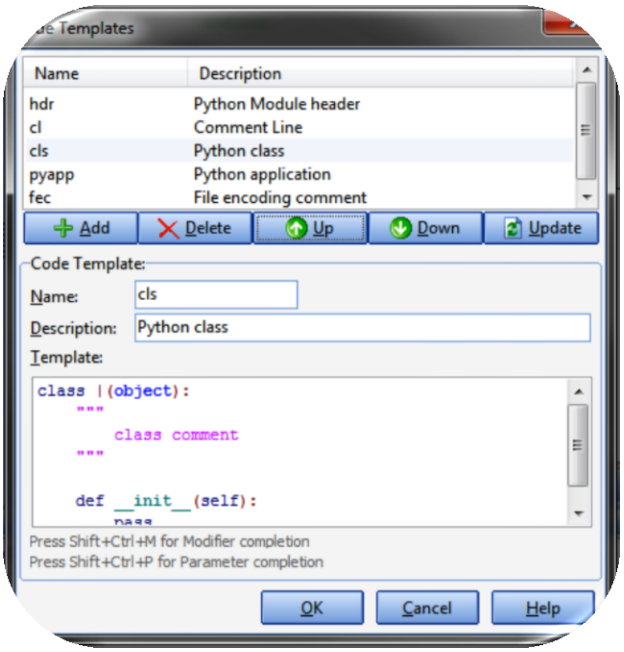
3.1 PREDLOGE

Predloga je del kode, ki si jo shranimo kot vzorec za večkratno uporabo. Ima statični del, ki je vedno enak, in dinamični del, ki ga z vsako uporabo predloge spremenimo (na primer ime metode, ime spremenljivke, vrednost spremenljivke). Vloga predlog je preprosta, pohitri nam pisanje kode, saj predlogo napišemo le enkrat, uporabljamo pa jo, kadar želimo. Okolja, ki podpirajo uporabo predlog, imajo enostavne predloge že napisane. Primer že napisane predloge vidimo na Slika 1, ki jo lahko izberemo, ko ustvarimo nov modul. V komentarju oziroma glavi modula takoj opazimo, da bo treba spremeniti vsaj ime modula, napisati, v kakšen namen se bo uporabljal in napisati pogoje (licenco), pod katerimi se sme uporabljati. Seveda spremenimo tudi vsebino metode.



Slika 1: Primer predloge v okolju PyScripter

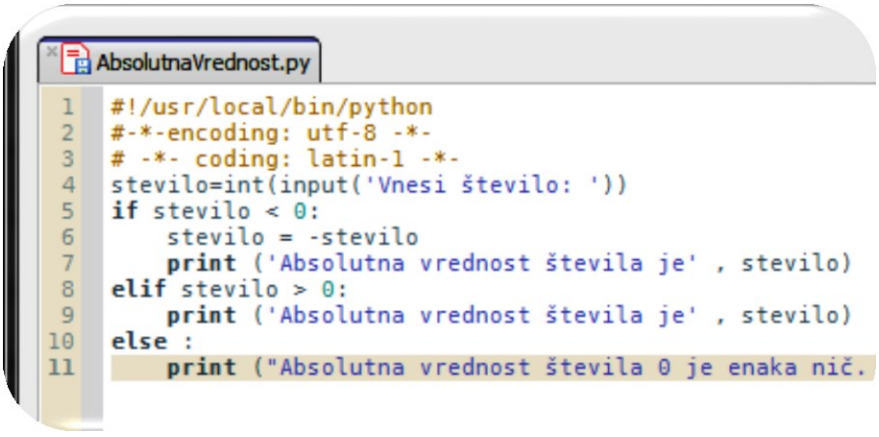
Poleg predlog, ki jih okolje ima, lahko ustvarimo svoje predloge. Na Slika 2 vidimo, katere predloge za pisanje modulov že imamo. Ravno tako vidimo, da si trenutno ogledujemo predlogo za pisanje razreda. Novo predlogo ustvarimo, če kliknemo na gumb »*Add*«, ki ga tudi vidimo na sliki. Ko to storimo, se polja »*Name*«, »*Description*« in »*Template*« izpraznijo in so tako pripravljena, da jih izpolnimo.



Slika 2: Okno za pregled in ustvarjanje predlog v okolju PyScripter

3.2 ŠTEVILČENJE VRSTIC

Številčenje vrstic je preprosto orodje, ki levo od kode izpiše zaporedno številko vrstice. To številčenje nam je v pomoč, ko smo v kodi naredili napako in nam sistem izpiše vrsto napake ter številko vrstice, kjer je bila ta napaka storjena.

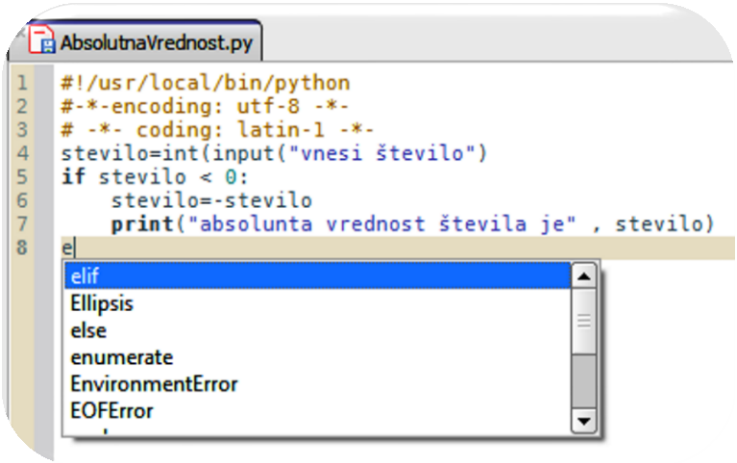


Slika 3: Številčenje vrstic v okolju IEP

Iz Slika 3 je razvidno, da ima okolje IEP orodje za številčenje vrstic. Tako vidimo, da se stavek »`if`« nahaja v peti, »`elif`« v osmi in »`else`« v deseti vrstici.

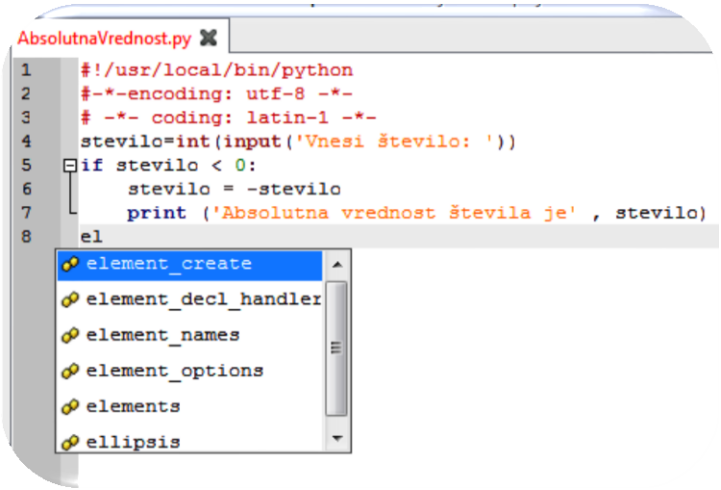
3.3 SAMODOPOLNJEVANJE KODE

Samodopolnjevanje kode je orodje, ki nam je v pomoč pri pisanju imen funkcij, spremenljivk, rezerviranih besed, imen naših funkcij in podobno. Od urejevalnika je odvisno, kako kakovostna je ta pomoč, oziroma kaj od naštetega nam ponuja.
V principu je delovanje tako, da ko vtipkamo del imena, nam urejevalnik ponudi seznam smiselnih imen za nadaljevanje vnosa.



Slika 4: Samodopolnjevanje kode v okolju IEP

Okolje IEP ima orodje za samodopolnjevanje kode, kar vidimo na Slika 4. Na isti sliki lahko opazimo, da nam ponudi seznam možnosti takoj, ko vtipkamo prvo črko. Po seznamu se pomikamo s pomočjo smernih tipk ali miške. Svoj izbor potrdimo tako, da pritisnemo tipko tab na tipkovnici ali s klikom miške.

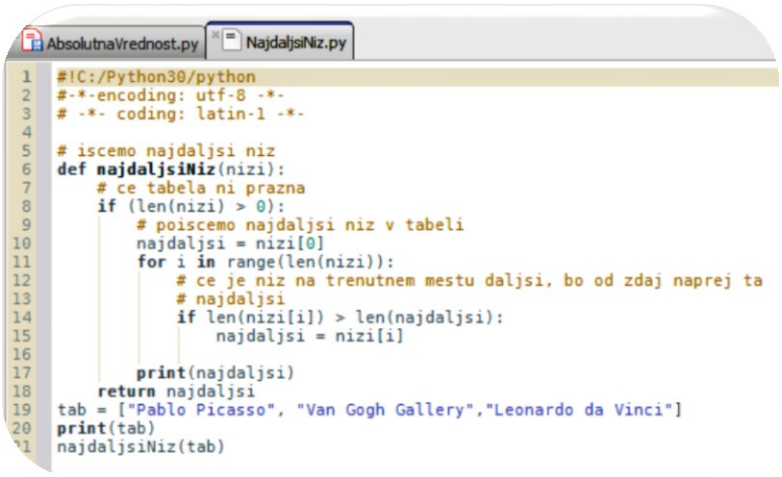


Slika 5: Samodopolnjevanje kode v okolju Geany

Na Slika 5 vidimo, da ima tudi okolje Geany orodje za samodopolnjevanje kode. Čeprav smo prvi črki dodali še drugo in s tem skrčili ponujeni seznam, besede »`elif`« na seznamu ni. To pomeni, da orodje za samodopolnjevanje kode v okolju Geany ne prikazuje rezerviranih besed.

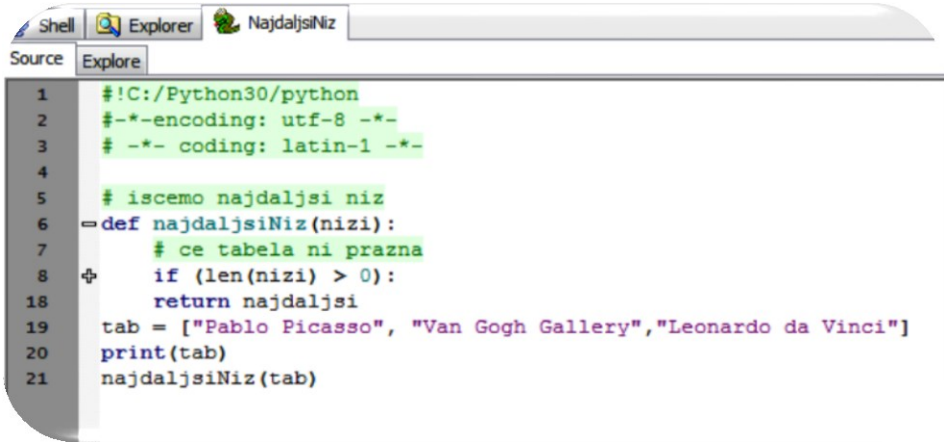
3.4 SAMODEJNO ZAMIKANJE KODE

Ena od prvih stvari, ki jih programer opazi, ko se uči programirati v Pythonu, je, da tu ne uporabljamo oklepajev ali posebnih rezerviranih besed, da označimo, kaj spada v posamezen blok. V Pythonu bloke določimo s tem, da vrstice zamikamo v desno. Število prostorov v zamiku je poljubno, pomembno je le, da je za vse stavke, ki sestavljajo blok, vedno enako.
V veliko pomoč pri zamikanju blokov nam je, če urejevalnik ponuja orodje za samodejno zamikanje kode. Vsakokrat, ko v urejevalniku vtipkamo dvopičje kot zaključek vrstice, se avtomatično pojavi zamik, običajno štirih znakov. Število znakov lahko v nekaterih urejevalnikih tudi spreminjamo. Ko želimo s pisanjem bloka zaključiti, skočimo v novo vrsto, pritisnemo tipko »back space« in se tako pomaknemo za nivo nazaj.
Orodje za samodejno zamikanje kode lahko, poleg zamikanja kode, samodejno označi tudi začetek bloka (lahko tudi začetek in konec bloka). Z označevanjem blokov pridobimo boljšo berljivost kode. Dodatna možnost orodja za samodejno zamikanje kode je, da lahko vsebino blokov tudi skrijemo. Skrivanje vsebine blokov nam je v pomoč predvsem pri daljših programih, saj s skrivanjem dobimo strnjen pogled že napisane kode. Posamezne dele kode si odpremo, če jo potrebujemo za nadaljnje pisanje.



Slika 6: Samodejno zamikanje kode v okolju IEP

Samodejno zamikanje kode v okolju IEP na Slika 6 je lepo vidno. Poleg samodejnega zamikanja kode se izriše tudi navpična črta, ki povezuje začetek in konec bloka.

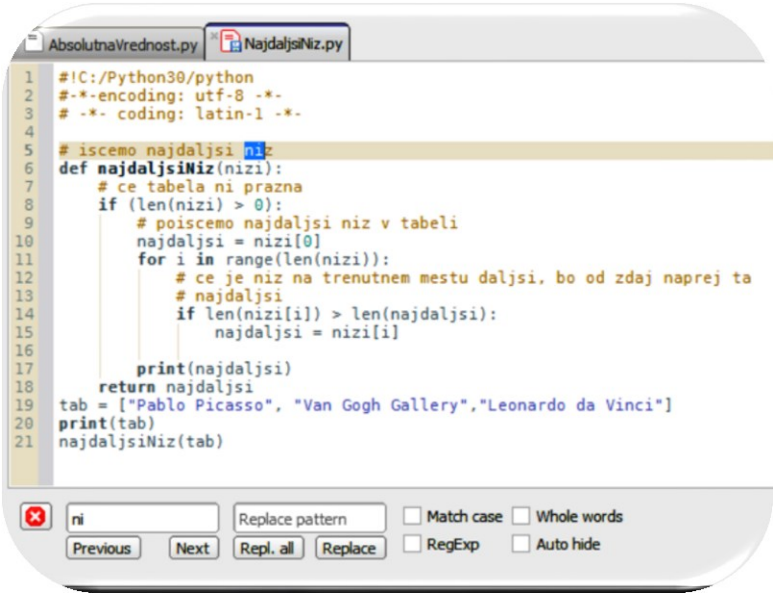


Slika 7: Samodejno zamikanje kode v okolju Boa constructor

Koda programa, ki jo vidimo na Slika 7, je enaka kodi programa, ki jo vidimo na Slika 6. Orodje za samodejno zamikanje kode v okolju Boa constructor namreč ponuja tudi skrivanje vsebine blokov. Tako je vsebina bloka, ki spada v stavek »`if`« trenutno skrita.
Označevanje blokov v okolju Boa constructor je drugačno od označevanja blokov v okolju IEP. Namesto označevanja začetka in konca bloka s pokončno črto, imamo tu levo od kode znak »−«, če je vsebina bloka vidna, in »+«, če je vsebina bloka skrita.

3.5 HITRO ISKANJE

Orodje za hitro iskanje je natanko to, kar pove njegovo ime. Pomaga nam pri iskanju imen spremenljivk, modulov, razredov, metod, projektov, paketov, rezerviranih besed. Kaj od naštetega lahko iščemo, na kakšen način in kaj z najdenimi besedami lahko počnemo, pa je odvisno od okolja, v katerem smo.

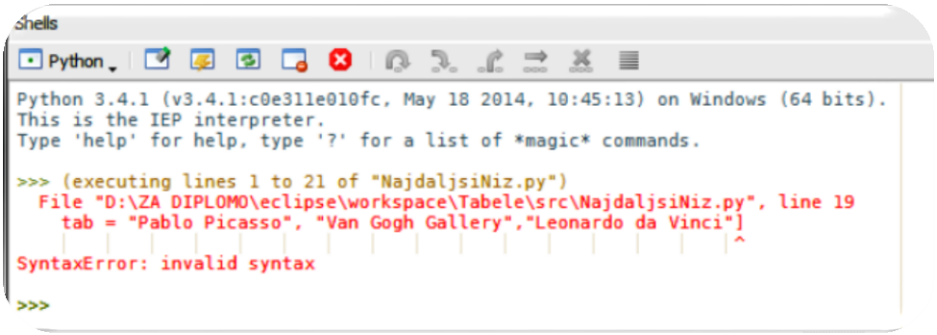


Slika 8: Orodje za hitro iskanje okolja IEP

Orodje za hitro iskanje okolja IEP vidimo na Slika 8, na spodnjem robu urejevalnika. Na sliki vidimo, da je beseda, ki jo iščemo, *»ni«* in da se ta beseda prvič pojavi v peti vrstici naše kode (beseda je označena z modro barvo). Med pojavitvami te besede prehajamo z gumboma *»Previous«* in *»Next«*. To besedo lahko zamenjamo z novo s pomočjo gumbov *»Repl.all«* in *»Replace«*. Razlika med njima je, da *»Replace«* zamenja besedo, ki je trenutno označena, *»Repl.all«* pa vse pojavitve te besede. Na voljo imamo štiri načine, katere pojavitve besede bomo videli. Tako označena možnost *»Match case«* prikazuje vse besede, ki se začnejo z ni. Možnost *»Whole words«* prikazuje samo iskano besedo (brez sestavljenk in izpeljank). Označen *»Auto hide«* nam avtomatično zapre iskalnik po desetih sekundah neaktivnosti. Zadnja nam ostane možnost *»RegExp«* (regularni izrazi), ki v primeru iskane besede ne da nobenih novih rezultatov. Če bi za iskano besedo napisali *»n.z«*, bi iskalnik poiskal vse besede s tremi črkami, ki se začnejo z *»n«* in končajo z *»z«*.

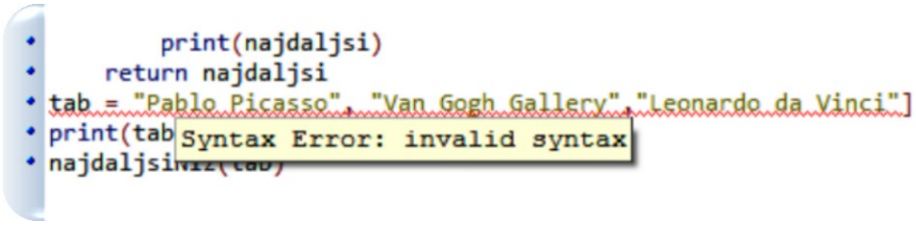
3.6 OZNAČEVANJE NAPAK

Vsak urejevalnik nas opozori na storjene napake. Urejevalniki s slabše razvitim orodjem za označevanje napak nas nanje opozorijo s pojavnim oknom ob zagonu programa in približnim mestom storjene napake. Tisti urejevalniki, ki imajo orodje za označevanje napak bolje razvito, nas na napake opozarjajo sproti. S kakšnim znakom je označena napaka in kakšno opozorilo ob tej napaki dobimo, je od urejevalnika do urejevalnika različno.



Slika 9: Označevanje napak v okolju IEP

V IEP pogrešamo označevanje napak medtem, ko pišemo kodo. Tako šele ob zagonu dobimo opozorilo, kot kaže Slika 9, o vrsti napake in kje napaka je.



Slika 10: Označevanje napak v okolju PyScripter

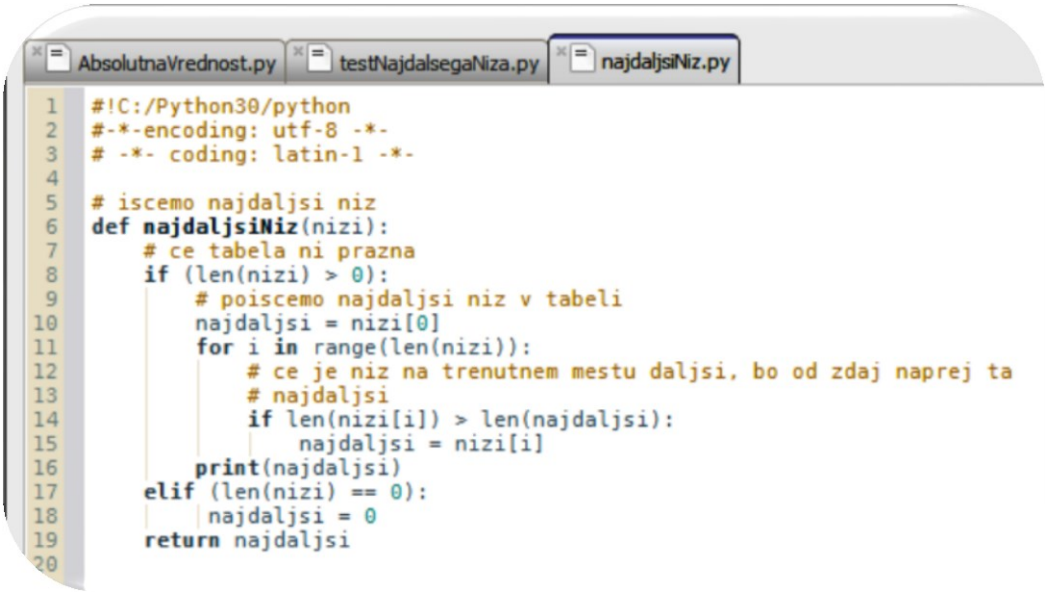
PyScripter nas med pisanjem kode takoj, ko smo napako storili, nanjo tudi opozori tako, da mesto napake vijugasto podčrta. Če se vijugasti črti približamo z miško, se pojavi okno, ki nam sporoči vrsto napake (kot prikazuje Slika 10).

