Небојша Васиљевић

МОДЕРНИЗАЦИЈА УЛОГЕ РАЧУНАРСКОГ ПРОГРАМИРАЊА У ОБРАЗОВАЊУ

Сведоци смо новог таласа увођења програмирања као општеобразовног садржаја у образовне системе многих земаља. Пример је Велика Британија где је од прошле школске године рачунарство обавезан предмет у свим основним школама почев од најмлађих узраста.

Претходни сличан талас отпочео је у време кућних и првих персоналних рачунара крајем седамдесетих и током осамдесетих година. У многим школама широм света почињало је да се учи програмирање, најчешће уз коришћење програмског језика *Basic*. Талас је окончан тако што се у већини школа на крају прешло на коришћење рачунара без програмирања. На крају су преовладале идеје технолошког описмењавања које су искључиво оријентисане на коришћење технологије.

Заправо је у целом том периоду рачунарском образовању приступано као познавању технике, с тим да је у почетку програмирање било саставни део коришћења техничке справе. На пример, поглавља корисничког упутства за Комодор 64 су била: Распакивање и повезивање, Тастатура Комодора 64, Коришћење софтвера, *Basic* – програмски језик, Програмирање у *Basic*-у, итд. Дакле, два поглавља након објашњења како се уређај укључује у струју и шта значи који тастер на тастатури следи увод у један програмски језик. Међутим, разлог томе није било разумевање значаја програмирања, већ чињеница да је *Basic* представљао основни кориснички интерфејс који се појављује када се рачунар укључи. Како је напредно коришћење рачунара престало да претпоставља елементе програмирања, тако је и програмирање нестало из информатичког образовања. То је слично као што је за возачки испит престало да се учи познавање рада четворотактног мотора.

Одређени број стручњака је и тада схватао да програмирање доприноси развоју начина размишљања и разумевању света око себе, баш као математика и физика, али то заправо није био преовлађујући разлог зашто се тада учило програмирање. Све време је техничко описмењавање било и остало доминантан концепт, само што тај концепт у почетку није могао да заобиђе одређени ниво програмирања као напредног облика коришћења технологије. Чак и тамо где је наставило да се учи програмирање у основним школама, приступ учењу програмирања није лако проналазио пут између општетехничког и професионално-програмерског приступа.

Нови талас увођења програмирања у школске курикулуме се заснива на чињеници да постоје одређене способности апстрактног размишљања које су потребне свима за живот и рад у дигиталном добу, а које се набоље стичу кроз програмирање. Дакле, примарни циљ учења програмирања као општеобразовног садржаја није оспособљавање за коришћење технике нити припрема за будуће професионално програмирање, већ стицање специфичних менталних способности.

На пример, неко лакше разуме, а неко теже да уколико систематски обележавамо стилове у *Microsoft Word* документу, моћи ћемо аутоматски да формирамо садржај и можемо аутоматизовати разне друге активности, попут корекције изгледа свих поднаслова. Слично, неко лакше разуме, а неко теже да у *Excel* табели треба увек да додамо нову колону са обележјем када желимо да обележавамо редове, уместо само да колоне бојимо у разне боје, јер ако само бојимо, после нећемо лако аутоматизовати филтрирање редова, сортирати, рачунати збирове по групама итд. Ово је добро да знају сви корисници апликација *Word* и *Excel*, а можемо приметити да особе које знају да програмирају спадају у категорију оних који лакше разумеју ово што причамо.

Примери попут претходна два илуструју једну менталну способност која је опште потребна у дигиталном добу, а која се може описати као генерално разумевање како треба поступити да би се будуће активности могле што боље аутоматизовати. Таква ментална способност се најбоље стиче програмирањем, а улога програмирања је слична улози бављења спортом на часовима физичког васпитања. Циљ физичког васпитања је стицање физичких способности и одговарајућих навика њиховог одржавања, а у ту сврху се практикују одређени спортови. При томе се ученици не припремају да постану активни спортисти нити се удубљују у техничке детаље спорта (детаљна правила, димензије терена, техничке карактеристике лопте, итд), већ се задржава фокус на стицању физичких способности.

