НАСТАВА МАТЕМАТИКЕ
LXV, 1-2 (2020), 8–29

Др Филип Марић

КОРЕЛАЦИЈА НАСТАВЕ ИНФОРМАТИКЕ, МАТЕМАТИКЕ И ФИЗИКЕ КРОЗ ПРОГРАМСКИ ЈЕЗИК PYTHON

1. Увод

Почевши од школске 2018/2019. године сви ученици основних школа у Србији у шестом разреду у склопу обавезног предмета „Информатика и рачунарство“ изучавају основне концепте програмирања коришћењем текстуалног програмског језика. У најновијим програмима наставе и учења за гимназије такође је приметна већа заступљеност тема из програмирања у односу на раније доминантне теме из области елементарне дигиталне писмености (пре свега коришћења апликативног софтвера).

Иако избор програмског језика није фиксиран наставним програмом, препорука је да се користи неки од савремених скриптних језика. Наставници и аутори уџбеника се углавном опредељују за програмски језик Python, пре свега због његове популарности, једноставности и богате колекције библиотека погодних за изучавање почетних корака програмирања. Ширу дискусију о предностима коришћења програмског језика Python у модернизацији улоге рачунарског програмирања у образовању дао је Н. Васиљевић у чланку [1]. Основе коришћења језика Python могу се видети у чланцима М. Чабаркапе [4,5].

Осим избора језика веома је значајан и избор апликативног програмског интерфејса (енгл. application programming interface, API). Уз неизбежно писање конзолних апликација са најједноставнијим, командно-линијским интерфејсом, препоручује се коришћење специјализованих АПИ за учење програмирања (нпр. робот Карел или корњача графика) и АПИ заснованих на 2д графици у координатама (нпр. библиотека Pygame).

Савремено образовање инсистира на брисањима класичне поделе између предмета и интегралном приступу решавању проблема. Увођење елемената програмирања у обавезан програм наставе отвара простор за нове начине корелације наставе рачунарства са наставом других предмета. Информатика нуди алатке које су корисне у свим сферама људског деловања. Програмирање, као уско специфична дисциплина, може да се примени и у друштвено-хуманистичким наукама

Проширено пленарно излагање на Државном семинару о настави математике и рачунарства, Београд, 08.02.2020.

(пре свега као механизам обраде података). Ипак, најјача и најприроднија је веза програмирања са наставом математике и одређеној мери физике и хемије. Често се дешава да су задаци који се решавају у настави математике и програмирања идентични, али се разликују технике решавања. Математички задаци су веома погодни за увођење концепата програмирања, а програмирање може допринети једноставнијем решавању математичких задатака (често експерименталном методом, која је запостављена у традиционалном математичком образовању) и премештању фокуса са механичког израчунавања на моделовање проблема.

2. Корелације у настави

У наставку ћемо кроз низ пригодних примера покушати да илуструјемо одређене могућности за јаче повезивање наставе информатике, математике и физике. Већина примера доступна је у приручницима за наставу програмирања у основној и средњој школи [2,3], на порталу `https://petlja.org`.

2.1. Основна аритметичка израчунавања

Већ приликом покретања развојног окружења за програмски језик Python покренута је командна линија у коју је могуће уносити изразе чија се вредност одмах израчунава. Подржане су све основне аритметичке операције над целим и над реалним бројевима (`+`, `-` за сабирање и одузимање, `*` за множење, `/` за реално дељење и `//` и `%` за целобројни количник и остатак), уз уобичајени приоритет и употребу заграда.

```
>>> 3 + 5
8
>>> 5 - 3
2
>>> 123 * 456
56088
```

```
>>> 123456 / 789
156.47148288973384
>>> 123456 // 789
156
>>> 123456 % 789
372
```

На тај начин Python директно замењује традиционални калкулатор и има га смисла користити где год се у настави иначе користи калкулатор (пре свега у настави физике и хемије, али и у математици, када се ради обрада података која и иначе подразумева допуштену употребу калкулатора).

Наредни значајни корак је употреба променљивих за смештање улазних и резултујућих величина, као и међурезултата израчунавања. На слици 1 (на следећој страни) приказано је решење наредног једноставног задатка у језицима C++ и Python.

Задатак 2.1.1. *Пера има 5 јабука, а Мика има две јабуке више од њега. Колико јабука имају заједно?*

Примећујемо да је решење у језику Python једноставније од онога у језику C++. Програм не садржи „сувишне делове'“, а променљиве је могуће користити

без потребе за претходном декларацијом (тип се одређује на основу вредности додељеној промењивој). Можемо слободно констатовати да се програм у програмском језику Python заправо скоро ни по чему не разликује од прецизно записаног математичког решења овог задатка. Стога се инсистирањем да ученици запишу решења задатака овог типа у облику програма заправо увежбава прецизан запис математичких решења, са чим, услед недоследности наставника приликом оцењивања, неки ученици имају проблема, чак иако умеју коректно да реше задатак. Рачунар постаје тај који контролише и оцењује коректност и прецизност ученикових модела решавања задатка.

C++

```
#include <iostream>
using namespace std;

int main() {
    int pera, mika;
    pera = 5;
    mika = pera + 2;
    zajedno = pera + mika;
    cout << zajedno << endl;
    return 0;
}
```

Python

```
pera = 5
mika = pera + 2
zajedno = pera + mika
print(zajedno)
```

Слика 1. Решење задатка 2.1.1 у програмском језику C++ и програмском језику Python

Приметимо, даље, да је за решавање задатка било довољно прецизно моделовати проблем, а да је израчунавање (у овом примеру сабирање бројева 5 и 2, а затим сабирање броја 5 са добијеним збиром 7) препуштено рачунару. Сасвим је јасно да механичко израчунавање развија одређене когнитивне вештине ученика и стога је са ученицима млађих узраста неопходно инсистирати на вештини рачунања без коришћења помоћних уређаја (рачуњаљки, калкулатора, рачунара). Међутим, у неком тренутку се може очекивати да су ученици ту вештину савладали и сврсисходно је фокус решавања проблема пребацивати на друге аспекте (као што се, на пример, у решавању задатака из физике израчунавање од почетка сматра неинтересантном, тривијалном вештином и допушта се употреба калкулатора).