3.7 TESTIRANJE PO SKLOPIH – UNITTEST

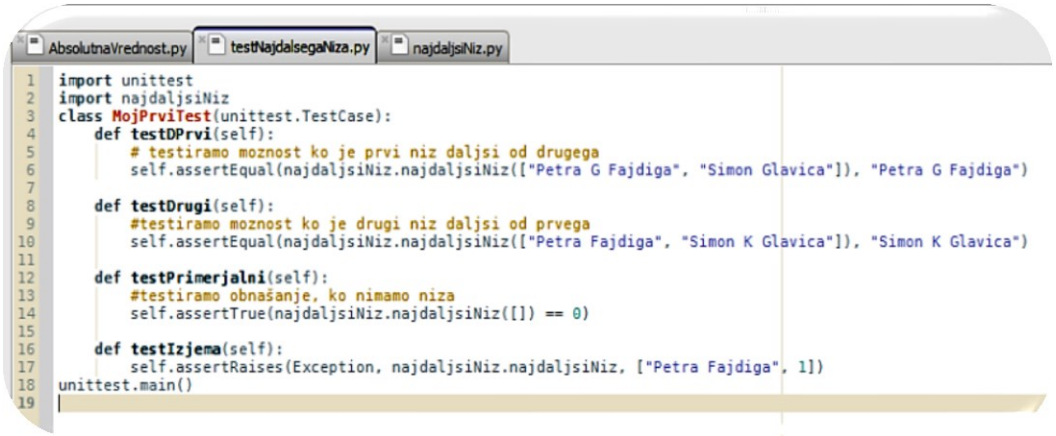
Unittest je orodje, ki nam pomaga preverjati delovanje napisane metode.
Preverjanje delovanja metode je pomembno, saj se tako prepričamo, da metoda vedno vrne pričakovani rezultat. Delovanje metode moramo preverjati vsakokrat, ko njeno kodo spremenimo. Za vnovično preverjanje moramo zopet uporabiti vse testne primere. Lahko se namreč zgodi, da na primerih, kjer je metoda pri prejšnjem testiranju delovala pravilno, zdaj vrača nepričakovane rezultate.
Ročno testiranje vsakega testnega primera posebej je zamudno in kaj hitro se zgodi, da na nekaj testnih primerov pozabimo. Da se to ne bi zgodilo, uporabimo unittest, ki omogoča, da vse testne primere napišemo v en program. Ta program zaženemo vedno, ko želimo testirati našo metodo.
Za pisanje testnega programa veljajo določena pravila, ki si jih bomo ogledali.
Uvoziti moramo modul unittest, ki ga imajo vsa okolja, ki podpirajo testiranje po sklopih. Uvozimo tudi modul, v katerem imamo napisano metodo, ki jo želimo testirati. Nato ustvarimo razred s poljubnim imenom, ki mora biti podrazred razreda *»unittest.TestCase«*. Znotraj razreda definiramo objektne metode, katerih ime se mora vedno začeti s test. Metoda mora imeti le parameter *»self«*. Vsaka metoda mora imeti enega od spodaj navedenih klicev metod iz modula unittest:

- *assertEquals (klicNašeMetode(), pričakovan rezultat)*, preverimo, ali je rezultat, ki ga metoda vrne, enak pričakovanemu rezultatu;
- *assertTrue (pogoj)*, preverimo, ali je določen pogoj izpolnjen;
- *assertRaises (Exception, imeNašeMetode, parametri)*, v primeru nepričakovanega vnosa sprožimo izjemo.

Ko napišemo vse potrebne metode za testiranje, moramo napisati še ukaz za klic *»unittest.main()«*, ki ga napišemo v istem nivoju kot razred.



Slika 11: Program, ki ga bomo testirali v okolju IEP



Slika 12: Testni program v okolju IEP

Slika 13: Rezultat testiranja z unittest v okolju IEP

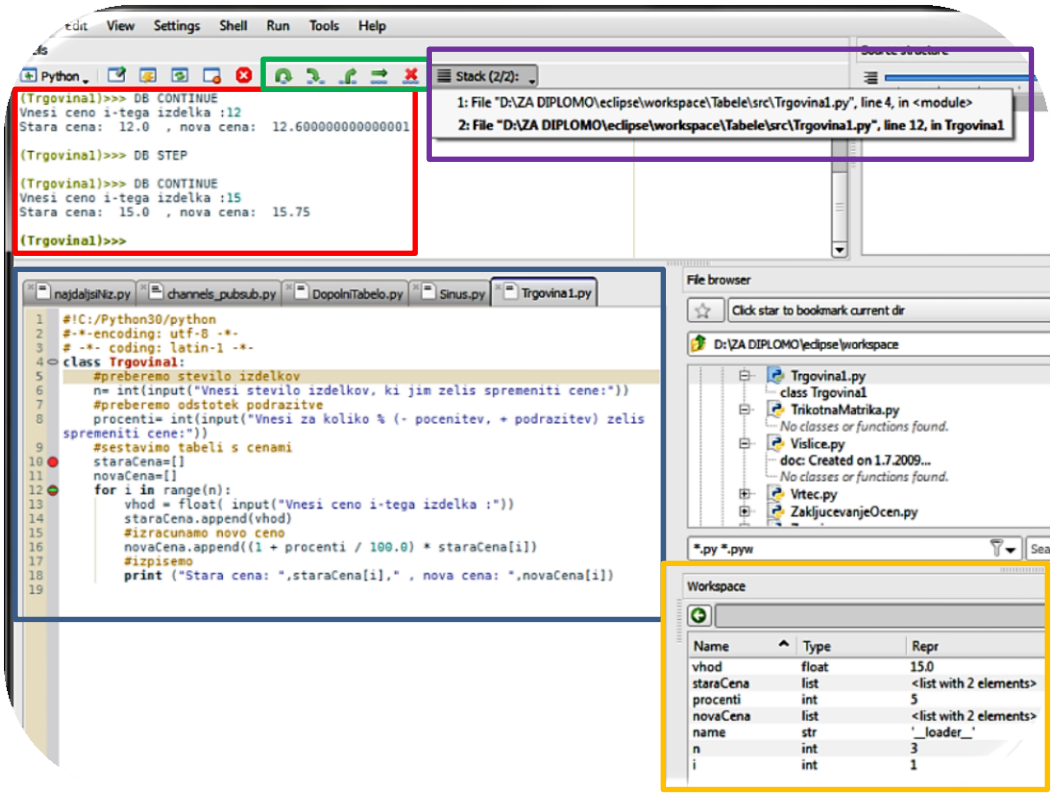
Program, ki ga bomo testirali, vidimo na Slika 11. Testni program prikazuje Slika 12. Tu vidimo, da imamo ustvarjen podrazred »`MojPrviTest`«, razreda »`unittest.TestCase`«, ter štiri objektne metode s parametrom »`self`«. Prvi dve metodi preverjata, ali je rezultat, ki ga metoda vrača, enak pričakovanemu. Tretja metoda preverja, ali je izpolnjen pogoj, četrta pa v primeru napačnega vnosa sproži izjemo. Na Slika 13 vidimo ukazno lupino, v kateri vidimo rezultat svojega preverjanja, koliko testov je bilo izvedenih in v kolikem času se je izvedel testni program. Vidimo tudi, da so vse testne metode delovale v skladu s pričakovanji in da se je proces testiranja zaključil brez napak.

3.8 PYTHON RAZHROŠČEVALNIK

Razhroščevalnik je orodje, ki ga uporabljamo za preizkušanje pravilnosti delovanja programa in odpravljanje napak v programu tako, da s sledenjem programa dobimo natančen vpogled v njegovo delovanje.

Programu lahko sledimo korakoma, vrstico za vrstico. Ker pa je to zamudno in včasih nepotrebno, saj smo o pravilnem delovanju nekaterih delov kode prepričani, so nam na voljo prekinitvene točke. S postavitvijo prekinitvene točke poskrbimo, da bo program ob zagonu izvedel vse vrstice kode do prekinitvene točke. Od tu naprej imamo različne možnosti nadaljevanja. Pri tem uporabljamo gumbe za kontrolo razhroščevanja (katere gumbe imamo na voljo, kako izgledajo in kako so poimenovani, je odvisno od okolja, v katerem razhroščevanje izvajamo).
Gumbi za kontrolo razhroščevanja (navedli bomo samo en primer poimenovanja vsakega gumba, funkcionalnost pa ostaja nespremenjena):

- »Go« zažene program oziroma nadaljuje izvajanje programa do prekinitvene točke ali do konca programa;
- »Step« izvede naslednji korak (stavek, vrstico). Če je mesto prekinitvene točke stavek,
  se izvede vrstica. Če je tu klic metode, razhroščevalnik vstopi v metodo in v prvi vrstici metode počaka, da nadaljujemo;
- »Over« izvede naslednji korak, tako kot »Step«, s to razliko, da ne vstopi v metodo, ampak jo izvede v celoti;
- »Step return« izvede vse korake v trenutni metodi in se vrne na mesto njenega klica;
- »Force step into« vsili posamezen korak;
- »Out« izstopi iz metode, v kateri je in počaka na nadaljnji ukaz;
- »Return to cursor« izvede vse korake do mesta, kjer se nahaja kurzor;
- »Quit« zaključi trenutno razhroščevanje;
- »View Breakpoints« je pogled na prekinitvene točke;
- »Mute Breakpoints« začasno izključi vse nastavljene prekinitvene točke.

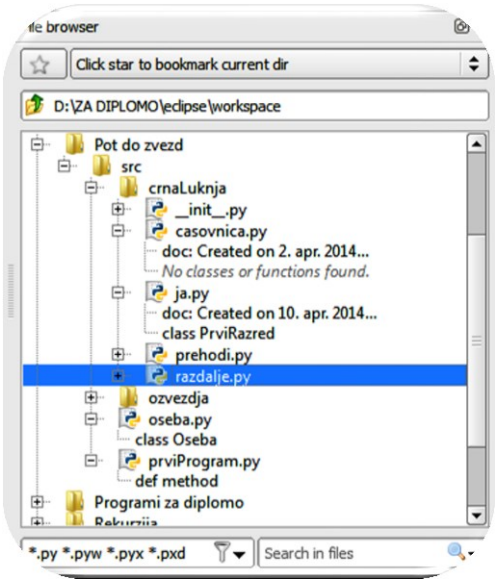


Slika 14: Razhroščevanje v okolju IEP

Na Slika 14 vidimo primer razhroščevanja v okolju IEP. Rdeč okvir označuje ukazno lupino, kjer vnašamo vhodne podatke, vidimo podatke, ki se izpisujejo, tu vidimo tudi opozorila o napakah. V orodni vrstici, v zelenem okvirju, so gumbi za kontrolo razhroščevanja. Prav tako v orodni vrstici, v vijoličnem okvirju, imamo stanje sklada (*»stack«)*. Sklad je del pomnilnika, ki deluje po sistemu *»LIFO«* (Last in First out). Običajno vsak nadaljnji vhod zabeleži, ko z razhroščevanjem vstopimo v eno izmed struktur in se s tem pomaknemo za en nivo globlje. Ko na določenem nivoju končamo z razhroščevanjem, se prav zaradi sklada, ki beleži zgodovino, lahko pomaknemo za nivo višje (ne na začetek) in tam nadaljujemo z razhroščevanjem, sklad pa nivo, iz katerega smo izstopili, zbriše iz pomnilnika. Modul, ki ga razhroščujemo, vidimo v modrem okviru. Tu lahko opazimo, levo od kode, rdečo piko, ki označuje prekinitveno točko. Nekoliko nižje vidimo rdečo piko, ki vsebuje zelen minus. To je ravno tako prekinitvena točka, zelen minus pa sporoča, da pregledujemo kodo od tu naprej. Rumen okvir označuje okno *»workspace«*, v katerem so navedena imena spremenljivk iz modula, ki ga razhroščujemo. Prav tako vidimo tip posamezne spremenljivke in njeno trenutno zavzeto vrednost.

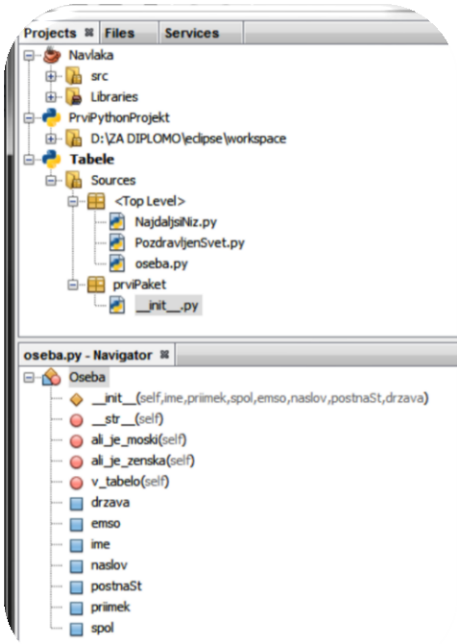
3.9 PROJEKTNO DREVO

Projektno drevo je abecedno (po datumu nastanka, po pomembnosti) urejen seznam projektov, vključno z njihovo vsebino (kaj je projekt in kaj njegova vsebina, poglej v razdelek 2).



Slika 15: Pregledovalnik datotek okolja IEP

Okolje IEP pravzaprav nima projektnega drevesa. Ima le pregledovalnik datotek, v katerem vidimo mape z imeni projektov, paketov, vidimo module in njihove razrede ter metode.



Slika 16: Projektno drevo okolja NetBeans

Projektno drevo okolja NetBeans je razdeljeno na dva dela, kot vidimo na sliki Slika 16. Zgornji del *»Projects«* prikazuje projekte, ki smo jih ustvarili, njihove pakete in podpakete ter module. Spodnji del *»Navigator«*, kjer vidimo strukturo posameznega modula (razrede, metode, spremenljivke).

4 RAZVOJNA OKOLJA

Pogledali bomo nekaj razvojnih okolij, preverili, kje jih dobimo, za kateri operacijski sistem so namenjena, ali so plačljiva ali brezplačna ter kakšna razvojna orodja ponujajo. Še več razvojnih okolij in njihov kratki opis dobimo na uradni strani programskega jezika Python, in sicer na naslovu https://wiki.python.org/moin/PythonEditors.

	BOA	ECLIPSE	EDITRA	GEANY	IDLE
ŠTEVILČENJE VRST	✓	✓	✓	✓	×
PREDLOGE	✓	✓	×	×	×
SAMODEJNO ZAMIKANJE KODE	✓	✓	✓	✓	✓
SAMODOPOLNJEVANJE KODE	✓	✓	✓	✓	✓
HITRO ISKANJE	✓	✓	✓	✓	×
OZNAČEVANJE NAPAK	✓	✓	×	×	✓
RAZHROŠČEVANJE	✓	✓	×	×	✓
PROJEKTNO DREVO	✓	✓	✓	✓	×
TESTIRANJE PO SKLOPIH	×	✓	×	×	×
PLAČLJIVO OKOLJE	×	×	×	×	×

Tabela 1: Primerjava okolij glede na njihova orodja (1. del)

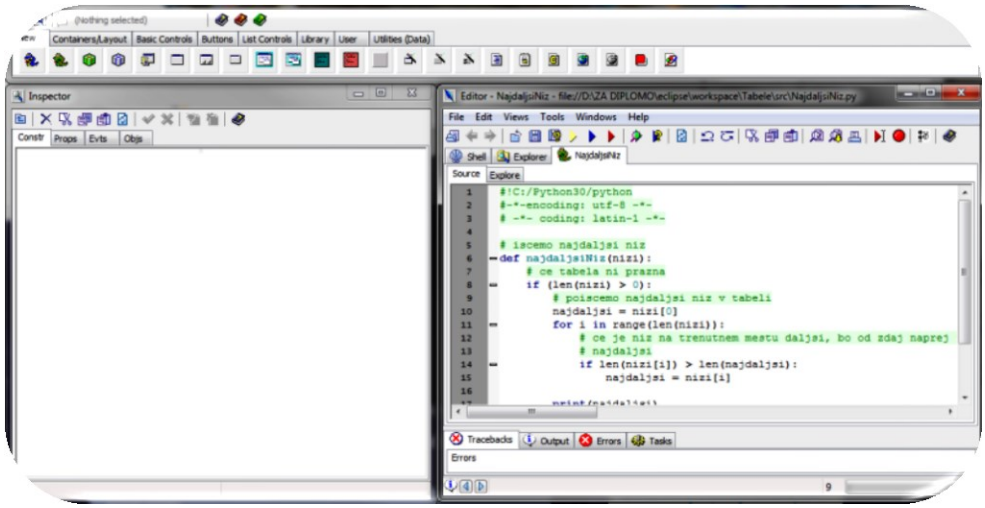
	NETBEANS	PYCHARM	PYSCRIPTER	SPYDER	IEP
ŠTEVILČENJE VRST	✓	✓	✓	✓	✓
PREDLOGE	✓	✓	✓	×	×
SAMODEJNO ZAMIKANJE KODE	✓	✓	✓	✓	✓
SAMODOPOL-NJEVANJE KODE	✓	✓	✓	✓	✓
HITRO ISKANJE	✓	✓	✓	✓	✓
OZNAČEVANJE NAPAK	✓	✓	✓	✓	×
RAZHROŠČEVANJE	✓	✓	✓	✓	✓
PROJEKTNO DREVO	✓	✓	✓	✓	×
TESTIRANJE PO SKLOPIH	✓	✓	✓	×	✓
PLAČLJIVO OKOLJE	✓	✓	×	×	×

Tabela 2: Primerjava okolij glede na njihova orodja (2. del)

Zgornji tabeli prikazujeta vsa okolja, ki jih v diplomi opazujemo, in prav tako opazovana orodja. Na ta način smo prikazali, katera orodja ima posamezno okolje. Dobili smo tudi prvo potrditev, da se okolja razlikujejo, pa čeprav le v naboru orodij.

BOA CONSTRUCTOR

V okolje Boa Constructor sta združena Python IDE in wxPython GUI graditelj (grafični uporabniški vmesnik). Programiranje z wxPython gui graditeljem je vizualno, kar pomeni, da kliknemo na ikono in se koda napiše sama (na Slika 17, v zgornjem robu, vidimo ikone za vizualno programiranje). Dobimo ga na uradni spletni strani http://boa-constructor.sourceforge.net/. Podpira operacijska sistema Windows in Mac OS X.
Boa Constructor se razlikuje od ostalih okolij predvsem zaradi vizualnega programiranja (kar smo omenili že zgoraj). Orodja, ki jih opazujemo, pa se ne razlikujejo od orodij v ostalih okoljih. V razdelku 3.4 smo si že ogledali orodje za samodejno zamikanje kode.



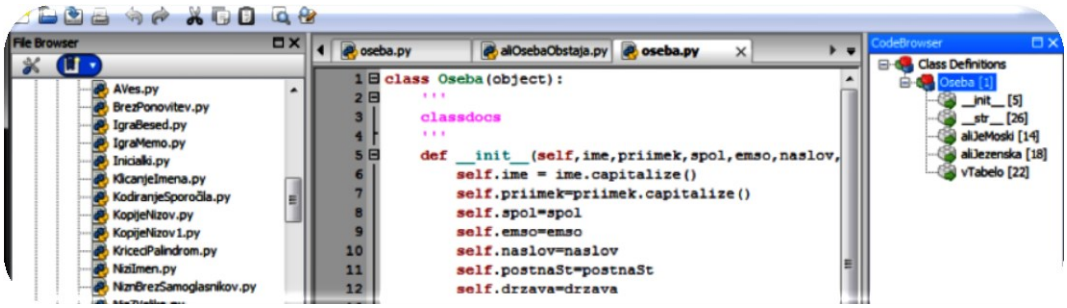
Slika 17: Boa Constructor

ECLIPSE

Eclipse je razvojno okolje za več programskih jezikov. Prenesemo ga z uradne spletne strani http://www.eclipse.org/ (tu naj opozorim na to, da se je med nastajanjem diplomske naloge stran spremenila in da je na voljo novejša različica Eclipse, zato obstaja možnost, da se nekateri opisi ne bodo ujemali z zadnjo različico). Za programiranje v Pythonu moramo prenesti še PyDev vtičnik z uradne strani http://pydev.org/. To okolje lahko uporabljamo v operacijskih sistemih Windows, Linux in Mac OS X. Orodja bomo natančno spoznali v razdelku 6.4.

EDITRA

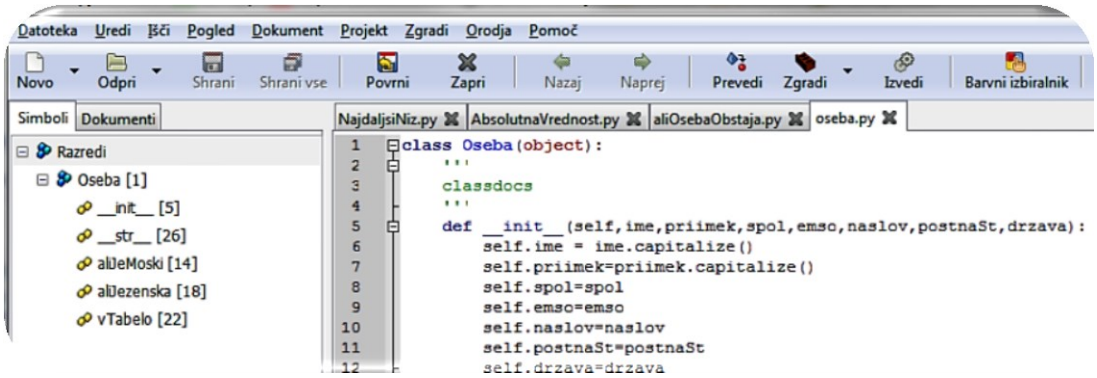
Editra je, ravno tako kot Eclipse, razvojno okolje za več programskih jezikov (podpira preko 60 programskih jezikov). Prenesemo ga z uradne spletne strani http://editra.org/. Deluje v operacijskih sistemih Windows, Linux in Mac OS X. Je preprosto razvojno okolje in enostavno za uporabo, vendar pogrešamo nekatera osnovna orodja, kot so označevanje napak, razhroščevalnik, unittest, predloge. Zanimivo je orodje projektno drevo, ki je razdeljeno na dva dela. Prvi del je pregledovalnik datotek, v katerem imamo pogled na projekte in module (levo od modulov na Slika 18). Z dvoklikom na modul pa se le-ta odpre. Drugi del projektnega drevesa je pregledovalnik kode (desno od modulov na Slika 18), ki prikazuje razred, metode in spremenljivke za modul, ki ga opazujemo. Za vsako od struktur poleg nje napiše, v kateri vrstici je bila deklarirana.



Slika 18: Projektno drevo

GEANY

Tudi Geany je razvojno okolje za več programskih jezikov. Prenesemo ga z uradne spletne strani http://www.geany.org/. Namenjen je operacijskim sistemom Windows, Linux in Mac OS X. Geany je po svojem videzu in enostavnosti za uporabo zelo podoben okolju Editra. Prav tako tu pogrešamo osnovna orodja za odkrivanje napak in predloge. Orodje za samodopolnjevanje kode smo si ogledali v razdelku 3.3. Njegova posebnost pa je, da je preveden v slovenski jezik, kot vidimo na Slika 19.



Slika 19: Okolje Geany

IDLE

IDLE je razvojno okolje za programski jezik Python, prenesemo ga skupaj s programskim paketom Python z uradne spletne strani https://www.python.org/. Uporabljamo ga lahko v operacijskih sistemih Windows, Linux in Mac OS X. Orodja, ki jih ima IDLE, bomo spoznali v razdelku 5.2.