Домени примене и алати

Преостаје питање како учити децу да размишљају као програмери, а да их при томе не оптерећујемо стварима које су потребне само професионалним програмерима нити да се задржавамо на бројним техничким детаљима који се налазе „испод хаубе“. Добар одговор на то питање је суштина модерног приступа учењу програмерања. Два кључна дела тог одговора су:

1. Избор алата који ћемо користити за програмирање, што обично укључује програмски језик и развојно окружење у коме ученик обликује и извршава програм.
2. Избор једног или више домена примене програмирања одакле ће се црпити проблеми кроз чије решавање учимо програмирање. Без доброг одговора на овај део питања остаје нам или програмски језик са својим бројним техничким детаљима, који онда постаје централна тема изучавања или класични проблеми које знамо из сопственог рачунарског образовања, што нас води ка професионално-програмерском приступу учењу програмирања.

Та два дела одговора су међусобно повезана, јер проблеме из изабраног домена примене треба да је могуће довољно једноставно и ефикасно решавати коришћењем изабраног алата, тј. карактеристике алата могу да отворе могућност избора одређених домена примене.

Прве кораке у програмирању могуће је направити и са специјализованим алатима који користе сопствени посебно дизајниран програмски језик, а који се може заснивати на слагању графичких елемената (визуелни програмски језик). Како у таквим случајевима ми бирамо алат у целини, неки алати овог типа, као што је *Scratch*[1] могу представљати добар избор за прве кораке у програмирању, посебно код млађих узраста. *Scratch* је посебно прилагођен спектру домена примене који обухвата једноставно дводимензионо цртање, дводимензиону анимацију и дводимензионе рачунарске игре, а све са могућношћу укључивања мултимедијалних елемената попут звука и видеа. Алати који су оријентисани ка таквим доменима примене обично интегришу едитор слике, звука и анимације са могућношћу задавања делова програмског кода којим се контролише понашање одређених објеката[2].

Алати као што је *Scratch* могу бити добар избор за прве кораке у програмирању у млађим узрастима, мада пре или касније у учењу програмирања долазимо и до класичног програмског језика.

Неки типични домени примене које срећемо у наставној пракси, то јест категорије проблема које решавамо програмирањем, су:

- *Занимљиви дводимензиони цртежи* који су погодни за програмско генерисање, где је посебна категорија такозвана корњачина графика[3] чији се корени обично везују за програмски језик *Logo*[4], `http://www.cs.berkley.edu/bh/`, при чему је подршка за корњачину графику такође део стандардне библиотеке неких програмских језика као што су *Python*[5] или *Small Basic*[6].
- *Дводимензиона анимација и рачунарске игре*, за шта се могу користити специјализовани развојни алати за израду дводимензионих игара као што је *GameMaker*[7] (користи сопствени визуелни програмски језик), специјализоване библиотеке као што је *Alegro*[8] (програмски језик `C/C++`) или стандардани апликативни интерфејси за дводимензиону графику у оквиру графичких кори-

[1] `http://scratch.edu/`

[2] Најпознатији широко коришћени алат овог типа, али који није посебно прилагођен потребама учења програмирања је Флеш (енгл. *Flash*, `http://www.adobe.com/products/flash.html`)

[3] Такозвана корњачина графика представља метод програмирања дводимензионе графике који се заснива на метафори кретања објекта који оставља траг, за шта је првобитно коришћен лик корњаче, од чега потиче назив. Суштина је да се без познавања правоуглог координатног система и тригонометријских функција, а са једноставним програмским конструкцијама, могу нацртати занимљиви геометријски облици.

[4] *Computer Science Logo Style, Brian Harvey, MIT Press (3 volumes)*

[5] `http://www.python.org/`

[6] `http://smallbasic.com/`

[7] `http://www.yoyogames.com/`

[8] `https://www.allegro.cc/`

сничких интерфејса као што су *Java 2D* (програмски језик *Java*) и *Windows GDI+* (*C++*, *C#* или *Visual Basic*).