Прикажимо још неколико примера текстуалних задатака директно преузетих из тренутно важећих збирки задатака из математике и физике и њихових решења у програмском језику Python.

Задатак 2.1.2. *Кина је 2018. године имала 1 427 647 786 становника и процењује се да ће у наредне 3 године расти по стопи од 0,59% годишње. Колико ће становника Кина имати 2021. године?*

Решење захтева разумевање процентног рачуна.

```
stopa_rasta = 0.59
kina2018 = 1427647786
kina2019 = kina2018 * (1 + stopa_rasta / 100)
kina2020 = kina2019 * (1 + stopa_rasta / 100)
kina2021 = kina2020 * (1 + stopa_rasta / 100)
print(kina2021)
```

За разлику од претходног задатка у којем се изгубила одређена вредност тиме што ученици нису морали да ручно извршавају аритметичке операције, у овом задатку је потпуно јасно да ручно рачунање представља озбиљну препреку у решавању те да решавање задатка уз помоћ рачунара има много више смисла него без њега.

Решавање задатака на овај начин има смисла и у настави физике, где је употреба калкулатора дозвољена од самог почетка. Илуструјмо то једним једноставним задатком из редовног градива за 6. разред.

Задатак 2.1.3. *Возило је за 90 минута прешло 100 километара, а затим је за наредних сат времена прешло још 60 километара. Колика је његова средња брзина на том путу?*

```
t_ukupno = 90 / 60 + 1 # h
s_ukupno = 100 + 60 # km
v_sr = s_ukupno / t_ukupno # km/h
print(v_sr)
```

2.1.1. Решавање задатака у општим бројевима

У претходним задацима улазни параметри су били фиксирани у самом тексту задатака и изрази у решењима су садржали конкретне бројевне константе. Изражавање решења у „општим бројевима“, тј. „извођење формуле“ омогућава да се истовремено реши цела фамилија задатака са различитим улазним параметрима (у изразима тада не фигуришу конкретне вредности, већ промењљиве). С обзиром на то да се у настави математике практично истовремено изучава и моделовање проблема и техника конкретног рачунања, велика већина задатака је формулисана над конкретним улазним подацима и не инсистира се довољно на извођењу формула над „општим бројевима“, пре преласка на крајње израчунавање решења. Добри наставници физике су често много доследнији у овоме него математичари и инсистирају на том да се конкретне улазне величине не смеју уврстити док се не изведе коначна формула. Ако у програмима подаци нису дати у старту, него се учитавају приликом покретања програма, тада се решавање задатака своди на извођење формула и њихово кодирање у програмском језику. То је тешко, јер је решавање задатака у општим бројевима когнитивно захтевније него решавање задатака са конкретним бројевима, али је вредно труда. Илуструјмо ово кроз неколико задатака из редовног градива за ниже разреде ОШ.

Задатак 2.1.4. *Јова склапа играчкице од лего-коцкица. Жели да склопи један ауто за који су му потребна 4 точка, 8 великих и 4 мале коцке и један бицикл за који су му потребна 2 точка и 6 малих коцки. Ако је цена точка 79 динара, цена мале коцке 59, а цена велике коцке 99 динара, напиши програм који израчунава колико је динара потребно Јови да би купио све потребне делове.*

Пошто су сви улазни параметри познати, задатак се може решити писањем једног дугачког израза.

```
>>> 4*79 + 8*99 + 4*59 + 2*79 + 6*59
```

С друге стране, реално је очекивати да се цене могу у неком тренутку променити, па је вредно труда написати програм који изражава укупну цену аутомобила и бицикла у функцији задатих цена коцкица и точкова.

```
cena_tocak = 79; cena_velika = 99; cena_mala = 59;
cena_automobil = 4*cena_tocak + 8*cena_velika + 4*cena_mala
cena_bicikl = 2*cena_tocak + 6*cena_mala
cena_ukupno = cena_automobil + cena_bicikl
print(cena_ukupno)
```

За разлику од претходног, ово решење је много разумљивије, а омогућава да се на веома лак начин добије резултат и након промене основних цена. На крају, уместо да се цене задају на почетку програма пре његовог покретања (што је честа пракса приликом писања скриптова у скриптним језицима какав је Python), те цене се могу учитати приликом покретања програма.

```
cena_tocak = int(input())
cena_velika = int(input())
cena_mala = int(input())
```

Задаци у којима се тражи извођење и кодирање формула са општим бројевима саставни су део такмичења из програмирања за ученике 5. и 6. разреда. Искуство показује да чак и најбољи ученици (они који учествују на такмичењима), често имају проблем са решавањем оваквих задатака, као и да је проблем управо у недостатку одговарајућих математичких вештина (јер ученици нису навикли да решавају задатке у којима улазни подаци нису конкретно задати). На пример, преко 30% петака, учесника квалификација за информатичка такмичења 2019/2020. године није решило наредни задатак и то пре свега јер нису умели да коректно ураде математички део (задатак је са програмерског становишта потпуно тривијалан).

Задатак 2.1.5. *Андрија и Бојан су у кафићу попили по један (исти) сок. Андрија је дао a, а Бојан b динара, а добили су кусур од k динара. Како треба да поделе кусур да би једнако платили?*

Два сока су коштала $a + b - k$, а један сок $s = (a + b - k)/2$ динара. Према томе, Андрија треба да добије $a - s$ а Бојан $b - s$ динара.

```
andrija_dao = float(input())
bojan_dao = float(input())
kusur = float(input())
sok = (andrija_dao + bojan_dao - kusur) / 2
andrija_kusur = andrija_dao - sok
bojan_kusur = bojan_dao - sok
print(andrija_kusur, bojan_kusur)
```

Инсистирање на решавању проблема у општим бројевима може разоткрити неке једноставније поступке од оних које ученици користе. Илуструјмо ово једним примером из збирке задатака из математике за 6. разред.

