Математички факултет
Универзитет у Београду

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Мастер рад

Погодности увођења програмског језика Пајтон у систем образовања

израдио: Гојко Гојковић

ментор: доц. др Мирослав Марић

Београд, 2015

Аутор: Гојко Гојковић 1019/2013

Наслов рада: Погодности увођења програмског језика Пајтон у систем образовања

Ментор: доц. др Мирослав Марић
Математички факултет
Универзитет у Београду

Чланови комисије: проф. др Душан Тошић
Математички факултет
Универзитет у Београду

проф. др Владимир Филиповић
Математички факултет
Универзитет у Београду

Датум: _____

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Садржај:

Увод 5
Упоређивање Пајтона са осталим програмским језицима 6
Програмски језик Паскал (Pascal). 6
Програмски језик Вижуал Бејсик (Visual Basic) 7
Програмски језик C 8
Програмски језици C++/C# 9
Први кораци у програмирању у Пајтону и другим програмским језицима 9
Увођење основних појмова програмирања у Пајтону 13
Основне операције 13
Појам примитивних типова 15
Декларација променљивих 17
Блокови наредби и опсег променљивих 20
Сложени типови 21
Петље и контрола тока 27
Скуповне листе 30
Функције 32
Модули и уграђене функције 32
Функција main() 34
Писање функција за поновну употребу 35
Дефинисање функција 36
Ламбда функције 39
Класе 42
Модули за обраду и приказ података 45
Пакет за нумеричку обраду података 45
Проширен пакет за обраду података 48
Пакет за графички приказ података 49
Имплементација алгоритама у Пајтону 52
Бинарна претрага 53
Сортирање пребројавањем 55

Сортирање обрнутим вишеструким разврставањем 57
Грахамов алгоритам 62
Интерполација 65
Закључак 70
Литература: 72

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Увод

Програмски језик Пајтон (Python) је настао крајем 80-их година двадесетог века, као надоградња на програмски језик Еј-Би-Си (ABC). Основна идеја аутора Пајтона, Хида ван Росума (Guido van Rossum) је била да створи скриптни језик који је једноставан за учење и може лако да се унапређује [1]. Главни циљ овог рада је да покаже да је Пајтон заиста лак програмерски језик за учење и да је стога погодан за увођење у систем образовања.

Брзина којом почетник може да усвоји основна знања из програмирања, уз минимални број линија кода, чини овај програмски језик погодним за особе које се први пут срећу са програмирањем. Опширна библиотека готових модула уједно га чине компетентним за конструисање много захтевних пројеката у разним сферама информатике.

У сврху анализе предности и мана Пајтона, упоредно ће бити анализирани и актуелни програмски језици у образовању. Први део рада се бави предностима које програмски језик Пајтон има као језик за почетнике, посебно у основним и средњим школама, док су другом делу разматране предности за студенте, нарочито ако су са овим језиком упознати у претходном образовању. Преко упоређивања Пајтона са псеудокодом, програмским језиком C и Матлабом (Matlab [20]) на конкретним примерима, видеће се у којој мери може да се унапреди образовање увођењем програмског језика Пајтон.

Формално говорећи, Пајтон је интерпретерски, објектно орјентисани програмски језик високог нивоа са интуитивним програмским кодом, који подржава процедурални, објектно оријентисани и функционални програмерски стил.

У сврху истраживања за овај рад коришћена је верзија Пајтона 3.4. Појава Пајтона 3 је донела многе новине у синтакси, али је задржана компатибилност са старијим верзијама, тако да скоро сви програмски кодови наведени у овом раду могу да се покрену и у верзији 2.7. Евентуална некомпатибилност може да се јави при увозу модула (класа или пакета). Овакви случајеви су ретки и биће посебно истакнути у примерима.

[1] Venner, Bill, The Making of Python: A Conversation with Guido van Rossum, Part I, januar 2003, http://www.artima.com/intv/pythonP.html (приступљено: јун 2015)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Упоређивање Пајтона са осталим програмским језицима

Једно од битних питања система образовања у области информатике јесте који програмски језик учити као први језик. Такав језик би требало да задовољава неколико критеријума, од којих би могли да се издвоје следећи: једноставност учења, развијање „програмерског“ начина размишљања, добра основа за напредније програмирање, као и брз прелазак на друге програмске језике. Уз то је за почетнике од нарочите важности задовољавање општег критеријум доброг програмског језика, што подразумева могућност брзе и лаке имплементације проблема који треба решити. Овај последњи захтев подразумева и ефикасно развијање алгоритама из теоријских модела.

Посматрано са становишта популарности програмских језика у софтверској индустрији, Пајтон је високо котиран. По истраживању TIOBE Index софтверске компаније, Пајтон се у марту 2015. године налазио на 8 месту најпопуларнијих језика[2]. Стога је извесно да је Пајтон погодан језик за почетнике и за ученике који желе да наставе да се баве програмирањем и после завршеног школовања.

Пре упуштања у конкретно поређење програмерских кодова, биће размотрено шта би актуелни програмски језици у образовању могли да пруже ученицима.

Програмски језик Паскал (Pascal).

Као језик дизајниран управо за учење програмирања, Паскал је врло често коришћен језик за почетнике који даје одличну основу за даље програмирање, концизан је и строг, али и поред тога релативно једноставан за прихватање. Са друге стране, Паскал има врло ограничену стварну примену и данас ће се ретко ко упустити у озбиљне пројекте или макар делове пројеката написаних на овом језику. Објектно орјентисане верзије паскала као што је Објектни Паскал у Делфију проширују могућности овог програмског језика, али са још увек ограниченом применом изван учионице. Осим тога, ове варијанте језика захтевају додатни труд образлагања концепта објектно оријентисаног стила што је значајна препрека за увод у програмирање. На тему потенцијала Паскала Брајан Керинган (Brian Kernighan) каже: „Сматрам да је погрешно користити Паскал ван оквира своје почетне намене. У својој основи Паскал је језик за играње, погодан је за учење, али не и у сврху правог програмирања“.[3]

[2] TIOBE Index http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[3] Kernighan, Brian W. *Why Pascal is Not My Favorite Programming Language*, AT&T Bell Laboratories, април 1981, http://www.lysator.liu.se/c/bwk-on-pascal.html (приступљено 2014)

Додатна препрека за проширивање знања о програмирању помоћу Паскала је мноштво варијација синтаксе на различитим платформама. Ово нарочито може бити збуњујуће за почетнике када се узме у обзир и строгост кода овог програмског језика.

Иако строге синтаксе, при писању програма у Паскалу не постоји разлика између имена променљивих писаних мали и великим словима. Могуће је да су дизајнери ово увели намерно како би смањили број грешака, те подстакли почетнике да брже напишу кôд, али проблем је што овакав запис може да створи лошу навику за касније програмирање. Скоро сви програмски језици данас праве ову врсту разлике из мноштва разлога.

Још једно од наслеђа Паскала је и раздвајање функција од процедура, односно функција које не враћају вредност. У Пајтону све функције имају скривену (динамичну) повратну вредност, па уколико функција не враћа вредност повратна вредност је „празна“ (`None`), слично као `void` у програмским језицима C,C++, Јава и другим. Иако у коду Пајтона постоји разлика између функција и метода, односно самосталних функција и функција унутар класа, ова два појма су структурално иста и у оквиру својства класе, корисник не може да осети неку значајну разлику између ова два појма при писању програма.

Паскал пружа ограничене могућности за програмирање ниског нивоа. Самим тим пружа и мало могућности за усвајање знања о начину на који рачунар чува и обрађује податке на машинском нивоу.

Програмски језик Вижуал Бејсик (Visual Basic)

Вижуал Бејсик (у даљем тексту ВБ) је такође језик дизајниран као језик за почетнике за програмирање и наставља да буде заступљен у образовању. Као директан наследник Бејсика (BASIC) носи са собом неколико значајних побољшања, која би требало да омогуће почетнику брзо савладавање програмирања, од којих је најистакнутији графички кориснички интерфејс (GUI), са могућношћу коришћења готових графичких елемената и њиховог укључивања у програм, преко методе „превуци и отпусти“ (drag and drop). Овакав приступ при учењу програмирања може бити привлачнији за ученике од строгог Паскала, али има и своје мане. Захваљујући елементима који се убацују на форму, уместо писања одговарајућих блокова програмерског кода, ученик заобилази врло важне компоненте програмирања. Ово значи да ће у том процесу учења бити изостављени врло битни мисаони процеси, који су почетнику неопходни да би заиста разумео програмирање и усвојио неопходне рачунарске концепте.

Ипак, овај програмски језик је добар избор за учење у основној школи, јер је на том нивоу довољан да уведе ученика у програмирање, без стварања одбојности према истом. Касније се „погрешан“ начин размишљања ипак може исправити

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

дубљим упуштањем у структуру, било ВБ, било неког другог погодног програмског језика.

Програмски језик Вижуал Бејсик долази у пакету Мајкрософт Вижуал Студија (Microsoft Visual Studio). Употреба овог пакета у школама и за индивидуални рад ученика и студената је бесплатна, али пуна верзија МВС, која се користи у комерцијалне сврхе, се наплаћује.

Треба узети у обзир и да се подршка за ВБ окончала 2005. године и да се даље унапређивање наставља само у оквиру Мајкрософтовог ВБ-а (Visual Basic . NET). Из овог разлога се нека шира употреба овог програмског језика код нових генерација ставља под знак питања. Подразумева се да је већ више година у току миграција на VB .NET, али за сада још увек постоји врло јака подршка одржању ВБ-а, односно враћању подршке за овај језик.

Програмски језик С

Програмски језик С дуги низ година представља чврсту основу за учење било ког другог модерног програмског језика. Његова ефикасност при извршавању и сажетост програмерског кода, учиниле су да се овај програмски језик одржи у мање више истој форми 40 година, што је у пољу информатике изузетно дуг период. Знања из програмирања, које почетник може да усвоји учећи С, су свеобухватна и велика. Оно што се јавља као проблем у методичком смислу, јесте брзина којом се то знање усваја као и оперативне могућности просечног ученика по усвајању тог знања. Са једноставним програмима обично нема много проблема, али чим се дође до мало захтевнијих концепата, ученици се збуњују и обесхрабрују. Усвајање напредних знања из С-а представља значајан проблем, чак и у оквиру високог образовања. Његову неадекватност као првог програмског језика подржава и чињеница да је данас изузетно мало заступљен у основним и средњим школама.

Осим што пружа свеобухватно знање о програмирању, синтакса језика С је добра основа за водеће објектно орјентисане програмске језике С++, С# и Јаву(Java) као и скриптне језике PHP и ЈаваСкрипт (JavaScript).

Важно је напоменути да је најчешћи Пајтонов интерпретер ЦеПајтон (CPython) написан у овом програмском језику и за боље разумевање рада програма писаних у Пајтону, неопходно је познавање С-а. Штавише, за разумевање структуре најосновнијих појмова, као што су листе, које у Пајтону замењују обичне низове, неопходно је и разумевање саме структуре података листа и њихове имплементације у програмском језику С. Међутим, почетницима у програмирању овакав концепт је сувише далек и њихово програмерско умеће на нивоу основне или средње школе не би требало да буде значајно осиромашено услед ове потешкоће. Творци Пајтона су вешто сакрили сложене типове од програмера, чиме се оставља места за касније образлагање. Са друге стране, када се говори о

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

високом образовању, програмски језик C је неопходан за разумевање суштине Пајтона.

Програмски језици C++/C#

Последњих година јавља се тенденција увођења програмских језика C++ и C# у школски систем, без пружања одговарајућег предзнања из C-а. Графички кориснички интерфејс, чија употреба је устаљена у школама при учењу програмски језика C++ и C#, се користи као „пречица“ ка радној верзији програма. На овај начин се у великој мери заобилазе основе објектно орјентисаног програмирања, па касније употпуњавање знања захтева већи труд од ученика. Овакав приступ је погодан само за мотивисање ученика, али увек постоји могућност да та мотивација нагло спласне онда када ученици морају да употребе неграфичке методе за употпуњавање програма. Са друге стране, упознавање са овим програмским језицима, преко неког процедуралног језика, захтева знатно више радних сати и то је често неизводљиво у оквиру неспецијализованих школа.

Једна од главних предности програмског језика Пајтон је што његова парадигма подразумева и процедурално и објектно орјентисано програмирање. Са становишта програмера, Пајтон пружа могућност процедуралног програмирања све до тренутка када се јави потреба за увођењем класа. Чињеница да су чак и примитивни типови заправо објекти,је сакривена од почетника, све док предавач не процени да је ученик спреман за такву врсту знања.

Први кораци у програмирању у Пајтону и другим програмским језицима

У овој секцији разматраће се како се ученик упознаје са Пајтоном по први пут и колико брзо може да се осети слободним у писању на овом језику.

Како би ученици могли да утврде знање стечено на часу преко домаћих задатака или групних пројеката, битно је да се алат за програмирање инсталира једноставно и да буде бесплатан. Инсталација интерпретера и пратећег едитора у Пајтону у потпуности испуњава овај критеријум: скидањем и покретањем инсталационог фајла са званичног сајта[4], добија се целокупни пакет неопходан да би се почело са програмирањем. Није потребан ручни унос путање системских променљивих, нити инсталација било ког додатног софтвера. На овај начин ученик не мора да троши време на успутне појединости и може директно да се

[4] https://www.python.org/

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

сконцентрише на задатке програмирања, што је значајан подстрек за сваког почетника.

При покретању програма компилација и покретање се обављају секвенцијално, односно ученици не морају одвојено да праве датотеку за покретање да би видели резултат свог рада.

Едитор за Пајтон омогућава и писање краћих, најчешће једнолинијских, програма директно преко интерактивне конзоле (Python Shell) и овакав приступ је на почетку врло практичан. Писање преко конзоле се не разликује значајно од писања у одвојеном едитору, само што је излаз одмах испод кода програма. У наставку следи пример првог програма, написаног директно преко конзоле.

```
print ("Zdravo! Uspešno je izvršen prvi program")
>>>Zdravo! Uspešno je izvršen prvi program
```

Да би се написала једна линија програма, заиста је потребна само једна линија, за разлику од других језика за почетнике, који сви захтевају да се компајлеру проследе и неке додатне информације. Уопштено говорећи, број линија кода у Пајтону је знатно мањи него у другим популарним програмским језицима.[5]

Како је то приказано у овом примеру, називи променљивих и вредности стрингова нису ограничени само на ASCII карактере. Већина светских писама, укључујући и ћирилично, могу се користити равноправно без потребе за додатним подешавањима.

Исти уводни програм у Паскалу би могао да има следећи облик:

```
program Hello;
begin
  writeln ('Zdravo! Uspesno je izvrsen prvi program ');
  readln
end.
```

Програм у Паскалу је доста комплекснији и форсира почетника да на самом почетку учи доста додатних елемената програмског језика. На крају тела програма често је неопходна и помало збуњујућа команда `readln` која има за сврху да омогући кориснику да заиста види излаз, односно спречава аутоматско затварање прозора по завршетку програма.