- *Тродимензиона анимација и рачунарске игре*, за шта постоје и специјализовани алати намењени почетницима који уче програмирање, попут алата *Allice*[9] (сопствени визуелни програмски језик) и алати намењени професионалној примени који се могу користити и за учење програмирања, као што је *Unity*[10] (*C#* или *JavaScript*).
- *Кориснички интерфејси* у облику форми и компонената које се распоређују на формама, за шта се у образовној пракси код нас најчешће користе алати *Visual Studio* (*C#, Visual Basic*) и *Delphi* (*Object Pascal*).
- *Израда веб страна са интерактивним могућностима*, за шта се обично користи програмски језик *JavaScript* заједно са језицима *HTML* и *CSS* за опис садржаја веб стране.
- *Детаљи самог програмског језика који се користи* такође могу представљати извор идеја за задавање проблема, што је посебно изражено код програмског језика *C/C++*, при чему је врло важно да се сувише не задржимо на изучавању програмског језика, уместо изучавања програмирања.
- *Елементарни алгоритамски проблеми* који се често формулишу као питање након препричаног стварног или замишљеног догађаја, за шта се може користити било који класичан програмски језик.
- *Такмичарско програмирање* има сличности са претходно описаним доменом, а специфично је по томе што је крајњи домен примене учешће на такмичењу где се програмирање примењује да би се постигао што бољи такмичарски резултат. На вишим нивоима такмичења, а посебно међународним, доминира програмски језик *C++*, а подељена су мишљења у вези тога да ли млађе такмичаре треба од почетка припремати у језику *C++* или прво користити програмски језик који их мање оптерећује.
- *Програмирање робота*, као што је *Lego Mindstorms*[11] са одговарајућим алатима за програмирање.
- *Прављење електронских и електро-механичких склопова који се контролишу рачунарским програмом*, за шта се све више користи мали јефтини рачунар *Raspberry Pi*[12] заједно са програмским језиком *Python*.
- *Домени примене програмирања у градивима других предмета*, као на пример: обрада резултата експеримента, статистичка израчунавања, графичко представљање података, задаци где формуле које чине решење задатка треба да преточимо у програм, анализа текста итд.

Свакако да претходно наведена листа није коначна јер се рачунари све више користе у разним сферама живота и рада, а постоји и мноштво занимљивих домена

[9] `http://www.alice.org/`
[10] `http://unity3d.com/`
[11] `http://mindstorms.lego.com/`
[12] `http://www.raspberrypi.org/`

примене који могу послужити као извор идеја за проблеме који се могу решавати програмирањем.

Када размишљамо о модернизацији учења програмирања, главно је да разумемо значај избора проблема који ће се решавати, јер се креативност и мотивација за решавање проблема доминантно испољава у домену примене, а не у програмирању самом по себи. Наравно, осим уколико се не бавите проблемима као што је пројекторавње алгоритама, али ако кренемо таквим размишљањем брзо ћемо склизнути у професионално-програмерски приступ учењу програмирања.

На пример, код програмирања робота, исти програм који контролише робота може да контролише и виртуелни тродимензиони модел истог таквог робота који се приказује на екрану. Из перспективе програмског кода, у питању је један те исти програм, али је из перспективе домена примене разлика једнако велика као када возите авион на симулатору и када у реалности возите авион. Осећај ученика шта је постигао је потпуно другачији, другачији је ефекат на његову мотивацију и потпуно се разликује оно што ће на крају ученик запамтити.

Слично можемо приметити да је код такмичарског програмирања домен примене câмо такмичење на коме ученик жели да постигне што бољи резултат и мотивисан је да се такмичи са осталима. Било би незанимљиво учити решавање такмичарских задатака и тај тип проблема се не би ни препознао као нека посебна област да нема реалних такмичења. Све у свему, кључно питање код идентификације домена примене јесте у ком тренутку ученик осећа задовољство решавањем проблема.

Без заснивања учења програмирања на занимљивим проблемима из разних домена примене, за већину ученика програмирање ће бити или досадно или претешко. Општеобразовни циљ учења програмирања треба да буде да уз мало програмирања ученик уме да решава проблеме из разних области.

Домени примене програмирања у градивима других предмета

Градиво других предмета представља значајан а недовољно искоришћен извор занимљивих проблема који се могу решавати уз мало програмирања.