Задатак 2.1.6. *а) Координата тачке M је 3, а координата тачке N је 9. Одреди координату тачке S која је средиште дужи MN.*

б) Координата тачке M је -1, а координата тачке N је 5. Одреди координату тачке P која је средите дужи MN.

Ови задаци су једноставни па се решење очекује само у облику координате средишње тачке (не очекује се поступак решавања). Ученик до тачног решења првог дела задатка може доћи следећим поступком: „Од 3 до 9 имам 6. Пола од тога је 3. Када се од 3 померим удесно за 3, дођем у 6“. Програм заснован на овом поступку био би формулисан на следећи начин:

```
M = 3; N = 9
S = M + (N - M) / 2
print(S)
```

Иако интуитивно веома јасан и природан, овај поступак је компликованији од следећег, који ће употребити знатно мањи број ученика. Заиста, колико год концепт аритметичке средине два броја био природан особама са математичким искуством, деци може бити неинтуитивно да сабирају два броја да би нашли њихово средиште – то је нешто што треба да науче.

```
M = 3; N = 9
S = (M + N) / 2
print(S)
```

Што се тиче другог дела задатка, ситуација је још компликованија. Ученици ненавикнути на рад са негативним бројевима често до решења долазе веома компликованим, заобилазним путем: „Од -1 до 0 имам 1, и од нуле до 5 имам 5, па је то укупно 6. Пола од тога је 3 и када се од -1 померим за 3, до нуле ми оде 1 и онда још 2 и дођем у тачку 2“. Инсистирањем на писању програма који решава задатак без обзира на знак улазних величина ученици се наводе на то да усвоје најједноставније поступке за решавање проблема, а у овом конкретном примеру да схвате да се појмови растојања тачака и аритметичке средине два броја у неизмењеном облику преносе са позитивних рационалних на све рационалне бројеве.

У физици је сасвим уобичајено изражавање формула у општим бројевима. Илуструјмо то једним примером задатка са општинског такмичења за 6. разред.

Задатак 2.1.7. *Облак и аутомобил су се кретали дуж истог правца, један другом у сусрет. Облак је био кружног облика полупречника* 5 km *и кретао се брзином од* 50 km/h. *Аутомобил се кретао брзином од* 30 km/h. *Колико километара је аутомобил возио кроз олују, ако је прошао кроз њен центар?*

```
v_az = 30 # km/h  # brzina automobila u odnosu na zemlju
v_oz = 50 # km/h  # brzina oblaka u odnosu na zemlju
r_o = 5 # km      # poluprecnik oblaka
s_ao = 2 * r_o    # put koji automobil predje kroz oblak
v_ao = v_az + v_oz # brzina automobila u odnosu na oblak
t_ao = s_ao / v_ao # vreme koje automobil putuje kroz oblak
s_az = t_ao * v_az # put koji automobil predje u odnosu na zemlju
                   # dok se krece kroz oblak
print(s_az)
```

Нагласимо да су сви претходни задаци били такви да се резултати директно израчунавају на основу датих улазних величина (применом неких једноставнијих или компликованијих формула). С друге стране, у одређеним задацима се решења добијају постављањем и решавањем једначина или система једначина. Такви задаци се не могу директно решавати применом класичног програмирања (јер класични програмски језици, па ни Python, немају могућност директно решавања једначина). У таквој ситуацији често је потребно једначину решити (на симболичком нивоу), да би се онда крајње израчунавање препустило рачунару. Илуструјмо ово једним примером.

Задатак 2.1.8. *Ако је збир неке три странице правоугаоника* m*, а збир неке друге три странице истог правоугаоника* n*, написати програм којим се одређује обим и површина тог правоугаоника.*

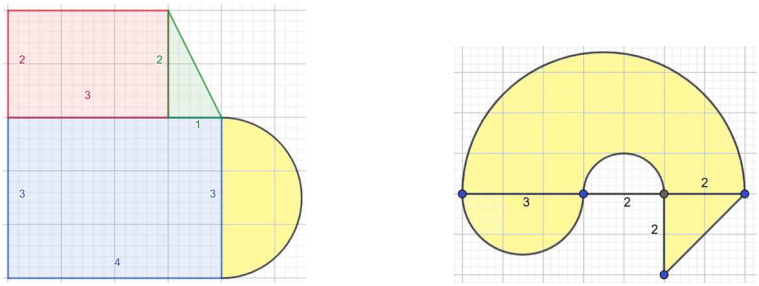
Обележимо странице правоугаоника са a и b. Тада је обим правоугаоника $2 \cdot a + 2 \cdot b$, а површина је $a \cdot b$. Ако је први збир три странице правоугаоника $2 \cdot a + b$ онда је збир друге три странице $a + 2 \cdot b$. Тако добијамо две једначине $2 \cdot a + b = n$ и $a + 2 \cdot b = m$ из којих можемо одредитии a и b. Један начин да се дође до решења је да се реши систем једначина неком од класичних метода решавања система једначина, чиме се добија да је $a = \frac{2n-m}{3}$ и $b = \frac{2m-n}{3}$, на основу чега се веома једноставно одређују обим и површина као $O = 2(a + b)$ и $P = ab$.

```
n = float(input())  # zbir neke tri stranice pravougaonika 2a+b
m = float(input())  # zbir neke tri stranice pravougaonika 2b+a
a = (2 * n - m) / 3 # stranica a
b = (2 * m - n) / 3 # stranica b
O = 2 * (a + b)     # obim pravougaonika 2(a+b)
P = a * b           # povrsina pravougaonika ab
```

2.2. Увођење појма функције

Дефинисање функција је један од основних корака разлагања проблема на потпроблеме и грађења апстракција у програмирању. Ученицима појам функције није довољно близак и имају потешкоће са његовим усвајањем Функције у почетку могу бити веома једноставне и изграђене на основу најпознатијих математичких формула. Пожељно је увести их у примерима где се исто израчунавање понавља на много места у програму.