У ВБ овај једноставан програм би изгледао овако:

[5] Summerfield, Mark, *Rapid GUI Programing with Python and QT*, Prentice Hall, септмебар 2007

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
Public Module Hello
  Sub Main()
    Console.WriteLine("Zdravo! Uspesno je izvrsen prvi program")
  End Sub
End Module
```

Слично као и код Паскала, од ученика се захтева додатно предзнање да би могли да разумеју како се извршава овај програм. ВБ је додатно проблематичан јер у самом старту захтева објашњавање објектно оријентисане парадигме. Већина наставника у школи заобилази овај напредни концепт уз опаску „тако треба“. Чак и ако се после вишемесечног учења предавач врати назад на анализу програмерског кода и објашњавање појма објекта, ефекат разумевања је тиме умањен. Сакривајући напредне концепте, творци Пајтона су дали могућност предавачу да остави појашњења за погодан тренутак, односно онда када се јави права потреба за њима. На крају можемо примети да синтакса ВБ-а захтева два блока наредби, што изискује додатна објашњења.

Ова проста апликација би у програмском језику C би могла да има следећи облик:

```
#include <stdio.h>
main(){
        printf("Zdravo! Uspesno je izvrsen prvi program");
}
```

Знатно краће него претходна два примера, али опет захтева од ученика да усвоји појмове библиотека и значење `main()` функције, па чак и самог концепта функција. Осим тога, како је већ поменуто раније, проблеми са учењем овог програмског језика настају касније при писању комплекснијих програма. Само мало комплексније команде, као што је исписивање вредности промењљиве захтевају компликованију синтаксу, за чије пуно разумевање је пожељно предзнање о адресама промењљивих.

На крају овог дела биће размотрено шта је потребно објаснити ученику при првим корацима у раду са Пајтоном и колико је овакав поједностављен приступ оправдан. С обзиром на једноставност кода, једини нови појмови су: излазна конзола, уграђена функција `print` и аргумент функције.

Питање које се овде отвара јесте да ли ученик после овог првог задатка може да схвати шта је заправо његов програм. Пошто Пајтон преко интерактивне конзоле заобилази и датотеку изворног кода и извршну верзију програма, ученици не могу да виде свој програм као независан део од конзоле. Са становишта других програмских језика ово може изгледати као озбиљна мана која

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

може да наведе ученике на погрешан пут. Није једноставно дати одговор на питање да ли овај привид има реалне импликације, односно да ли заиста удаљава ученике од програмирања. Ако је суштина учења програмирања у могућности конкретизовања апстрактних задатака које желимо да извршимо, односно у реализацији „мисаоних модела“, онда Пајтон у потпуности испуњава овај задатак. У крајњој линији чак и када имамо извршну верзију програма, она је у великој мери део друге целине, на првом месту оперативног система, али у извесној мери и едитора, на коме се обавља писање, односно преглед програма.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Увођење основних појмова програмирања у Пајтону

Упознавање ученика са програмирањем по први пут често носи са собом пуно потешкоћа за наставника. У овом одељку ће бити детаљно прегледани неки од нових појмова, које би ученик требао да усвоји како би савладао основе програмирања у Пајтону. Паралелно ће бити поменуте и предности и мане у односу на увођење истих појмова у другим програмским језицима.

Основне операције

Операције у Пајтону се у великој мери поклапају са осталим програмским језицима. Наместо пуне листе, овде ће бити изнете једино разлике у односу на популарне језике у школама.

Операције сабирања и множења имају нешто ширу примену него што је то уобичајено. Наиме, операција сабирања се може користи за спајање низова (листа) и стрингова, док се операција множења може користити за дуплирање елемената листе. На пример за формирање нула-матрице формата 4×4 може се користити следећа синтакса:

```
M = [[0]*4]*4
```

Оператор % остатка при дељењу, као честа операција при учењу програмирања, у Пајтону је проширена тако да буде математички конзистентна са негативним бројевима. Тако да израз `-1 % 2` даје тачан резултат 1, док Паскал, C, C++ и Јава дају нетачан резултат -1. Ово је погодно при тражењу остатка при дељењу израза за који није сигурно да ли ће увек узети позитивну вредност.

Пајтон има и посебну ознаку за целобројно дељење //. Ова операција је намењена првенствено за дељење целих бројева, али се може применити и на реалне бројеве. Резултат је и даље реалан само се иза децимале уписује нула, тако да на пример 5.9//2 даје резултат 2.0. Ова особина операције целобројног дељења има посебну сврху при писању у Пајтону, пошто су све промењљиве динамичног типа (више о томе у следећем одељку), па је теже само по типу промењљиве лако предвидети резултат „обичног“ дељења.

У синтакси Пајтона важи опште правило да треба да постоји само један начин да се опише неки исказ[6]. Ово није у потпуности тачно, али се тим иза овог језика

[6] У оригиналу: There should be one-- and preferably only one --obvious way to do it, The Zen of Python, Tim Peters. Пуна листа идеја водиља за Пајтон је доступна директно из Пајтон конзоле преко команде `import this`

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

потрудио да се то оствари у великој мери. Тако да је, на пример, из C-а задржана операција додавања вредности променљивој `a+=1`, али не и операција „повећања за један“ `a++`.

Ради лакше читљивости, логички оператори су изражени енглеским речима „and“, „or“ и „not“, док су бинарни оператори исти као у програмском језику C. Слично као у програмском језику Јава, резултат логичких поређења може да узме булове вредности „True“ и „False“. Логичка поређења са нумеричким вредностима 1 и 0, као што је то у програмском језику C, су и даље дозвољена.

Операције поређења > и < су проширене, омогућавајући нама природно математичко поређење три променљиве a<b<c уместо повезивања два израза `a<b and b<c`.

Пајтон има и оператор степеновања који се означава са две звездице на следећи начин `a**b`. Овај оператор се понаша слично као оператор „^“ из ВБ. Вредност експонента није ограничена једино на природне бројеве, већ се употребом негативних и рационалних вредности, ова операција може користити за скраћено израчунавање реципрочне вредности и корена бројева.

Врло једноставан кôд омогућава Пајтону да функционише и као калкулатор. Испод је наведено три примера једноставне употребе оператора степеновања.

```
2**7
>>>128
9**0.5
>>>3.0
82**-1
>>>0.012195121951219513
```

Исти задатак у програмском језику C и Паскалу би захтевао не само увођење одговарајућих библиотека са функцијама, већ и пропратне елементе сваког другог програма. ВБ има уграђену операцију степеновања, али је цео кôд ипак знатно дужи. Пајтон у овом случај ствара привид да није коришћена чак ни `print` функција.

Могућности Пајтоновог калкулатора се наравно не заустављају овде, већ је могуће формулисати и комплексне математичке изразе, али је управо на овим једноставним примерима наглашена једноставност са којом се извршава посао калкулатора.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Појам примитивних типова

Строго говорећи програмски језик Пајтон нема праве примитивне типове. Сви типови, укључујући и примитивне, су заправо објекти, слично као класе омотача (wrapper) примитивних типова у Јави. Битна разлика је у томе што се инстанце класа променљивих креирају аутоматски, али је конструктор могуће позвати и директно. Важна последица овакве структуре јесте да је величина додељене вредности променљивој практично неограничена. Наравно, постоји почетна резервисана меморија која зависи од рачунарског система на коме се програм извршава. У случају променљиве типа `int` величину резервисане меморије можемо добити на следећи начин:

```
import sys
sys.getsizeof(int())
>>> 24
```

Што значи да је почетна величина резервисане меморије за променљиву типа `int` на тестираном рачунару 24 бита. Функција заправо проверава величину инстанце класе `int`. По потреби меморија додељена некој променљивој се проширује аутоматски и практично је ограничена само хардвером, тачније величином RAM меморије. У теорији ово може да створи одређене проблеме при извршавању програма, али је капацитет данашњих рачунара толико велики да се проблеми овог типа јављају изузетно ретко. Неефикасност Пајтона је једна од главних мана овог програмског језика.

По устаљеној номенклатури актуелних програмских језика, могу се издвојити следећи примитивни типови: `string`, `int`, `float` и `boolean`. Пошто нема ограничења за вредности које нека променљива може да чува, нема ни потребе за декларисањем проширених или „штедљивих“ променљивих типа `longint`, `double`, `byte`, `char` и `short`. Посебно се може разматрати тип `char` пошто он заправо постоји једино као једноелементна ниска. Овде нарочито долази до изражаја беспотребно коришћење простора јер `sys.getsizeof(str())` на тестираном рачунару даје вредност од 51, што значи да је у случају стринга са једним елементом у најбољем случају неискоришћено чак 47 бајтова. Међутим, сваки следећи ASCII карактер у ниски заузима тачно један бајт, па су тако стрингови већи само за 50 бајтова од истих у програмском језику C. Подразумева се да ће у овом тексту бити коришћен појам карактер уместо појма једноелементна ниска.

Овако уведени примитивни типови имају додатан повод за издвајање од осталих објеката јер су, поред торки (уређених н-торки), о којима ће бити речи касније, једини непроменљиви типови.

Посебан примитивни тип у Пајтону су комплексни бројеви. Они се представљају природно: преко збира реалног и имагинарног дела, при чему се иза имагинарне нумеричке вредности додаје карактер j, што је у складу са

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

математичким записом комплексног броја. Реалној и имагинарној вредности броја може се приступити преко уграђених функција `.real` и `.imag`.

```
z=1+2j
z**2
>>>(-3+4j)
```

Булове променљиве могу узимати вредности `True` или `False`. Међутим, задржана је и проширена провера истинитости из програмског језика C, па се тако све нула вредности „`0`“, „`0.0`“ и „`0j`“ осталих примитивних типова тумаче као `False` при провери тачности. Осим тога и празна ниска, низ или речник заједно са `None` кључном речи су такође `False`. Све остале вредности се тумаче као `True`.

Ниске се могу обележавати или обичним, дуплим, наводницима или једноструким наводницима. Из методичких разлога би било добро држати се конзистенције у писању програма. Једно од могућих решења је да се наставник придржава синтаксе програмског језика C, односно да се користе апострофи за карактере и наводници за „праве“ ниске. Треба имати на уму да Пајтон исписује стрингове на излаз без наводника ако су самостални, или са једноструким наводницима уколико је потребно разликовати ниску од назива променљиве, као што је то случај унутар низова или других типова.

```
print("pas")
print(["papagaj"])
>>>pas
['papagaj']
```

Узимајући у обзир претходно излагање, јасно је да нема много смисла учити почетнике о величини резервисане меморије за променљиве. Довољно је само поменути да Пајтон за сваку променљиву заузима део меморије и проширује је по потреби. Ово не би требало да буде озбиљан методички проблем јер, уколико то није неопходно за писање програма, за ученике није неопходно да разумеју на који начин се тачно записују подаци у рачунару. Са друге стране пошто је, како ћемо видети у следећем одељку, ово скривено од програмера, нема никакве бојазни да би касније детаљно образлагање додељивање меморије могло да створи забуну.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Декларација променљивих

Врло битан појам за усвајање на почетку упознавања са програмирањем јесу променљиве, на првом месту њихова декларација и додела вредности. Сама додела вредности представља збуњујући чинилац на почетку јер, водећи се за математичким појмом једнакости, израз `a=5` ученици тумаче или као резултат или као проверу тачности тог израза.

Скоро сви програмски језици користе декларацију променљивих, па је усвајање значења доделе олакшано „уводом“ преко декларације. Наиме, када се декларише променљива `int a,` уз објашњење да се тиме обезбеђује одређено место у меморији за уписивање вредности за природни број, каснији израз доделе вредности има више смисла [9].

У програмском језику Пајтон не постоји експлицитно декларисање променљивих. Први пут када се додели вредност некој променљивој она је декларисана. У методолошком смислу ово представља известан проблем из горе наведеног разлога. Међутим, врло брзо, када ученик савлада ове елементарне појмове доделе вредности, погодности „аутоматског декларисања“ долазе до изражаја пошто драстично смањују комплексност кода. Ово такође поједностављује и организовање кодирања унапред, што у методичком смислу има и лоше и добре стране. Негативно може бити то што се ученик осећа исувише лагодно у организацији структуре програма и не осећа у потпуности строгост коју изискује програмирање. Са друге стране, смањена комплексност кода оставља више простора за рад на суштински битним концептима програмирања који могу да доведу до функционалног програма.

Имена променљивих не смеју садржати размак и по конвенцији се за раздвајање речи користи доња црта. Имена не могу да почињу бројем, али су неки од осталих карактера дозвољени. По конвенцији, променљиве које нису објекти почињу малим словом. Могуће је коришћење разноразних писама за називе променљивих, укључујући и ћирилицу, али се овакав приступ не препоручује, првенствено због забуна које могу да настану код карактера који имају исту визуелну репрезентацију. Тако би, на пример, било могуће декларисати две променљиве `a`, једну на ћирилици другу на латиници које би са стране компајлера биле различите иако изгледају идентично.

```
a = 5
a = 1
print(a,a)
>>> 1 5
```

Променљиве програмског језика Пајтон су динамичног типа. Осим уобичајених вредности примитивних и сложених типова, променљиве могу да узимају и вредност целе функције или модула, о чему ће бити речи у одељку о

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

функцијама. Пошто нема декларисања променљивих, то значи да некадашњој `int` промењивој може да се додели одмах `string` вредност или било која друга. Оваква пракса се не подржава, те би ваљало ученицима указати на ову чињеницу, односно од почетка их наводити да не мешају типове променљивих, јер то врло брзо може да створи нечитљив кôд. Следећи једноставан пример илуструје један вид мешања променљивих које би требало избегавати.

```
  a = 6
  print (a)
  >>> 6
  a = "Ja nisam broj i ne treba me dodeljivati brojčanoj
promenljivoj"
  print (a)
  >>> Ja nisam broj i ne treba me dodeljivati brojčanoj promenljivoj
```

Осим класичне декларације, Пајтон омогућава и истовремено додељивање почетних вредности за више променљивих, што често може олакшати писање кода.

```
  a,b,c = 1,2,3
```

Овакав запис подсећа на додељивање вредности уређеним паровима (тројкама, n-торкама). Како би се овај скраћени запис приближио ученицима, може се користити и израз са заградама.

```
  (a,b,c) = (1,2,3)
```

Пајтон подржава спајање две или више ниски у једну преко оператора „+“, што је врло згодно у појединим ситуацијама, али може и да створи одређене проблеме, нарочито када се упари са динамичним променљивим. Видећемо пример класичне грешке коју компајлер не препознаје као такву, али даје нетачан резултат. У овом примеру користи се функција `input`, која избацује на излаз стринг из аргумента и враћа вредност која се уноси са улаза. Може се примети и да се ове две често коришћене команде обављају у једном реду, док су за Паскал и C неопходне две.

```
  a = input("Unesite prvi sabirak: ")
  b = input("Unesite drugi sabirak: ")
  print("Zbir brojeva "+a+" i "+b+" je "+(a+b))
  >>>
  Unesite prvi sabirak: 4
  Unesite drugi sabirak: 1
  Zbir brojeva  4  i  1  je  41
```

Суштинска грешка овог програма је што је занемарена чињеница да `input()` функција увек враћа ниску, у овом случају карактер. Овај проблем се јавља јер промењиве немају унапред предодређен тип, па компајлер не проналази грешку

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

при додељивању карактера уместо броја. Овај конкретан проблем може да се реши пребацивањем улазне вредности у природни број, слично као што би се то урадило у другим програмским језицима. Преко операције за сабирање не може да се сабира стринг са бројем, па је потребно извршити још једно пребацивање после сабирања. Боље решење би била употреба особине `print` функције да прима више аргумената, чиме би задња линија кода изгледала овако.