NETBEANS

NetBeans ponuja hiter in enostaven razvoj namiznih, mobilnih in spletnih aplikacij. Prenesemo ga z uradne spletne strani http://netbeans.org/. Namestimo ga lahko na operacijskih sistemih Windows, Linux in Mac OS X. Osnovna orodja po videzu in uporabnosti spominjajo na orodja okolja PyCharm. Nekaj razlike je le v projektnem drevesu, ki smo ga že opisali v razdelku 3.9.

.

PYCHARM

Tudi PyCharm je razvojno okolje za več programskih jezikov, prenesemo ga z uradne spletne strani http://www.jetbrains.com/pycharm/. Izbiramo lahko med plačljivo (Professional) in brezplačno (Community) različico. Operacijski sistemi, ki jih Pycharm podpira, so Windows, Linux in Mac OS X. V razdelku 7.3 bomo opisali Pycharmova orodja.

PYSCRIPTER

PyScripter je razvojno okolje, ki je zelo priljubljeno med študenti FRI (Fakulteta za računalništvo in informatiko). Dobimo ga na naslovu https://code.google.com/p/pyscripter/. Windows je edini operacijski sistem, v katerem Pyscripter deluje. Osnovno orodje za označevanje napak smo si ogledali v razdelku 3.6, predloge pa v razdelku 3.1.

SPYDER

Spyder je razvojno okolje, ki ponuja številne pakete, namenjene pisanju programov, ki so prilagojeni potrebam različnih znanstvenih ved. Namestimo ga z uradne spletne strani https://code.google.com/p/spyderlib/. Je namenjen operacijskim sistemom Windows, Linux in Mac OS X.

IEP

IEP je preprosto, a učinkovito Python IDE okolje. Ime IEP je pravzaprav kratica za Interactive Editor for Python. Prenesemo ga lahko za operacijske sisteme Windows, Linux in Mac OS X z uradne spletne strani http://www.iep-project.org. Njegova orodja smo si že ogledali v razdelku 3.

5 IDLE

Kot smo omenili v razdelku 4, je IDLE okolje, ki ga dobimo, če namestimo programski paket Python z uradne strani http://www.python.org/.
Zaradi tega to okolje sreča in vsaj nekaj časa uporablja večina programerjev v programskem jeziku Python. Oglejmo si ga nekoliko podrobneje.

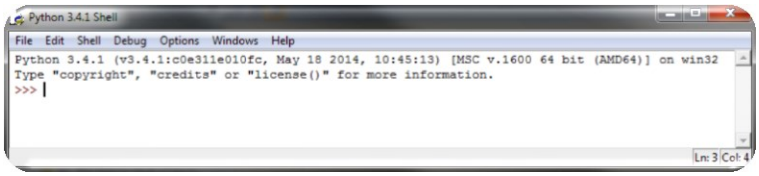
Na danem naslovu poiščemo najnovejšo različico in jo shranimo nekam na disk. Paziti moramo, da prenesemo različico, ki je namenjena našemu operacijskemu sistemu. Če imamo naložen operacijski sistem Windows, lahko prenesemo »`Python 2.x.x`« ali »`Python 3.x.x`«, kot kaže Slika 20 (x.x predstavlja različico, ki jo bomo naložili).

Slika 20: Prenos Python paketa za Windows

5.1 PRVI PROGRAM V IDLE

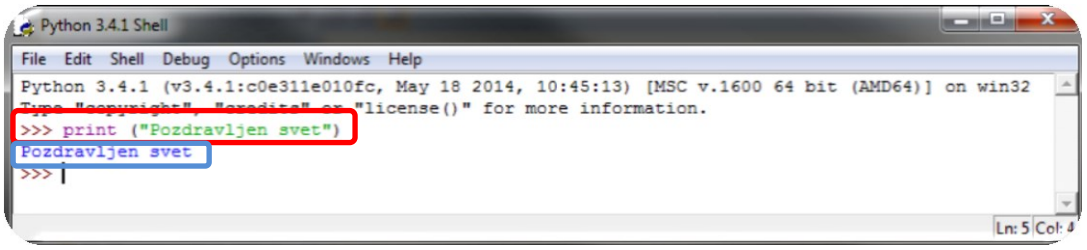
Ko poženemo IDLE (kot kaže Slika 21), se nam odpre »`Python Shell`« (Slika 22). To je nekakšno ukazno okolje, ki je primerno za računanje (kot računalo) in za pisanje krajših programčkov. Hkrati pa je to tudi okno, kjer so vidni rezultati, ko poženemo program. V nadaljevanju ga bomo imenovali ukazna lupina.

Slika 21: Pot do IDE



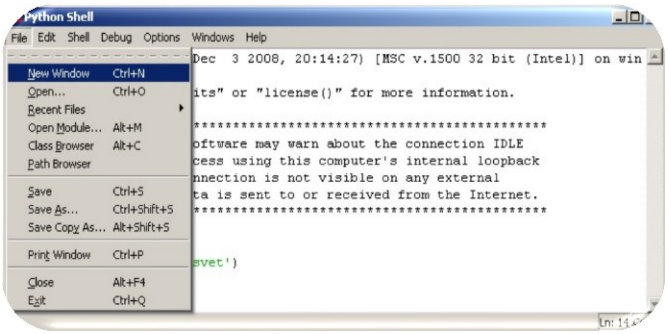
Slika 22: Ukazna lupina

Poizkusimo izpisati niz *»Pozdravljen svet«*. Na Slika 23 je z rdečo barvo uokvirjen ukaz, ki ga napišemo. Z modrim pa, če smo Python pravilno namestili, tisto, kar nam program izpiše.



Slika 23: Pozdravljen svet

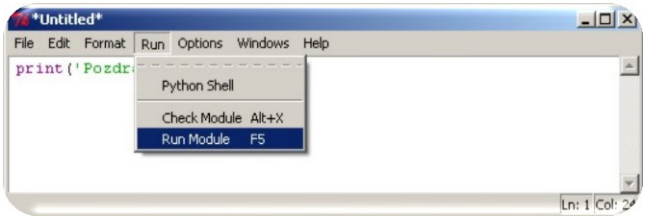
Kot kmalu ugotovimo, v ukazni lupini pišemo kodo vrstico za vrstico. Prejšnje vrstice ne moremo popravljati na njenem prvotnem mestu. Imamo pa možnost, da njeno vsebino prenesemo kot nov ukaz, ki ga lahko še uredimo. To storimo tako, da uporabimo kombinacijo tipk *»Alt + p«* (v obratno smer med vnesenimi ukazi pa se premikamo z *»Alt + n«*). Druga možnost, kako prikličemo vrstico, je, da kliknemo nanjo in pritisnemo *»Enter«*. Poleg tega napisane kode ne moremo shraniti v obliki programa, zato odpremo urejevalnik, kot je prikazano na Slika 24.



Slika 24: Kako odpremo urejevalnik

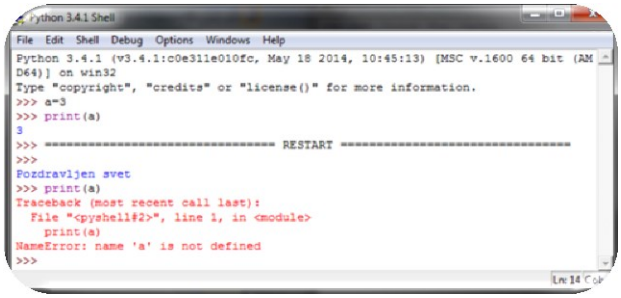
Tu lahko pišemo kodo, jo popravljamo, dopolnjujemo in shranimo. Shranjeno kodo lahko nato ponovno zaženemo kasneje, lahko pa jo uporabimo tudi kot del neke druge kode, kot modul.

Pokažimo še, kako program zaženemo. Ko napišemo kodo, jo moramo shraniti na disk. Da bi program zagnali, moramo v opravilni vrstici kliknit na »Run« in tam izbrati možnost »Run Module« (Slika 25). Druga možnost za zagon programa pa je, da pritisnemo tipko »F5« na tipkovnici. Če kode nismo shranili, se na zaslonu pojavi okno, ki nas opozori, da moramo to storiti (Slika 26). Če smo kodo pravilno napisali, se izvede in rezultat vidimo v ukazni lupini. Pred vsako izvedbo programa v urejevalniku se ukazna lupina ponovno zažene. To povzroči, da se vse, kar smo do tedaj definirali v ukazni lupini, izbriše.
Poglejmo si primer, s katerim pokažemo, kako to gre.
V ukazni lupini deklariramo spremenljivko `'a'`, ki ji dodelimo vrednost. Izpišemo jo na zaslon. Nato v urejevalniku zaženemo (Slika 25) napisani program (npr. Pozdravljen svet, Slika 28). Ukazna lupina se ponovno zažene in izpiše rezultat programa, ki smo ga zagnali (Slika 29). Zdaj želimo ponovno na zaslon izpisati spremenljivko `'a'`. Ker pa se je ukazna lupina medtem ponovno zagnala, spremenljivka `'a'` ni več deklarirana. Spremenljivke `'a'` ni več, zato je prevajalnik ne more najti in javi napako, da spremenljivka `'a'` ne obstaja (sporočilo o napaki Slika 27).



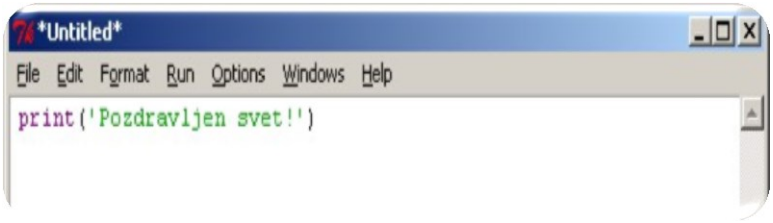
Slika 25: Run Module

Slika 26: Opozorilo

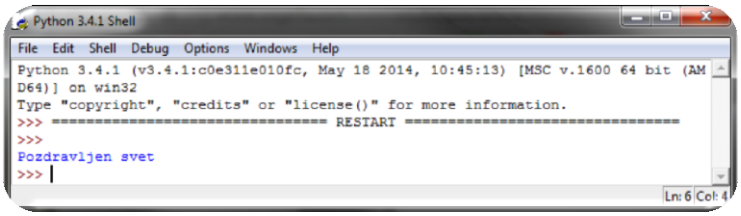


Slika 27: Napaka

Poglejmo si, kako s pomočjo urejevalnika napišemo program, ki izpiše niz (Slika 28) in kako je viden rezultat v ukazni lupini (Slika 29).



Slika 28: Koda programa



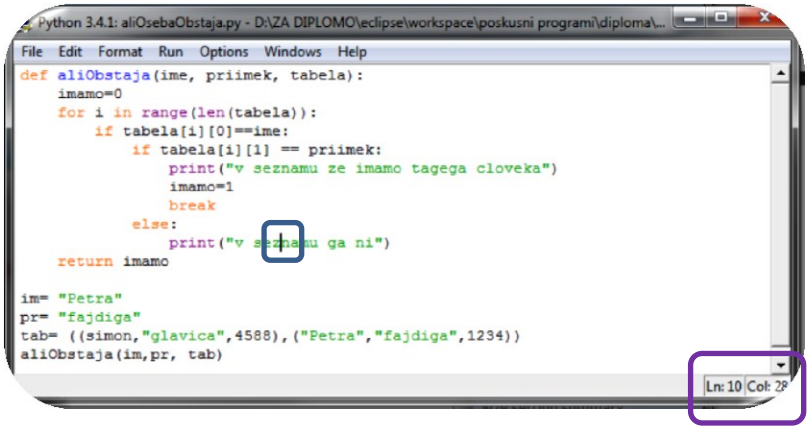
Slika 29: Rezultat programa

5.2 RAZVOJNA ORODJA V IDLE

IDLE se ne more ravno pohvaliti z veliko razvojnimi orodji, a si vseeno poglejmo, kaj nam ponuja.

5.2.1 Številčenje vrstic

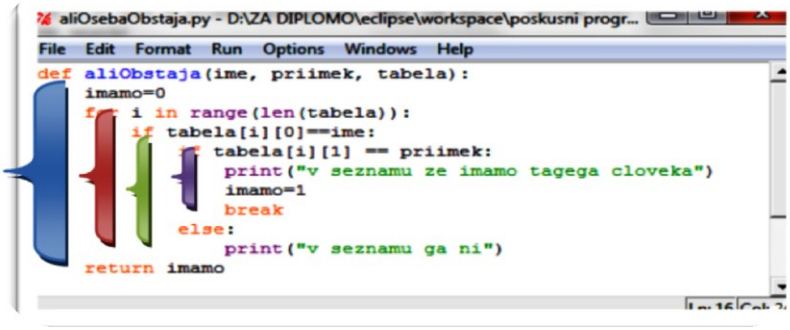
Številčenje vrstic v IDLE ni tako, kot bi si ga želeli. Namesto da bi imeli številčenje vrstic levo ob kodi, kot smo navajeni iz drugih okolij, imamo tu le v desnem spodnjem kotu okna podatka o vrsti in znaku, kjer se nahaja urejevalni kazalec (kurzor). Tako na sliki vidimo, da se s kurzorjem nahajamo v kodi (modri okvir na Slika 30), v desnem kotu (vijolični okvir na Slika 30) pa lahko razberemo, da se nahajamo pri osemindvajsetem znaku v deseti vrsti.



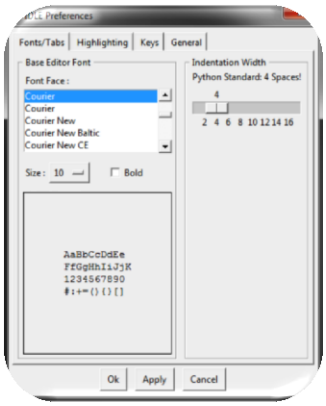
Slika 30: Mesto nahajanja

5.2.2 Samodejno zamikanje kode

V IDLE se bloki zamikajo tako, kot smo opisali v razdelku 2. Z oklepaji različnih barv so na Slika 31 označeni bloki.
Če nam zamik za 4 znake ne ustreza, ga lahko v IDLE *»Preferences«* spremenimo.



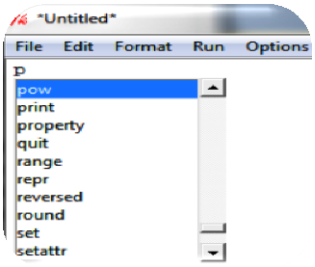
Slika 31: Bloki IDLE



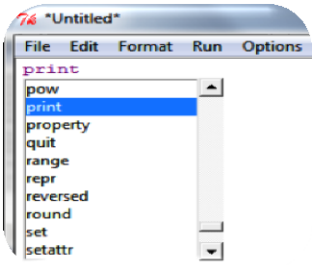
Slika 32: IDLE nastavitve

5.2.3 Samodopolnjevanje kode

IDLE ima vgrajeno pomoč pri pisanju imen funkcij in spremenljivk. Ko vtipkamo del imena in pritisnemo tipko *»tab«*, nam IDLE ponudi seznam smiselnih imen za nadaljevanje vnosa (Slika 33). Po seznamu se premikamo s smernima tipkama gor in dol. Pomik v katero koli stran (gor ali dol) povzroči, da se del naše kode dopolni, oziroma spremeni glede na trenutni izbor v seznamu (Slika 34).
Če pa obstaja le eno smiselno ime, ob pritisku na tipko tab IDLE kar sam opravi vnos. Na primer, če napišemo *»pri«* in pritisnemo tipko *»tab«*, bo IDLE sam vnesel ukaz *»print«*.



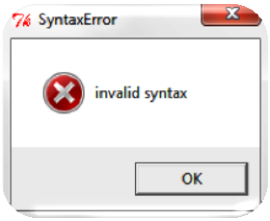
Slika 33: Spustni seznam



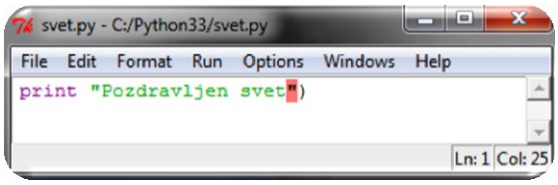
Slika 34: Izbor iskane besede

5.2.4 Označevanje napak

Kot vemo, so v kodi lahko sintaktične ali semantične napake (za razlago o napakah glej razdelek 2). Če smo se zmotili v sintaksi, se takrat, ko želimo zagnati program, prikaže pojavno okno, ki nam to tudi sporoči (*»invalid syntax«*). Ko kliknemo na *»OK«*, se vrnemo v urejevalnik. Tu opazimo, da se je mesto, kjer se je program ustavil, obarvalo rdeče, tudi kurzor se je prestavil na to mesto (Slika 36). To nam pove, da je napaka v tej vrstici, ali pa v že prejšnjih vrsticah.

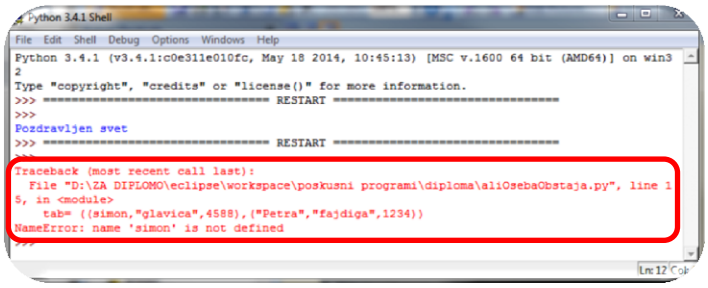


Slika 35: Sporočilo o sintaktični napaki



Slika 36: Prikaz mesta napake

Semantične napake, ki med izvajanjem kode povzročijo napako, zaradi katere se program ne more več izvajati naprej, se prikažejo v ukazni lupini kot »*error*« rdeče barve (Slika 37, rdeči okvir). V tem primeru dobimo podatek, v kateri vrstici je prišlo do napake in kakšne vrste napaka je.

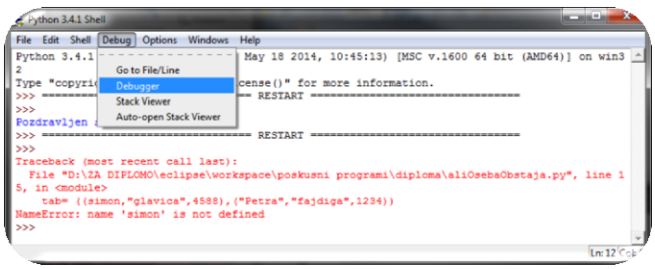


Slika 37: Napaka

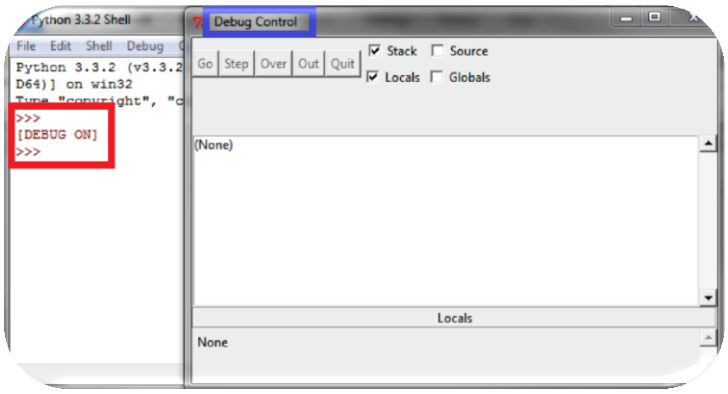
Opis dogajanja smo namerno poenostavili, saj Python omogoča programsko lovljenje teh napak. A opis tega sega izven okvira te diplomske naloge.

5.2.5 Python razhroščevalnik

Kaj razhroščevalnik je, smo povedali v razdelku 3.8, zato bomo tu opisali le, kako ga uporabljamo v IDLE.
Da bi delali z razhroščevalnikom, ga moramo aktivirati. To storimo tako, da v ukazni lupini v menujski vrstici kliknemo na »*Debug*«, v pojavnem oknu pa Debugger (kot kaže Slika 38). Odpre se nam novo okno »*Debug Control*« (Slika 39). V ukazni lupini dobimo sporočilo o vključenem razhroščevalniku »*[DEBUG ON]*« (Slika 39, rdeči okvir).