Задатак 2.2.1. *Израчунати површину сложених облика са слике 2.*



Слика 2. Облици чију површину треба израчунати

```
# povrsina pravougaonika
def P_pravougaonika(a, b):
    return a * b

# povrsina pravouglog trougla
def P_pravouglog_trougla(a, b):
    return a * b / 2

# povrsina kruga
def P_kruga(r):
    return r * r * math.pi

# povrsina polukruga
def P_polukruga(R):
    return P_kruga(R / 2) / 2
```

```
P1 = P_pravougaonika(2, 3) +
    P_pravougaonika(3, 4) +
    P_pravouglog_trougla(2, 1) +
    P_polukruga(3))

P2 = P_polukruga(7) -
    P_polukruga(2) +
    P_polukruga(3) +
    P_pravouglog_trougla(2, 2)

print(P1, P2)
```

У физици се, рецимо, могу дефинисати функције за конверзију јединица (што је честа потреба у реалним задацима). На пример, наредна функција конвертује брзину дату у километрима на сат у метре у секунди.

```
def ms(kmh):
    return kmh * 1000 / 3600
```

2.3. Геометрија

Изучавање геометрије се веома лепо може повезати с рачунарском графиком.

3.2.1. Корњача графика

Популаран апликативни програмски интерфејс кроз који се, још од програмског језика Logo из 1960-их, често уводе основни концепти програмирања је *корњача графика*. Курсор (популарна „корњача“) помера се по екрану и креира цртеж остављајући траг на екрану током свог померања. Програмер управља курсором помоћу следећих наредби:

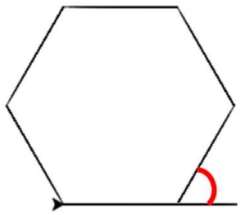
- `forward(x)` – иди напред x корака
- `left(a)` – окрени се улево a степени
- `right(a)` – окрени се удесно a степени

На пример, квадрат се може нацртати тако што се четири пута понови корак напред за дужину странице и окрет за 90 степени. Краћи програмски кЬд се добија ако се за понављање наредби употреби петља.

```
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
```

```
for i in range(4):# ponovi 4 puta
    forward(100)  #    idi napred 100 koraka
    left(90)      #    okret nalevo 90 stepeni
```

Природно уопштење претходног програма је да се нацрта правилан многоугао са n страница.



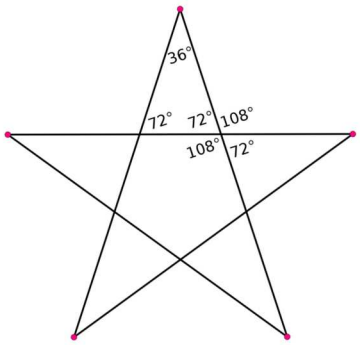
Да би се то десило, корњача у сваком кораку треба да се окрене за спољашњи угао тог многоугла, па је потребно одредити меру тог угла у степенима. Иако је могуће позвати се на теорему која се обично доказује у склопу наставе математике која тврди да је мера тог угла $\frac{360^\circ}{n}$ (ова теорема се обично доказује израчунавањем збира унутрашњих углова разлагањем n-тоугла на $n-2$ троугла), могуће је ту чињеницу једноставно илустровати и на језику корњаче. Наиме, корњача започиње цртање у неком положају, током цртања многоугла окрене се тачно n пута и на крају заврши цртеж у истом положају у ком је цртање започето. Пошто

се окренула тачно за пун круг тј. за $360°$ у n једнаких окрета, приликом сваког окрета корњача се окрене за $\frac{360°}{n}$.

```
n = 6
for i in range(n):# ponovi n puta
    forward(100)# idi napred 100 koraka
    left(360/n) # okret nalevo za spoljasnji ugao n-tougla
```

Многи математички задаци у којима се захтева израчунавање углова и дужина дужи могу се задати као проблеми корњача-графике. На тај начин се појачава мотивација ученика (углове не рачунамо само зато што се у задатку тако тражи, већ што су нам они потребни да бисмо нацртали неки интересантан цртеж). Решења тих геометријских проблема су директно проверива на рачунару, што омогућава учење уз помоћ експериментисања (у претходном програму ученици често погоде вредности за троугао, четвороугао, петоугао, шестоугао, па из тога уоче правилност).

Задатак 2.3.1. *Написати програм који исцртава звезду са 5 кракова.*



Ако се звезда црта са самопресецањем (како је приказано на слици), онда се црта 5 дужи, и након сваке корњача се окреће надесно, за величину суплемента оштрог угла при врху крака звезде.

Тај угао се може израчунати на следећи начин. У средини звезде налази се правилни петоугао, чији је сваки унутрашњи угао једнак 108 степени. То је могуће закључити тако што се тај петоугао подели на 5 подударних једнакокраких троуглова, чији углови при врху испуњавају цео круг од $360°$, па је сваки од тих углова једнак $360° : 5 = 72°$, док су углови на основици тих троуглова једнаки $(180° - 72°) : 2 = 54°$. Унутрашњи угао петоугла обухвата два таква угла, па је једнак $2 \cdot 54° = 108°$. Краци звезде су једнакокраки троуглови, чији су спољашњи углови на основици једнаки $108°$, па су унутрашњи углови на основици једнаки $180° - 108° = 72°$. Зато су унутрашњи углови при врху кракова звезде једнаки $180 - 2 \cdot 72° = 36°$. Дакле, приликом цртања звезде помоћу 5 дужи које се међусобно секу, корњача након сваке од 5 нацртаних дужи окреће надесно за угао од $180° - 36° = 144°$.

Након ове математичке анализе, сасвим је једноставно написати програм.

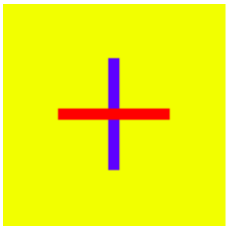
```
for i in range(5):
    forward(100)
    right(144)
```

2.3.2. 2д графика са кооординатама

Корњача графика је одлична техника за вежбање углова. Ипак, традиционално се рачунарска графика заснива на концепту координата. Велики број језика програмерима на располагање ставља неки механизам за исцртавање основних геометријских облика (тачака, квадрата, кругова, правоугаоника, елипси, кружних лукова) задавањем координата. Облици су смештени у Декартов правоугли координатни систем, с тим што се све дешава у једном квадранту коме је координатни почетак у горњем левом углу екрана, x-координата расте надесно, а y-координата расте наниже Треба бити обазрив, јер се у математици разматра другачије оријентисана y-оса.