```
print("Zbir brojeva",a,"i",b,"je",a+b)
```

Овакав кôд је читљивији и добро је водити се мање збуњујућим приступом, те сабирање користити само за бројчане променљиве унутар `print` функције. Осим тога, за разлику од спајања преко +, повезивање унутар ове функције, преко зареза, аутоматски ставља размаке између два аргумента на излаз, што је још једна олакшавајућа околност.

За потребе спајања две ниске у једну, у сврху додељивања трећој, операција сабирања је ипак погодна. Како елементи ниске, односно карактери, нису променљиви директно (immutable), морају се чувати у засебној варијабли. Ово значи да нису дозвољене промене типа `"абракадабра"[0]="п"` При чему је регуларно вршити додавање на следећи начин:

```
izraz= "Programski jezik"
izraz+=" Pajton"
print(izraz)
>>>Programski jezik Pajton
```

Овде је јасно да променљива `izraz`, после додавања, показује заправо на други стринг који је добијен уз помоћ првог.

Функција `print` пружа могућност исписивања вредности променљивих унутар текста за излаз на "C-овски" начин. Ово је добра методичка пракса јер пружа увод за касније прелажење на друге програмске језике. Додуше, постоји одређена неконзистенција када је овај начин излаза у питању. Стара синтакса је врло слична програмском језику C:

```
cena = 280
print("Đumbir se može naći za %d dinara" % cena)
Đumbir se može naći za 280 dinara
```

Ова синтакса је још увек важећа у верзији Пајтона 3.4.1, али је у плану пуна замена новом и флексибилнијом синтаксом.

```
cena_dj = 280
cena_kr = 70.5
valuta = "dinara"
print("Đumbir se može naći za {0} {2}, krastavac za {1} {2}".format(cena_dj,cena_kr,valuta))
>>> Đumbir se može naći za 280 dinara, krastavac za 70.5 dianra
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Функцији `format` се прослеђује низ са варијаблама из којих се чита вредност, а одређена вредност се унутар текста приказује позивањем индекса, који одговара том низу унутар витичастих заграда. Може се приметити да није неопходно обраћати пажњу на тип променљиве која се убацује унутар текста.

Овакав облик записа се поприлично удаљио од стандардног C-овског записа, али још увек није лош увод за каснији развитак почетника. Осим поменутих приступа, постоје још две варијације као мешавина две горе наведене, које су још увек актуелне и користе се паралелно, али се не разликују довољно да би се посебно наводиле.

Блокови наредби и опсег променљивих

У програмском језику Пајтон не постоји експлицитна ознака за крај реда. Стога се за крај реда користи просто нови ред у текстуалном документу, а за почетак новог блока нова линија увучена за четири празна места у односу на претходну. Пошто овај програмски језик не користи ни експлицитни карактер за крај реда, ни ознаку за почетак и крај блокова наредби, нешто што је у C-у, Паскалу и ВБ-у само естетског типа, у Пајтону је неопходност. Ово је углавном добра пракса која тера ученике да од самог почетка пишу уредан кôд. За неког ко има предзнање из других програмским језика ово може изгледати заморно на почетку, али је за почетнике у Пајтону ово природан приступ.

Скоро сви едитори који су намењени Пајтону аутоматски конвертују притисак дугмета „tab“ на тастатури у четири карактера, што значајно олакшава писање. Поједини едитори попут Нинџа Иде (Ninja IDE) користе центриране тачке (·) за размак, што додатно побољшава видљивост.

Почетак блока функције, класе, `if`, `while` или `for` наредбе се означава двотачком (:) пре новог реда. Почетак функције се означава кључном речи `def`, за којом следе име променљиве и аргументи унутар заграде:

```
def funkcija(argument1, argument2):
```

Како се променљиве декларишу простим додељивањем вредности, уколико се додели вредност неком директно непроменљивом типу, односно броју или торци унутар функције или класе, заправо се декларише нова променљива са истим именом. Овакав приступ може бити донекле збуњујући и захтева додатна објашњења. Да би се обезбедила промена вредности променљиве која важи и изван функције, то се мора посебно нагласити у сваком подблоку преко наредби `global` и `nonlocal`. Следећи пример илуструје који су ефекти коришћења ових наредби.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
def testiranje_opsega():
    def lokalna_izmena():
        niska = "lokalno"
    def nelokalna_izimena():
        nonlocal niska
        niska = "nelokalna"
    def globalna_izmena():
        global niska
        niska = "globalna"
    niska= "test niska"
    lokalna_izmena()
    print("Posle lokalne dodele dobijamo:", niska)
    nelokalna_izimena()
    print("Posle nelokalne dodele dobijamo:", niska)
    globalna_izmena()
    print("Posle globalne dodele dobijamo:", niska)

testiranje_opsega()
print("Van funkcije: ", niska)

>>> Posle lokalne dodele dobijamo: test niska
Posle nelokalne dodele dobijamo: nelokalna
Posle globalne dodele dobijamo: nelokalna
Van funkcije:  globalna
```

Обична замена је строго локалног карактера, док се промена вредности `global` промењиве врши на нивоу датотеке, али не и на нивоу спољне функције. Кључна реч `global` се може користи и унутар функције како би се нагласило да је нова додела (декларација) глобалног типа.

Сложени типови

Програмски језик Пајтон садржи мноштво уграђених типова који се могу користити слично као и типови који су раније окарактерисани као примитивни. За употребу типова листа, речника, торки и скупова није неопходно експлицитно креирање објеката. Сви елементи сложених типова су такође динамични, па могу да садрже мешавину других типова, укључујући инстанце објеката и функције. Декларисање се обавља директно, доделом вредности.

```
niz = [1, 3.14, ["drugi", 4], {"element skupa"}, ("elemnt torke")]
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Овакав приступ скраћује број линија кода и, што је још важније, омогућава ученицима да савладају напредније концепте, без залажења у специфичности различитих структура података.

У оквиру Пајтона се може издвојити посебна врста сложених типова који се називају итерабле (iterable). Класе итерабли све садрже `_iter_()` или `_getitem_()` функцију које им омогућавају да се користе при итерацијама, најважније при `for` петљама и скуповним листама (comperhension list), о којима ће бити речи касније. Најчешћи итерабилни типови у Пајтону су ниске, листе, торке, речници и скупови, при чему редослед елемената у последња два типа није предодређен редоследом додељивања елемената, па у појединим случајевима нису погодни за коришћење при итерацији.

Листе (Lists), односно низови, могу да садрже неограничени број елемената. Ово је постигнуто тиме што су низови у Пајтону заиста повезане листе, те су стога динамичног типа и могуће је проширивати их до граница капацитета рачунара. Ово опет омогућава ученицима да се користе великим низовима без потребе за предзнањем о показивачима. Показивачи у Паскалу се врло ретко обрађују у оквиру школе. Креирање повезаних структура у C-у, неопходних да би се обавиле радње са динамичним низовима, захтевају доста рада и представљају значајан изазов за почетнике.

У оквиру стандардне библиотеке Пајтона постоји мноштво метода за рад над листама, нарочито за брзо уписивање елемената у листу. Поред основних функција, међу којима се могу издвојити `list.append(x)`, `list.insert(i, x)`, `list.remove(x)`, `len()` и `list.reverse()`, сама синтаксе Пајтона има погодне скраћенице за рад са низовима. Већ је поменута употреба множења за додељивање истих елемената листи без експлицитне употребе петљи. На пример, команда `niz = [0]*5` се може употребљавати уместо класичне доделе `niz=[0,0,0,0,0]`, или преко `for` петље.

Елементима листе се приступа на уобичајени начин, навођењем индекса елемента. Поред позитивних индекса, Пајтон подржава и позивање елемената листе са негативним индексима, чиме се приступа елеменитма листе од краја, где је -1 последњи елемент низа.

Уколико се јави потреба за приступањем поднизу листе, Пајтон подржава операцију двотачке „:“. Ознака `[a:b:c]` примењена на низу означава следећу операцију: приступити елементима од a до b са кораком c. При чему се користи индексирање од нуле, па је стога укупан број елемената којима се приступа (b-a)/c, без укључивања b.

Ако се изостави друга двотачка, подразумевани корак је 1. Могуће је изоставити и неки или све од бројева a, b или c при чему су подразумеване вредности редом: почетак низа, крај низа и опет корак 1. За корак је могуће узети и негативну вредност чиме се елементима листе приступа од краја. Следи кратак пример који илуструје ову ситуацију.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
niz=[1,2,3,4]
print (niz[::-1])
>>>[4, 3, 2, 1]
```

Овакав приступ олакшава и измену елемената низа. Да би се сваки други елемент низа заменио неком цифром, довољно је позвати се на горњу синтаксу.

```
niz[1::2]=[5]*len(niz[1::2])
print(niz)
>>> [1, 5, 3, 5]
```

Последња наредба има следећи смисао: подниз са парним индексима листе `niz` заменити листом на десној страни.

За брзо креирање листи које садрже секвенцијалне бројеве користи се функција `range()`. Са истим значењем као и при одсецању, функција прима један, два или три параметра, за почетну вредност, крајњу вредност и корак. Прослеђивање само једног параметра подразумева да је та вредност гранична, док је почетна вредност `0` а корак `1`.

```
print(list(range(10)))
print(list(range(5,20,2)))
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[5, 7, 9, 11, 13, 15, 17, 19]
```

Експлицитно пребацивање у листу при исписивању на излаз је постала неопходност у верзији Пајтона 3, за разлику од верзије 2 где оваква врста конверзије није била неопходна. Ово није од великог значаја пошто се `range()` функција најчешће користи унутар петљи.

Важно је напоменути да, осим класичних низова, односно листи у Пајтону, постоји и посебна класа `arrays`, која има за сврху првенствено прављења строго типских низова и операција над њима. Овакви низови су сличнији низовима програмског језика C и могу да садрже само један тип, који је унапред назначен при декларацији променљиве. Овакви низови су меморијски ефикаснији и могу да форсирају употребу меморијски мање захтевних типова, чији су еквиваленти у C-у `byte`, `int`, `float` и `char`. При овоме се за означавање не-UNICODE карактера користи посебна синтакса.[7]

Торке (Tuples), односно уређење n-торке, су секвенце објеката врло сличне листама са битном разликом што нису промењљиве директно (inmutable), па им недостаје неколико функција које се могу применити на листама. Декларација је проста као за низове, само се уместо угласте заграде користе мале, баш као што се ови објекти означавају у математици.

[7] Пуно објашњење кодова типова као и функција ове класе је доступно на: https://docs.python.org/3/library/array.html

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Појам уређеног пара и уређене тројке је познат ученицима из математике и овај појам могу да прихвате можда и природније од листа. Треба само обратити пажњу да се направи јасна разлика између торки и листа.

Торке могу бити врло погодне у појединим ситуацијама. Ово се може видети у једноставном примеру замене вредности двеју променљивих a и b. Има више начина на које ово може да се реши, али обично захтева минимум три линије кода код програмских језика Паскал, C и ВБ. Један од најприроднијих начина је коришћење додатне променљиве.

```
t=a;
a=b;
 b=t;
```

Уз то, уколико се ова замена обавља више пута, погодно је написати додатну функцију која обавља замену. Пајтон има елегантно решење за овај проблем коришћењем уређеног пара. Кôд за замену два броја би био једнолинијски

```
(a,b)=(b,a)
```

Оваква врста доделе се може обавити и без заграда, али је, из методичких разлога, погодније користити пун облик уређеног пара. За ученике који су упознати са овим појмом оваква замена би требало да буде лакша за усвајање од класичне замене помоћу привремене вредности. Уколико је неки од циљева курса упознавање ученика са додељивањем вредности променљивама и могућношћу замене бројева преко функције, или програмирање ниског нивоа коришћењем битовског оператора ексклузивног или, ово је такође могуће обавити у оквиру синтаксе Пајтона. У ову сврху кодови замена вредности променљивих `a` и `b` би могли да изгледају овако.

Битовски оператор „^”:	Замена унутар функције:

a=2
b=5
a^=b
b^=a
a^=b
	a=2
	b=5
	def zameni():
	  global a,b
	  t=a
	  a=b
	  b=t

Речници (Dictionary) су још једна директно измењива структура за складиштење података у Пајтону и састоје се од парова кључ-елемент. У неким програмским језицима ова структура се зове асоцијативни низ (associative arrays), а у програмским језицима C# и C++, најближа структура је уређени речник (OrderedDictionary).

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Сви кључеви унутар речника морају бити јединствени. Осим тога, кључеви морају бити хешабилни (hashable) односно директно непроменљиви објекти: бројеви, ниске или торке. Декларација је директна и обавља се унутар витичастих заграда, где се прво наводи кључ, а затим елемент иза кључа.

```
recnik = {"ime":"Marija", "starost":14, "ocene":[5,2,4,5]}
```

Променљивама речника се приступа слично као елементима листе, само што се уместо редног броја елемента наводи кључ.

```
recnik["ocene"]
>>>[5, 2, 4, 5]
```

Две основне функције за рад са речницима су `.del()` и `.keys()`, где прва брише кључ или кључеве прослеђене у аргумент, док друга даје вредности свих кључева у речнику.

```
list(recnik.keys())
>>>['ocene', 'ime', 'starost']
```

Употреба речника у оквиру курса програмирања даје пуно могућности наставницима да помогну ученику да развије програмерски начин размишљања, без много залажења у напредне концепте.

Скуп (Sets) је структура која одговара математичким скуповима, где не постоји индексација елемената и сваки елемент у скупу може да се појави само једном. Додавање нових елемената, који већ постоје у скупу, преко функције `add()`, се игнорише. Скуп може да садржи вредности произвољног типа и декларише се набрајањем елемената унутар витичастих заграда.

```
skup = {"paprika", "paradajz", "krastavac"}
```

Пошто скупови подржавају операције уније, разлике, дисјункције и симетричне разлике, погодни су за неке једноставније примене на задатке из математике.

На крају овог одељка о променљивама треба нагласити да све директно измењиве варијабле, у које спадају листе, речници и скупови, прослеђују своју вредност функцији преко референце односно показивача. Експлицитна употреба показивача у Пајтону не постоји. Овакав приступ није нарочито методички погодан, јер захтева додатна објашњења о разлици у записивању вредности променљивих и одређених сложених типова.

Измене унутар функције, које се односе на промену вредности већ постојеће променљиве, биће глобалне, али додела вредности новог објекта (па макар и са истим именом), додељује нову референцу па су такве измене локалне. При томе су сами називи аргумената функције и промене над њима одвојене од променљиве која се мења. Следећи пример приказује ограничења и могућности измена над листама које се прослеђују као аргументи функције.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
lista1=[1,2,3]
lista2=['a','b','c']
def menja_liste(n1,n2):
    n1=[2,3,4]
    n2[:]=['e','f','g']
menja_liste(lista1,lista2)
print(lista1)
print(lista2)
>>>
[1, 2, 3]
['e', 'f', 'g']
```

Како је првој промењивој `n1` унутар функције додељен нови објекат, односно нова референцу на листу, прослеђена листа остаје неизмењена. У другој додели је дошло до замене свих чланова оног објекта на који показује промењива `n2`, па ће измена важити и изван функције.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Петље и контрола тока

У овом одељку разматраће се контрола тока у програмском језику Пајтон, уз стављање акцента на разлику у односу на остале програмске језике.

Наредба `if` у Пајтону се понаша на уобичајени начин. Иза кључне речи се уписује израз чија се тачност проверава, за којим следи двотачка и тело `if` наредбе. Пошто не постоји наредба `case(switch)` која се примењује у Паскалу, С-у и ВБ-у, наместо ње се користи вишеструка употреба `elif` наредби, праћене опционом `else` наредбом која се извршава уколико није испуњен ниједан од услова `elif` наредби.

```
a= int (input("Unesite broj "))
if a<0:
    a=0
    print("Negativna vrednost je promenjena na nulu")
elif a==0:
    print("Uneli ste nulu")
elif a==1:
    print("Uneta je jedinica")
else:
    print ("broj veci od jedan ostaje nepromenjen")
```

Недостатак `case` наредбе се оправдава тенденцијом оснивача Пајтона да нема сувишних начина за обављање истих наредби. И заиста `swich` наредба из програмског језика С може да се замени са вишеструким `else if` наредбама, али ово понекад може да произведе непрегледан кôд, првенствено због увлачења и блоковских заграда.