На пример, у наставном програму математике за осми разред основне школе постоји област „Графичко представљање података“ са следећим описом садржаја: „Представљање зависних величина табеларно и у координатном систему. Графичко представљање статистичких података у облику дијаграма (стубичастих, кружних, ...). Рачунање средње вредности и медијане. Поређење вредности узорка са средњом вредношћу“.

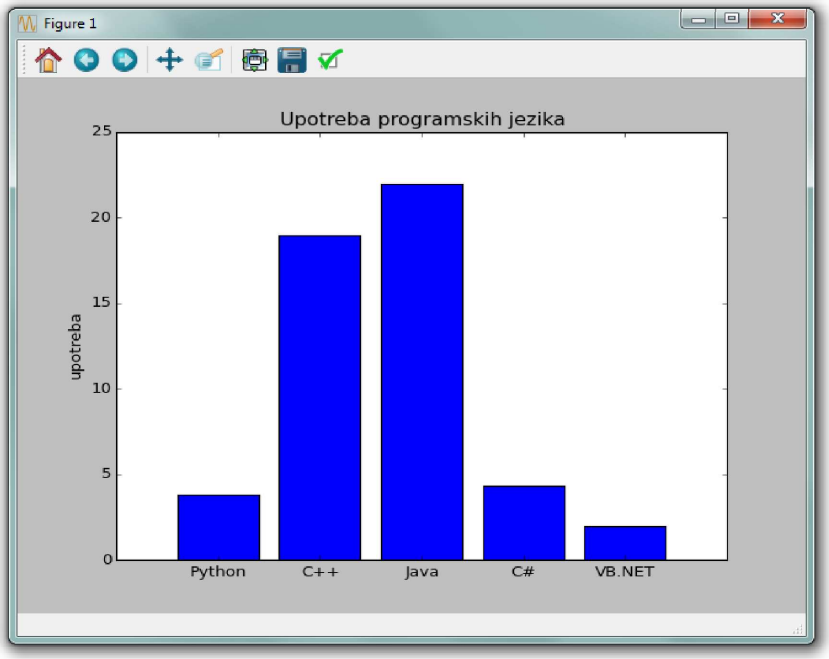
Проблеми који припадају на тај начин описаној области у пракси се решавају готово искључиво уз помоћ рачунара. За то се може користити алат као што је *Microsoft Excel* или неки од алата за статистичка, инжењерска односно научна израчунавања као што су *Matlab*[13], *SPSS*[14] или све популарнији алат отвореног

[13] `http://www.mathworks.com/products/mathlab/`
[14] `http://www.ibm.com/software/analytics/spss/`

кода кратког назива R[15], док од програмских језика опште намене у овој области све значајнију улогу има *Python*. При томе је *Python* генерално погодан за учење програмирања[16] и од недавно је најпопуларнији програмски језик у уводним курсевима програмирања на америчким универзитетима[17].

Следећи *Python* програм формира стубичасти дијаграм за податке о степену употребе појединих програмских језика:

```
from pylab import *
jezici = ['Python', 'C++', 'Java', 'C#', 'VB.NET']
upotreba = [3.8,19,22,4.3,2]
hor_pozicije = range(len(jezici))
bar(hor_pozicije, upotreba, align='center')
xticks(hor_pozicije, jezici)
ylabel('upotreba')
title('Upotreba programskih jezika')
show()
```



Слика 1. Стубичасти дијаграм формиран уз помоћ програма на програмском језику *Python*

[15] http://www.r-project.org/

[16] Н. Васиљевић, *Зашто је програмски језик Пајтон добар за учење програмирања*, Инфотека 14, 1 (јун 2013), 63–76.

[17] Esther Stein, *Python for beginners*, Commun. ACM 58, 3 (2015), 19–21, http://doi.acm.org/10.1145/2716560

Резултат рада програма је приказан на слици 1. За извршавање наведеног програма, поред основне инсталације *Python* окружења, потребни су и одређени додатни модули који се уобичајено користе за научна израчунавања (енгл. scientific computing). Најједноставнији начин да инсталирате *Python* заједно са разним додатним модулима који вам могу затребати је да инсталирате неку од *Python* дистрибуција као што је *Anaconda*[18], *WinPython*[19] или *Python(x,y)*[20].