За многе програмске језике постоје библиотеке које дају подршку за рад са 2д графиком. Једна таква библиотека у програмском језику Python је библиотека Pygame[1] која служи за програмирање једноставних 2д игара. Илуструјмо употребу ове библиотеке кроз неколико једноставних примера.

Задатак 2.3.2. *У центру прозора димензије 200 пута 200 пиксела, нацртај знак плус чије су дужине страница 100 пиксела. Дебљина линије је 10 пиксела, а боје подеси као на слици*[2].



Пошто се дужи цртају задавањем координата крајњих тачака, централно место решења је одређивање тих координата. Центар екрана има координате $(100, 100)$, па крајње тачке вертикалне дужи имају координате $(100, 50)$ и $(100, 150)$, а док крајње тачке хоризонталне дужи имају координате $(50, 100)$ и $(150, 100)$ (од централне тачке померамо се за 50 пиксела, што је пола дужине дужи, по одговарајућој координати у одговарајућем смеру).

```
# bojimo pozadinu u zuto
prozor.fill(pg.Color("yellow"))
# vertikalna plava linija duzine 100 piksela
```

[1] `https://pygame.org`

[2] Верзија овог чланка с илустрацијама у боји може се наћи на сајту часописа Настава математике `https://dms/rs/casopis-nastava-matematike/`

```
pg.draw.line(prozor, pg.Color("blue"), (100, 50), (100, 150), 10)
# horizontalna crvena linija duzine 100 piksela
pg.draw.line(prozor, pg.Color("red"), (50, 100), (150, 100), 10)
```

Задатак 2.3.3. *Напиши програм који црта небо, сунце и облак, као на слици.*

Сунце и облаци се добијају цртањем кругова. Кругови се цртају задавањем координата центра и полупречника.

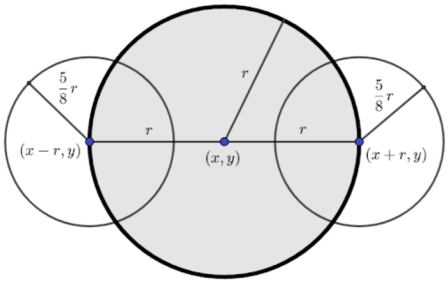
```
# bojimo pozadinu u plavo
prozor.fill(pg.Color("skyblue"))
# crtamo sunce
pg.draw.circle(prozor, pg.Color("yellow"), (100, 100), 80)
# crtamo oblak od tri kruga
pg.draw.circle(prozor, pg.Color("white"), (200, 200), 80)
pg.draw.circle(prozor, pg.Color("white"), (120, 200), 50)
pg.draw.circle(prozor, pg.Color("white"), (280, 200), 50)
```

Релативно задавање координата и димензија. Већа флексибилност програма се постиже када се координате одређују у односу на неке кључне тачке, уместо да се наведу у апсолутном облику. На пример, када се плус нацрта на следећи начин, његов положај се не мења ако се промени димензија прозора, док се његова величина једноставно мења изменом само једног броја у програму. Поступак израчунавања крајњих тачака дужи исти је као и раније, једино што су сада све формуле изражене у општим бројевима.

```
(cx, cy) = (sirina / 2, visina / 2) # koordinate centra ekrana
duzina = 100 # duzina linije je 100 piksela
debljina = 10 # debljina linija je 10 piksela
# vertikalna plava linija
pg.draw.line(prozor, pg.Color("blue"),
                 (cx, cy - duzina/2), (cx, cy + duzina/2), debljina)
# horizontalna crvena linija
pg.draw.line(prozor, pg.Color("red"),
                 (cx - duzina/2, cy), (cx + duzina/2, cy), debljina)
```

Положај објекта одређен је положајем једне његове кључне тачке (она се понекад назива *сидро*). Величина објекта је задата помоћу једног бројевног параметра (*фактора скалирања* у односу на неку основну величину). На пример,

код облака сидро може бити у центру великог круга, а величина може бити одређена полупречником тог круга. Дефинисањем процедура које као параметре имају положај, величину и боју облика, можемо једноставно нацртати више сличних облика.

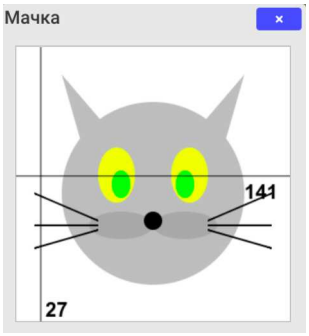


```
# procedura koja crta oblak na datoj poziciji, date velicine
# u datoj nijansi sive boje
def oblak(x, y, r, siva):
    boja = [siva, siva, siva] # nijansa sive boje
    # centralni veliki krug
    pg.draw.circle(prozor, boja, (x, y), r)
    r_malo = round(5 * r / 8) # poluprecnik manjih krugova
    # levi manji krug
    pg.draw.circle(prozor, boja, (x-r, y), r_malo)
    # desni manji krug
    pg.draw.circle(prozor, boja, (x+r, z), r_malo)

# crtamo nekoliko oblika razlicite velicine i nijanse sive boje
oblak(240, 200, 40, 180)
oblak(270, 250, 50, 210)
oblak(230, 100, 50, 230)
```

Симетрије. Често су објекти симетрично распоређени у односу на неку праву линију. Ако је права постављена хоризонтално или вертикално, симетрично пресликавање се врши прилично једноставно. На пример, тачка која је симетрична тачки (x, y) у односу на вертикалну праву $x = x_0$ је $(x_0 + (x_0 - x), y)$ ако је $x_0 > x$, тј. $(x_0 - (x - x_0), y)$ ако је $x_0 < x$ (поново, деци је много једноставније да симетрично пресликају тачку са конкретним координатама, него да изкажу општу формулу).