Имплементација `for` петље у Пајтону се у извесном смислу значајно разликује од актуелних програмских језика у образовању. Уместо експлицитног бројача са ограничењима и кораком, у оквиру Пајтона, итерација се имплементира преко посебног скупа сложених типова који се назива итерабле. Овај тип `for` петљи се јавља као алтернативни начин итерације и у програмским језицима Јава и ЈаваСкрипт.

Најчешће итерабле у Пајтону су листе, ниске и речници.

```
for broj in [2,3,4]:
print (broj)
>>>2
3
4
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    recnik ={"ime":"Marija", "starost":24, "ocene":[9,6,7,8]}
for kljuc in recnik:
    print(kljuc, recnik[kljuc])
>>>
starost 24
ime Marija
ocene [9, 6, 7, 8]
```

Као што је раније поменуто, редослед којим се узимају елементи из речника може да буде другачији од редоследа којим се уносе, на шта треба обраћати пажњу.

Уместо једноструких вредности за промењљиву у коју се уписује појам, могуће је користити и више вредности одједном, које редом пролазе кроз више низова истовремено коришћењем `zip` наредбе. Петља се зауставља код краће од две итерабле.

```
for broj,rec in zip(range(20,30),["kisa","sneg","grad"]):
    print(broj,rec)
>>>
20 kisa
21 sneg
22 grad
```

За нумерисање елемената низа кроз који се пролази користи се `enumerate()` функција:

```
for indeks,rec in enumerate(["kisa","sneg","grad"]):
    print(indeks,rec)
>>>
0 kisa
1 sneg
2 grad
```

Осим итерације кроз низ, `for` петља у Пајтону поседује још једну специфичност: опциону `else` грану. Тело ове гране се извршава уколико је успешно извршена итерација последњег члана низа.

По потреби, уместо промењљиве у коју се уписује вредност може да се користи и празно место за промењљиву „ _ “. Овакве ситуације су у пракси врло ретке.

```
for _ in range(3):
    print(9)
>>>9
9
9
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Кључна реч `in` може да се користи и независно од `for` петље, онда када треба проверити да ли се одређени елемент налази у листи, торки или скупу, као и да ли речник садржи одређени кључ. Оваква провера враћа Булову вредност `True` или `False`.

```
niz=[1,2,3]
1 in niz
>>> True
4 in niz
>>> False
```

Оваква употреба наредбе `in` је у складу са провером за улажење у следећу итерацију `for` петље која заправо проверава да ли постоји следећи елемент у низу.

Наредба `while` је слична као у осталим програмским језицима, уз одговарајућу синтаксу. Она такође подржава `else` наредбу која се извршава једном, онда када услов `while` наредбе узме вредност `False`.

Петље у Пајтону подржавају и `break` наредбу која прекида даљу итерацију. Позивањем ове наредбе унутар петље долази до прескакања `else` гране, уколико постоји.

```
broj_ocena = int(input("Unesite broj ocena "))
niz = []
x=1
while x<=broj_ocena:
    ocena= int(input("Unesite ocenu: "))
    x=x+1
    if ocena<1 or ocena>5:
        print ("Dozvoljeno je unositi samo ocene od 1 do 5, prekid petlje")
        break
        niz.append(ocena)
    else: #Ovaj deo koda se  izvršava samo ako su uneti svi elementi niza
    print ("unete su ocene: "+str(niz))
>>>
Unesite broj ocena 4
Unesite ocenu: 4
Unesite ocenu: 2
Unesite ocenu: 3
Unesite ocenu: 5
unete su ocene: [4, 2, 3, 5]
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Да би се добио тачан одговор на питање да ли је овакав приступ `for` петљи добар у методичком смислу, морало би да се обави опширно паралелно испитивање у школама у којима се учи Пајтон и неки други програмски језик. Ипак, оно што може да се примети је да код оваквих врсти петљи има мање посла него код класичних, јер није неопходно обраћати пажњу на услов, већ само на садржај низа из кога се ишчитавају елементи. Уосталом, огромна већина `for` петљи барата управо са низовима и у таквим ситуацијама је приступ који користи Пајтон практичнији.

Скуповне листе

Пајтон омогућава коришћење скуповних листи (comperhension list) које служе као моћан алат за брзо формирање листа. Синтакса за прављење оваквих листа је слична математичком запису изабирања елемената из неке колекције. Ове листе користе `for` петљу унутар листе, речника или торке, која пролази кроз елементе неке друге итерабле. Често је то листа која се добија `range`() функцијом. На тај начин се, на пример, чување квадрата првих 10 бројева може обавити у само једној линији.

```
  [x**2 for x in range(1,11)]
  >>> [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Овакав начин рада са низовима далеко је интуитивнији од уобичајеног и ближи је математичком концепту низова које ученици усвајају у школи.

Уместо операције над елементима листе, која је основа за скуповну листу, може се користити и било која функција. Осим тога дозвољено је укључивање додатног `if` услова који има сврху филтера за елементе унутрашње итерабле.

Скуповне листе подржавају и угњеждене петље, тако да је могуће приступити истовремено свим елементима вишедимензионог низа и од њих направити низ ниже димензије. Наредни пример илуструје проширене могућности скуповних листа, прво преко прављења таблице множења, а затим преко креирања листе која садржи само парне елементе из ове таблице.

```
  M=[[i*k for i in range (1,6) ] for k in range(1,6)]
  print("Tablica mnozenja:",M)
  print("Parni elementi:",[el for unutrasnja in M for el in
unutrasnja if el%2==0
  >>>
  Tablica mnozenja: [[1, 2, 3, 4, 5], [2, 4, 6, 8, 10], [3, 6, 9,
12, 15], [4, 8, 12, 16, 20], [5, 10, 15, 20, 25]]
  Parni elementi: [2, 4, 2, 4, 6, 8, 10, 6, 12, 4, 8, 12, 16, 20,
10, 20]
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Речници такође подржавају скуповне листе, али захтевају укључење једног параметра више.

```
print({x : y for x,y in zip(range(1,7),list("recnik"))})
>>>{1: 'r', 2: 'e', 3: 'c', 4: 'n', 5: 'i', 6: 'k'}
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Функције

Добра употреба и самостално писање функција су једна од основних вештина које програмер почетник треба да савлада како би могао да напише прегледан и функционалан програм. Функције су основни градитељски блокови сваког доброг програма. Као и у другим програмским језицима, ова област је врло широка и у следећим одељцима биће пружен само основни увид у функције у Пајтону.

Осим основних функција директно доступних по инсталацији Пајтона, касније ће бити наведени и неки додатни пакети, који могу да потпомогну рад са овим програмским језиком.

Модули и уграђене функције

Пајтон поседује мноштво уграђених функција чије се дефиниције могу наћи у стандардној документацији која долази са програмом. Није лоше направити разлику између функција којима се приступа директно и оних које су доступне у оквиру додатних модула.[8]

Издвојене датотеке које садрже дефиниције класа и функција у Пајтону се називају модули. Пуна листа уграђених функција је сувише опширна и стално се проширује тако да ће у овом раду бити наведене само неки од њих.[9]

Како би функција из неког другог модула била увезена у програм, користи се команда `import`. Следећи пример илуструје употребу функција из врло корисног модула `math`.

```
import math
   print (math.sqrt(9))
```

Ако се нека функција користи више пута, може се најавити коришћење те функције из модула, чиме се избегава даље помињање самог имена модула.

```
from math import factorial
print (factorial(10))
```

Овакав приступ је можда бољи за ученике пошто одлаже употребу тачке као позива за атрибут или функцију класе. Са друге стране ако се у истој датотеци, у коју је увезена функција дефинише нова функција са истим именом, она ће заменити увезену. При укључивању једне функције ово се лако може спречити,

[8] Пуна листа функција доступна је на адреси: https://docs.python.org/3/library/functions.html

[9] Листа модула укључених у основу инсталацију Пајтона доступна је на адреси https://docs.python.org/3/py-modindex.html

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

али Пајтон омогућава и увођење групе функција или класа коришћењем оператора звезда:

```
from math import *
```

Чиме се врши увоз свих функција модула `math`.

Све функције у Пајтону, укључујући и оне који су чланови класа, су виртуелне, дакле могу бити замењене другом функцијом са истим именом. Стога овакав приступ треба избегавати на почетку рада. Осим тога ученике треба упућивати да увозе само функције које користе и због ефикасности извршавања кода.

При великим пројектима са много модула Пајтон пружа могућност груписања модула у пакете. Пакет је ништа више од списка датотека са модулима који му припадају. Конкретним модулима се приступа преко назива пакета праћеног оператором тачка и именом модула. Алтернативно, модулима унутар пакета се може приступити и преко пара кључних речи „`from… import…` “ исто као што се приступа функцији унутар модула или елементима класе. Овакав приступ је добар јер са стране програмера који користи постојећи пакет при увозу нема суштинске разлике, па тешко може доћи до забуне, осим ако се не планира модификација над модулима увезеног пакета.

Укључивање модула није неопходно обавити на самом почетку датотеке. Увоз потребне класе или функције се може извршити било где унутар кода, што понекад може побољшати ефикасност кода. Са друге стране, уколико написану функцију треба самостално укључити у оквиру неког другог програма, сви модули које она користи морају бити увезени унутар саме функције која се увози. Почевши од верзије Пајтона 3.0 није дозвољено директно укључивање свих функција из модула унутар друге функције преко оператора звезда. Овај приступ је изабран како би се избегла неефикасност која може да настане при „ланчаном“ увозу низа функција, чиме се нагомилава број непотребних увоза.

У случају функције за коју се предвиђа да ће се ретко позивати и једина је унутар датотеке која користи увезени модул, погодно је користити локално укључивање модула. Мана оваквог приступа је што се донекле умањује читљивост кода када то није неопходно, односно онда када претпостављамо да се та функција неће користити самостално у оквиру неког другог програма.

Понекада је погодно преименовати увезени модул или саму функцију. Ово може бити обављено при самом укључивању коришћењем `as` команде, или директном доделом после увоза промењиве. У следећем примеру је приказано три начина за преименовање функција.

```
import math as mata
from statistics import mean as prosek
koren=mata.sqrt
print(mata.sqrt(9))
print(koren(64))
print(prosek([1,5,2]))
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
>>>
3.0
8.0
2.6666666666666665
```

Преименовањем метода и функција може унапредити читљивост кода, код полазника програмирања млађе животне доби који нису упознати са појединим стручним појмовима на енглеском језику. Осим тога почетно преименовање функција смањује могућност да дође до случајне замене неке раније дефинисане функције или модула.

За примену неке функције „споља“ над појединачним елементима неке итерабле користи се функција `map()`, која за први аргумент узима функцију, а за други итераблу над којом се примењује. Елементи итерабле се редом узимају за аргументе оперативне функције.

```
from math import sqrt
brojevi=['1','11','111']
kvadrati=[1,4,9,16,25]
print(list(map(int,brojevi)))
print(list(map(sqrt,kvadrati)))
>>>
[1, 11, 111]
[1.0, 2.0, 3.0, 4.0, 5.0]
```

Функција main()

Пајтон подржава и укључивање и специјалне функције `main()`. Она није неопходна за покретање програма, односно модула као у програмском језику C, али је важна када се увози програм, класа или функција као модул у неки други програм. Ако датотека, која се увози, садржи неки кôд који се извршава по покретању модула, тај кôд ће се извршити и при самом увођењу у оквиру другог модула. Нека је дата датотека под називом `stampa`, која има следећу структуру.

```
def stampa_iz_datoteke():
    ....
def stampa_iz_klipborda():
    ....
print ("Ovo je modul stampanje")
```

Употреба функције из ове датотеке у неком другом програму може да се омогући преко команде `import stampa`. Међутим, `print()` функција ће

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

аутоматски бити покренута при увозу уколико се не налази унутар `main()` функције, па ће се добити излаз који вероватно није пожељан. Овај проблем се решава издвајањем извршног дела програма помоћу главне функције.

```
def main():
    print ("Ovo je modul stampanje")
if __name__ == "__main__": main()
```

Испод дефинисања функције постоји додатна информација за интерпретер која у телу `if` наредбе наводи име главне функције. Име `main()` функције је дакле произвољно, мада се овај назив препоручује ради конзистентности са другим програмским језицима. На овај начин главни део кода се покреће само по покретању модула, али не и при употреби `import` наредбе.

Писање функција за поновну употребу

Курс програмирања не може да буде потпун без упознавања почетника са могућношћу да сам направи функције. Пајтон програмски језик је бесплатан и отвореног кода. Ову чињеницу треба обелоданити ученицима и назначити им да њихове функције, уколико су довољно добро написане, једног дана могу да постану део уграђених функција библиотеке Пајтона. Ово може да буде значајан подстрек за експериментисање при писању програма, што је уједно и најефикаснији начин за развијање програмерског начина размишљања.

Оног тренутка када се направи нека функција, класа или променљива и сачува се у оквиру засебне датотеке, она бива додавана у списак документације Пајтона на локалном рачунару. На овај начин наставник и ученик имају брз преглед постигнутог напретка на часу или у оквиру домаћих задатака. У ову сврху погодне су и портабилне верзије Пајтона које ученик може да носи са собом на преносној меморији заједно са ранијим задацима. На тај начин се посао обављен у школи врло једноставно се наставља на рад домаћих задатака и обратно. Поновна употреба ранијих програма је једна од важних аспеката програмирања и није лоше упознати ученике са њом од самог почетка.

Додатна погодност по питању мобилности јесте могућност програмирања на Пајтону на паметном мобилном уређају и таблету преко програма као што су КуПајтон (QPython) за Андроид или Пајтониста (Pythonista) за iOS. Овакав приступ се може учинити привлачним ученицима и пружа још једну додатну мотивацију за рад.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Дефинисање функција

Као што је већ поменуто, функције се дефинишу помоћу кључне речи `def` праћене именом промењиве са произвољним бројем аргумената. Функције које се дефинишу као део класе називају се методама. Може се и очекивати да нема декларисања повратне вредности, пошто су све промењиве динамичног типа.

Једна нерекурзивна функција, која исписује првих `n` вредности Фибоначијевог низа са почетним вредностима $F_0 = 0$ и $F_1 = 1$, у Пајтону би могла да има следећи облик:

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print (a)
        (a, b) = (b, a+b)
n= int(input("Unesite granicu za fibonačijev niz: "))
fib(n)
```

Као што се може приметити, коришћење уређеног пара драстично смањује број линија кода потребних да би се обавио тражени задатак.

Исти програм у Паскалу би могао да изгледа овако:

```
PROGRAM Fibonaci;
PROCEDURE fib(n:integer);
    VAR prvi, drugi, suma : integer;

    BEGIN
        suma := 1;
        prvi := 0;
        drugi := 1;
        writeln(drugi);
        while suma<n do
        begin
            writeln(suma);
            suma := prvi + drugi;
            prvi := drugi;
            drugi := suma;
        END;
END;

VAR n:integer;
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
BEGIN
   readln(n);
   fib(n);
END.
```

Овакав кôд је знатно дужи и захтева више времена. Почетници морају да обраћају пажњу на декларацију променљивих и кабасте блокове наредби, што им одвраћа пажњу од саме суштине програма.