Проблем сличан претходно описаном је обрада резултата експеримената у физици. Ако смо извели експеримент у коме смо мерили времена проласка објекта кроз неколико тачака, обрада података из тог експеримента, заједно са исцртавањем дијаграма би могла да се обави следећим програмом:

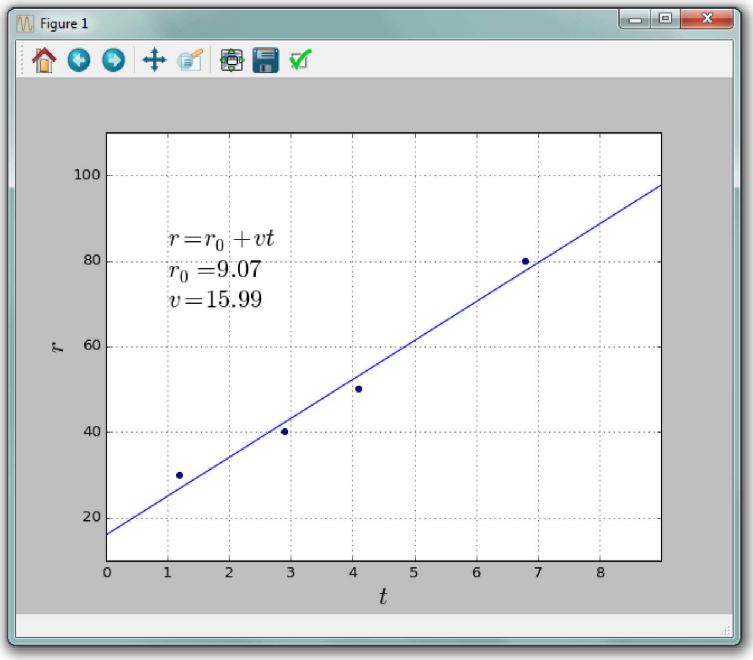
```
from pylab import *
vremena = [1.2, 2.9, 4.1, 6.8]
polozaji = [30, 40, 50, 80]
v, r0 = polyfit(vremena, polozaji, 1)
s = "$r = r_0 + vt$\n$r_0=0:.2f$\n$v=1:.2f$".format(v, r0)
text(1,70,s, fontsize = 20)
plot([0, 9], polyval([v, r0],[0, 9]))
scatter(vremena, polozaji)
xlim(0,9)
xticks(arange(0, 9, 1))
grid(True)
xlabel('$t$', fontsize = 20)
ylabel('$r$', fontsize = 20)
show()
```

Резултат рада претходног *Python* програма је приказан на слици 2. У претходном програму смо користили линеарну регресију (позив функције `polyfit`, где трећи параметар говори да је полином степена један), рачунање вредности полинома у задатој листи тачака (позив функције `polyval` рачуна вредности линеарне функције у тачкама 0 и 9), цртање линијског дијаграма (позив функције `plot`), цртање тачкастог дијаграма (позив функције `scatter`), као и испис текста (позив функције `text`). Приликом исписа текста могу се задавати математичке формуле у TEX-овској нотацији између знакова за долар. Остали позиви функција подешавају начин исцртавања дијаграма.

Ако желимо да формирамо документ који садржи *Python* кôд, резултате израчунавања (графички, табеларно и сл) и текст са додатним објашњењима, можемо користити алат *Jupyter*[21] који омогућава структуирање израчунавања у свеске (енгл. notebook) као што то чини *Matlab*.

Претходна два примера би могла да се реализују и са алатима који не захтевају програмирање. Међутим, чињеница да је израчунавање специфицирано

[18] `http://continuum.io/`
[19] `http://winpython.github.io/`
[20] `http://python-xy.github.io/`
[21] `http://jupyter.org/`



Слика 2. Обрада резултата експеримента

програмским језиком је значајна без озбира што је структура програма у датим примерима тривијална (чиста линијска структура). На овај начин ученик се навикава да се изражава у формалној нотацији и упознаје се са значајем могућности да нешто у програму промени па да пусти израчинавање из почетка, што је битна карактеристика алгоритамског изражавања.