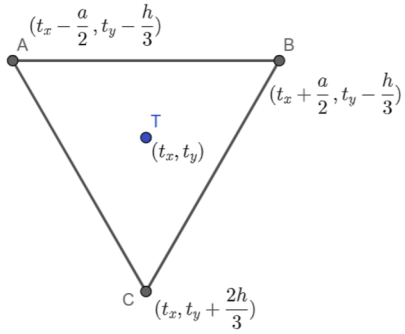
На пример, приликом цртања мачке приказане на наредној слици, апликација допушта да се очитају x координате само у левом делу прозора, док се координате тачака у десном делу морају израчунати применом симетрије у односу на средину екрана $x_0 = 50$.



2.3.3. Израчунавање координата применом Питагорине теореме

Да би се израчунале координате, понекад је потребно примењивати формуле које се изучавају у старијим разредима ОШ.

Задатак 2.3.4. *Напиши програм који исцртава саобраћајни знак приказан на слици. Шта тај знак поручује учесницима у саобраћају?*



За цртање једнакостраничног троугла потребно је применити израчунавање висине Питагорином теоремом и теорему о положају тежишта.

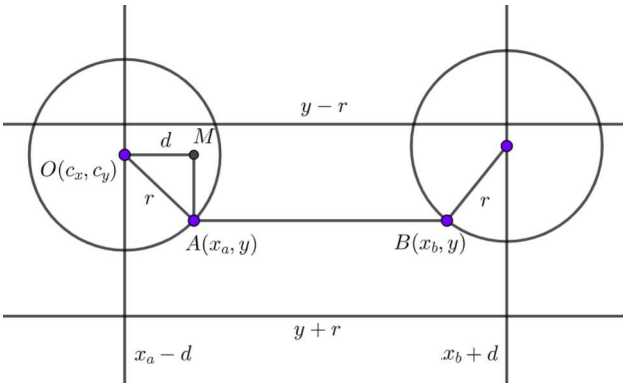
```
def jedakostranicni_trougao(tx, ty, h, boja):
    a = h * 2 / math.sqrt(3)     # duzina stranice
    # koordinate temena - teziste deli visinu u odnosu 1 :  2
    A = (tx - a/2, ty - h/3)
    B = (tx + a/2, ty - h/3)
    C = (tx, ty + 2*h/3)
    pg.draw.polygon(prozor, boja, [A, B, C])
```

Задатак 2.3.5. *Важан задатак у игрици у којој лоптица пролази кроз препреке је испитивање да ли се круг и правоугаоник секу. Дефинисати функцију која то испитује.*

Иако је овај задатак могуће решити применом Питагорине теореме и елементарних својстава координата, он је ученицима прилично тежак, јер решење захтева моделовање проблема на какво нису навикли у редовној настави математике.

С обзиром на димензије, круг сече правоугаоник ако и само ако сече неку од његових ивица. Стога је потребно дефинисати функције које одређују да ли круг сече вертикалну, односно да ли круг сече хоризонталну дуж.

Када круг са центром $O(c_x, c_y)$ полупречника r сече хоризонталну дуж AB, ако је $A(x_a, y)$ и $B(x_b, y)$? Пресек може да постоји само ако је вертикално растојање између центра круга и хоризонталне праве на којој се налази дуж мање или једнако r тј. пресек може да постоји само ако важи $|c_y - y| \leqslant r$, односно $y - r \leqslant c_y \leqslant y + r$. Ако је овај услов испуњен, пресек може, а не мора да постоји – то зависи од x-координате центра c_x. Гранични положаји центра се одређују на основу кругова који додирују дуж. На основу Питагорине теореме можемо израчунати да је то интервал $[x_a - d, x_b + d]$, где је $d = \sqrt{r^2 - (c_y - y)^2}$.




```
def krugSeceHorizontalnuDuz(cx, cy, r, xa, xb, y):
    if abs(cy - y) > r:
        return False
    d = math.sqrt(r**2 - (cy - y)**2)
    return xa - d <= cx and cx <= xb + d
```

Пресек са вертикалним дужима се може испитати аналогно. Ипак, елегантније решење је примена осне симетрије око праве $y = x$. Том трансформацијом се: произвољна тачка (x, y) пресликава у тачку (y, x) круг са центром у тачки (c_x, c_y) полупречника r пресликава у круг са центром у тачки (c_y, c_x) полупре-

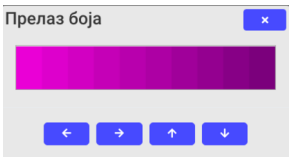
чника r вертикална дуж са теменима (x, y_a) и (x, y_b) пресликава у хоризонталну дуж са теменима (y_a, x) и (y_b, x).

```
def krugSeceVertikalnuDuz(cx, cy, r, x, ya, yb):
    return krugSeceHorizontalnuDuz(cy, cx, r, ya, yb, x)
```

2.3.4. Конструисање линеарних функција

Конструисање линеарних функција се веома често потребно у решавању задатака из различитих области. Илуструјмо неколико примена ове важне технике у настави програмирања рачунарске графике.

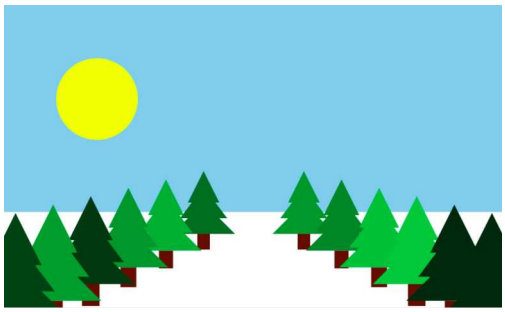
Задатак 2.3.6. *Написати програм који приказује тзв. градијент, тј. прелаз између две насумично одређене боје.*