Програмски језик C би могао да има следећу имплементацију овог задатка.

```
#include <stdio.h>

void fib(n){
    int t1=0, t2=1, suma=0;

    printf("Fibonacijev niz: %d %d ", t1, t2);
    suma=t1+t2;
    while(suma<n){
        printf("%d ",suma);
        t1=t2;
        t2=suma;
        suma=t1+t2;
  }
}
int main(){
    int n;
    printf("Unesite granicu: ");
    scanf("%d",&n);
    fib(n);
    return 0;
}
```

Овај кôд је велики напредак, али још једном, носи са собом сличне проблеме као и Паскал.

Графички приступ ВБ-а обично подразумева прво убацивање елемената на форму пре писања кода. У примеру који следи се претпостављају три елемента на форми „`Button1`“, „`TextBox1`“ и „`Label1`“. Пропратни кôд који исписује елементе Фибоначијевог низа у `Label1` могао би да изгледа овако:

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
 Public Class Form1
     Private Sub Button1_Click(sender As Object, e As EventArgs)
Handles Button1.Click
         Dim a As Integer = 0
         Dim b As Integer = 1
         Dim fib As Integer = 1
         Dim n As Integer = Int(TextBox1.Text)
         Do
           Label1.Text = Label1.Text + fib.ToString &
ControlChars.NewLine
             fib = a + b
             a = b
             b = fib
         Loop While fib < n
     End Sub
 End Class
```

Овакав кôд је кратак и графички кориснички интерфејс јесте визуелно примамљивији за ученике, али садржи неколико појмова који захтевају додатна објашњења за почетника. Осим тога, и поред коришћења готових елемента на форми, писање оваквог програма захтева више времена од писања верзије у Пајтону и садржи више линија кода. Значајан део кода је аутоматски убачен, што одузима могућност ученицима да буду у потпуној контроли писања програма. Осим тога, подразумевана поставка Microsoft Visual Studio-а даје предлоге за појмове који би могли да се користе, што, иако врло погодно у каснијем програмирању, може да спречи ученике да дођу до неопходних програмерских сазнања.

Може се приметити да програм писан у Пајтону користи и једну променљиву мање, чиме донекле надомештава своју неефикасну употребу меморијског простора. Такође, граница за одређивање јако великих Фибоначијевих бројева је у Пајтону знатно већа него у осталим програмским језицима, када се ради о горе наведеној једноставној структури функције. Ипак ова ограничења су у оквиру потреба курса програмирања и у другим програмским језицима.

Угњеждене функције омогућавају и прављење фамилије функција, где унутрашња функција обавља неку радњу, а спољна функција заправо враћа референцу на унутрашњу функцију са одређеним параметром.

Ово се може видети на следећем примеру.

```
  def familija(n):
      def operacija(x):
          return x**n
      return operacija
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
kvadriranje=familija(2)
kub=familija(3)
print(kvadriranje(3))
print(kub(3))
>>>
9
27
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Ламбда функције

Пајтон подржава и једнолинијске ламбда (lambda) функције које су погодне за скраћено дефинисање функција и за креирање фамилије функција. Испод су наведена два различита начина за дефинисање функције који имају исти ефекат.

```
def funk(x): return x**3
lam=lambda x : x**3
```

Обе функције се позивају на исти начин и дају исти резултат:

```
print(funk(3))
print(lam(3))
>>> 27
27
```

Због иначе једноставног дефинисања функција у Пајтону, ламбда у свом основном облику није од великог значаја. Међутим ако функција враћа ламбда функцију као повратну вредност то може довести до сличних резултата као у примеру са угњежденим функцијама али у још компактнијем облику [10]:

```
def inkrement(n):
    return lambda x:x+n

inkrement5=inkrement(5)
inkrement11=inkrement(11)

print(inkrement5(100))
print(inkrement11(100))

print(inkrement(10)(100))
>>> 105
111
110
```

Прва функција прави различите функције зависно од прослеђеног параметра. Ове функције надаље могу да се користе независно једна од друге. Последња линија указује на могућност директног позивања прве функције, прослеђивањем два параметра редом. Редослед којим се ово обавља је прво креирање функције са прослеђеним параметром, а затим повезивање параметризоване функције над прослеђеним аргументом.

Овај приступ се може даље проширити прављењем листе фамилије функција. Следећи пример илуструје листу која чува функције степеновања, чији је параметар степена одређен индексом листе.

```
lista=[]
for i in range(5):
    lista.append(lambda x:x**i)
for i in range(5):
   print(lista[i](3))
>>>
1
3
9
27
81
```

Ламбда функције имају значајну примену и при давању додатних информација одређеним уграђеним функцијама као што су `sorted(), min()` и `max()`, што ће бити образложено преко следећих примера.

Нека је дата листа тачка задатих преко уређених парова коју треба сортирати. Уколико се `sorted()` функцији проследи само низ тачака она ће вратити те тачке сортиране по првој координати, односно по другој уколико су прве координате једнаке. Уколико је потребно сортирање примарно по другој координати може се користити други, опциони, параметар `sorted()` функције који има улогу кључа. Овај други параметар мора бити функција која прима један аргумент и враћа кључ по ком ће бити обављено сортирање. У овом случају је ламбда функција од велике помоћи, пошто је врло могуће да се та „кључ“ функција неће користити нигде другде осим за специфичне потребе сортирања:

```
tacke=[(8,3),(4,2),(2,3),(5,8)]
print(sorted(tacke))
print(sorted(tacke,key=lambda tacke:tacke[1]))
>>>
[(2, 3), (4, 2), (5, 8), (8, 3)]
[(4, 2), (8, 3), (2, 3), (5, 8)]
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Кључ-функција може бити и нека од уграђених функција које су погодне за дату ситуацију. За сортирање речи може се на пример користити `str.lower` функција која у конкретном случају изједначава мала и велика слова при сортирању:

```
  print(min("Ana je prva, ali ne mora da bude".split()))
  print(min("Ana je prva, ali ne mora da
bude".split(),key=str.lower))
  >>>Ana
  Ali
```

На овај начин се добија правилан лексикографски распоред, пошто су мала и велика слова најчешће исте важности при сортирању.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Класе

Наставни програм програмирања у школама ретко оставља места за обрађивање класа и објеката, као опширне, али важне теме у област програмирања. Лакоћа синтаксе Пајтона може да омогући ученику да брзо и ефикасно усвоји основна знања о класама и објектима. Чак и сам увод у ове појмове може бити од помоћи ученицима који одлуче да наставе да се баве програмирањем.

Класе се дефинишу преко кључне речи `class`, праћене називом и телом класе. Формулисање инстанце класе се обавља помоћу функције конструктора `__init()__` која може да садржи произвољан број аргумената. Уколико се не формулише конструктор унутар класе, Пајтон аутоматски прави основни конструктор без параметара. Сваки конструктор при дефинисању има за први аргумент референцу на објекат, који се по конвенцији зове `self`, али може носити и било који други назив. Иза овог аргуметра се декларишу остали „прави“ аргументи. При позивању конструктора аргументи се прослеђују као да први аргумент не постоји. Могуће је имати само један конструктор по класи. Овај недостатак се надомешћује тако што се неком од параметара додели подразумевана вредност, чиме се она при прављењу примерка класе може изоставити уколико сви остали аргументи, почевши од тог, такође имају подразумевану вредност. Почетна вредност може бити `None`, чиме се елиминишу проблеми сувишног додељивања вредности при прављењу примерка класе. Овај метод је примењен у следећем примеру:

```
class multiKon:

    def __init__(self, info, parametar=None, parametar2=None):
        if parametar:
            self.info = info+ str(parametar)
        else:
            self.info =info

m1=multiKon("Instanca bez parametra")
print(m1.info)
m2=multiKon("Instanca sa parametrom ", 111)
print(m2.info)
>>>
Instanca bez parametra
Instanca sa parametrom 111
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Кључна реч `self` представља референцу на примерак класе и слична је кључној речи `this` у Јави, C++ и C#, као и кључној речи `Me` у ВБ-у.

Методе, као функције унутар класа, разликују се од осталих по томе што им је при дефинисању први аргумент увек `self`, исто као и код конструктора. При позивању функције над инстанцом, први аргумент из дефиниције се изоставља.

Следећа класа има три једноставна метода:

```
from statistics import mean as ar_sredina

class ucenici:
    def __init__(self,ime = "",prezime="",ocene=[0],jmbg=None):
        self.ime=ime
        self.prezime = prezime
        self.ocene=ocene
        self.__jmbg=jmbg

    def prosek(self):
        return ar_sredina(self.ocene)
    def stampaJMBG(self):
        return self.__jmbg

def main():
    ucenik1=ucenici("Ivo","Andric",[4,5,3,5],2134)
    ucenik2=ucenici("Branko","Radicevic")
    print("Ucenik:",ucenik1.ime,ucenik1.prezime,"ima prosek
",ucenik1.prosek())
    print("Ucenik:",ucenik2.ime,ucenik2.prezime,"ima prosek
",ucenik2.prosek())
    print(ucenici("Petar","Petrović",None,9999999).stampaJMBG())

if __name__ == "__main__": main()
>>>
Ucenik: Ivo Andric ima prosek  4.25
Ucenik: Branko Radicevic ima prosek  0.0
9999999
```

Додељивање подразумеване вредности пружа могућност изостављања неког од последњих параметара. За одређивање аритметичке средине коришћена је функција `mean()` из модула `statistics` укључене у стандардну библиотеку од верзије Пајтона 3.0. Употреба `main()` функције унутар датотека, које садрже класе, је пожељна јер је уобичајено да се класе налазе у више различитих датотека.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Уколико је то погодно, могуће је направити привремену класу и истовремено позвати променљиву или методу из класе, као што је учињено у претходном примеру. При последњем штампању је изостављено и додељивање оцена ученику, преко додељивања празне вредности, како би променљива `jmbg` узела правилну вредност. Овакав приступ „прескакања“ аргумената је редак и обично указује на потребу за оптимизацијом кода.

Сви чланови класе без атрибута су јавни (public), односно може им се приступити изван класе. Додавање водећих доњих линија __ члан класе постаје ограничен на употребу унутар класе и њених потомака. Често се после имена локалних променљивих додају две или три доње линије, што је чисто естетске природе. У сврху заштите података, овим променљивама је уобичајено да се приступа преко метода унутар класе. У претходном примеру је декларисана променљива `__jmbg` заједно са функцијом која исписује њену вредност. Покушај директног приступа овој променљивој преко инстанце класе, условио би грешку при компилацији, као да променљива `jmbg` не постоји у класи. Другим речима, сакривена је, не само вредност приватне променљиве, већ и само постојање. Једини начин приступа приватној променљивој класе је преко метода те класе или њених потомака.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Модули за обраду и приказ података

Модули који ће бити наведени у овом одељку служе првенствено за мотивацију и припрему ученика за напредније програмирање. Ови, као и многи други модули, могу се укључити после преузимања одговарајућих датотека. Ове датотеке се могу преузети појединачно или у оквиру неког од пакета који многа подешавања обављају аутоматски. За потребе овог рада коришћена је Анаконда (Anaconda) која се користи проширеном интерактивном конзолом Ајпајтон (IPython) [14].

Пакет за нумеричку обраду података

Пакет НамПај[10] (NumPy) је пакет који садржи велики број функција и објеката који олакшавају нумеричке прорачуне. Синтакса која је уско везана за овај пакет се често држи синтаксе Матлаба, што омогућава брз прелазак са једног радног окружења на друго. Ово ће нарочито доћи до изражаја при употреби модула за штампање скица и графикона, који ће бити обрађени касније.

Основна структура пакета НамПај је више димензиони нумерички низ, који се у својој основи не разликује значајно од обичних листа у Пајтону, али подржава више могућности. Слично као поменута уграђена структура `array`, ови низови подржавају јединствени тип промењиве, уз проширену функционалност.

После укључивања одговарајућег модула и функције, најприроднији начин за прављење НамПај низа (у наставку: нумеричког низа) је прослеђивање листе функцији `array()`.

```
from numpy import array
a=array([5,6,7])
print(a)
>>> [5 6 7]
```

Тип промењиве у нумеричком низу је одређен типом промењивих у прослеђеној лист. Уколико су прослеђени различити типови у оквиру листе, нумерички низ ће аутоматски претворити остале елементе у „најшири“ тип, где су природни бројеви најужи а ниске најшире.

```
from numpy import array
a=array([5, 6, 7.5])
b=array([3, 2+1j, 1])
c=array([6, 1, 'niska'])
```

[10] Скраћено од Numerical Python (Нумерички Пајтон)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
print(a)
print(b)
print(c)
>>>
[ 5.   6.   7.5]
[ 3.+0.j  2.+1.j  1.+0.j]
['6' '1' 'niska']
```

Као што се може приметити, при штампању реалних бројева унутар нумеричких низова, изоставља се децимални део уколико је нула, што није случај са листама.

Тип промењиве се може и експлицитно назначити преко додатног аргумента:

```
array([9,10,11], dtype=float)
```

Оно што нумеричке низове чини погоднијим за рад са вишедимензионим низовима је штампање у матричном облику.

```
from numpy import array
M=array([[1,2,3],[4,5,6],[7,8,9]])
print(M)
>>>
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Постоји и мноштво функција које омогућавају скраћено прављење нумеричких низова. У доњем примеру су наведене неке од њих.

```
from numpy import arange,linspace,reshape
niz=arange(12) #poput range funkcije
ekvidistatniNiz=linspace(0,1,11)# 11 brojeva u rasponu od 0 do 1
matrica3x4 = arange(12).reshape(4,3) #predimenzionisanje niza u
matricu 4x3


print(niz)
print(ekvidistatniNiz)
print(matrica3x4)
)


>>> [ 0  1  2  3  4  5  6  7  8  9 10 11]
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1. ]
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Подржане су и функције са именима која се поклапају са одговарајућим функцијама у Матлабу `zeros()` и `ones()`. Ове функције додељују свим елементима низа као почетну вредност нулу односно јединицу.

Све операције над нумеричким низовима се обављају над појединачним елементима. Бинарне операције са два или више низова исте димензије се такође обављају елемент по елемент. Ово понекад може да проузрокује неочекивано понашање, нарочито при операцијама над матрицама.

```
from numpy import array
a=array([[5,6],[1,3]])
print(a+9)
print(a**2)
>>>
[[14 15]
 [10 12]]
[[25 36]
 [ 1  9]]
```

Међутим, постоји посебна структура матрице `matrix()` која дозвољава нормалне операције над матрицама.

```
from numpy import matrix
a=matrix([[1,0],[0,1]])
b=matrix([[9,6],[1,3]])
print(a*b)
>>>
[[9 6]
 [1 3]]
```

На крају, може се још поменути да нумерички низови подржавају и већину функција које подржавају листе, као што су `.sum()`, `.min()`, `exp()`, `sqrt()`, `sin()` итд.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Проширен пакет за обраду података

Продужење претходног пакета НамПај је СајПај (SciPy[11]), који даје нове могућности за обраду података, нарочито оне врсте која спада под научни домен. Овај пакет омогућава напредне математичке операције као што су интеграција, диференцирање, операције линеарне алгебре, више димензиону обраду слика итд. [13].

У наставку је неколико примера који илуструју могућности овог пакета. У првом примеру је приказано прављење, исписивање и основне операције над полиномима коришћењем функције `poly1d()`, која као аргумент прима листу коефицијената полинома и прави одговарајући полином.