И поред тога што је програмска структура тривијална, ови примери нису сасвим тривијални, потребно је доста тога објаснити ученику пре него што буде оспособљен да самостално решава сличне проблеме. Такође, овакве примере даље можемо проширити гранањима, итерацијама и слично и то у мери у којој желимо да ученику представимо сложеније програмске структуре.

На пример, код *Python* програма који генерише стубичасти дијаграм можемо приметити да није прегледно да подаци о називима програмских језика и о њиховој употреби буду у две одвојене листе, па можемо променити програм тако да на почетку стоји:

```
podaci = [['Python', 3.8], ['C++', 19], ['Java', 22],
          ['C#', 4.3], ['VB.NET', 2]]
```

Затим можемо поставити питање шта треба урадити да би промењљиве `jezici` и `upotreba` добиле исте оне вредности које су имале у претходној верзији програма. Могући одговор је:

```
jezici = []
upotreba = []
for p in podaci:
    jezici.append(p[0])
    upotreba.append(p[1])
```

Примећујемо једноставност рада са елементарним структурама података у програмском језику *Python*. При томе *Python* омогућава и следећи начин одговора на постављено питање:

```
jezici = [p[0] for p in podaci]
upotreba = [p[1] for p in podaci]
```

Ту смо користили синтаксу за конструкцију листе која је аналогна математичкој нотацији за конструкцију скупова, попут $\{f(p) \mid p \in P\}$. Без обзира што се употребљена синтакса за конструкцију листе може сматрати напредном могућношћу *Python*-а и што представља елемент стила функцијског програмирања, за ученика таква синтакса може бити сасвим разумљива и елементарна јер је аналогна нотацији која је ученику позната из математике.

Овде смо се сусрели за чињеницом да се неке технике програмирања и елементи програмског језика традиционално сврставају у напредне, а да су аналогни концепти ученицима добро познати из математике. Типичан пример за то је рекурзија, где ученици релативно рано у оквиру наставе математике упознају рекурзивне дефиниције функција, а са друге стране се рекурзија традиционално сматра напредном могућношћу програмског језика још од времена када први виши програмски језици нису подржавали рекурзију.

Илустровали смо шта значи искористити домен примене програмирања заједно са добрим алатом. Алат који смо користили у овом случају је програмски језик *Python* са скупом модула (тј. програмских библиотека) који подржавају научна израчунавања[22]. Без занимљивог домена примене и одговарајућег алата, користећи само могућности које доноси сам програмски језик, са чисто линеарном структуром програма ученицима не бисмо могли много тога занимљивог приказати и журили бисмо да их научимо сложенијим програмским структурама како бисмо уопште имали шта да им причамо. С друге стране и домен примене треба да има смисла, да сам по себи представља нешто што је вредно изучавати, због чега домени примене из постојећег градива разних предмета имају значајан потенцијал.

Овакви кратки програми који су писани за решавање одређеног задатка користећи готове могућности високог нивоа, где програмски кôд замењује команде које бисмо у неком другом случају задавали кроз кориснички интерфејс, спадају у категорију програма које називамо скриптама.

Скрипте типично немају додатан корак компилације између уређивања текста програма и извршавања. Програмске језике који су погодни за програмирање скрипти називамо скриптним језицима. Главна техничка карактеристика

[22] `http://www.scipy.org/`

скриптних језика је да спадају у програмске језике који се интерпретирају, а не компилирају, мада се „испод хаубе“ може десити одређена компилација пре извршавања, али тако да онај ко прави и користи скрипте о томе не мора да води рачуна.

Скриптни језици су посебно погодни за учење програмирања уз примере из специфичних домена примене. Разлог томе је што они који развијају библиотеке за скриптни језик за подршку одређеним доменима примене имају на уму случајеве коришћења библиотеке где у кратком скрипту треба решити практичан проблем.

Многе апликације имају могућност аутоматизовања операција које нуде корисницима тако што корисницима нуде и могућност писања скрипти. На пример, *Microsoft Office* апликације за ту сврху користе програмски језик *Visual Basic for Application (VBA)*.