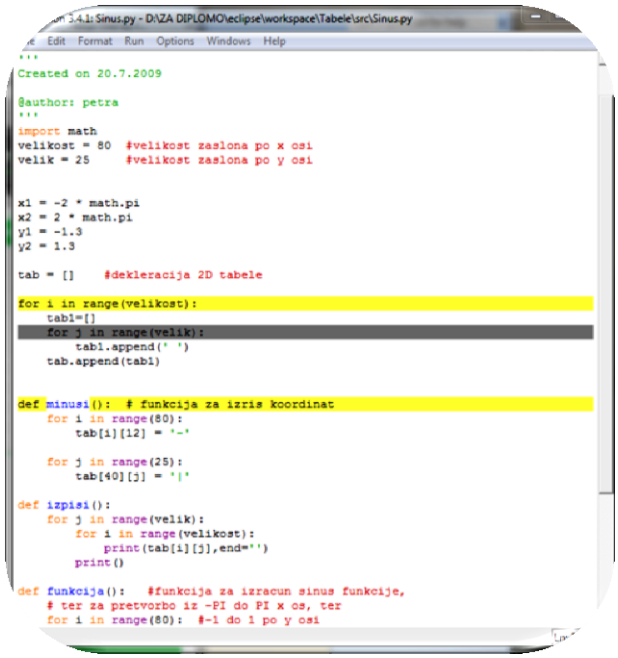


Slika 38: Odprtje razhroščevalnika



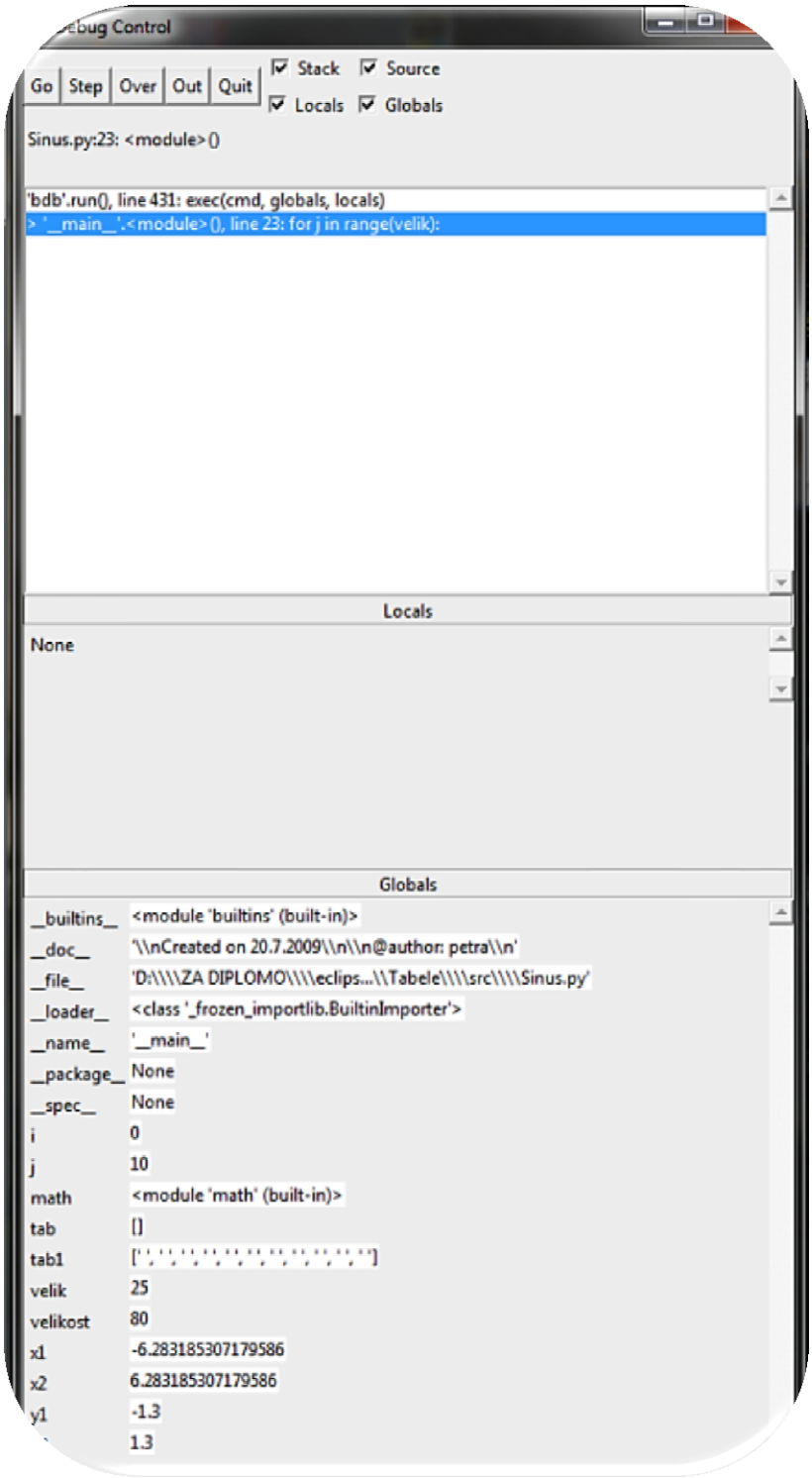
Slika 39: Okno za razhroščevanje

Nato odpremo modul, ki ga želimo pregledati: »*Ukazna lupina* → *File* → *open*→ *našModul.py*«. Na mestu, kjer želimo natančno pregledovati delovanje kode, z desnim gumbom miške kliknemo in v pojavnem oknu izberemo »*Set Breakpoint*«. Tako postavimo prekinitveno točko in vrstica se obarva rumeno. Prekinitvenih točk lahko nastavimo, tudi več.
Po končanem nastavljanju prekinitvenih točk zaženemo naš modul (z »*F5*« oziroma z »*Run module*«) in preklopimo na okno »*Debug Control*«. Zdaj lahko začnemo s pregledovanjem kode s pomočjo gumbov za kontrolo razhroščevanja (glej razlago razdelka 3.8). Če želimo takoj preiti do prekinitvene točke, pritisnemo na »*Go*«. Ko se program ustavi ob prekinitveni točki, nadaljujemo izvajanje kode, tako da uporabljamo gumbe za kontrolo razhroščevanja. Gumbi, ki jih IDLE ima, so: »*Go*«, »*Step*«, »*Over*«, »*Out*« in »*Quit*«.
V modulu se sivo obarva vrstica, v kateri se trenutno nahajamo, vrstice s prekinitvenimi točkami, kot smo že povedali, se obarvajo rumeno (kot vidimo na Slika 40).



Slika 40: Program, ki ga bomo razhroščevali

V oknu za kontrolo razhroščevanja (na Slika 41) smo si vklopili prikazovanje »*Stack*« (kaj je stack, smo spoznali v razdelku 3.8), »*Locals*«, »*Globals*« (locals in globals prikazujeta trenutne vrednosti lokalnih in globalnih spremenljivk) in »*Source*« (ko vklopimo source, se v ospredje – če ga imamo recimo postavljenega za oknom control – prikaže modul, ki ga razhroščujemo).



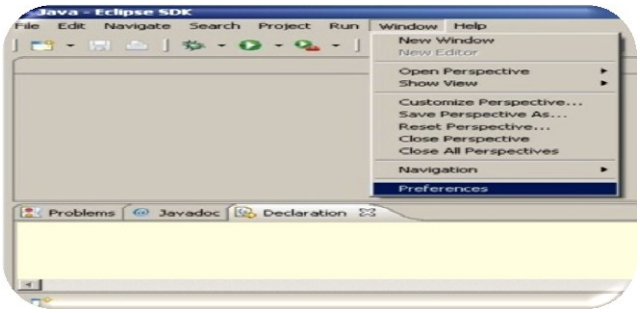
Slika 41: Kontrolno okno za razhroščevanje

6 ECLIPSE

V razdelku 4 smo spoznali, kje in kako dobimo Eclipse.
Eclipse je odprtokoden IDE (kaj pomeni odprtokodnost, je opisano v razdelku 2, IDE smo opisali v razdelku 2). Je tako imenovano »orodje vse v enem«, ki omogoča pisanje, urejanje, prevajanje, tolmačenje in poganjanje programov. To lahko počnemo v različnih programskih jezikih, kot so Java, Python, C, C++. Prvotno pa je bil Eclipse namenjen le programskemu jeziku Java.
Eclipse je precej priljubljen med programerji, saj nudi, kot smo povedali, programiranje v različnih programskih jezikih. Njegova priljubljenost raste tudi zaradi Eclipse tehnološkega projekta. Pri projektu si prizadevajo, da bi Eclipse spoznala širša populacija, spodbuja raziskovanje in izobraževanje množic. Deluje pa tudi kot dom za zamisli, ki so na poti, da postanejo pomemben podprojekt Eclipse.

6.1 PRIPRAVA ECLIPSE ZA PYTHON

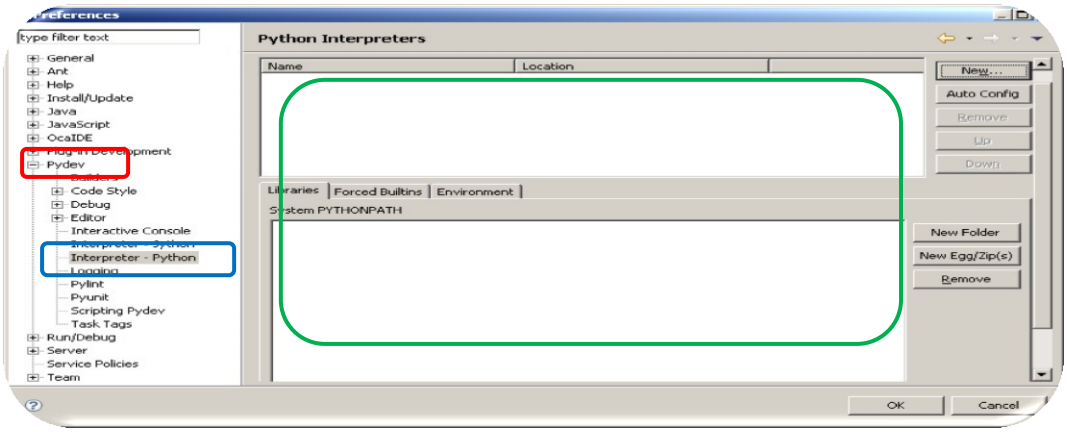
Okolje Eclipse privzeto ne podpira programiranja v Pythonu. Ker pa je to okolje zelo razširjeno, so se pojavili dodatki, ki Eclipse nadgradijo, tako da je mogoče v tem okolju uporabljati tudi Python. Na ta način programerji lahko uporabljajo okolje, ki ga poznajo od prej pri pisanju programov v jeziku Java, tudi za programiranje v Pythonu.
Da bi v Eclipse delali s Pythonom, moramo prej na svoj računalnik namestiti še odprtokodni paket PyDev. Dobimo ga na naslovu http://pydev.org.
Na svoj disk prenesemo stisnjeno mapo (datoteka.zip), ki jo moramo odpakirati. V tej mapi sta dve mapi (*»features«* in *»plugins«*). Vsebino map prenesemo v istoimenski mapi, kjer imamo nameščen program Eclipse.
Zdaj nam preostane le še, da v Eclipse nastavimo pot do tolmača. Izbiramo lahko med tremi tolmači in njihovimi verzijami. Python tolmač, prenesen z uradne strani, IronPython in Jython.
V opravilni vrstici kliknemo na *»window«* in izberemo možnost *»preferences«* (Slika 42).



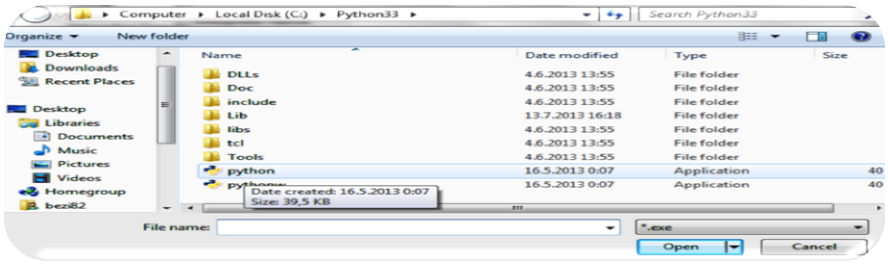
Slika 42: Odpremo nastavitve

Odpre se nam novo okno, kjer kliknemo na *»+«* pred nastavitvijo *»PyDev«* (glej rdeč okvir na Slika 43). S tem dobimo nove izbire, nato kliknemo na *»interpreter-Python«* (glej moder okvir na isti sliki). Zdaj se v desnem delu okna z nastavitvami prikažejo trenutne nastavitve poti do tolmača za Python. Na Slika 43 (zelen okvir) vidimo, da pot še ni nastavljena. Zato jo nastavimo. Kliknemo na gumb *»New«*. Ponovno se nam odpre novo okno (Slika 44), s pomočjo katerega poiščemo datoteko *»python.exe«*. Ko datoteko

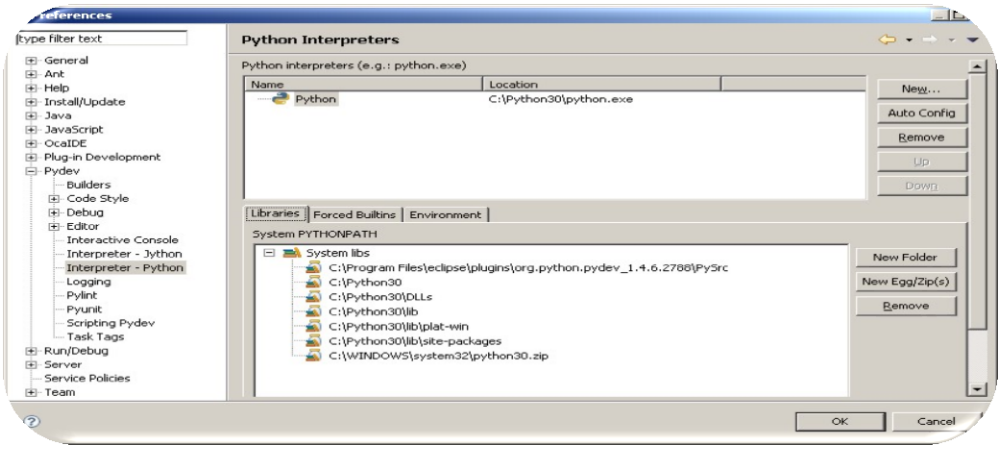
najdemo, še kliknemo na gumb *»open«* in že lahko vidimo spremembe. Na Slika 45 vidimo, da se v polju *»Python interpreters«* izpiše pot do datoteke *»python.exe«*. V zavihku *»Libraries«* (knjižnice), ki je bil sprva tudi prazen, so zdaj izpisane poti do sistemskih knjižnic.



Slika 43: Nastavljanje poti



Slika 44: Python.exe



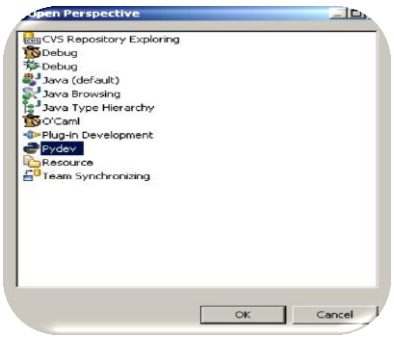
Slika 45: Nastavljena pot

6.2 PRVI PROGRAM V ECLIPSE

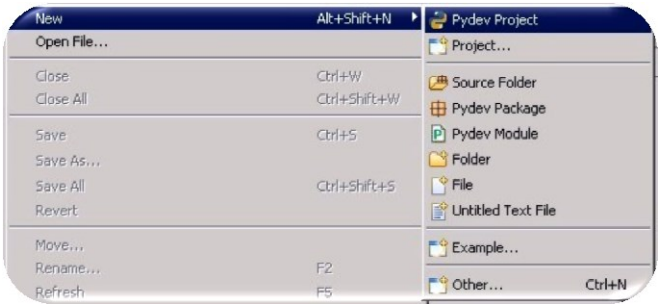
Zdaj, ko smo Eclipse uspešno nastavili tako, da je mogoče pisati tudi programe v Pythonu, si oglejmo, kako napišemo in poženemo program v Eclipse.
Spremeniti moramo delovno okolje, iz Jave v PyDev (Slika 46). Vsako delovno okolje nam namreč ponuja orodja, ki se za izbran programski jezik najpogosteje uporabljajo. Na sliki si poglejmo izbor orodij za delo s Pythonom (Slika 48).

Slika 46: Sprememba delovnega okolja

Odpre se nam novo okno (Slika 47), kjer poiščemo »*Pydev*«.

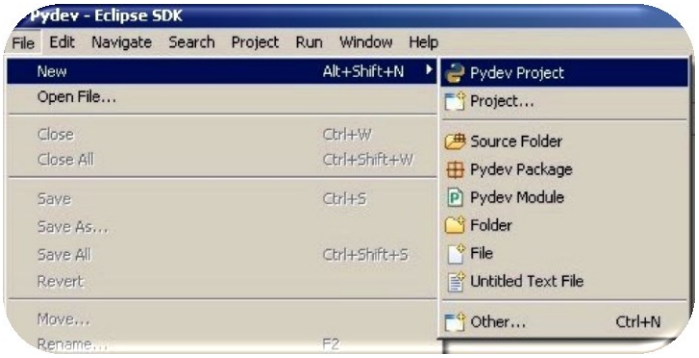


Slika 47: PyDev



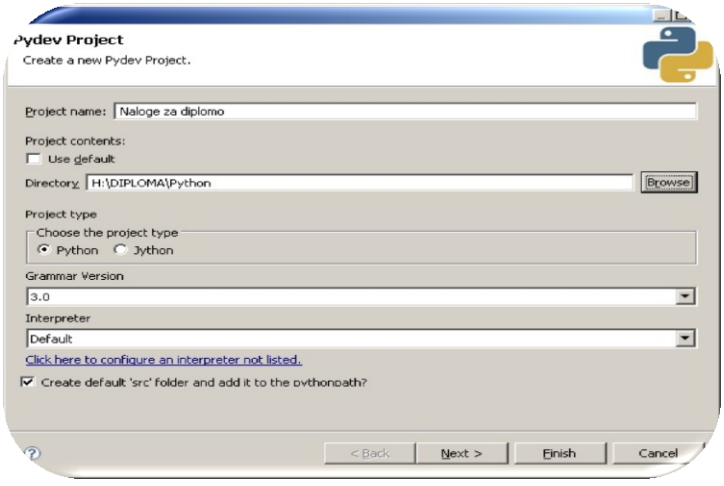
Slika 48: Osnovne izbire, ki jih ponuja okolje PyDev

Če smo navajeni dela v okolju IDLE, kjer le odpremo urejevalnik in začnemo pisati kodo, bomo tukaj malo zmedeni. V okolju Eclipse se vse dela s projekti. Projekt je skupek datotek, ki skupaj predstavljajo rešitev določene programske naloge. Če za rešitev programa potrebujemo le eno datoteko, se nam bo verjetno zdelo vse skupaj prezapleteno. Vendar je dejstvo, da večino programskih rešitev sestavlja več delov – definicije razredov, uporabniški vmesnik, programske knjižnice, testni programi … Takrat pa je primerna organizacija nujna. In to nam omogočajo projekti.
Ker projekta še nimamo, ga moramo ustvariti.
V orodni vrstici kliknemo »`File`«. Z miško se pomaknemo na »`New`«. Odpre se nov zavihek, kjer kliknemo na »`Pydev project`« (Slika 49).



Slika 49: Nov projekt

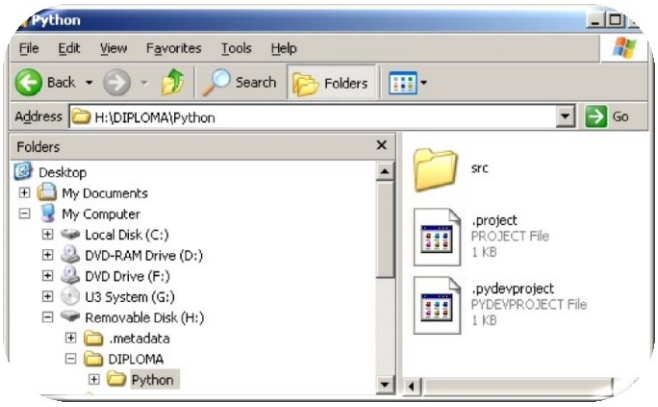
Odpre se novo okno (Slika 50). V prvo okence vpišemo ime projekta. V drugem okencu nastavimo pot do mape na disku, kamor bomo shranili svoj izdelek. V polju »`Grammar Version`« izberemo različico 3.0 oz. tisto različico Pythona, ki ga uporabljamo. Ko smo vse nastavili, kliknemo še na gumb »`Finish`«.



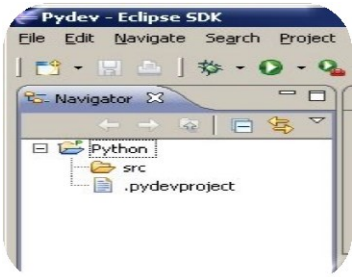
Slika 50: Nastavitve projekta

Če pogledamo na disk, kamor smo shranili projekt (Slika 51), vidimo, da se je tam pojavila mapa z imenom »`src`« ter dve datoteki »`.project`« in »`.pydevproject`«. V to mapo bomo od zdaj naprej shranjevali vse programe.

Da smo ustvarili nov projekt, je razvidno tudi v Eclipse. V levem zgornjem kotu *»navigatorja«* (navigator je eden izmed zavihkov v Eclipse) se pojavi mapa (Slika 52). Mapa je poimenovana tako, kot smo poimenovali projekt. V tej mapi je še ena mapa, imenovana »*src*«.

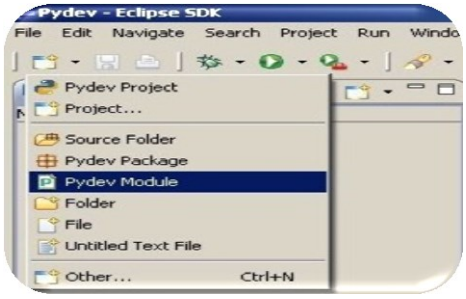


Slika 51: Shranjen projekt

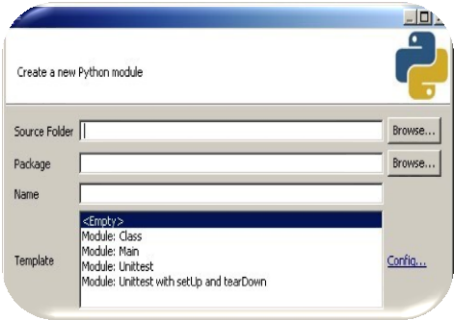


Slika 52: Projekt, viden v navigatorju

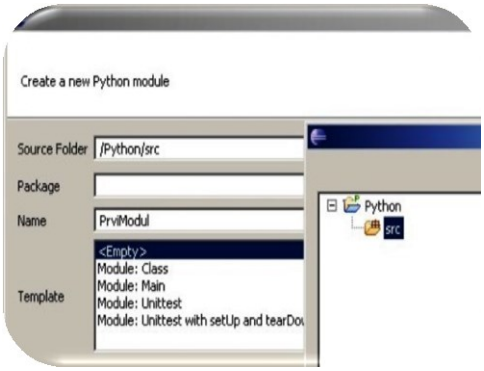
Da bi pričeli s pisanjem programa, moramo ustvariti še nov modul. V orodni vrstici kliknemo *»file«* in kliknemo na polje *»Pydev Module«* (Slika 53). Odpre se nam novo okno Slika 54: Kreiranje modula). V prvem okencu nastavimo pot do projekta, ki mu bo modul pripadal (Slika 55). Ker imamo trenutno le projekt Python, ga izberemo. Drugo okence pustimo prazno. V tretje okence vpišemo ime svojega modula. Ko končamo z nastavljanjem, kliknemo še na gumb *»Finish«*. Ponovno pogledamo v *»navigator«*. Videli bomo novo datoteko in ob njej ime ravnokar ustvarjenega modula (Slika 57). Na isti sliki lahko tudi opazimo, da se je v osrednjem delu urejevalnika pojavil zavihek z imenom modula. To je prostor, kjer bomo pisali kodo programa.