Претпоставимо да треба да одредимо n нијанси којима ћемо попунити правоугаонике (при чему је почетна нијанса једнака првој, а завршна нијанса једнака другој насумично одређеној боји). Свака се боја представља комбиновањем црвене, зелене и плаве компоненте (користи се тзв. RGB модел боја) и сва обрада се врши посебно за црвену, посебно за зелену и посебно за плаву компоненту боје. Претпоставимо да почетна боја у себи садржи R_p, а да крајња садржи R_k јединица црвене боје. Потребно је конструисати линеарну функцију која слика редни број нијансе i у количину црвене боје у тој нијанси, тако да је за $i = 0$ вредност те функције једнака R_p, а за $i = n - 1$ вредност те функције једнака R_k. Елементарном математиком (на пример, решавањем система једначина по непознатим коефицијентима линеарне функције) добијамо да је тражена функција $R(i) = R_p + i(R_k - R_p)/(n - 1)$. На исти начин одређујемо и количину зелене, односно плаве боје у свакој нијанси i.

```
n = 10     # broj nijansi
sirina_polja = sirina / n
visina_polja = visina
(rp, bp, gp) = nasumicna_boja()   # nasumicno odredjena pocetna boja
(rk, bk, gk) = nasumicna_boja()   # nasumicno odredjena krajnja boja
for i in range(0, n):
    r = round(rp + i*(rk - rp)/(n - 1))
    b = round(bp + i*(bk - bp)/(n - 1))
    g = round(gp + i*(gk - gp)/(n - 1))
    pg.draw.rect(prozor, (r, g, b),
                 (i*sirina_polja, 0, sirina_polja, visina_polja))
```

Задатак 2.3.7. *Напиши програм који распоређује јелке као на слици.*



Претпоставимо да на располагању имамо процедуру `jelka` која црта јелку на основу задатих координата централне тачке дна стабла, висине и нијансе боје. Претпоставимо да промењиве `sirina` и `visina` садрже ширину и висину екрана, док промењива `horizont_y` садржи y-координату линије хоризонта. Горње стабло левог дрвореда почиње на 40% ширине екрана и 10% висине екрана испод хоризонта и висине је 150 пиксела. Свако наредно стабло је од претходног лево за 7,5% ширине екрана и испод за 5% висине екрана, при чему је више за 20 пиксела. На основу ових параметара можемо конструисати линеарне функције које за стабло са редним бројем i одређује x-координату, y-координату и димензију (то што су стабла распоређена дуж једне *праве линије* указује на то да је и функција која описује зависност y у односу на x линеарна, међутим, на овом месту се не конструише та функција, већ линеарне функције које описују зависност координата од вредности параметра који је у овом случају редни број стабла).

```
# crtatmo levi drvored - linearne funkcije
for i in range(broj_stabala):
    x = 0.4 * sirina - i * 0.075 * sirina
    y = (horizont_y + 0.1 * visina) + i * 0.05 * visina
    dim = 150 + i * 20
    jelka(x, y, dim, random.uniform(0.2, 2.0))
```

Десни дрворед се може нацртати по истом принципу (применом симетрије у односу на средину екрана). Ипак прикажимо решење које је мало више у духу императивног програмирања у коме се у сваком кораку извршавања петље мењају вредности промењивих (што није својствено математици).

```
# crtamo desni drvored - azuriranje promenljivih
x = sirina / 2 + 0.1 * sirina
y = horizont_y + 0.1 * visina
dim = 150
for i in range(broj_stabala):
    jelka(x, y, dim, random.uniform(0.2, 2.0))
    x += 0.075 * sirina
    y += 0.05 * visina
```

```
dim += 20
```

Интересантно је продискутовати да је особина која суштински карактерише линеарне функције та да за константни пораст параметра функције тј. променљиве x, константно расте и вредност функције тј. променљива y.

2.4. Анимације и симулације

Осим статичких слика, програмски је релативно једноставно реализовати и анимације које се добијају исцртавањем слика у правилним временским интервалима. Библиотека PyGame чију смо употребу за 2д графику приказали у претходном поглављу може се користити и за креирање анимација. Анимације се могу одлично искористити за прављење физичких симулација (поготово из области кретања). Прикажимо један пример, инспирисан задатком који је ученицима био постављен на општинском такмичењу.

Задатак 2.4.1. *Гепард је неопажено пришао импали до растојања од* 200 m*, и брзином од* 120 km/h *појурио ка њој. Истог тренутка, импала је почела да бежи брзином од* 90 km/h*. Након* 12 s *од почетка трчања, импала је наишла на предео обрастао жбуњем. Импале су одлични скакачи, што им омогућава да наставе трчање истом брзином. Када је гепард наишао на исти овај предео обрастао жбуњем, морао је да смањи брзину на* 100 km/h*, и том брзином је наставио потеру. Гепард је одличан и веома брз тркач, али на кратким релацијама, и није у могућности да јури плен дуже од* 1 km*. Да ли је гепард успео да стигне импалу?*

Израда сваке анимације има неколико делова.

- Потребно је дефинисати променљиве које описују стање објеката на екрану („сцени“), које се мењају током трајања анимације.
- Потребно је дефинисати функцију која црта сцену на основу тренутних вредности променљивих (ова функција се позива у правилним временским интервалима).
- Потребно је дефинисати функцију која описује како се у задатом временском тренутку израчунавају вредности променљивих. То може бити било на основу времена t протеклог од почетка анимације, било ажурирањем вредности променљивих на основу времена Δt протеклог од претходног исцртавања.

У нашем примеру, две основне променљиве су x-координата гепарда x_g и x-координата импале x_i. Иако се оба објекта у задатку посматрају као материјалне тачке, у анимацији морамо користити слике које имају своје ширине, па ћемо претпоставити да те координате представљају координате средина слика. Претпоставићемо да је координатни систем постављен тако да се гепард на почетку кретања налази у координатном почетку (тј. да је $x_g = 0$, а да се импала

налази на положају $x_i = 200$ (јер је на основу текста задатка на почетку она удаљена 200 метара од гепарда). Ове променљиве иницијализујемо на почетку програма, заједно са одређеним бројем константи које представљају параметре задате текстом задатка. Жбуње се не помера током анимације, па је његов положај константан. Међутим, он нам није задат у тексту задатка па га морамо израчунати на основу информације да импала за 12 секунди долази до жбуња (приликом израчунавања морамо водити рачуна о конверзији јединица).

```
# ulazni parametri
v_gs = 120    # km/h     brzina geparda po savani
v_is = 90     # km/h     brzina impale po savani
v_gz = 100    # km/h     brzina geparda po zbunju
v_iz = v_is   # km/h     brzina impale po zbunju

x_g = 0          # m     pocetna pozicija geparda
x_i = 200        # m     pocetna pozicija impale
max_x_g = 1000   # m     razdaljina koju geprad moze da pretrci
t_iz = 12        # s     vreme potrebno da impala dotrci do zbunja
# izracunavamo pocetni polozaj zbunja
s1_i = t_iz * ms(v_is)   # m   put koji impala pretrci do zbunja
x_z = x_i + s1_i         # m   pocetni polozaj zbunja
```