```
from scipy import poly1d
p=poly1d([4,1,0,3])
print("Osnovni polinom: \n",p)
print("Proizvod polinoma: \n",p*p)
print("Zbir polinoma: \n",p+p)

>>>
Osnovni polinom:
    3     2
4 x + 1 x + 3
Proizvod polinoma:
     6     5     4      3     2
16 x + 8 x + 1 x + 24 x + 6 x + 9
Zbir polinoma:
    3     2
8 x + 2 x + 6
```

Полиноми се исписују у два реда, тако да су у првом реду експоненти у одговарајућој колони, а у другом коефицијенти. Операције множења и сабирања се добро понашају са полиномима. Уколико је неопходно нагласити да се ради о операцијама сабирања и множења над полиномима, могу се користити и функције `polyadd()` и `polymul()`.

Функција `polyval(p,x)` даје вредност полинома у тачки, док функција `roots(p)` враћа нуле прослеђеног полинома у облику листе.

Пајтон омогућава и израчунавање извода полинома са функцијом `polyder(p[,m])`, која за први аргумент узима полином, а за други опциони ред извода. Подразумевани ред извода је један.

[11] Скраћено од Scientific Python

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
from scipy import poly1d,polyder
p=poly1d([4,1,0,3])
print(polyder(p))
print(polyder(p,2))
>>>
    2
12 x + 2 x
 24 x + 2
```

Поред тражења извода полинома функције, постоји могућност израчунавања извода у тачки било које функције преко `derivative()` функције из овог пакета.

Пакет за графички приказ података

У оквиру стандардне библиотеке Пајтона, укључен је модул `turtle` (корњача), који омогућава једноставно исцртавање графичких елемената на платну, по систему који имитира повлачења оловке по папиру. За употребу овог графичког модула није неопходно познавање координатног система и он има првенствено забавну сврху. Стога тај модул неће бити овде детаљно разматран већ ће бити анализиран један од алата за исцртавање графика шире намене.

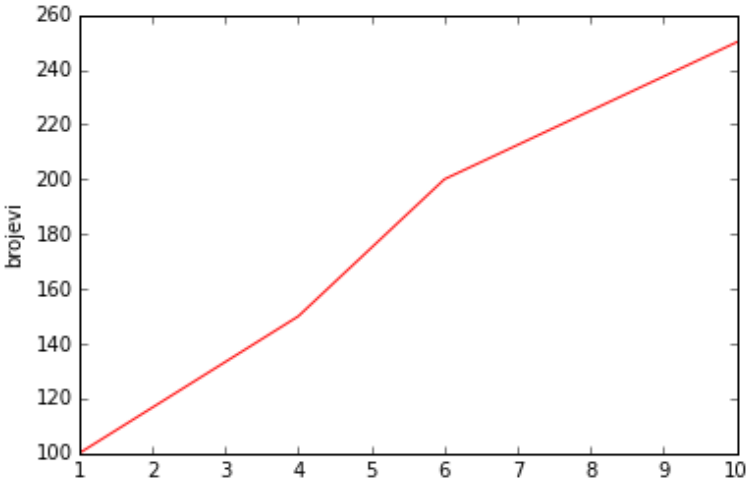
Пакет модула МатПлотЛиб (Matplotlib) је колекција која омогућава рад са функцијама чија се синтакса и примена у великој мери поклапа са сличним функцијама из програмског пакета Матлаб. Ове функције пружају могућност штампања графикона, па их је добро укључити у наставу јер графички приказ резултата рада ученика може позитивно утицати на мотивацију ученика.

Овај пакет користи нумеричке низове и све стандардне листе се у позадини пребацују у овај тип низова. Рестрикција на униформне листе је природна, пошто се координате графика изражавају јединственом врстом променљивих. Мала разлика у односу на обичне нумеричке низове је то што су низови, који су имплементирани у овом пакету, увек бројеви и сви карактер-бројеви се аутоматски пребацују у нумеричке вредности, што омогућава олакшано учитавање података са улаза и из датотека, тако да низ `['1','4','6','10']` прећутно постаје `[1,4,5,10]`.[11]

Основна функција овог пакета је `.plot()` која дозвољава штампање графика на основу аргумената.

```
from matplotlib import pyplot
pyplot.plot([1,4,6,10],[100,150,200,250],'r-')
pyplot.ylabel('brojevi')
pyplot.show()
>>>
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs



Слика 1 – Табеларна функција

Први аргумент се односи на вредности x-осе, други на y-осу, док трећи, опциони аргумент, даје информацију о начину на који ће графикон буди штампан. Први и други аргумент морају имати исти број елемената. Уколико се проследи само једна листа, подразумева се да су то вредности y-осе, док су подразумевани елементи друге осе природни бројеви који почињу са нулом, а завршавају се са једним бројем мање од дужине прослеђеног низа y-координата.

Аргумент о формату исписа за први број карактер означава боју, док други означава графички приказ „тачака“. У случају линије, тачке се не истичу, већ су само повезане дужима као што се види на примеру. Подразумеване вредности су плава боја и линија.

Уколико је потребно означити осе које се штампају, употребљавају се функције `.xlabel()` и `.ylabel()`, за x y осу, респективно.

За добијање графика неке математичке функције, за x координату се прослеђује жељени опсег бројева а за y координату функција од жељеног низа бројева. У овом тренутку је јасна предност примене операције над појединачним члановима нумеричког низа.

Пакет `pyplot` пружа могућност исцртавња више графика одједанпут преко функције `subplot()`. Ова функција узима три аргумента, први означава број редова, други број колона и трећи позицију у матрици.

У следећем примеру се имплементира исписивање графика полиномне функције другог и трећег степена.

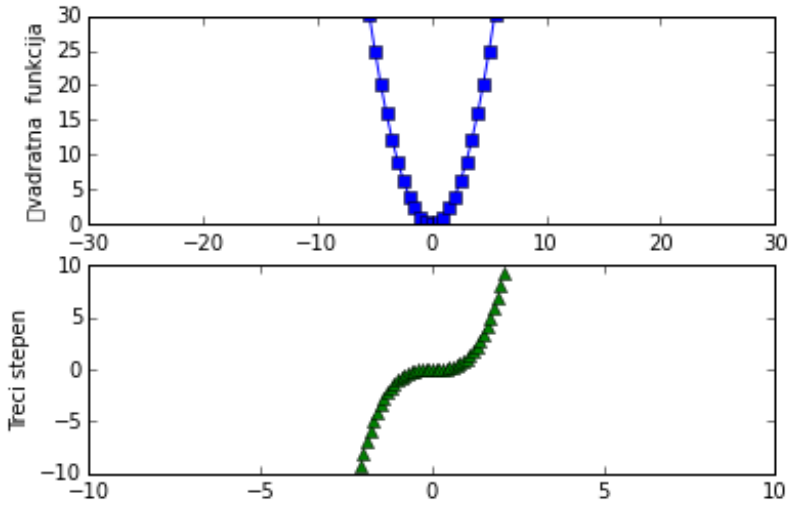
```
t=numpy.arange(-30 , 30, 0.5)


pyplot.subplot(2,1,1)
pyplot.axis([-30,30,0,30])
pyplot.plot(t,t**2, 'bs-')
pyplot.ylabel('Kvadratna funkcija')
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
pyplot.subplot(2,1,2)
t=numpy.arange(-10 , 10, 0.1)
pyplot.axis([-10,10,-10,10])
pyplot.ylabel('Treci stepen')
pyplot.plot(t, t**3, 'g^')

pyplot.show()
```



Слика 2 – функције другог и трећег степена

Како би график функције користио вредности распона на оси у односу 1:1 у сврху добијања реалистичног графика искоришћена је функција `axis()` која дефинише границе видљивог графика. Ова функција за аргументе узима листу од четири елемента, где прва два означавају горњу и доњу границу графика за x осу, а друга два границе за y осу. У противном би однос био нарушен и за довољно велике вредности y t добио би се нерепрезентативан график, па чак и права линија.

Може се приметити како параметри за графичку репрезентацију тачака могу да примају више од једног аргумента, у конкретном случају квадратна функција је представљена и преко квадрата који означавају тачке и преко дужи које их спајају.

Претходно излагање је само кратак увид у могућности које пружа овај пакет. Мноштво уграђених функција, могућност рада са Латех (LateX) синтаксом за записивање формула директно на графике, као и други начини за презентацију података, нарочито статистичких, чине да Пајтон буде компатибилан да се такмичи са Матлабом и другим језицима, када је у питању исцртавање графикона.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Имплементација алгоритама у Пајтону

Претходне главе су биле углавном усмерене ка испитивању могућности Пајтона да замени тренутно актуелне програмске језике у систему образовања, пре свега у оквиру основних и средњих школа. У овој глави ће бити разрађена мало напреднија тема, преко анализе предности овог програмског језика за имплементацију алгоритама.

Како је суштина програмирања у самом алгоритму и подацима, често се при учењу алгоритама излази из оквира „живих“ програмских језика и прелази се на писање много читљивијег псеудокода. У следећим примерима, уколико није наглашено другачије, биће коришћен псеудокод заснован на програмском језику Паскал из књиге [2]. Ефикасност овог конкретног псеудокода неће бити разматрана и изабрана је као референтна јер се налази у оквиру стандардне литературе Математичког факултета у Београду.

На почетку треба напоменути да је могућност извршавања кода алгоритма ипак од великог значаја при учењу алгоритама, првенствено услед могућности директне провере исправности алгоритма, али и због лакшег усвајања знања, услед потенцијала за директну примену.

Иако се књиге о алгоритмима најчешће пишу на псеудокоду, вежбе и испити се на скоро свим факултетима изводе помоћу програмских језика C, C++ или Јави. То значи да студенти при учењу новог алгоритма морају да размишљају у два паралелна правца. Коришћење поменутих програмских језика приморавају студенте да размишљају о структурама података и пратећим процедурама писања конкретног језика, наместо о конструкцији самог алгоритама. По речима Паи Чоуа “Пајтон представља базично алгоритамски језик...” [12], и идеја овог одељка је управо да прикаже да алгоритми писани у Пајтону нису ништа лошији од псеудокода, када је у питању читљивости и лакоћа имплементације. Подразумева се да је за разумевање ових алгоритама неопходно познавање синтаксе језика, што је само још један мотив за увођење Пајтона у основно и средње образовање.

[12] Pai H. Chou, *Algorithm Education in Python* (2002), стр. 1

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Бинарна претрага

На почетку се анализира алгоритам бинарне претраге који је по речима професора М. Живковића „... за област алгоритама оно што је точак за механизме: она је једноставна, елегантна, неизмерно важна, и откривана више пута.“[13].

```
  Алгоритам 1.0 –Псеудокод- Бинарна претрага
  Ulaz: X (niz od n brojeva, uređenih neopadajuće) i z (broj koji se
traži).
  Izlaz: Poz (indeks i takav da je X[i] = z, ili 0 ako takav indeks
ne postoji).
  begin
     Poz := Nadji(z, 1, n);
  end
  function Nadji(z, Levi, Desni) : integer
  begin
  if Levi = Desni then
  if X[Levi] = z then Nadji := Levi
  else Nadji := 0
  else
  Srednji = ⌈(Levi + Desni)/2⌉;
  if z < X[Srednji] then
  Nadji = Nadji(z, Levi, Srednji – 1)
  else
  Nadji = Nadji(z, Srednji, Desni)
  end
```

Овај псеудокод подразумева постојање низа X са бројевима ван самог алгоритма у неком виду глобалне променљиве. Ово не умањује драстично читљивост алгоритма, али се при имплементацији у C-у на ово мора обраћати пажња. У скоро свим примерима псеудокода индексирање низова почиње са 1, па стога постоји неслагање по индексима између псеудокода са једне и Пајтона и програмског језика C са друге стране

Иако се ради о псеудокоду где се ретко користи декларација променљивих, овај алгоритам се ипак строго држи Паскалове синтаксе, па је тиме форсиран тип повратне вредности функције.

Кôд бинарне претраге у Пајтону би могао да изгледао овако:

[13] Миодраг Живковић, *Алгоритми* (2000) стр. 79

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
"""Алгоритам 1.1 -Пајтон- Бинарна претрага"""
def binarna_pretraga(lista, element):
    def trazi_rekurzivno(levi, desni):
        if desni < levi:
            return -1

        srednji = (levi + desni)//2
        if element > lista[srednji]:
            return trazi_rekurzivno(srednji + 1, desni)
        elif element < lista[srednji]:
            return trazi_rekurzivno(levi, srednji - 1)
        else:
            return srednji
    return trazi_rekurzivno(0, len(lista) - 1)
```

Угњеждена функција омогућава да позив рекурзивне функције има само два елемента, уместо четири. Ово је не само визуелно лепше, већ је и ефикасније у смислу ресурса. Може се приметити да је употребом `elif` наредбе умето `swich-case`, дошло до одређеног губитка на читљивости.

Број линија кода у Пајтону је заправо мањи него у псеудокоду. Коришћење операције целобројног дељења је исто толико једноставно као и операција заокруживања на цео број. Да је кôд Пајтона захтевао заокруживање на већи број, могао је бити примењен метод додавања јединице или функције `ceil(x)`, дефинисане у оквиру стандардне библиотеке у склопу модула `math`.

Алгоритам написан у Програмском језику C може да има следећи облик:

```
/*Алгоритам 1.2 -C- Бинарна претрага*/
int bin_pretraga(int niz[], int levi, int desni, int element){

    int srednji;
    if (levi > desni){
        return -1;
    }
    srednji = (levi + desni) / 2;

    if (niz[srednji] == element){
        return srednji;
    }
    else if (niz[srednji] > element){
        bin_pretraga(niz, levi, srednji - 1, element);
    }
    else if (niz[srednji] < element){
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
            bin_pretraga(niz, srednji + 1, desni, element);
        }
    }
```

Како се може и очекивати од C-а, кôд алгоритма 1.2 је врло компактан и од Пајтона се разликује у само неколико нијанси. При тражењу средњег елемента, није на први погледа јасно о ком се дељењу ради, јер треба обраћати пажњу на типове променљивих које учествују у операцији. Пошто већина компајлера за C не подржава угњеждене функције, рекурзивни позив функције позива два елемента више.

Оно што захтева мало више пажње, при имплементацији алгоритма бинарне претраге, је дужина низа који се прослеђује, односно крајњи десни крај при претрази. Њега је тешко одредити директно у C-у и најчешће се примењује решење које захтева од корисника ручно уношење броја чланова низа. Ово можда изгледа као ситна ствар, али треба имати на уму да се са сваким скраћивањем кода смањује могућност грешке и, што је још важније, оставља више простора за студента да се бави самим алгоритмом. Осим тога, могућност функција писаних у Пајтону да се снађу са дужином низа унутар функције, чини ове функције далеко флексибилнијим, у смислу величине низа који се претражује.

Пошто је јасно да се преполовљава низ који се претражује у сваком кораку рекурзије, сва три кода алгоритма јасно указују на сложеност од O(log n), за низ величине n. Број `if` блокова је исти, те ни у овом аспекту нема неких замерки везаних за Пајтон. Сва три кода добро осликавају теоријски модел претраге, који и јесте у својој суштини рекурзиван.

Сортирање пребројавањем

Сортирање пребројавањем (countig sort) је врло елегантан метод сортирања без директног поређења, са одличним временом извођења. Овај алгоритам сортира n елемената у распону од 0 до k, али га је могуће је једноставно модификовати за рад над негативним бројевима, транслацијом пре и после сортирања. За сваки елемент `x` пребројава се колико елемента није веће од `x`, што омогућава том елементу да дође на одговарајуће место у сортираном низу. Следећи псеудокод [8] чува информације о „бројању“ унутар низа `C` [14] .