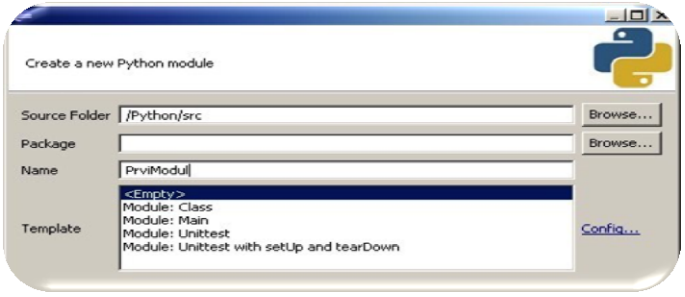
Slika 53: Nov modul



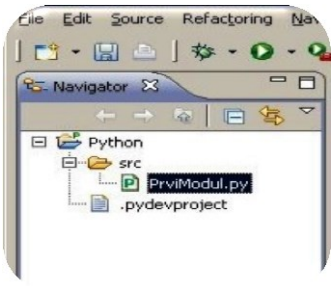
Slika 54: Kreiranje modula



Slika 55: Pot do projekta

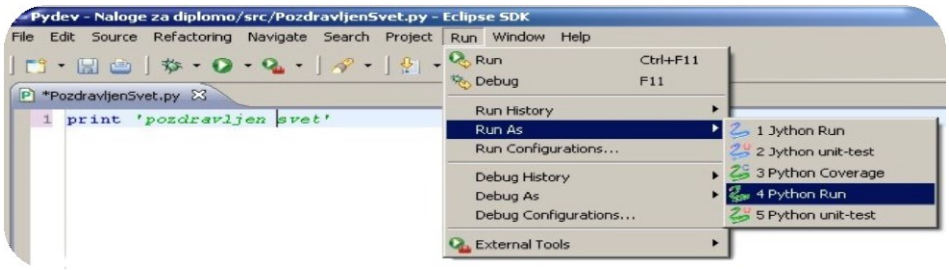


Slika 56: Nastavitve modula

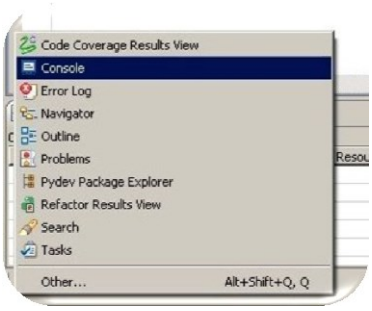


Slika 57: Modul, viden v navigatorju

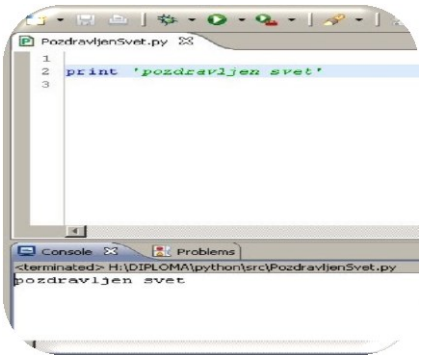
Vedno, ko bomo začeli pisati nov modul, bomo ponovili postopek od Slika 53 naprej, razen v primeru uporabe predloge (več o predlogah bomo razložili v poglavju 6.4.1). Zdaj imamo vse pripravljeno, da napišemo program in ga zaženemo. Ko napišemo kodo, v orodni vrstici kliknemo na »`run`«. Izberemo možnost »`Run As`«, ter »`Python Run`« (Slika 58). Odpremo tudi ukazno mizo (Slika 59), kjer bomo videli rezultat svojega programa (Slika 60).



Slika 58: Koda in zagon programa



Slika 59: Kako odpremo ukazno mizo



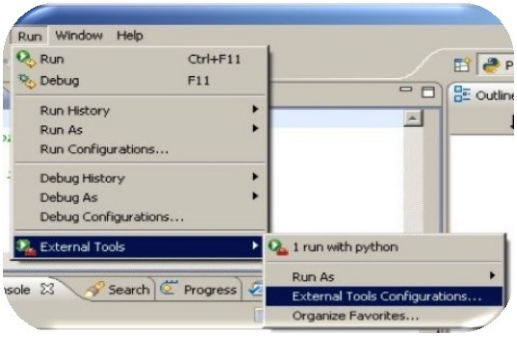
Slika 60: Rezultat programa, spodaj levo

Kot smo omenili, je seveda za pisanje programa, ki ne naredi nič drugega, kot izpiše niz »*Pozdravljen svet*«, to zamuden in zapleten postopek. A pri »pravih« programih nam bo ustrezna organizacija prišla še kako prav. Ker pa nas v diplomski nalogi prvenstveno zanimajo razvojna okolja kot taka in ne razvoj programskih rešitev, tu ne bomo pokazali zgledov, kjer bi se uporaba projektov kot takih izkazala za nujno.

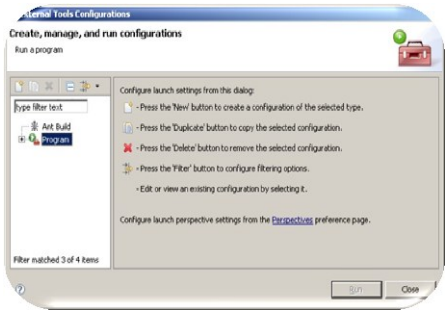
6.3 DODATNE NASTAVITVE V ECLIPSE

Ob vnosu podatkov uporabnika (preko ukaza »*input*«) včasih pride do težav. Te težave nastopijo le, če uporabljamo operacijski sistemi iz družine Microsoft Windows. Operacijski sistem Windows za konec vrstice uporablja dva zloga z vrednostjo 10 in 13. Operacijski sistemi, kot so Linux, Unix in Mac OS X pa uporabljajo samo en zlog vrednosti 13. Zaradi te razlike »*input*« včasih ne deluje tako, kot bi želeli. Tem težavam se lahko izognemo. V nadaljevanju bomo opisali nastavitve, ki nam to omogočajo.
V opravilni vrstici kliknemo na »*Run*«, z miško se pomaknemo na možnost »*External Tools*«. Kliknemo na »*External Tools Configurations*« (Slika 61).
Odpre se nam novo okno (Slika 62). V levem zgornjem kotu novega okna kliknemo na prvo ikono (Slika 63).
V desnem delu nastavitvenega okna se nam odprejo nove nastavitvene možnosti (Slika 64).
V prvo okence, »*Name*«, vpišemo neko smiselno ime za našo nastavitev.
V drugo okence, »*Location*«, vpišemo pot do tolmača »*python.exe*«.

V tretje okence, »*Working Directory*«, vpišemo ukaz *${container_loc}*. Ta ukaz zažene ukazno lupino v projektu, v katerem se nahaja trenutno izbrani Paython modul. Ko izberemo drug modul in ga zaženemo, s temi nastavitvami, se pot do projekta v katerem je ta modul, samodejno nastavi.
V četrto okence, »*Arguments*«, vpišemo ukaz: *-i ${resource_name}* (»*resource_name*« predstavlja ime katerega koli modula). Ta nastavitev vključi interaktivni način, tako da ukazna lupina (»*Console*«) posnema vedenje ukazne lupine v IDLE.
Ko smo končali z nastavljanjem, kliknemo na gumb »*Run*«. Za vse nadaljnje zagone modulov kliknemo na ikono  v orodni vrstici.

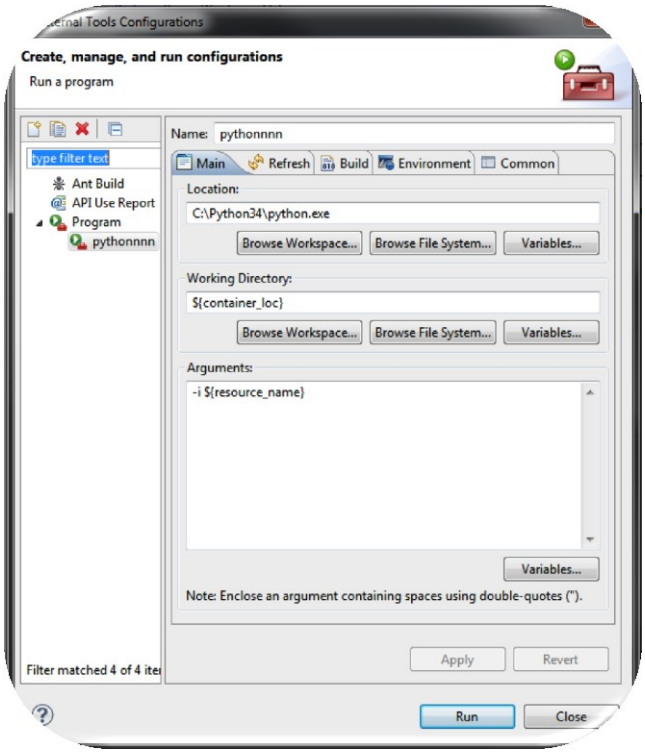


Slika 61: Dodatne nastavitve



Slika 62: Nastavitve zagona programa

Slika 63: Ikona

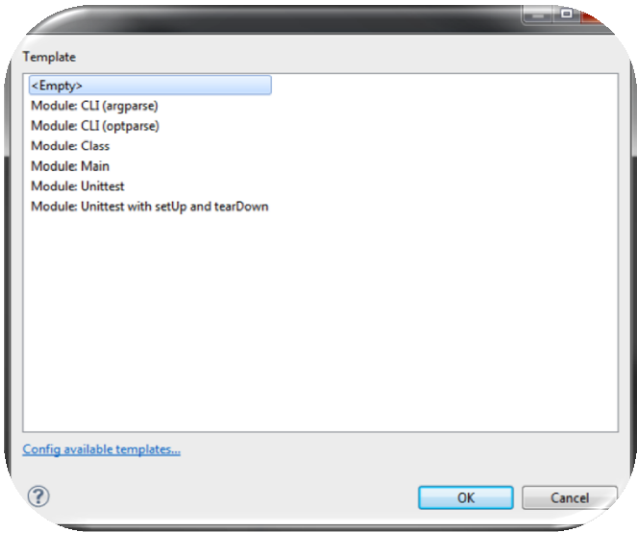


Slika 64: Nastavitve zagona programa

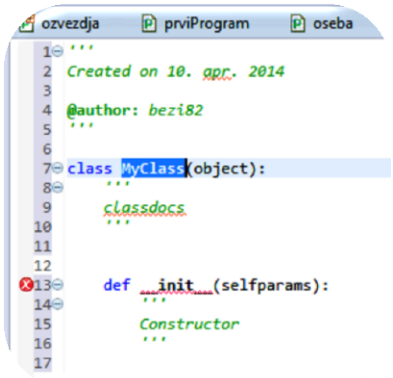
6.4 RAZVOJNA ORODJA ZA PYTHON V ECLIPSE

6.4.1 Predloge

Kaj so predloge, smo si pogledali v razdelku 3.1.
Predloge uporabljamo na dva načina, kot predlogo za nov modul in predlogo kot del kode znotraj modula.
Kadar se odločimo, da bomo ustvarili modul, imamo poleg praznega modula na voljo še šest predlog (Slika 65). Izbira predloge je odvisna od tega, kaj želimo z modulom doseči. Če bo naš modul služil kot razred, bomo seveda izbrali predlogo *»Module: Class«.*
Tako smo ustvarili modul, ki že vsebuje osnovne sestavine razreda (Slika 66). Poleg osnov definicije razreda imamo pripravljen prostor za dokumentacijo tega razreda ter konstruktor. Preostane nam le, da izpolnimo spremenljive dele predloge, kot je poimenovanje razreda, na kar nas opozarja modro obarvan tekst na Slika 66 ter kurzor, ki stoji poleg obarvanega mesta. S tipko »*Tab*« se pomikamo po preostalih mestih, potrebnih popravka.

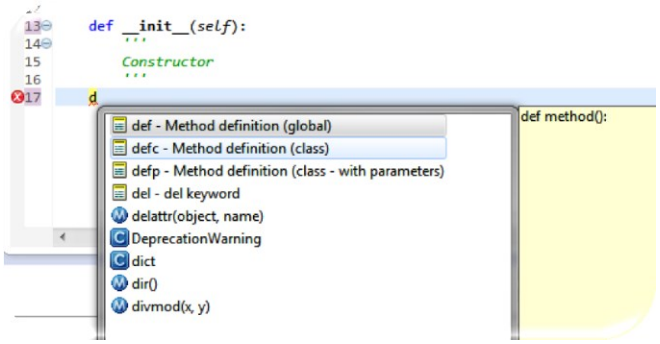


Slika 65: Predloge za nov modul

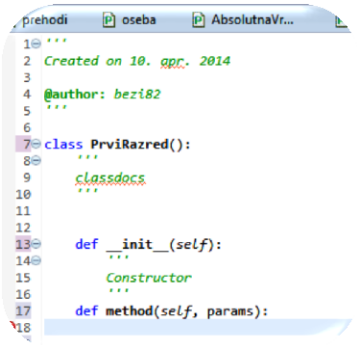


Slika 66: Predloga razreda

Denimo, da želimo v razred zdaj dodati neko metodo. V ta namen bomo uporabili drugo vrsto predloge, tisto, ki nam ustvari del kode. Definicije metod se v Pythonu začno z d, zato vpišemo prvo črko (v tem primeru d). Odpre se okno za samodopolnjevanje kode (o samodopolnjevanju kode smo govorili v razdelku 3.3). Do seznama predlog in njihovih opisov, ki se začnejo na želeno črko, pridemo, če v pojavnem oknu pritisnemo tipki »*Ctrl+space*«. Tu dvakrat kliknemo na tretjo možnost »*defp*«, ki je predloga za metode s parametri (Slika 67). Preostane nam le, da metodo poimenujemo, določimo imena parametrov in napišemo vsebino metode (Slika 68). Dostop do celotnega seznama predlog nam je omogočen, če dvakrat zapored pritisnemo kombinacijo tipk »*Ctrl+space*«.



Slika 67: Seznam predlog in njihovi opisi

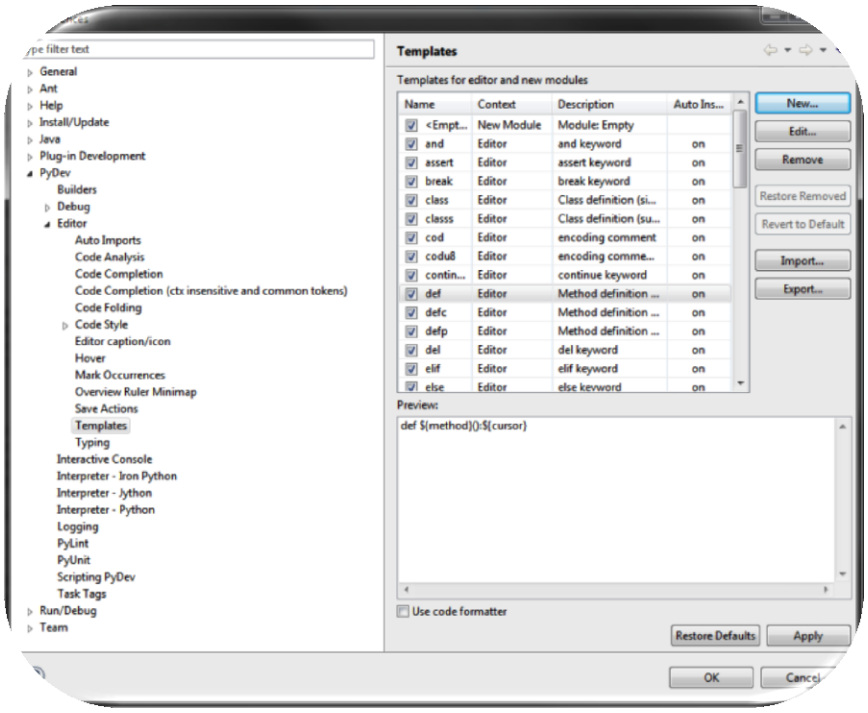


Slika 68: Predloga znotraj modula

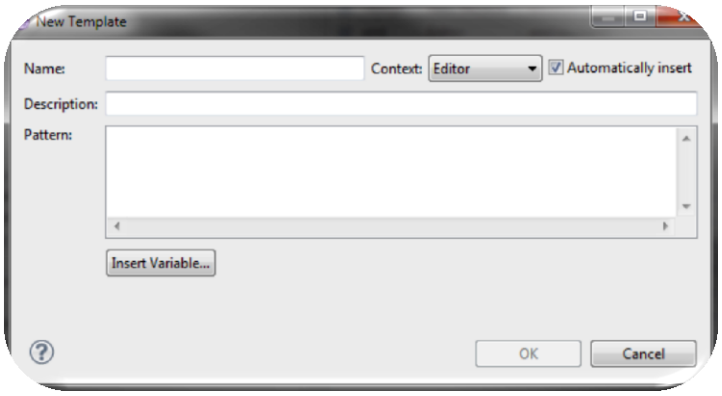
Nove predloge lahko ustvarimo tudi sami. To naredimo tako, da kliknemo na »*Windows → Preferences → PyDev → Editor → Templates →new...*«, ki je ob desnem robu seznama predlog (Slika 69). Odpre se novo okno (Slika 70), ki ga izpolnimo tako:

- v polje »*Name*« vnesemo ime predloge;
- v spustnem seznamu »*Context*« izberemo med »*Editor*« in »*New Modul*«, odvisno od namena uporabe predloge. V primeru »*new modul*« se ustvarjena predloga vidi (uporabi) pri kreiranju novega modula. Če uporabimo možnost Editor, pa bomo uporabljali predlogo znotraj že kreiranega modula. Te predloge so vidne v seznamu predlog, ko pišemo kodo modula (kot je razvidno iz slike Slika 72).
- v polje »*Description*« vnesemo opis predloge;
- v polje »*Pattern*« vnesemo kodo predloge;
- kliknemo na »*Insert Variable*«, da dobimo seznam parametrov, ki so na voljo. Tu govorimo predvsem o parametrih, kot so datum, čas, leto, uporabniško ime tistega, ki je modul ustvaril in podobno.

S klikom na gumb »*OK*« smo končali z ustvarjanjem predloge. Ta se pojavi na seznamu in je od zdaj naprej na voljo za uporabo.



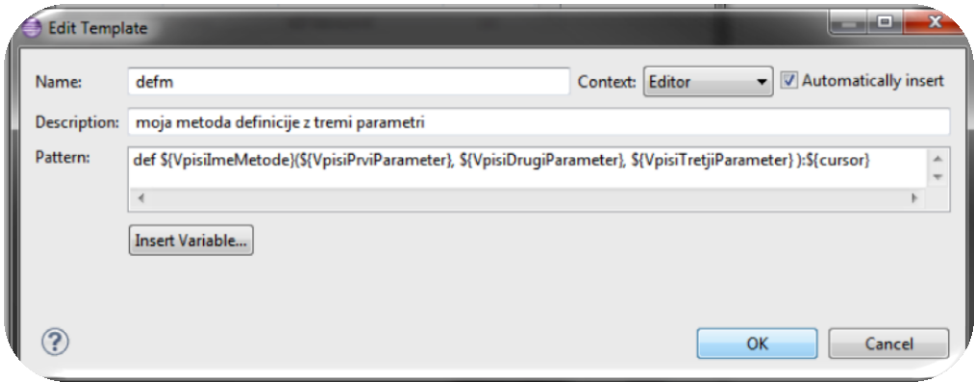
Slika 69: Predloge



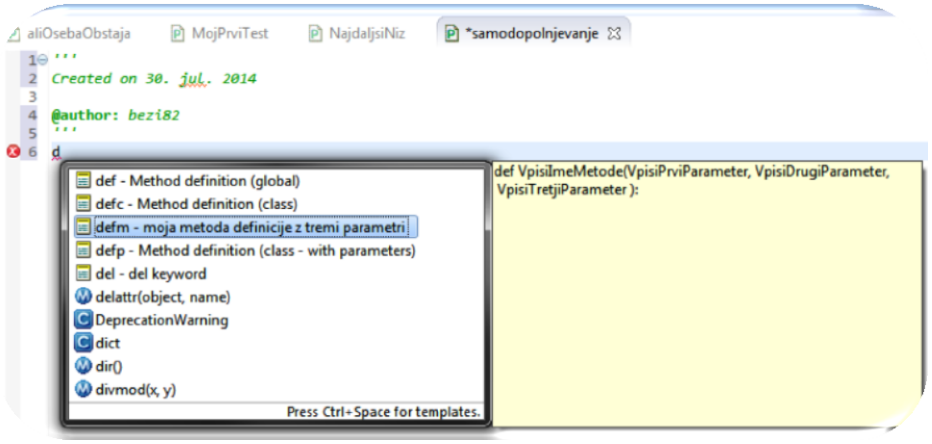
Slika 70: Kreiranje predloge

Koda predloge, kot smo že omenili, ima statični del in spremenljivi del.
Statični del predloge so pravzaprav rezervirane besede in znaki, kot *»def«* za metodo ali *»class«* za razred, oklepaj ali dvopičje. Te dele preprosto napišemo, brez posebnosti.
Spremenljivi del ima obliko: *»${SmiselnoPovemoKajMoramoVpisati}«*. Če povemo z besedami, vsak spremenljivi del vsebuje znak za dolar *»$«*, znotraj zavitih oklepajev napišemo, kaj moramo vstaviti oziroma kaj je manjkajoči del.
Poglejmo si še na primeru, kako napišemo predlogo za metodo s tremi parametri.
Kje ustvarjamo nove predloge, smo se že naučili zgoraj. Na Slika 71 vidimo, da smo si za ime predloge izbrali *»defm«*, za opis smo napisali *»moja metoda definicije s tremi parametri«*. V za vzorec pa smo napisali *»def ${VpisiImeMetode}(${VpisiPrviParameter},*

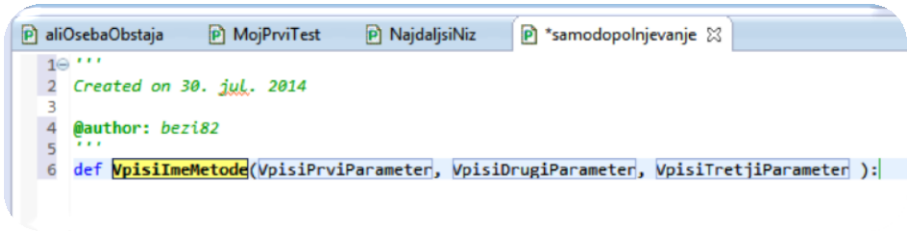
${VpisiDrugiParameter},${VpisiTretjiParameter})
:${cursor}«.
Ustvarjeno predlogo najdemo v seznamu predlog, kot kaže Slika 72. Kako je vidna metoda v našem modulu, prikazuje Slika 73.