Предност програмског језика *Python* у односу на програмске језике попут *VBA* јесте чињеница да је *Python* програмски језик опште намене који је генерално добар за учење програмирања и за који постоји широки спектар модула за разне домене примене. Када ученик савлада програмски језик *Python*, уз мали додатни напор упознаје разне такве модуле.

То наравно не значи да нема смисла упознавати се и са специфичним механизмима аутоматизације рада у појединим апликацијама, а посебно за апликације које се већ обрађују у настави. Овим дотичемо још један домен примене из постојећег градива разних предмета, а то су рачунарске апликације које у настави обрађујемо са корисничког аспекта, а у којима постоје одређене могућности аутоматизације обраде. У тој категорији се посебно истиче *Excel*. Било би корисније да ученик научи да добро користи формуле у *Excel*-у, а да се на рачун тога смањи број осталих апликација које ученик упознаје кроз наставу.

Простор за примену програмирања у градивима разних предмета треба тражити и на местима где је примена технологије у решавању задатака стигла до калкулатора. За почетак, ако *Python* користимо у интерактивном режиму (*Python shell*), онможе да нам замени калкулатор. На пример:

```
>>> 2+2
4
```

„`>>>`“ је одзивни знак (енгл. *prompt*) који стоји испред онога што смо откуцали. Након притиска на *Enter* у следећој линији се исписује резултат.

Калкулатор често користимо у решавању задатака попут: „Ако је човек током времена $t = 3{,}5\,\mathrm{sec}$ прешао пут од $s = 4{,}5\,\mathrm{m}$, колика му је просечна брзина?“ Формула за брзину је $v = s/t$, па кад заменимо бројеве и то унесемо у „калкулатор“ добијамо:

```
>>> 4.5/3.5
1.2857142857142858
```

Пајтон је „паметнији“ од обичног калкулатора, па можемо писати и:

```
>>> t=3.5
>>> s=4.5
>>> s/t
1.2857142857142858
```

Ако имамо резултате пет мерења времена за које је човек претрчао 100 метара, овако бисмо могли да израчунамо брзине:

```
>>> merenja = [15.3, 11.7, 21.9, 13.2, 14.6]
>>> s = 100
>>> [s/t for t in merenja]
[6.5359477124183005, 8.547008547008547, 4.566210045662101,
7.575757575757576, 6.8493150684931505]
```

Сваки задатак из физике у коме дате величине немају задате конкретне вредности, тако да задатак треба решити у општим бројевима, може се формулисати као програмерски задатак. Ако при решавању задатка треба разликовати више случајева, тада програмско решење укључује гранање, на пример, ако треба разликовати да ли се две куглице међусобно сударају пре или након што прва удари у зид. Није тешко направити пример где су нам при решавању задатка потребни итерација и низови, довољно је да имамо већи број објеката са различитим својствима или да имамо већи број фаза у догађају из задатка (на пример више сударања објеката).