[14] Низ C има индексирање `[0…k]` док низови A и B имају индексе [1…n]. Ова неконзистенција од стране аутора је највероватније настала због специфичних потреба овог алгоритма.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
Алгоритам 2.0 –Псеудокод - Сортирање пребројавањем(A,B,k)
Нека је C[0…k] нови низ
for i = 0 to k
        C[i]=0
for j = 1 to A.length
        C[A[j]]=C[A[j]]+1
//C[i] сада садржи број елената који су једнаки i
for i =1 to k
        C[i]=C[i]+C[i-1]
//C[i] сада садржи број елемената који су мањи или једнаки i
for j  = A.length downto 1
        B[C[A[j]]] = A[j]
        C[A[j]]=C[A[j]]-1
```

У алгоритму 2.0 је изостављена тривијална претпоставка да прослеђени низ B није мањи од A. Овај алгоритам је заиста врло кратак и једноставан за имплементацију када се има на уму основна идеја.

Следи опис истог алгоритам у Пајтону:

```
"""Алгоритам 2.1 -Пајтон- Сортирање пребројавањем (A,k)"""
def sortiranje_prebrojavanjem(A,k):
    C=[0]*k
    for x in A:
        C[x]=C[x]+1
    for i in range(1,k):
        C[i]=C[i-1]+C[i]
    B=[None]*n
    for x in reversed(A):
        B[C[x]-1]=x
        C[x]=C[x]-1
    return B
```

Операција множења низова у оквиру Пајтонове синтаксе омогућава избегавање две петље за формирање почетних вредности низова. На овај начин добијен је кôд са истим бројем линија као и псеудо варијанта, без губитка на читљивости. Пајтонова `for` петља се овде показала бољом, пошто се заобилажењем индексирања поједноставило тело петљи. На овај начин програмерски кôд садржи једну заграду мање при додељивању вредности. Велики број заграда често може да збуни програмера, нарочито почетника, па је свака олакшица у овом смеру добродошла. Уместо прослеђивања локалне листе B, директно су јој додељене вредности унутар функције, што је олакшано операцијом множења. Уместо наведеног приступа може да се користи и `.insert()` функција, али она не би довела до једноставнијег кода, евентуално незнатно ефикаснијег.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Сама идеја алгоритма није нарушена и, за некога ко је упознат са синтаксом Пајтона, јасно је на који начин се врши сортирање. Све временски захтевне петље су присутне и пружају добар увид у сложеност алгоритма.

Алгоритам у програмском језику C:

```
/*Алгоритам 2.2 -C- Сортирање пребројавањем */
#include<stdio.h>
main(){
    int i,j,k,n;
    int A[100],B[100],C[100];
    scanf("%d%d",&n,&k);
    for (i=0;i<n;i++)
        scanf("%d",&A[i]);
    for (i=0;i<k;i++)
         C[i]=0;
    for (j=0;j<n;j++)
        C[A[j]]=C[A[j]]+1;
    for (i=1;i<k;i++)
        C[i]=C[i]+C[i-1];
    for (j=n-1;j>=0;j--){
        B[C[A[j]]-1]=A[j];
        C[A[j]]=C[A[j]]-1;
    }
 }
```

Овако представљен алгоритам врло је сличан наведеном псеудокоду, па стога не захтева даља објашњења. Може се приметити да, изузимајући декларацију променљивих и уношење вредности, ова верзија није нарочито дужа од оне написане у Пајтону, али је мање читљива, услед већег броја угњеждених заграда.

Временска сложеност овог алгоритма је врло добра и износи O(n+k). Сортирање пребројавањем је најефикасније примењивати на великом броју цифара из релативно малог опсега, односно ако је k=O(n), чиме је потребно време извршења Θ(n).

Сортирање обрнутим вишеструким разврставањем

Овај алгоритам са сложеним именом, познатији у енглеском говорном подручју као *radix sort* се заснива на размештању елемената који се сортирају у прегради по цифрама, почевши од цифре најниже вредности. Овај алгоритам се може користити за сортирање бројева са месним записом, као и сортирање ниски по карактерима.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Имплементације овог алгоритма у Паскаловом псеудокоду:

```
  Алгоритам 3.0 –Псеудокод- СОВР
  Ulaz: X (niz od n celih nenegativnih brojeva sa po k cifara).
  Izlaz: X (sortirani niz).
  begin
     Pretpostavljamo da su na početku svi elementi u globalnoj listi
GL
     {GL se koristi zbog jednostavnosti; lista se može realizovati u
X}
      for i := 0 to d – 1 do
      {d je broj mogućih cifara; d = 10 u dekadnom sluqaju}
  inicijalizovati listu Q[i] kao praznu listu;
      for i := k downto 1 do
          while GL nije prazna do {razvrstavanje po i-toj cifri}
              skini x iz GL; {x je prvi elemenat sa liste}
              c := i{ta cifra x; {gledano sleva udesno}
              ubaci x u Q[c];
              for t := 0 to d – 1 do {objedinjavanjee lokalnih
lista}
                   uključi Q[t] u GL; {dodavanje na kraj liste}
      for i := 1 to n do {prepisivanje elemenata iz GL u niz x}
           skini X[i] iz GL
  end
```

Алгоритам 3.0 подразумева да је познат очекивани број цифара елемената, односно број са највише цифара. Алгоритам се лако модификује да прима један аргумент мање и сам проналази број цифара највећег елемента. Ово је имплементирано у алгоритму написаном на Пајтону.

```
  """Алгоритам 3.1 –Пајтон- СОВР """
  def pronadji_cifru(el,i,osnova):
      for j in range(i):
          el=el/osnova
      return int(el%osnova)

  def sortiranjeObjedinjavanjem(GL,osnova):
      maks_cifra = len(str(max(GL)))
      for i in (list(range(maks_cifra))):
          Q=[[]for _ in range(osnova)]
          while len(GL) > 0:
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
            x =GL[0]
            GL = GL[1:len(GL)]
            c = pronadji_cifru(x,i,osnova)
            Q[c].append(x)
        GL=[el for unutrasnja in Q for el in unutrasnja]
    return GL
```

Имплементација у Пајтону је изведена у компактном облику, по цену читљивости, а у сврху наглашавања могућности које Пајтон има при раду са листама. За почетак, обједињени су улазни низ `X` и глобална листа `GL`. Функција `pronadji_cifru()` је једноставна помоћна функција која враћа цифру на месту `i` елемента `el`.

При проналажењу броја са максималним бројем цифара међу бројевима у `GL` коришћена је уграђена функција `max`, пошто је очигледно да, ако је листа позитивна, највећи елемент нема мање цифара него највећи број.

Додељивање почетних вредности листе односно матрице `Q` се обавља преко скуповне листе која креира одређени број празних листа унутар друге листе, чији број зависи од основе система. На овај начин, индекс врста одговара вредностима тренутних цифара чији се бројеви смештају у матрицу, што је у декадном систему од 0 до 9.

Претпоследња линија главне функције је нарочито специфична, јер илуструје могућност истовременог проласка кроз све елементе матрице. Ова скуповна листа уписује редом вредности врста (или колона) у привремену променљиву `unutrasnja` из које потом ишчитава све елементе уписујући их у листу `GL`.

Алгоритам 3.1 се лако може проширити и за рад над негативним бројевима. За одређивање максималног броја цифара користи се претраживање по апсолутним вредностима елемената.

```
  maks_cifra = len(str(max([abs(i) for i in GL])))
```

Остатак наведеног кода већ сортира негативне и позитивне елементе независно једне од других у правилном редоследу, стога их на крају треба само раздвојити и негативне поставити на почетак листе. Коришћењем скуповних листи ово се обавља у три реда.

```
neg=[el for el in GL if el<0]
    GL=[el for el in GL if el>=0]
    GL=neg+GL
```

Скуповне листе заобилазе спољну примену `for` петљи. Овакав приступ је и интуитиван, јер ако се прави петља само како би се обавиле неке радње над низом, онда је потпуно разумљиво да петља буде део низа, а не обрнуто, како је то

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

уобичајено у програмским језицима. Употреба ових петљи не ремети могућност анализе сложености алгоритма, пошто је у сваком тренутку очигледан број корака петље у оквиру скуповне листе.

Дословна имплементација овог алгоритма у програмском језику C има значајне препреке. Уколико се не користи структура повезаних листа, врло је тешко конструисати динамичну матрицу за расподелу елемената у зависности од текуће цифре. Програмски језик C не даје могућност несиметричних дводимензионалних низова. Једно решење за овај проблем је означавање фиктивног краја низа специјалним карактером, који наравно мора бити истог типа као и елементи који се распоређују. Стога у обзир долазе или екстремне вредности одговарајућег типа променљивих, на пример 2^{16}-1 за `unsigned short` или нека негативна вредност нпр. -1 уколико се сортирају позитивни бројеви. Овакав приступ је очигледно непрактичан, те се уместо матрице најчешће користи низ који пребројава елементе са одређеном цифром на текућем месту, слично као код сортирања пребројавањем (counting sort). Међутим, овај приступ се удаљава од основне идеје алгоритма представљене имплементацијом у псеудокоду.

Стога би аналогни алгоритам у програмском језику C могао да има следећи облик:

```
/*Алгоритам 3.2 –C- СОВР */
#include <stdio.h>

int cifra(int a,int mesto, int osnova){
    int i;
    for(i=1;i<=mesto-1;i++)
        a=a/osnova;
    return(a%osnova);
    }
int maks_cifa(int a[], int n, int osnova){
    int c=1,i;
    for (i=0;i<n;i++){
        if (c<br_cifara(a[i]))
            c=br_cifara(a[i]);
    }
    return c;
}
int br_cifara(int x,int osnova){
    int c=1;
    do {
       x=x/osnova ;
       c++;
```

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    }while (x>0);
    return c;
}
void prolaz(int a[],int n,int mesto,int osnova){
     int pc[10]; int pomocni[100]; int i,d;
    for(d=0;d<osnova;d++)
        pc[d]=0;
    for(i=1;i<=n;i++)
        pc[cifra(a[i],mesto,osnova)]+=1;

    for(d=1;d<osnova;d++)
        pc[d]=pc[d]+pc[d-1];
    for(i=n;i>=1;i--){
        pomocni[pc[cifra(a[i],mesto,osnova)]]=a[i];
        pc[cifra(a[i],mesto,osnova)]-=1;
        }
    for(i=1;i<=n;i++)a[i]=pomocni[i];
}
void radixSort(int a[],int N,int osnova){
     int p,cif;
    maks_cifa(a,N,osnova);
    for(p=1;p<=5;p++)
        prolaz(a, N, p,osnova);
}
```

Имплементација у програмском језику C је знатно дужа. Као прво за одређивање максималног број цифара, која се у Пајтону проналази у једном реду, треба употребити две петље. У алгоритму 3.2 то је обављено у оквиру функција `maks_cifa` и `br_cifara` преко директног бројања цифара и тражења максимума међу елементима низа. Алтернативно се може користити приступ сличан као оном у Пајтону, преко функције `itoa()`, која претвара број са прослеђеном основом у ниску, али при проналажењу броја карактера опет треба да се искористи петља преко које се проналази крај речи. У случају програмског језика C овај приступ није значајно бољи од наведеног. Уместо проналажења стварне максималне вредности цифара, ограниченост опсега примитивних типова у програмском језику C може се обавити имајући у виду максимални број цифара које програм може да обради за специфичан тип, на пример 9 за `unsigned long`. Како је овај број константа, она не утиче на укупну временску сложеност алгоритма, али ипак има одређени утицај на ефикасност.

Укупна временска сложеност овог алгоритма је O(nk), пошто се главна петља алгоритма извршава k пута, унутар које је n пролазака кроз сваки елемент низа и то два пута у случају псеудо-кода и Пајтона.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Грахамов алгоритам

Грахамов алгоритам (Graham scan) је врло ефикасан алгоритам за одређивање конвексног омотача произвољног скупа тачака у равни или простору. Прво се одабира екстремна тачка а затим се остале тачке сортирају према неком критеријуму, најчешће према нагибу праве која спаја почетну тачку са осталим. У главном делу алгоритма тачке се повезују по сортираном редоследу уз корекцију, уколико је нарушена конвексност многоугла.

Следећи псеудо алгоритам подразумева сортирање тачака у правцу казаљке на сату почевши од крајње десне доње тачке.

```
  Алгоритам 4.0 –Псеудокод- Грахамов алгоритам
  Ulaz: p1, p2, . . . , pn (skup tačaka u ravni).
  Izlaz: q1, q2, . . . , qm (konveksni omotač tačaka p1, p2, . . . ,
pn).
  begin
      Neka je p1 tačka sa najvećom x-koordinatom (a najmanjom y-
koordinatom, ako ima više tačaka sa najvećom x-koordinatom);
      Sortiranje tačaka po uglu koji zaklapaju sa x osom. Neka je
redosled p1, p2, . . . , pn;
      q1 := p1;
      q2 := p2;
  q3 := p3; {put P se na početku sastoji od p1 , p2 i p3}
      m := 3;
      for k := 4 to n do
          while ugao između – qm–1 – qm – i – qm – pk–) ≥ π do
              m := m – 1;
          m := m + 1;
          qm := pk
  end
```

Имплементација овог алгоритма у псеудокоду је прилично једноставна, међутим за функционалан алгоритам потребно је обично знатно више линија кода. С синтакса захтева дефинисање структуре тачке, док Пајтон омогућава рад са листом уређених парова. Највећу тежину алгоритма носи сортирање тачака и по питању временске сложености и по питању имплементације.