Slika 71: Kreiramo predlogo s tremi parametri



Slika 72: Ustvarjena predloga, vidna v seznamu predlog

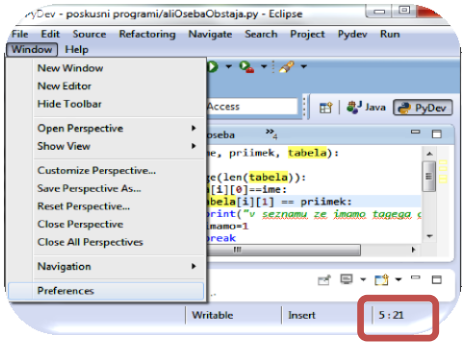


Slika 73: Predloga, vstavljena v modul

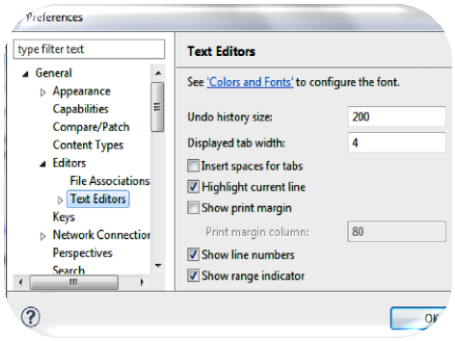
6.4.2 Številčenje vrstic

Ravno tako kot v IDLE v desnem spodnjem kotu najdemo informacijo o tem, v kateri vrstici in katerem stolpcu trenutno smo (v rdečem okvirju). Vendar pa si lahko v

Eclipse vklopimo tudi številčenje vrstic poleg kode programa. V orodni vrstici poiščemo razdelek »*Window*«, tu se spustimo do »*Preferences*« (Slika 74). Odpre se novo okno, kjer kliknemo na razdelek »*General*«, nato na »*Editors*« ter »*Text Editors*«. Tu poiščemo »*Show line numbers*« in izbiro označimo (Slika 74).



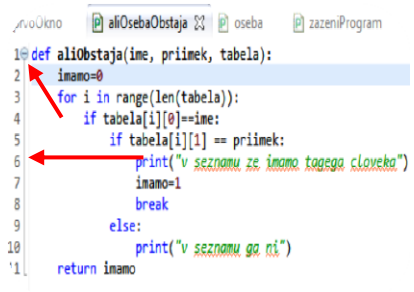
Slika 74: Informacija o položaju



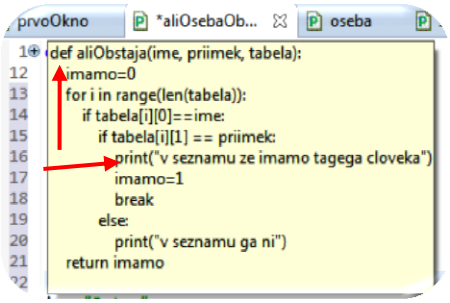
Slika 75: Vklop številčenja

6.4.3 Samodejno zamikanje kode

Eclipse, prav tako kot IDLE, avtomatično zamika bloke, vendar pa gre tudi korak dlje. Vsebino bloka, ki predstavlja definicijo metode, namreč lahko skrijemo ali pa ponovno prikažemo. Ta način skrivanja metod pride v poštev predvsem pri daljših programih, da na primer lahko vidimo vse metode, ki smo jih že napisali, na eni strani. Vsak začetek bloka za definicijo metode označi s krogom, ki vsebuje znak »−« (puščica na Slika 76). Ko smo znotraj metode, se lahko kadar koli postavimo ob levi rob in prikazala se bo črta, ki prikazuje začetek in konec bloka (Slika 76). Če na krog kliknemo, se znak »−« spremenil v »+«. Koda, ki je znotraj bloka metode, se skrije. Od nje ostanejo le tri pike. Ko se z miško postavimo na krog, lahko vidimo vsebino metode (prikaz na Slika 77). Z miškinim klikom na znak »+« ponovno prikažemo celotno kodo.



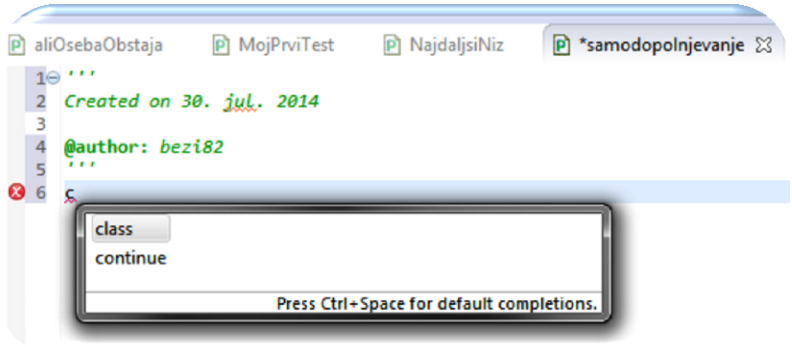
Slika 76: Blok s kodo



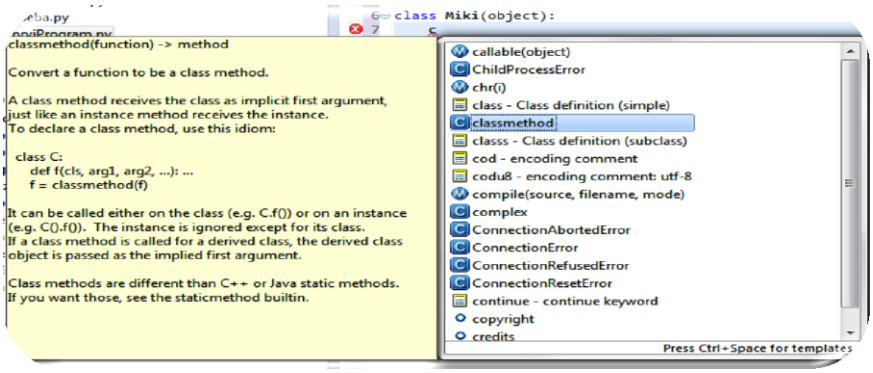
Slika 77: Prikazana koda bloka

6.4.4 Samodopolnjevanje kode

Eclipse ima, ravno tako kot IDLE, vgrajeno pomoč pri pisanju imen funkcij in spremenljivk. Tudi v tem primeru je Eclipse boljši, saj pri dopolnjevanju ponuja tudi rezervirane besede in imena naših funkcij.
Ko vtipkamo prvo črko, nam samodejno ponudi seznam smiselnih imen za nadaljevanje vnosa (Slika 78). Z dopisovanjem črk zožimo izbor kandidatov. Če smo pravega kandidata našli, pritisnemo »enter« in koda se samodejno dopolni.
Če s ponujenim nismo zadovoljni ali nismo prepričani o načinu uporabe, pritisnemo kombinacijo tipk »*CTRL+Space*«. Tako poleg že ponujenega seznama dobimo dopolnjen seznam, skupaj z opisi uporabe (Slika 79).



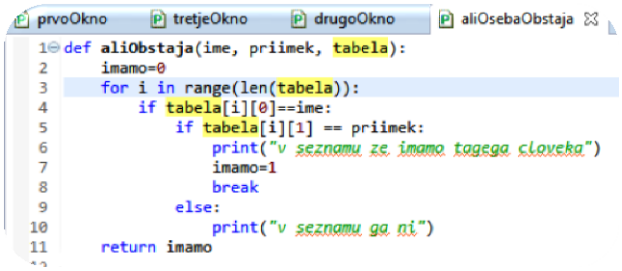
Slika 78: Seznam za samodopolnjevanje kode



Slika 79: Dopolnjen seznam in razlaga

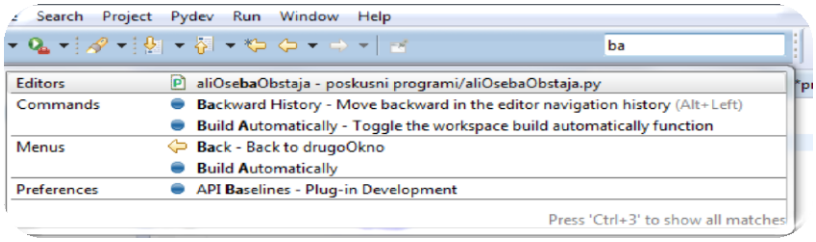
6.4.5 Hitro iskanje

Eclipse ponuja več vrst iskanja. Tu se lahko postavimo s kurzorjem na spremenljivko in že imamo označene vse pojavitve želene spremenljivke (z rumeno barvo označena spremenljivka tabela na Slika 80).



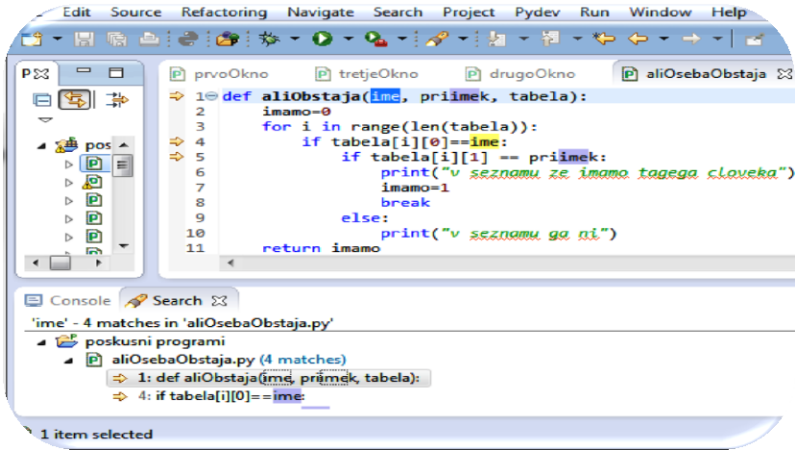
Slika 80: Najdene pojavitve spremenljivk

Druga možnost za iskanje je, da v orodni vrstici v okencu za hitro iskanje vnašamo črke želene besede. Prikazal se bo seznam, ki se bo z vsakim novim vnosom črke zožil. V tem seznamu dobimo seznam vsega, kar je iskalnik v Eclipse zaznal. To so lahko imena projektov, razredov, definicij, spremenljivk pa tudi orodja. Razvrščena so po področjih, v katera spadajo (Slika 81).



Slika 81: Hitro iskanje

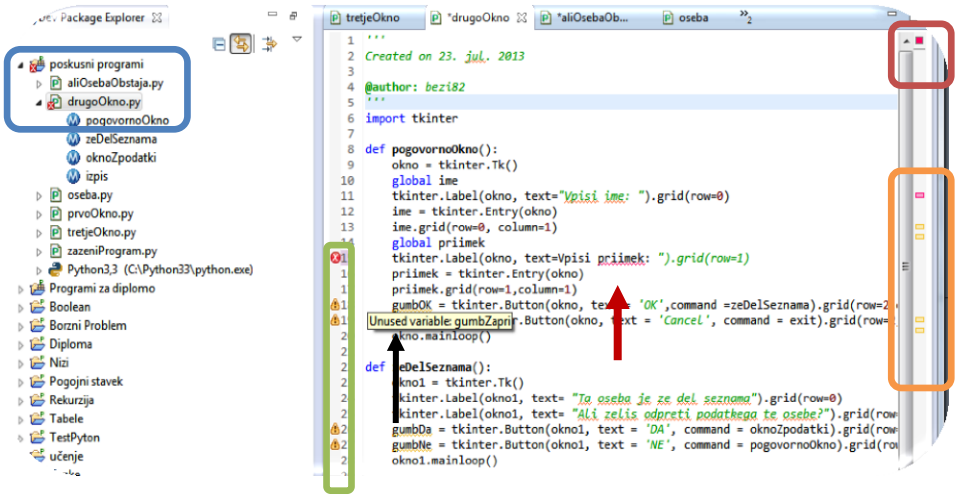
Tretja možnost iskanja je uporaba izbire »*search*«, ki jo najdemo v opravilni vrstici. Tu imamo zopet tri možnosti iskanja. Med njimi si bomo ogledali le zadnjo, to je iskanje po besedilu (»*text*«).
V modulu označimo želeno besedo, nato kliknemo na »*Search*«, se spustimo do »*text*« ter v na novo odprtem seznamu kliknemo na »*File*«. Poleg zavihka »*Console*« se pojavi in odpre zavihek »*Search*«, v katerem se izpišejo rezultati svojega iskanja. Izpišejo se vse vrstice kode, v katerih se pojavi iskani niz (iskani niz na Slika 82 je ime).



Slika 82: Iskani niz »ime«

6.4.6 Označevanje napak

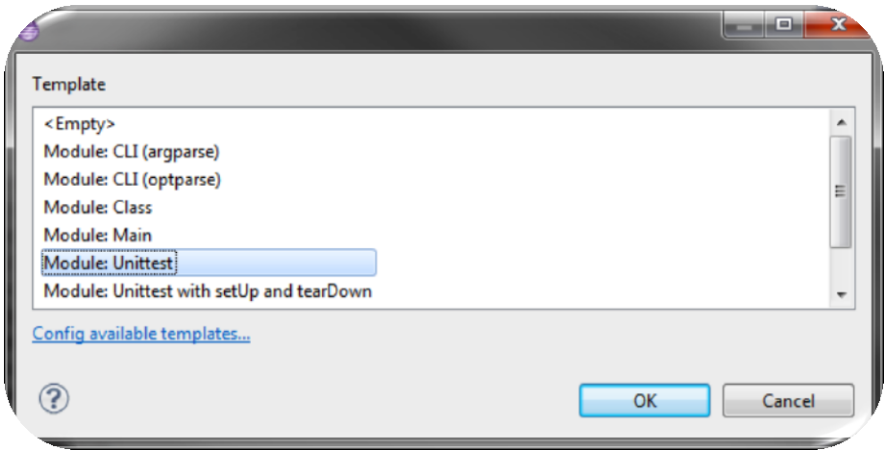
Eclipse ponuja številna opozorila, ki nas opozarjajo na napake v kodi.
Opozorila o napakah so podana v rdeči in rumeni barvi. Napaka, prikazana rdeče, pomeni, da je prišlo do sintaktične napake in jo moramo nujno popraviti. Rumena oznaka je le opozorilo, da se lahko ob zagonu programa sproži izjema, da uporabljamo v programu spremenljivko le v prireditvenem stavku potem pa je nikoli ne uporabimo in podobno.
Mesto napake se vijugasto podčrta v barvi napake (na prikaz napake kaže rdeča puščica na Slika 83). Če se postavimo z miško na mesto napake, se prikaže natančen opis le-te (na isti sliki nanj kaže črna puščica).
Levo od kode se napaka označi na dva načina. Opozorilo rdeče barve se prikaže v obliki kroga z znakom »*x*«, medtem ko se opozorilo rumene barve prikaže kot trikotnik z znakom »*!*« (obe napaki sta vidni na Slika 83 v zelenem okviru). V obeh primerih se, ko se znaku približamo z miško, prikaže kratek opis napake.
Ob desnem zgornjem robu kode se ob napaki pojavi kvadrat ustrezne barve (na isti sliki rdeč okvir), ki nam prikaže število napak oziroma opozoril. Vzdolž kode se prikažejo pravokotniki ustrezne barve (Slika 83, oranžne barve). Pravokotnikov je toliko, kolikor napak smo naredili. Ob kliku na pravokotnik se kurzor premakne na mesto napake.
V podatkovnem drevesu se na mapi paketa in na samem modulu ravno tako prikaže znak za opozorilo, da je v njem napaka (Slika 83, okvir modre barve).



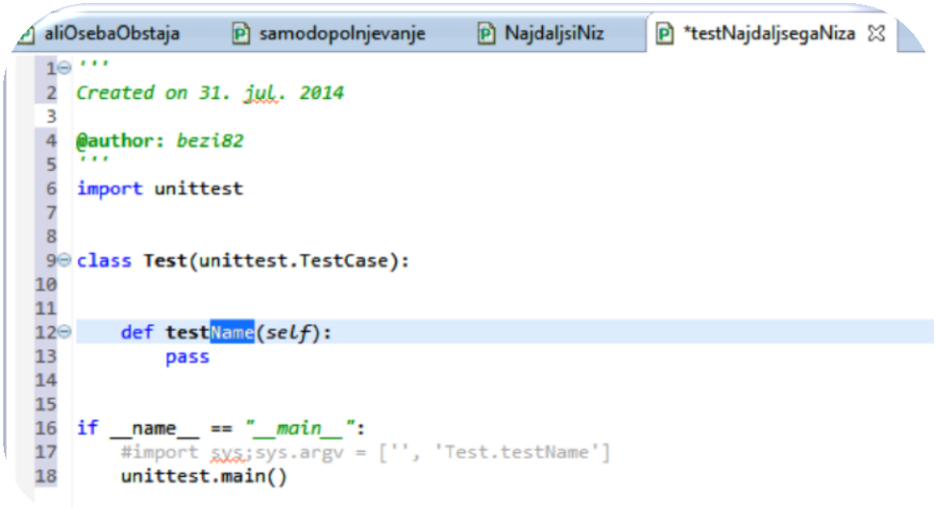
Slika 83: Prikaz napak

6.4.7 Testiranje po sklopih – unittest

Testiranje po sklopih smo spoznali že v razdelku 3.7. Tu omenimo le edino razliko pri pisanju unittest programa, ki je v tem, da nam ni treba pisati cele kode, saj imamo na voljo predlogo za unittest modul. Kot vemo, imamo pri kreiranju novega modula ponujen nabor predlog, ki ga vidimo na Slika 84. Ena izmed njih je predloga za kreiranje unittest modula. Kako je videti tako kreiran modul, vidimo na Slika 85. Pisanje kode za testni program nadaljujemo tako, kot smo se naučili v razdelku 3.7.



Slika 84: Unittest predloga za modul



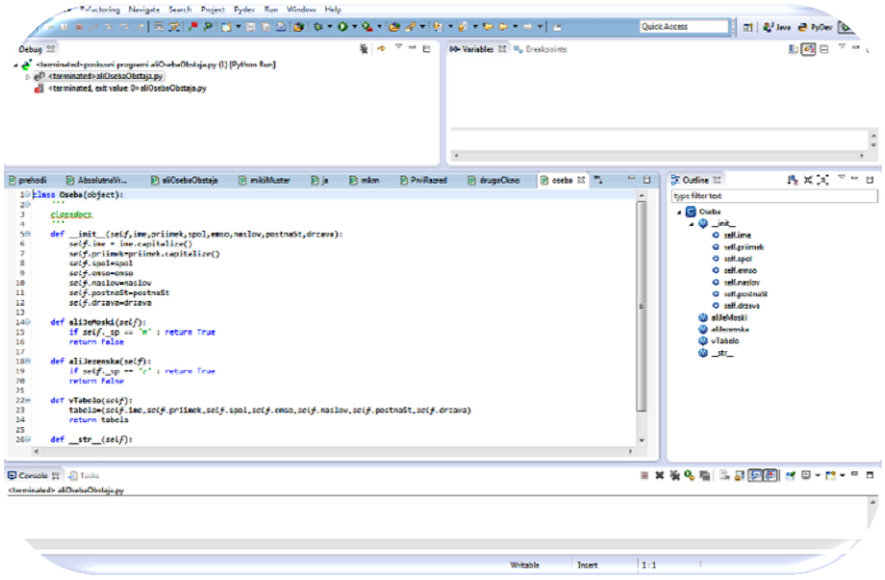
Slika 85: Unittest modul

6.4.8 Python razhroščevalnik

Kaj je razhroščevalnik, smo izvedeli v razdelku 3.8. V kakšnih primerih uporabljamo razhroščevalnik, smo izvedeli v razdelku 2.

Za delo z razhroščevalnikom si odpremo delovno okolje »*Debug*«, tako kot smo preklopili iz »*Jave*« na »*PyDev*« (Slika 86). Pogled se nam spremeni, kot kaže Slika 87.

Slika 86: Sprememba delovnega okolja

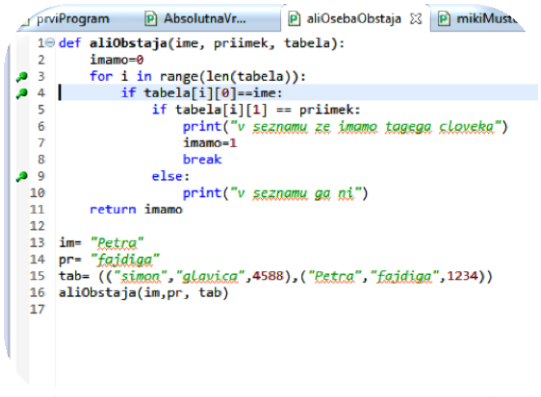


Slika 87: Delovno okolje razhroščevalnika

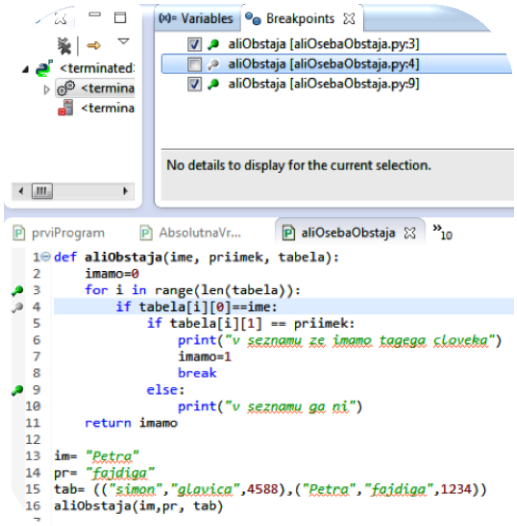
Razhroščevalnik je zdaj pripravljen na razhroščevanje zadnjega zagnanega modula. Če to ni modul, ki ga želimo pregledati, v orodni vrstici kliknemo na puščico, ki je del ikone »*Debug*« (Slika 88). Odpre se spustni seznam, iz katerega izberemo želeni modul. Če modula ni na seznamu, se spustimo do »*Debug As*« in izberemo »*Python Run*«.

Slika 88: Ikona Debug

Prekinitvene točke postavljamo z dvoklikom na številko vrstice ali levo od nje, kjer želimo začeti z natančnejšim pregledom kode. Postavimo lahko poljubno število točk, ki se označijo zeleno (Slika 89). Točko lahko kadar koli odstranimo z dvoklikom nanjo. Dodatna možnost, ki nam jo Eclipse ponuja, je, da postavljeno točko tudi začasno izklopimo. Kliknemo na zavihek »*Breakpoints*«, kjer vidimo seznam postavljenih točk. Odkljukamo točko, ki jo želimo začasno izklopiti, in ta se obarva sivo (Slika 90).