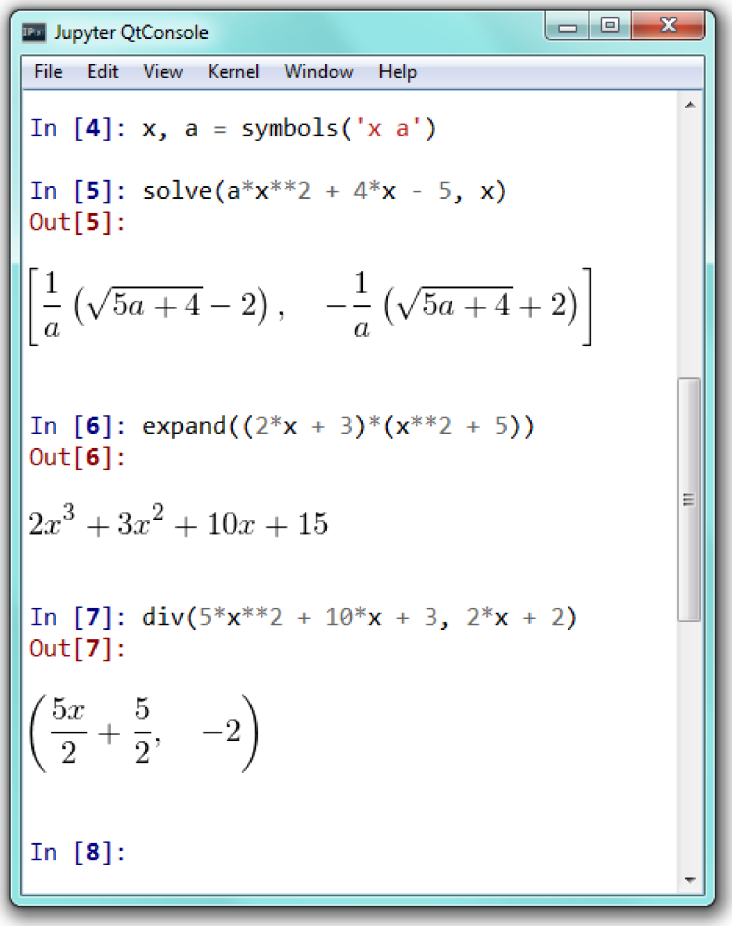
У програмском језику *Python* је могуће и симболичко рачунање уз помоћ *SymPy*[23] модула. Примери коришћења *SymPy* модула у оквиру *Jupyter QtConsole* су приказани на слици 3. Ученицима можемо задавати задатке у чијем се решавању *Python* користи као симболички калкулатор, али тако да и ученици морају да направе неке математичке закључке.

Осим примера које смо овде изложили, *Python* са одговарајућим додатним модулима има добру подршку за статистичка израчунавања, линеарну алгебру, нумеричке методе, али и подршку за домене примене који нису толико непосредно везани за математику, као што је обрада слике, обрада геопросторних података или обрада природног језика.

Такмичарско програмирање

Паралелно са приближавањем основних идеја програмирања ширем кругу ученика кроз погодан избор домена примене и алата који се користе, ученицима код којих се препозна таленат за програмирање треба омогућити да знатно темељније науче програмирање још у основној школи. По том питању постоји сличност са музичким образовањем: талентовани ученици у музичким школама су у стању да савладају много сложеније градиво него њихови вршњаци у оквиру редовне наставе музичког образовања.

[23] `http://www.sympy.org/`



Слика 3. Примери коришћења *SymPy* модула у оквиру *Jupyter QtConsole*

Као и у могим другим предметима, почевши од математике, такмичења су важно средство да се најталентованији ученици заинтересују и да развијају свој таленат у одређеној области. Такмичења имају својих недостатака и ограничења, која се огледају у чињеници да се област изучавања сужава на проблеме који су погодни за такмичења и да од ученика захтевају посебне такмичарске вештине. То све треба имати у виду, али то нису довољни разлози да се негира значај такмичења за подстицање развоја талентованих ученика. По томе се такмичења из програмирања суштински не разликују од такмичења из других предмета.

Специфично за такмичења из програмирања је да припрема за такмичење захтева посебан приступ учењу програмирања за ученике у основним и средњим школама и да не постоји довољно литературе која је прилагођена за такав приступ, што поставља високе затхтеве пред наставника који треба да припрема ученике за такмичење.

Закључак

Главни правац модернизације улоге програмирања у образовању треба да буде приближавање доменима примене. Потребно је направити спој програмирања и корисничког приступа учењу алата. Код корисничког приступа доминира домен примене: у програму за цртање правимо занимљиве цртеже, у програму за обраду видео материјала правимо филмове, итд. Разлог због кога постоји потреба да шира популација има основне представе о програмирању треба уједно да буде и водиља како програмирање учити као општеобразовни предмет, а то је као практично средство за решавање разних проблема.

Сваки приступ учењу програмирања претпоставља постојање алата који омогућавају такав приступ. Након што изађемо из оквира специјализованих алата добро прилагођених за прве кораке у програмирању, као што је *Scratch*, долазимо до тога да нам је потребан скриптни језик који је истовремено довољно једноставан за учење, довољно изражајан и над којим су надограђени алати за разне домене примене. При томе као посебно значајан извор домена примене треба имати у виду градива других предмета.

E-mail: nebojsa.vasiljevic@gmail.com