Испод је дата једна имплементација Грахамовог алгоритма у Пајтону:

```
  """Алгоритам 4.1.1 –Пајтон- Грахамов алгоритам"""
  import sys
  #Funkcija koja pronalazi krajnju desnu i donju tačku
  def ekstremna(tacke):
      maks=tacke[0]
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
      for t in tacke:
          if t[0]>maks[0] or (t[0]==maks[0] and t[1]<maks[1]):
              maks=t
      return maks


  #Za odredjivanje udaljenosti od početne tačke, dovoljno je pronaći
kvadrat rastojanja
  def udalj(p,q):
      return (p[0]-q[0])**2+(p[1]-q[1])**2


  def sortira(tacke):
      prva=ekstremna(tacke)
  #Rečnik u kome će se čuvati težinske vrednosti tačaka u odnosu na
nagib
      tabela={}
  #Kako vertikalne prave imaju nedefinisan nagib njihove vrednosti
čuvamo u posebnoj promenljivoj vertikalne
      vertikalne=[]
      for t in tacke:
          if t[0]!=prva[0]:
              nagib=(t[1]-prva[1])/(t[0]-prva[0])
              if not nagib in tabela:
                  tabela[nagib]=t
  #Ukoliko dve tačke imaju isti nagib vrši se pomeraj za minimalnu
vrednost realnog broja za Pajton15


              elif udalj(prva,t) > udalj(prva,tabela[nagib]):
                   tabela[nagib+sys.float_info.min]=t
              else:
                  tabela[nagib-sys.float_info.min]=t
          else: vertikalne.append(t)


  #sorted() funkcija primenjena nad rečnikom vraća sortiranu listu
ključeva
  #Lista se obrće jer se u smeru kazaljke na satu nagibi zapravo
smanjuju
      kljucevi=sorted(tabela,reverse=True)
      for i in range(len(tabela)):
          tacke[i]=tabela[kljucevi[i]]
```

[15] sys.float_info.min je reda 10^{-308}

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
  #Dopunujemo listu tačaka sa prestalim tačakama sa istom x
vrednosti kao početna
      vertikalne.sort(reverse=True)
      tacke[len(tabela):len(tacke)]=vertikalne
      return tacke
  #Provera da li je došlo do neželjenog skretanja pri formiranju
poligona preko znaka vektorskog proizvoda"""
  def levo_skretanje(p,q,r):
      if(p[0]-q[0])*(r[1]-q[1])-(p[1]-q[1])*(r[0]-q[0])>0:
          return True
      else: return False


  def graham(tacke):
      tacke=sortira(tacke)
      omotac=tacke[0:2]
      for t in tacke[2:]:
  #Ukoliko nova tačka narušava željeni smer, izbacujemo predhodne
tačke iz omotača
          while levo_skretanje(t,omotac[-1],omotac[-2]):
              omotac.pop()
          omotac.append(t)
      return omotac
```

Алгоритам 4.1.1 се доследно води наведеним псеудо алгоритмом 4.0 (и савета изворне књиге [2]). У омотачу се налазе све граничне тачке, дакле и оне које се налазе на страницама многоугла.

Укупна сложеност овог алгоритма је $O(n \log n)$, јер је ограничена доњом границом за сортирање тачака. Само формирање алгоритма, након сортирања, захтева $O(n)$ времена пошто се свака тачка уклања највише једанпут. Позивање уграђених `sorted()` и `.sort()` функција не нарушава сложеност алгоритма пошто се позивају константан број пута и обављају се за време $O(n \log n)$. Избегнуто је коришћење уграђених функција за одређивање нагиба и растојања, како би се добио бољи увид у ефикасност.

Други начин за имплементацију Грахамовог алгоритма је преко измењене верзије представљене од стране А.М. Андруа[16], која сортира тачке по x координатама уместо по углу и затим проналази омотач истовременим проналажењем горњег и доњег дела омотача. Овај алгоритам је драстично краћи и боље користи могућности Пајтона.

[16] Andrew, A. M. (1979). "Another efficient algorithm for convex hulls in two dimensions"

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
  """Алгоритам 4.1.2 –Пајтон- Грахамов алгоритам Андру"""
  import functools

  def skretanje(p, q, r):
      if(q[0] - p[0])*(r[1] - p[1]) - (r[0] - p[0])*(q[1] - p[1])>0:
          return 1 #skretanje na levo
      else:
          return -1
  def _keep_left(omotac, r):
      while len(omotac)>1 and skretanje(omotac[-2], omotac[-1],r)!=
1:
          omotac.pop()
  # Proveravamo da li je omotac[-1] != r kako bi smo pokrili
degenerativne slučajeve
      if not omotac or omotac[-1] != r:
          omotac.append(r)
      return omotac

  def convex_omotac(tacke):
  #Vraća tačke konveksnog omotač u smeru suprotnom kazaljki na satu
      tacke = sorted(tacke)
      donji= functools.reduce(_keep_left, tacke, [])
      gornji = functools.reduce(_keep_left, reversed(tacke), [])
      #Pri proširivanju donjine uključujemo prvu i zadnju tačku
      donji.extend(gornji[i] for i in range(1, len(gornji) - 1))
      return donji
```

Интерполација

На крају одељка о алгоритама, биће обављено кратко разматрање о могућностима Пајтона у имплементацији интерполационих полинома. Посебно, биће разматран потенцијал Пајтона у замени једног од најпопуларнијих инжењерских алата: Матлаба (MATLAB). Како је Матлаб моћан скриптинг језик са много могућности потпуна компарација би могла да буде рад за себе. Овде ће бити упоређена само два једноставна примера који осликавају разлике и сличности ова два језика.

За почетак је важно навести да одређени пакети Матлаба нису бесплатни, што је у старту знатна препрека за индивидуални рад студената и ученика. И поред

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

широког спектра могућности, Матлаб ипак пружа ужу функционалност од Пајтона, чији су сви додаци бесплатни и лако доступни.

Један од лакших интерполационих полинома нумеричке математике је Лагранжев интерполациони полином. Матлаб користи индексирање низова почевши од 1. Имплементација функције која израчунава вредност прослеђене променљиве, на основу података у датотеци `tablica`, има следећи облик.

```
%Алгоритам 5.1 -Матлаб- ЛИП
function Lagr1 = Lagr1(x)
tablica;
n=length(X);
L=0;
for i=1:n
    p=1;
    for j=1:n
        if j~=i
            p=p*(x-X(j))/(X(i)-X(j));
        end
    end
 L=L+p*Y(i);
end
Lagr1=L;
```

Подаци у датотеци `tablica` су записани у следећем облику

```
X=[0 0.2 0.4 0.6 0.8 1];
Y=[1 2 3 4 5 6];
```

Доследна имплементација у Пајтону је врло сличног минималног кода, уз додатно изостављање ознака за крај линија и блокова наредби.

```
"""Алгоритам 5.2 -Пајтон- ЛИП"""
from tablica import *
def lagranz(x):
    n=len(X)
    L=0
    for i in range(n):
        p=1
        for j in range(n):
            if j!=i:
                p=p*(x-X[j])/(X[i]-X[j])
        L+=p*Y[i]
```


Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
    return L
```

Датотека `tablica` се наравно држи синтаксе Пајтона која се од Матлабове разликује у раздвајању елемената зарезима.

```
X=[0, 0.2, 0.4, 0.6, 0.8, 1]
Y=[1, 2, 3, 4, 5, 6]
```

Као што се може приметити, овај алгоритам је имплементиран без увођења додатних модула.

За потребе писања функције која враћа цео полином у облику:

$$C_n x^n + C_{n-1} x^{n-1} + \cdots + C_1 x + C_0$$

погодно би било коришћење неког од модула за рад са полиномима као што је СајПај. У ту сврху прво се анализира верзија претходног алгоритма у Матлабу који враћа полином као вредност.

```
%Алгоритам 6.1 -Матлаб- ЛИП полином

function LPolinom = LagrPoly()

Tablica;
proizvod = 1;
n=length(X);
suma = 0;
for i = 1:n
    for j=1:n
        if i~=j
           proizvod=conv(proizvod,[1 -X(j)]/(X(i)-X(j)));
        end
    end
    proizvod=proizvod*Y(i);
    suma=suma+proizvod;
    proizvod=1;
end
LPolinom=suma;
End
```

Имплементација алгоритма 6.1 је врло слична претходној верзији, с тим што променљива производ акумулира коефицијенте полинома, наместо коначне вредности преко функције `conv()`.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Имплементација у Пајтону је такође врло елегантна. Једина битна разлика је што у Пајтону у једном тренутку мора да се нагласи да ће операције бити вршене над полиномом преко функције `poly1d()` из модула СајПај. Са друге стране, није неопходна посебна функција како би се нагласило множење полинома, већ Пајтонов интерпретер препознаје оператор множења као одговарајуће множење полинома.

```
"""Алгоритам 6.2 -Пајтон- ЛИП полином"""
from scipy import poly1d
from tablica import *

def LagrP():
    proizvod=1
    n=len(X)
    suma=0
    for i in range(n):
        for j in range(n):
            if i!=j:
                proizvod=proizvod*poly1d([1, -X[j]]) /(X[i]-X[j])
        proizvod=proizvod*Y[i]
        suma=suma+proizvod
        proizvod=1
    return(suma)
```

Излаз који даје Пајтонова интерактивна конзола је у великој мери прецизнији и читљивији од оне коју пружа Матлаб, који низове и полиноме третира једнако, па је стога излаз просто низ коефицијената, без ознаке о ком степену се ради. Излаз претходног алгоритма би могао да изгледа овако у Матлабу:

```
ans =

    7.5000  -19.5000   15.0000    1.0000
```

Где се подразумева да је први број коефицијент уз највећи степен. Уколико исписивање степена није експлицитно наглашено у самом програму, њихове вредности се добијају бројећи од позади од нуле. У овом случају је очигледно да је највећи степен 3, али при дужим полиномима ово може да представља проблем.

Пајтон, односно пакет СајПај, са друге стране уз вредности коефицијената додаје и променљиву x са одговарајућим параметром. Ово се обавља у два реда:

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

```
>>>
     3         2
7.5 x - 19.5 x + 15 x + 1
```

Овај запис је далеко читљивији од излаза који пружа Матлаб. Уз то, ако је неки од коефицијената нула Пајтон их неће штампати, што је ближе математичком запису полинома.

Захваљујући овом додатном модулу Пајтон излази на крај са радом над полиномима исто толико добро као и Матлаб.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Закључак

Настава програмирања у школама би требала да прати динамику којом се развијају информатичке и комуникационе технологије. Увођење нових тема за обраду у школи носи са собом и нове потешкоће за предавача. У овом раду су представљене предности програмског језика Пајтон, у сврху бољег установљавања да ли су ове предности довољне да би се овај језик увео школски систем.

Програмски језик Пајтон и сви његови додаци су потпуно бесплатни, инсталација је врло једноставна и обавља се на сличан начин као и инсталирање било које друге апликације. Пајтон пружа могућност ученику да брзо почне са програмирањем од тренутка упознавања са језиком. Ово је нарочити подстицај за самостални индивидуални рад.

Минималистички кôд и једноставна синтакса чини Пајтон погодним за талентоване почетнике и у основној школи, практично од тренутка када науче основе коришћења рачунара и енглеског језика. Чињеница да се ученик може упознати са синтаксом Пајтона директно из конзоле, на сличан начин на који би се упознавао са напреднијим калкулатором, може омогућити врло благ прелаз на учење напреднијих концепата програмирања. Ово је значајна предност у односу на актуелне програмске језике у образовању који у самом почетку захтевају објашњавање додатних концепата програмирања.

Основне операције у Пајтону су интуитивне и лако прихватљиве почетницима. Иако се у основи мало разликују од других програмских језика, њихова проширена функционалност као што је сабирање ниски и умножавање читавих низова, могу подстаћи ученика на критичко размишљање без бојазни да би прелазак на друге програмске језике могао да створи значајне потешкоће.

Значајна олакшица за почетнике у програмирању су и динамични типови у Пајтону. Преко овог приступа типови података су сакривени од програмера, што омогућава бржи увод у суштинске програмерске концепте. На тај начин, ученик ће само привремено бити ускраћени за неопходна знања о типовима променљивих и алокацији меморије, дајући предавачу већу слободу да конструише наставни програм, сагласно индивидуалним могућностима ученика.

Синтакса Пајтона је тако конструисана да форсира почетника да пише уредан кôд, пошто су блокови наредби одређени размацима односно њиховим увлачењем у односу на претходни део кода. Ова зависност од стила писања кода може се показати проблематичном код почетника, али на овај начин ученик може да усвоји добру праксу писања читљивих програмерских кодова.

Декларација функција у Пајтону је слична са осталим програмским језицима па омогућава лак прелазак почетника и на друге програмске језике. Угњеждене функције у појединим случајевима значајно могу побољшати кôд, нарочито код примене унутар рекурзивних функција. Обимна и стално растућа библиотека додатних модула пружа добру основу за напредније програмирање.

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Класе у Пајтону су такође сличне са класама у другим програмским језицима и омогућавају увођење појмова класа и објеката релативно рано у наставном програму програмирања. Упоређен са други актуелним језицима у програмирању, Пајтон је једини који омогућава истовремено увођење процедуралног, објектно оријентисаног, па чак и функционалног стила програмирања. При учењу Пајтонове синтаксе творци овог програмског језика су оставили места да се почетник сконцентрише на специфичне почетничке проблеме процедуралног програмирања, без потребе за механичком употребом концепата који би били касније разјашњавани. Ово уједно даје и већу слободу наставницима при прављењу плана наставе.

Пајтон се показао као врло добар језик за писање алгоритама. У односу на програмски језик C кôд је доста краћи и често читљивији, истовремено пружајући добар увид у временску сложеност алгоритма. Упоређен са псеудокодом, у појединим случајевима, кôд Пајтона је чак и краћи и подједнако читљив. У случајевима када је Пајтон комплекснији од псеудокода, долази до изражаја могућност покретања и тестирања програма. Пајтон се у примерима показао и као бесплатна и добра замена за Матлаб.

На самом крају погодно је поменути и речи Лај Хетланда на тему потенцијала овог језика: „Пајтон је језик који уме да вам се склони са пута док пишете програме. Омогућава имплементацију функционалности без много муке и допушта писање програма који су јасни и читљиви“ [17].

Узимајући у обзир горе наведено може се примети да је Пајтон заиста врло добра замена за актуелне програмске језике. Његово увођење у школски програм може значајно потпомоћи информатичкој образованости ученика и смањити анксиозност коју овај део информатике често носи са собом, истовремену дајући добру основу за напредније програмирање.

[17] Magnus Lie Hetland, *Beginning Python – From Novice To Professional, (* New York: Apress, 2008), XXIX

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

Литература:

[1] Васиљевић, Небојша, "Зашто је програмски језик Пајтон добар за учење програмирања" објављено у *ИНФОтека*, број 1, јун 2013 http://infoteka.bg.ac.rs/pdf/Srp/2013-1/Programskijezik.pdf (приступљено: август 2014)

[2] Живковић, Миодраг, *Алгоритми.,* Београд: Математички факултет (2000)

[3] Kernighan, Brian W. *Why Pascal is Not My Favorite Programming Language*, AT&T Bell Laboratories, април 1981

[4] Hetland,Lie Magnus, *Beginning Python – From Novice To Professional.* New York: Apress (2008)

[5] Lutz, Mark. *Learning Python*. Sebastopol, California: O'Reilly Media (2013)

[6]Chou, Pai H. "Algorithm Education in Python". Paper presented at the Proceedings of the Python 10 Conference, Alexandria, VA, February 2002

[7] Summerfield, Mark, *Rapid GUI Programing with Python and* QT. Prentice Hall (2007)

[8] Cormen, Thomas H. ; Leiserson,Charles E. ; Rivest, Ronald L. i Stein, Clifford, *Introduction to Algorithms 3, 3rd edition*. Cambridge, Massachusetts: The MIT Press (2009)

[9] Comparing Python and C, Part 1, Integer Variables https://www.daniweb.com/software-development/python/threads/71008/comparing-python-and-c-part-1-integer-variables (приступљено: септембар 2014)

[10] Python: Lambda Functions, http://www.secnetix.de/olli/Python/lambda_functions.hawk (приступљено: септембар 2014)

[11] Pyplot tutorial. http://matplotlib.org/users/pyplot_tutorial.html (приступљено: децембар 2014)

[12] Python Software Foundation. https://www.python.org/ (приступљено: август 2014)

[13] SciPy Tutorial, http://docs.scipy.org/doc/scipy-0.14.0/reference/tutorial/index.html (приступљено: децембар 2014)

[14] Anaconda IDE, https://store.continuum.io/cshop/anaconda/ (приступљено: новембар 2014)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs

[15] The Python Standard Library. https://docs.python.org/3/library/ (приступљено август 2014)

[16] Tutorials Point. http://www.tutorialspoint.com/python/ (приступљено: новембар 2014)

[17] The Future of Java and Python. http://www.hartmannsoftware.com/Blog/Articles_from_Software_Fans/The-Future-of-Java-and-Python (приступљено новембар 2014)

[18] Switzer, Tom, *2D Convex Hulls: Graham Scan* http://tomswitzer.net/2010/03/graham-scan/ (приступљено: новембар 2014)

[19] Visual Basic Guide. http://msdn.microsoft.com/en-us/library/2x7h1hfk%28v=vs.90%29.aspx (приступљено: септембар 2014)

[20] Mathworks, Matlab software http://www.mathworks.com/products/matlab (приступљено : јануар 2015)

[21] Venner, Bill, *The Making of Python: A Conversation with Guido van Rossum, Part I*, januar 2003, http://www.artima.com/intv/pythonP.html (приступљено: јун 2015)

Virtual Library of Faculty of Mathematics - University of Belgrade
elibrary.matf.bg.ac.rs