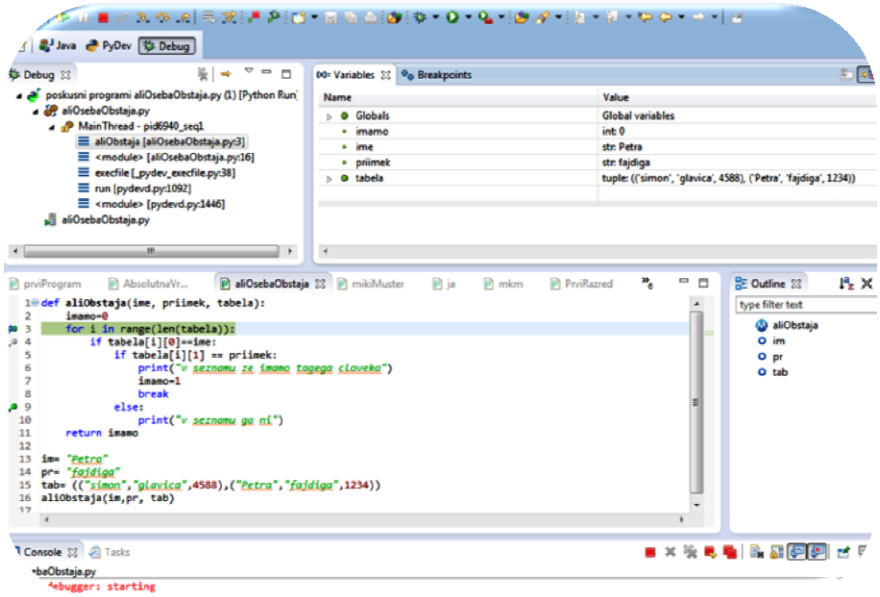


Slika 89: Prekinitvene točke



Slika 90: Začasen izklop prekinitvene točke

Zdaj s klikom na ikono »*Debug*« poženemo razhroščevanje. Razhroščevalnik se zaustavi na prvi izvajalni vrstici in v zeleno pobarva vrstico, v kateri trenutno smo. V zavihku »*Variables*« dobimo podatke o tem, katere spremenljivke imamo in kakšne so njihove trenutne vrednosti. Zavihek »*Console*« nam sporoča, da je razhroščevalnik zagnan (Slika 91).



Slika 91: Razhroščevanje

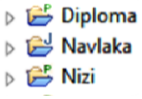
Različne možnosti nadaljevanja opravljamo z ikonami v orodni vrstici (Slika 92).
Ikone za kontrolo razhroščevanja smo spoznali v razdelku 3.8, tu pa jih le navedimo: »*Resume*«, »*Terminate*«, »*Step Into*«, »*Step Over*«, »*Step Return*«.

Slika 92: Ikone za nadzor razhroščevanja

Razhroščevalnik v Eclipse je veliko preglednejši, kot je razhroščevalnik v IDLE. Poleg tega ima Eclipse možnost, da začasno izklopimo nekatere prekinitvene točke, česar IDLE ne omogoča.

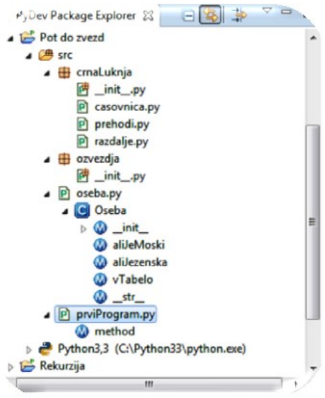
6.4.9 Projektno drevo

Eclipse lahko prikaže projektno drevo (kaj je projektno drevo, izvemo v razdelku 3.9). Privzeto se nahaja ob levem robu okolja (Slika 94). Lahko pa ga tudi poljubno prestavljamo. To storimo tako, da kliknemo in držimo miškin gumb na »*PyDev Package Explorer*« in ga prenesemo, kamor želimo.
Pogled na projektno drevo je prednastavljen na »*project*«, kar pomeni, da vidimo abecedno urejene projekte.
Projekti so prikazani v obliki map, ki so označene z začetnico programskega jezika (Slika 93: Projekti).



Slika 93: Projekti

S klikom na trikotnik na levi strani projekta se nam odpre struktura le-tega (glej razlago projekta v razdelku 2). Z vsakim dvoklikom na katero koli izmed struktur se pomikamo globlje po strukturi projekta. Najgloblji nivo je struktura modula (struktura modula so v modulu napisani razredi, metode, spremenljivke …). Ko dvokliknemo na modul, se le-ta odpre. Če dvokliknemo na katero koli od struktur modula, pa se modul odpre in kurzor se postavi na mesto strukture (če dvokliknemo na metodo »`aliJezenska`« s Slika 94, se bomo znašli v modulu »`oseba`«, v razredu »`Oseba`«, v vrstici, kjer je deklarirana metoda »`aliJezenska`«).



Slika 94: Projektno drevo

Iz Slika 94 lahko razberemo, da imamo v projektnem drevesu dva projekta: »`Pot do zvezd`« in »`Rekurzija`«. Da imamo tudi dva paketa: »`crnaLuknja`« in »`ozvezdja`«. Ta paketa imata modul »`_init_.py`«. V celotnem projektu imamo 5 modulov, 1 razred »`Oseba`« in 6 metod. Opazimo pa tudi, da imamo podatke o uporabljenem tolmaču.

7 PYCHARM

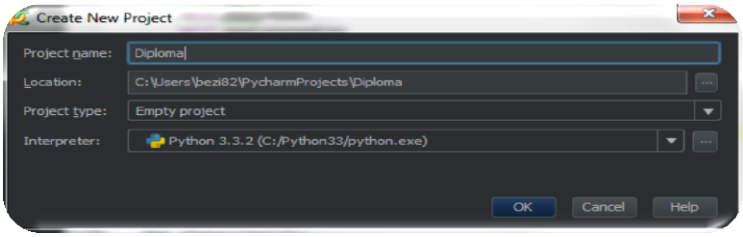
Kje in kako dobimo Pycharm, smo povedali že v razdelku 4.
Razvijalci PyCharma pravijo, da je programiranje z njim, kot da bi programiral v paru, saj PyCharm bdi nad tvojim početjem in ti nudi pametne nasvete, ki se prikažejo v obliki prižgane žarnice.

7.1 NAMESTITEV PYTHONA V PYCHARM

Po namestitvi PyCharma na računalnik dodatni programi in nastavitve niso potrebni, zato lahko takoj začnemo s pisanjem prvega programa. Le včasih se zgodi, da PyCharm ni sam ugotovil, da je na našem računalniku nameščen tolmač za Python: to zaznamo pri odpiranju novega projekta, saj je vrstica prevajalnika (»*interpreter*«) prazna (o novem projektu bomo govorili v naslednjem razdelku). Takrat moramo ročno dodati pot do tolmača.

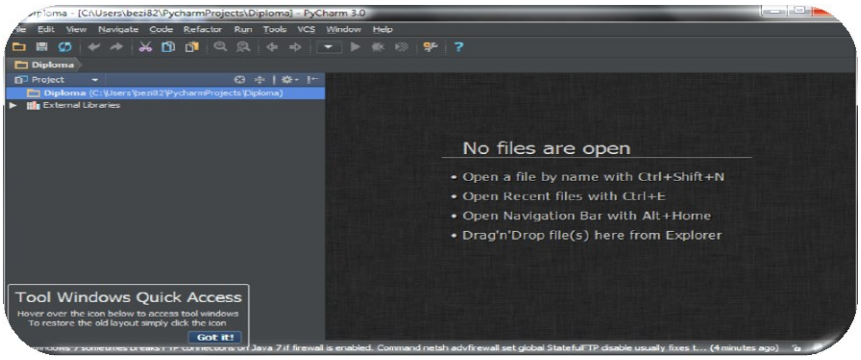
7.2 PRVI PYTHON PROGRAM V PYCHARM

V PyCharmu se, ravno tako kot v Eclipse, dela s projekti. Ker projekta še nimamo, ga moramo ustvariti. Da bi projekt ustvarili, moramo narediti sledeče: v opravilni vrstici kliknemo »*File*« → »*New Project*«.
Odpre se novo okno (Slika 95), kjer v prvo okence vpišemo ime svojega projekta. V drugem okencu nastavimo pot do mape na disku, kamor bomo shranili svoj izdelek. V tretjem okencu izberemo, kakšne vrste bo naš projekt. Glede na našo odločitev se bo odprlo ustrezno delovno okolje. Če tega še ne vemo, lahko odpremo prazen projekt. Zadnje okence je namenjeno poti do tolmača za Python, ki jo ročno dodamo, če je še ni. Pot si Pycharm zapomni, tako nam je pri naslednjem novem projektu ni treba ponovno nastavljati. Iz slike je razvidna nastavljena pot.



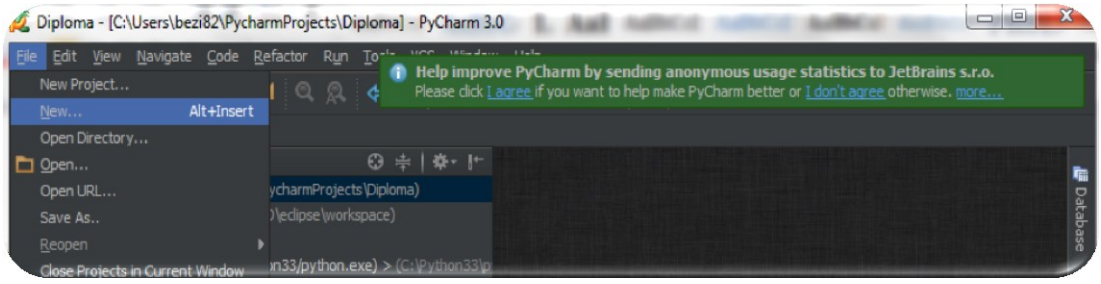
Slika 95: Nov projekt

Da smo ustvarili nov projekt, je razvidno v PyCharm. V levem delu PyCharma, v nekakšnem navigatorju, se pojavi mapa (Slika 96). Mapa je poimenovana, kakor smo poimenovali projekt.



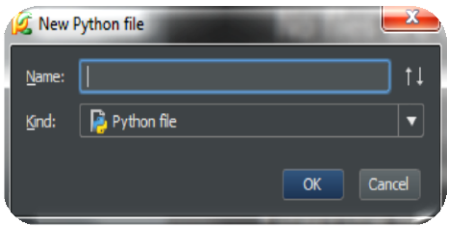
Slika 96: Navigator

Da bi začeli pisanje programa, moramo ustvariti še nov modul. V opravilni vrstici kliknemo *»File«* in kliknemo na izbiro *»New«* (Slika 97). Odpre se nam novo okno (Slika 98), tu izberemo *»Python File«*. Ponovno se odpre novo okno. Tu v prvo okence vpišemo ime modula (Slika 99) in kliknemo *»OK«*.



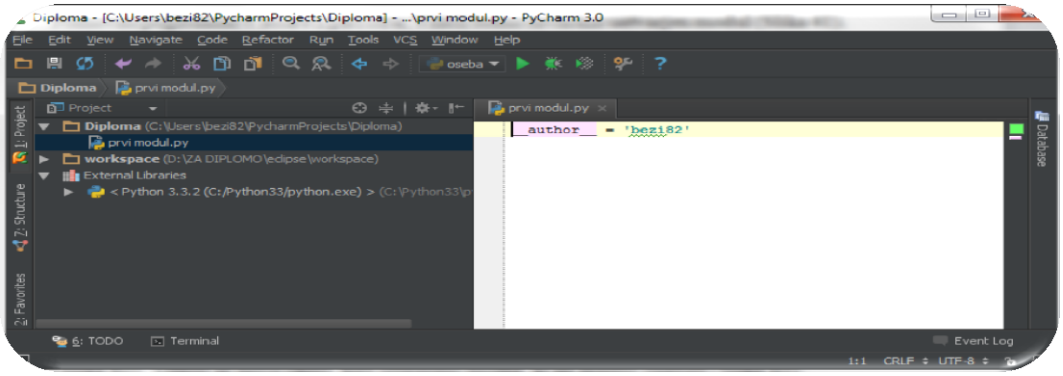
Slika 97: Nov modul

Slika 98: Python dokument



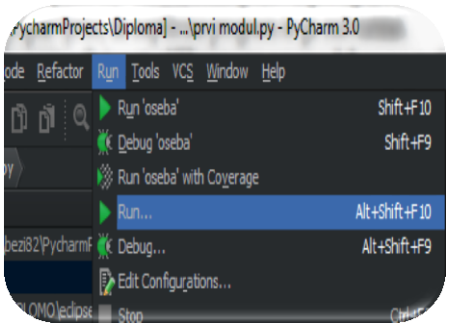
Slika 99: Poimenovanje modula

Ponovno pogledamo v *»navigator«*. Videli bomo ravnokar ustvarjen modul (Slika 100). Na isti sliki lahko tudi opazimo, da se je v osrednjem delu urejevalnika pojavil zavihek z imenom modula. To je prostor, kjer bomo pisali kodo programa.



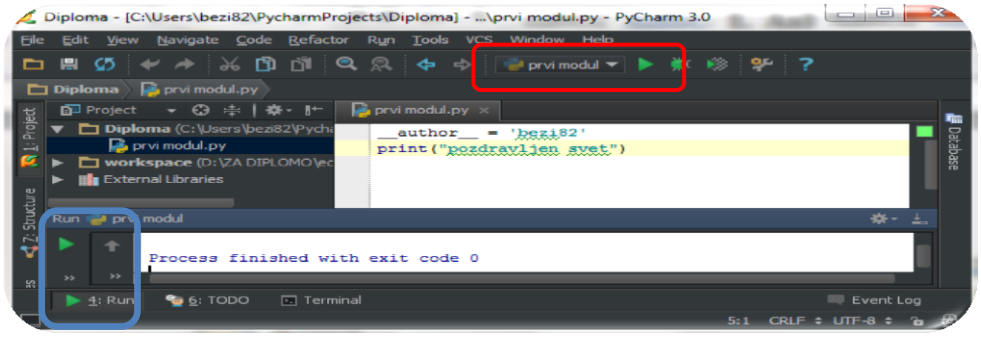
Slika 100: Ustvarjen modul

Vedno, ko bomo začeli pisati nov modul, bomo ponovili postopek od Slika 97 naprej.
Zdaj imamo vse pripravljeno, da napišemo program in ga zaženemo.
Ko napišemo kodo, v opravilni vrstici kliknemo na *»run«*. Izberemo možnost *»Run...«* (Slika 101). Odpre se novo okno, kjer izberemo modul, ki ga želimo zagnati (Slika 102).
V spodnjem delu okna se avtomatično odpre okno, v katerem vidimo rezultat ravno zagnanega modula.



Slika 101: Zagon modula

Slika 102: Izbira modula za zagon



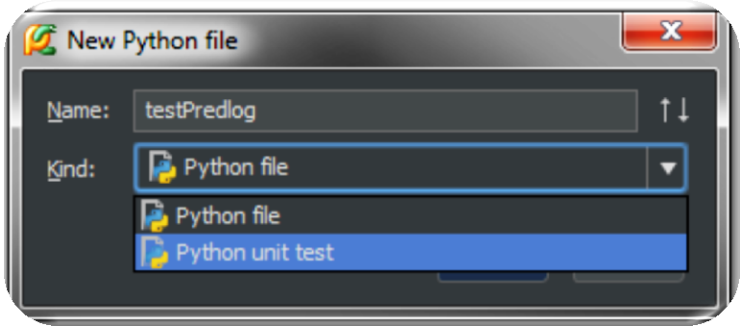
Slika 103: Rezultat modula

Ko smo modul prvič zagnali, se v orodni vrstici poleg zelene puščice za zagon izpiše ime pravkar zagnanega modula (v rdečem okvirju na Slika 103). V okencu z imenom modula se shranjujejo imena zagnanih modulov. Med njimi izbiramo, če kliknemo na sivo puščico in dobimo spustni seznam s temi imeni. S klikom na zeleno puščico nato zaženemo izbrani modul.
Prav tako lahko s klikom na zelene puščice v desnem spodnjem kotu (modri okvir na isti sliki) ponovno zaženemo naš modul.

7.3 RAZVOJNA ORODJA

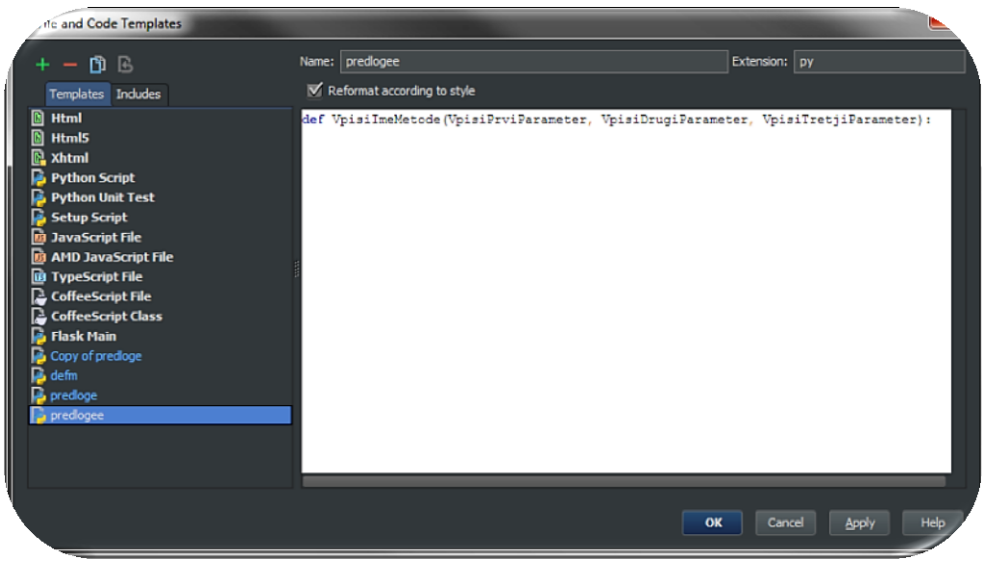
7.3.1 Predloge

Kaj so predloge, smo spoznali v razdelku 3.1.
V Pycharmu imamo na razpolago le štiri predpripravljene predloge. Od teh štirih predlog za potrebe te diplome prideta v poštev le dve. Obe predlogi sta na razpolago pri kreiranju novega modula (Slika 104). Prva »`Python file`« predloga ima le eno vrstico, ki je podatek o avtorju. Druga »`Python unit test`« je predloga za unittest.



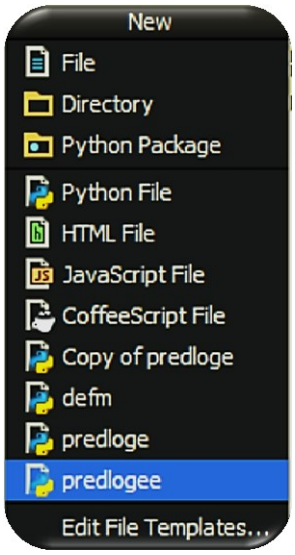
Slika 104: Izbira predloge za nov modul

Zaradi omejenega števila predlog si želimo, da bi lahko kreirali svoje predloge. To sicer lahko storimo, a ponovno smo omejeni, saj lahko kreiramo le predloge za nov modul. Poglejmo si, kako to storimo. Prva možnost je, da ustvarimo modul, vanj vpišemo želeno kodo za predlogo. Nato v opravilni vrstici kliknemo na »*Tools → Save File as Template*«. Odpre se novo okno (Slika 105), v katerem lahko predlogo preimenujemo in popravljamo. Druga možnost je, da v opravilni vrstici kliknemo na »*File → settings*«. V novem oknu, ki se odpre, poiščemo možnost »*File and Code Templates*« ter kliknemo nanjo. Po tej poti ponovno pridemo do okna, ki ga vidimo na Slika 105. Tu kliknemo na znak plus in izpolnimo polja »*Name*« za ime predloge in »*Extension*«, tu vpišemo »*py*«, saj kreiramo predlogo v jeziku Python. V bel prazen prostor, kjer je na Slika 105 napisana koda in je zdaj prazen, napišemo kodo za predlogo. V Pycharmu ni mogoče uporabljati parametrov, kot smo lahko videli v Eclipse. Zato moramo parametre ustrezno poimenovati, da bo jasno razvidno, da moramo ta del spremeniti (na sliki Slika 105 vidimo, da za poimenovanje vsakega parametra uporabimo besedo »*vpiši*«, tako vemo da moramo ta parameter spremeniti).



Slika 105: Kreiranje predloge

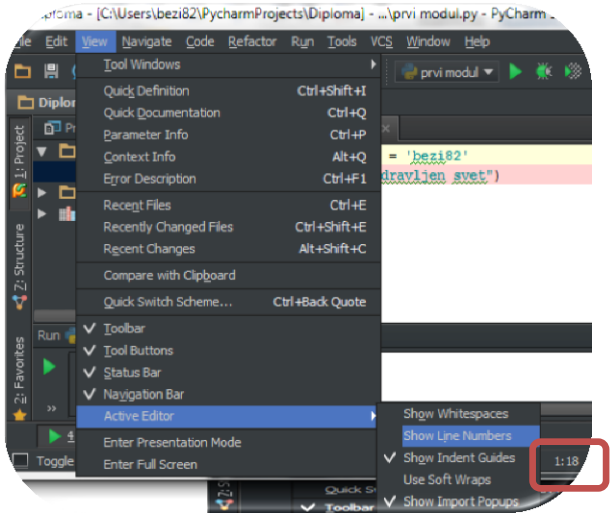
Ko želimo predlogo uporabiti, v orodni vrstici kliknemo na »*File → New*«, odpre se novo okno, kot kaže Slika 106.



Slika 106: Izbor predloge

7.3.2 Številčenje vrst

Ravno tako kot v Eclipse in IDLE v desnem spodnjem kotu najdemo informacijo o tem, v kateri vrstici in katerem stolpcu se trenutno nahajamo (Slika 107 v rdečem okvirju).
Samo številčenje vrstic kode moramo, podobno kot v Eclipse, posebej vklopiti (to naredimo le enkrat). V orodni vrstici poiščemo razdelek »`View`«, tu se spustimo do »`Active Editor`« ter odkljukamo »`Show Line Number`«.



Slika 107: Številčenje vrst kode

7.3.3 Samodejno zamikanje kode

PyCharm, podobno kot Eclipse, zamika bloke. Vsak začetek in konec bloka metode označi s trikotnikom. Ko se z miško postavimo na začetni trikotnik, se s črto poveže s

končnim trikotnikom in obratno. Če na trikotnik kliknemo, se začetni trikotnik spremeni v kvadrat z znakom »+«. Koda, ki je znotraj bloka metode, se skrije. Od nje ostanejo le tri pike. Ko se z miško postavimo na te pike, lahko vidimo vsebino metode (prikaz na Slika 110). Z miškinim klikom na znak »+« ponovno prikažemo celotno kodo (Slika 108 s trikotniki in kodo, Slika 109 z znakom + brez kode).

```
def aliObstaja(ime, priimek, tabela):
    imamo=0
    for i in range(len(tabela)):
        if tabela[i][0]==ime:
            if tabela[i][1] == priimek:
                print("v seznamu ze imamo tagega cloveka")
                imamo=1
                break
            else:
                print("v seznamu ga ni")
    return imamo
```

Slika 108: Bloki

```
def aliObstaja(ime, priimek, tabela):...
```

Slika 109: Skrita koda

Bloke zank in »`if`« stavkov preprosto poveže s pokončno črto.

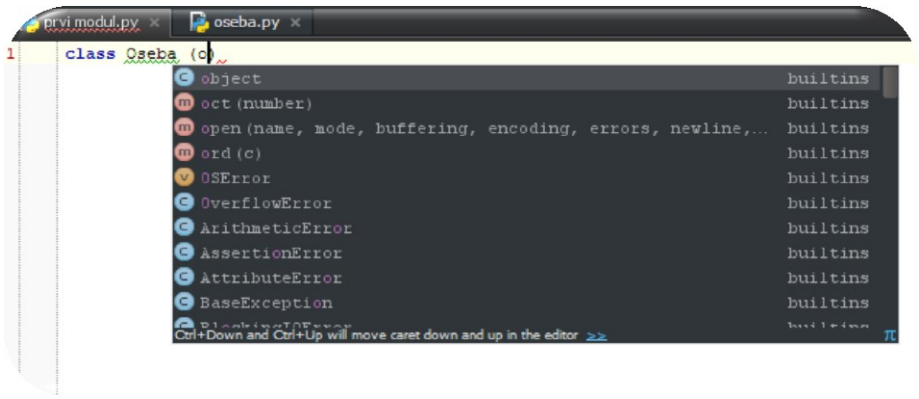
```
def aliObstaja(ime, priimek, tabela):...
    imamo=0
    for i in range(len(tabela)):
        if tabela[i][0]==ime:
            if tabela[i][1] == priimek:
                print("v seznamu ze imamo tagega cloveka")
                imamo=1
                break
            else:
                print("v seznamu ga ni")
    return imamo
```

Slika 110: Prikazana koda

7.3.4 Samodopolnjevanje kode

Kaj je samodopolnjevanje kode smo opisali v razdelku 3.3. V PyCharmu deluje samodopolnjevanje kode na podoben način, kot je to v Eclipse, le da je tu lepši izgled.

Že pri vnosu prve črke nam samodejno ponudi seznam smiselnih imen za nadaljevanje vnosa (Slika 111). Po seznamu se lahko pomikamo z smernimi tipkami na tipkovnici. Z dopisovanjem črk zožimo izbor kandidatov. Če smo pravega kandidata našli, pritisnemo »*enter*«, tipko »*tab*« ali dvokliknemo z miško in koda se samodejno dopolni.
Če s ponujenim nismo zadovoljni, pritisnemo kombinacijo tipk »*CTRL+Space*«. Tako poleg že ponujenega seznama dobimo dopolnjen seznam. Recimo, da izberemo za prvo črko o, potem dobimo seznam smiselnih imen, ko pa želimo dopolnjen seznam, se ta dopolni tako, da dobimo seznam vseh besed, ki vsebujejo črko o.



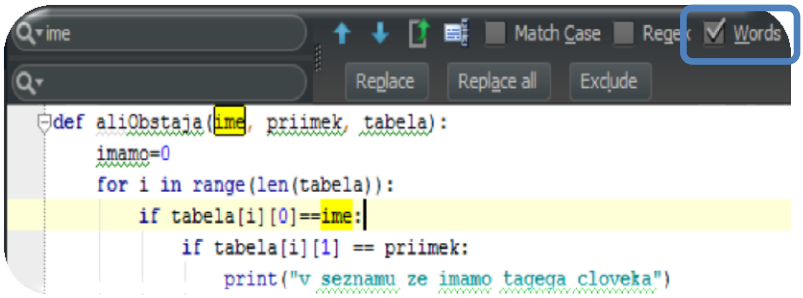
Slika 111: Seznam

Iz slike Slika 111 je tudi razvidno, da ima vsaka ponujena beseda svojo oznako. Tako je razred oziroma razredna struktura označena z modrim krogom, znotraj katerega je črka c. Metode so označene z rdeče pobarvanim krogom in črko m, ključne besede z oranžno pobarvanim krogom in črko v.

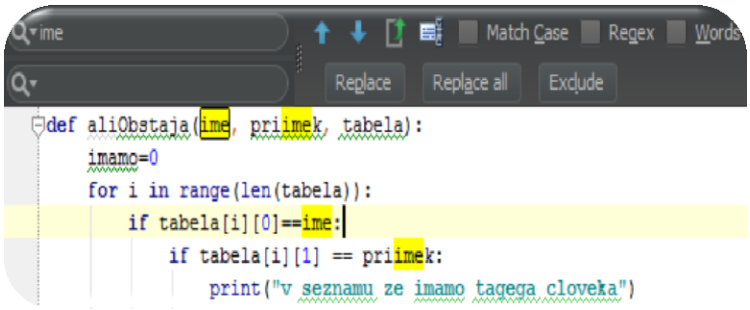
7.3.5 Hitro iskanje

Da bi omogočili hitro iskanje, moramo v orodni vrstici pritisniti na ikono s povečevalnim steklom. Kot vidimo na sliki Slika 111 imamo odprta dva modula, ki sta predstavljena kot zavihka (prvi »*modul.py*« in drugi »*oseba.py*«). Hitro iskanje je potrebno vključiti v vsakem zavihku (modulu) posebej. Iščemo lahko besedo z vsemi pojavitvami. Tu je mišljeno tudi, ko se iskana beseda pojavi znotraj druge besede (Slika 112 prikazuje izključno iskano besedo ime, Slika 113 prikazuje vse pojavitve besede, tudi ko se beseda ime pojavi v besedi priIMEk). Če želimo najti izključno iskano besedo, moramo obkljukati »*words*« (modri okvir na Slika 112). Dobljeni rezultati se obarvajo rumeno. Pregledujemo jih lahko na več načinov:

- predhodna pojavitev niza (modra puščica navzgor);
- naslednja pojavitev niza (modra puščica navzdol);
- poišči vse (zelena puščica navzgor).



Slika 112: Izključno niz

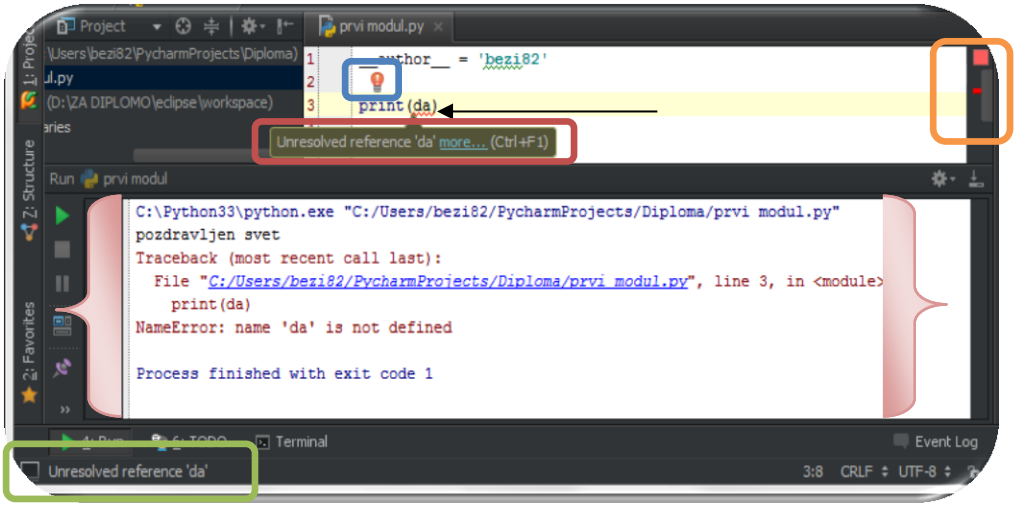


Slika 113: Vse pojavitve niza

Vse pojavitve najdenega niza lahko zamenjamo z novim nizom (*»Replace all«* na Slika 113). Tudi točno določen najden niz lahko zamenjamo z novim nizom (*»Replace«* na isti sliki). Posamezno pojavitev niza lahko izključimo iz zamenjave (*»Exclude«*), tako se bodo vse pojavitve niza ob uporabi *»Replace all«* zamenjale z novim nizom, razen tistih, ki smo jih izključili.

7.3.6 Označevanje napak

PyCharm ponuja številne namige, ki nam omogočajo, da se izognemo napakam v kodi. Ne da bi kodo poganjali, vseskozi vrši analizo napisane kode. Rezultate preverjanja nam poda kot nekakšen semafor na desni strani kode (Slika 114 v oranžnem okvirju). Dokler vidimo zelen kvadratek, vemo, da v kodi nimamo napake.
Kadar se pojavi rumen kvadratek, to pomeni, da je PyCharm v kodi zaznal mesto, kjer bi lahko prišlo pri izvajanju programa do zaustavitve. V primeru sintaktične napake pa se kvadratek obarva rdeče, kar pomeni, da moramo to napako obvezno popraviti.
Prav tako nam na mestu, kjer se je pojavila napaka, kodo vijugasto podčrta (na Slika 114 na to opozarja puščica). Ko se z miško približamo podčrtanemu mestu, se pojavi okno s kratkim opisom napake (Slika 114 v rdečem okvirju). Če je opis prekratek in še vedno ne vemo, kakšno napako je zaznal, imamo možnost *»več«* (*»more«*), ki nam poda daljši opis.
Malo kasneje se pojavi poleg dotičnega mesta žarnica (na Slika 114 modri okvir). Žarnica sveti v rumeni ali rdeči barvi, odvisno od vrste napake. Poleg žarnice se pojavi kazalec, ki nam ponuja nasvet oziroma namig, kako naj napako popravimo, če nanj kliknemo.
Tudi v levem spodnjem kotu vidimo kratek opis napake (na Slika 114 zelen okvir). Če po vseh opozorilih nismo odpravili napake in smo program vseeno pognali, se v oknu za izpis prikaže popoln opis napake (znotraj rdečega oklepaja, Slika 110).



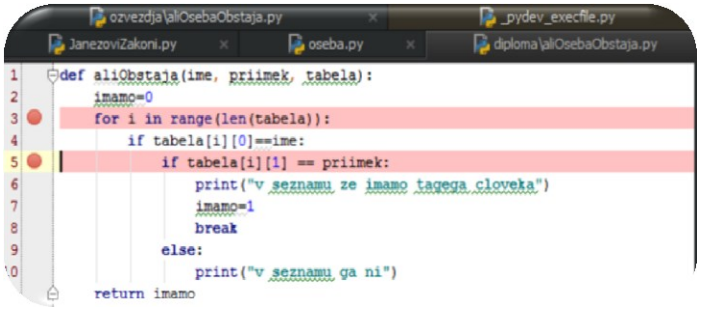
Slika 114: Opozarjanje na napake v PyCharm

7.3.7 Testiranje po sklopih – unittest

Testiranje po sklopih poteka, ravno tako kot v Eclipse, s pomočjo predloge za kreiranje unittest modula. Pravila, ki se jih moramo držati pri pisanju unittest programa, smo spoznali v razdelku 3.7.

7.3.8 Python razhroščevalnik

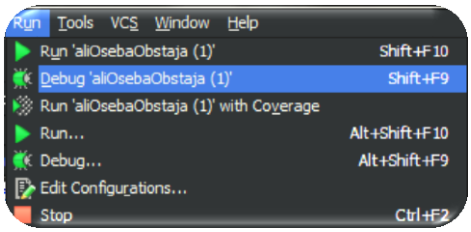
Razhroščevalnik smo spoznali že v razdelku 3.8. Zdaj pa si poglejmo, kako razhroščevalnik deluje v Pycharmu.
Prekinitvene točke postavljamo s klikom desno od številke vrstice, kjer želimo pričeti z natančnejšim pregledom kode (Slika 115). Postavimo lahko poljubno število točk, ki se označijo z rdečo piko in vrstica se obarva rdeče. Točko lahko kadar koli odstranimo s klikom nanjo.



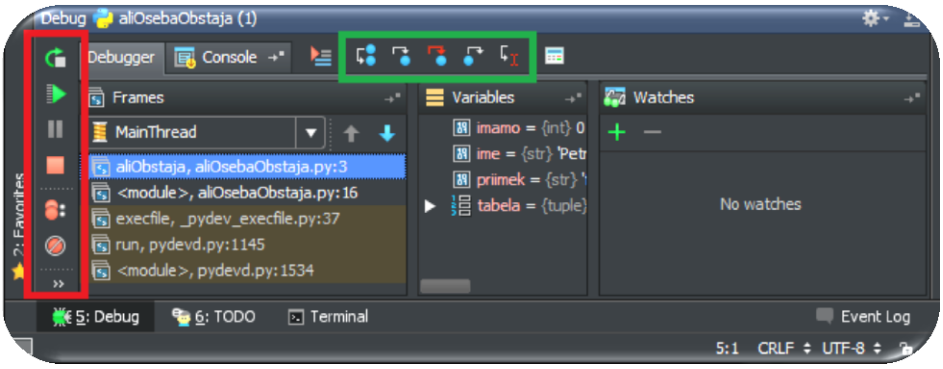
Slika 115: Postavitev prekinitvenih točk

Zdaj kliknemo na *»Run«* v opravilni vrstici ter v spustnem seznamu na *»Debug našModul«* (Slika 116). V oknu pod urejevalnikom kode, kjer smo prvotno

opazovali rezultate zagnanih modulov, se odpre okno za razhroščevanje z vsemi orodji (Slika 117).

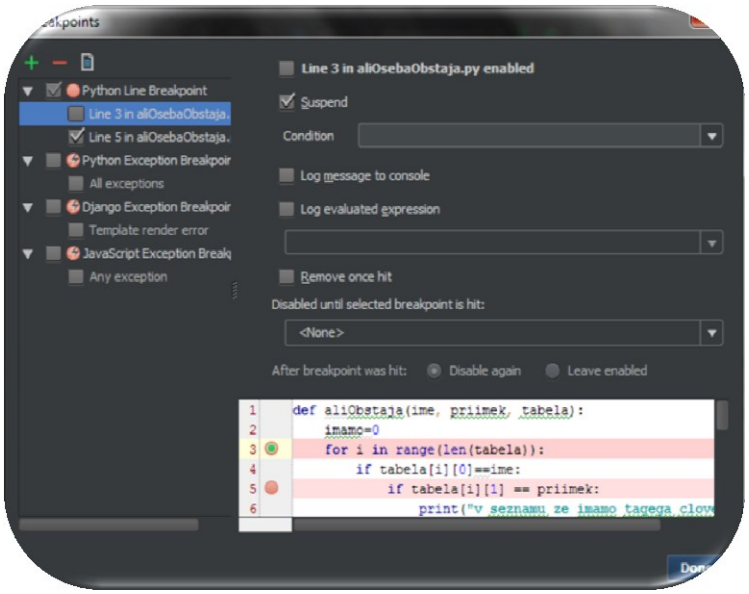


Slika 116: Zagon razhroščevalnika



Slika 117: Razhroščevalnik

Na levem robu razhroščevalnika (na Slika 117, rdeč okvir) imamo osnovna orodja (njihovo delovanje smo spoznali v razdelku 3.8), kot so: *»Rerun«*, *»Stop«*, *»Mute Breakpoints«* in *»View Breakpoints«*. Sledi podrobna razlaga slednje.
Pri kliku na *»View Breakpoints«* se odpre novo okno, v katerem imamo podatke o nastavljenih prekinitvenih točkah (Slika 118). Tu lahko posamezno točko začasno izključimo. To lahko storimo tudi v oknu, ki se pojavi, ko z desnim miškinim gumbom kliknemo na točko. Začasno izključena točka se znotraj rdeče pike obarva zeleno.



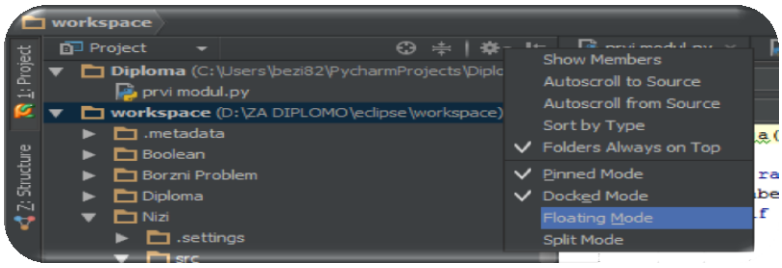
Slika 118: Nastavljanje prekinitvenih točk

Razhroščevalnik se zaustavi na prvi prekinitveni točki, kjer v modro pobarva vrstico v kateri smo. V zavihku »*Variables*« dobimo podatke o tem, katere spremenljivke imamo in kakšne so njihove trenutne vrednosti. Zavihek »*Console*« nam sporoča, da je razhroščevalnik zagnan.
Različne možnosti nadaljevanja izvršujemo z ikonami v orodni vrstici okna za razhroščevanje (Slika 117, zeleni okvir). Možnosti, ki so na voljo, so:
»*Step Over*«, »*Step Into*«, »*Force step into*«, »*Step Out*« in »*Run to Cursor*«. Vse naštete možnosti smo, prav tako kot osnovna orodja, že spoznali v razdelku 3.8.

Razhroščevanje v PyCharmu je boljše od razhroščevanja, ki ga ponuja IDLE zaradi enakih razlogov kot v Eclipse (kar smo napisali v razdelku 6.4.8). PyCharm je tudi boljši od obeh, saj nam ponuja več možnosti za nadzor razhroščevanja, kot primer navedimo možnost »*Run to Cursor*«.

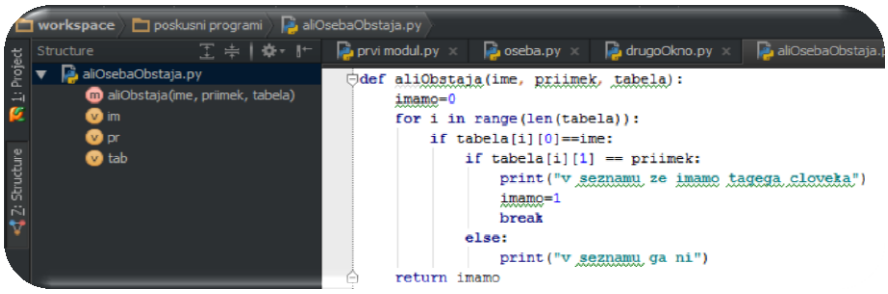
7.3.9 Projektno drevo

Kot smo omenili že pri orodju Eclipse, s pomočjo projektnega drevesa lahko hitro in enostavno poiščemo modul, ki ga želimo.
Projektno drevo v PyCharmu se privzeto nahaja ob levem robu okolja. Lahko pa ga tudi poljubno prestavljamo. To storimo tako, da kliknemo na »*koleščekk*«, prikaže se seznam, na katerem poiščemo »*Floating mode*« (Slika 119).
Za razliko od Eclipse pa ima PyCharm ločena projektno in strukturno drevo.
Pogled projektnega drevesa je prednastavljen na »*project*«, kar pomeni, da imamo pogled na abecedno urejene projekte. Nastavimo pa si ga lahko tudi na »*project files*« in dobimo pogled na abecedno urejene datoteke. Obstaja pa tudi tretja možnost »*problems*«, pri kateri dobimo abecedno ureditev map, v katerih so moduli, ki jih moramo popraviti.



Slika 119: Projektno drevo

Pod ikono za vklop in izklop projektnega drevesa imamo ikono za vklop in izklop strukturnega drevesa (Slika 120). Ta nam za vsak modul prikaže njegovo strukturo (struktura modula so v modulu napisane metode, spremenljivke, razredi). Ko kliknemo na poljubno strukturo, se kurzor pojavi na mestu, kjer smo jo definirali.



Slika 120: Strukturno drevo

8 ZAKLJUČEK

V diplomski nalogi smo spoznali številna okolja in nekatera osnovna orodja, ki jih le-ta ponujajo. Pobližje smo si ogledali tri okolja, kako izgledajo, kako delujejo, katera orodja imajo in kako ta orodja uporabljamo.
Pregled okolij in njihovih orodij smo naredili z namenom, da bi ugotovili, katero okolje je najboljše, najbolj enostavno in hitro za programiranje v programskem jeziku Python.
Enovitega odgovora pravzaprav ni. Veliko je okolij, ki imajo vsa opazovana orodja, a pri vsakem lahko najdemo pomanjkljivost. Pozabili smo tudi na neznanko, za katero ni univerzalne rešitve. Ta neznanka je človeški faktor, ki ima odločilen vpliv na izbiro razvojnega okolja. Vsak programer ima, zaradi svojega stila programiranja, namreč svoj pogled na to, katera orodja mora okolje obvezno imeti, katera orodja so zaželena in katerih orodij ne potrebuje.

9 VIRI IN LITERATURA

[1] Burd, B., *Eclipse for dummies*. Canada: Wiley Publishing, Inc., 2005. The instant notes series. ISBN: 0-7645-7470-1.

[2] Summerfield, M., *Programming in Python 3: A Complete Introduction to the Python Language*. USA: Addison-Wesley, Pearson Education, Inc:; 2008. ISBN-13: 978-0-13-712929-4.

[3] Razvojna okolja za programiranje v Pythonu
https://wiki.python.org/moin/PythonEditors
zadnji obisk 18.7.2014

[4] Razvojno okolje Boa Constructor
http://boa-constructor.sourceforge.net/
zadnji obisk 18.7.2014

[5] Razvojno okolje Eclipse
http://www.eclipse.org/
zadnji obisk 18.7.2014

[6] Dodatne nastavitve za Eclipse
http://www.rose-hulman.edu/Users/faculty/young/CS-Classes/resources/Eclipse/shells-in-pydev.htm
Zadnji obisk 18.7.2014

[7] Python IDE za Eclipse
http://pydev.org/
zadnji obisk 18.7.2014

[8] Razvojno okolje Editra
http://editra.org/
zadnji obisk 18.7.2014

[9] Razvojno okolje Geany
http://www.geany.org/
zadnji obisk 18.7.2014

[10] Programski paket Python
https://www.python.org/
zadnji obisk 18.7.2014

[11] Razvojno okoje NetBeans
http://netbeans.org/
zadnji obisk 18.7.2014

[12] Razvojno okolje PyCharm
http://www.jetbrains.com/pycharm/
zadnji obisk 18.7.2014

[13] Razvojno okolje PyScripter
https://code.google.com/p/pyscripter/
zadnji obisk 18.7.2014

[14] Razvojno okolje Spyder
https://code.google.com/p/spyderlib/
zadnji obisk 18.7.2014

[15] Razvojno okolje IEP
http://www.iep-project.org
zadnji obisk 18.7.2014