Јако је добро да се све променљиве чувају у својим природним јединицама (на пример, положај треба да буде задат у метрима, а не у пикселима, а време у секундама, а не у фрејмовима), а да се конверзија изврши приликом исцртавања. То прерачунавање се по правилу врши једноставним линеарним функцијама и у програму је потребно чувати коефицијенте тих функција. У нашем примеру ћемо претпоставити да се један метар представља помоћу једног пиксела, па нећемо памтити фактор за прерачунавање метара у пикселе. С друге стране, анимацију ћемо подесити тако да се у једној секунди промени 30 слика и ту константу ћемо памтити у посебној променљивој.

```
fr_s = 30 # 1/s # broj frejmova u jednoj sekundi animacije
```

Сада можемо дефинисати функцију која исцртава објекте на сцени. Да би се сличица исцртала потребно је навести њен горњи леви угао. Пошто знамо x-координату њене средине, да бисмо одредили x-координату њене леве ивице, потребно је од x-координате средине одузети пола ширине слике. Пошто желимо да се сви објекти налазе на дну екрана, y-координату горње ивице сваког објекта одредићемо тако што од висине екрана одузмемо висину објекта.

```
# slicice koje cemo koristiti
gepard = pg.image.load("gepard.png")   # slicica geparda
impala = pg.image.load("impala.png")   # slicica impale
def crtaj():
    # brisemo prethodnu scenu
    prozor.fill(pg.Color("white"))
    # crtamo zbunje
```

```
    pg.draw.rect(prozor, pg.Color("green"),
                 (x_z, visina - 10, sirina - x_z, 10))
    # crtamo geparda i impalu
    prozor.blit(gepard, (x_g - gepard.get_width() / 2,
                 visina - gepard.get_height()))
    prozor.blit(impala, (x_i - impala.get_width() / 2,
                 visina - impala.get_height()))
```

На крају, дефинишимо функцију која помера гепарда и импалу. Померање ћемо вршити тако што ћемо x-координату увећавати за производ брзине (коју одређујемо у зависности од подлоге тј. у зависности од тога да ли се животиња креће по савани или по жбуњу) и протеклог времена Δt, које лако израчунавамо тако што знамо број сличица које се приказују у једној секунди.

```
def pomeri():
    global x_i, x_g   # globalne promenljive koje se menjaju
    if x_g >= x_i:    # ako je gepard stigao impalu
        return        # zaustavljamo animaciju
    dt = 1 / fr_s # s # vreme proteklo izmedju dva frejma
    # pomeramo geparda
    if x_g < max_x_g:      # ako se gepard jos nije umorio
        # brzina geparda u zavisnosti od podloge
        v_g = v_gs if x_g < x_z else v_gz
        # izracunavamo mu novi polozaj
        x_g = x_g + ms(v_g) * dt
    # pomeramo impalu
    # brzina impale u zavisnosti od podloge
    v_i = v_is if x_i < x_z else v_iz
    # izracunavamo joj novi polozaj
    x_i = x_i + ms(v_i) * dt
```

3. Закључци

У чланку је показан низ примера задатака из математике и физике за основну школу, који су решавани у коришћење рачунара, писањем програмског кода у језику Python. У сличном духу могуће је формулисати и компликованије примере, примерене настави у средњим школама (они би обухватали примену тригонометрије или сложенијих закона физике). Осим коришћења рачунара за рачунање и израду анимација и симулација, још једно интересантно поље за повезивање математике и програмирања је примена математичке логике и техника доказивања теорема на резоновање о коректности алгоритама.

Потребно знање програмирања за решавање задатака приказаних у овом чланку је веома елементарно и ученик за њихово решавање не мора да познаје пуно компликованих детаља програмског језика, нити да влада напредним рачунарским алгоритмима (довољно је познавање употребе променљивих, израза, једноставног гранања, петљи и дефинисања и позивања функција, а у области геометрије и

анимације познавање основног програмског интерфејса одабране библиотеке за 2д графику). Све ове теме су обухваћене редовном наставом програмирања у 6. и 7. разреду основне школе.

Централни фокус свих приказаних задатака је, дакле, на математичком моделовању проблема. Колико год да су математички концепти који се користе у приказаним задацима елементарни и увелико обухваћени редовним програмом наставе математике, искуство показује да ученици (па и наставници) веома тешко излазе на крај са њима. Тежину представља то што су сви задаци конструктивистиче природе: од ученика се тражи да изведе формулу или да дефинише функцију која задовољава тражена својства. Од ученика се тражи да примени научено на решавање проблема са каквим се није раније срео, што је само по себи проблематично. То је додатно појачано чињеницом да настава математике под притиском оцењивања користи интензивно „вентил шаблонских задатака“, који омогућава да ученици стекну добре оцене применом углавном репродуктивних знања, без примене концепата који се обрађују и без њиховог креативног комбиновања. Уместо инсистирања на математичком моделовању и извођењу формула, настава често остаје заробљена у механичкој примени унапред датих образаца и у рутини нумеричког и симболичког израчунавања (иако су у последње време приметне одређене промене у позитивном смеру). Искуство аутора на пољу наставе информатике показује да, када стечена математичка знања треба да примене на креативан начин у решавању информатичких задатака, ученици исказују повећано интересовање, али успех често изостаје, јер нису навикнути на такав приступ.

Појава програмирања као обавезног предмета у основним школама и промена приступа настави програмирања у гимназијама су нова реалност која отвара значајан потенцијал за осавремењавање наставе математике и физике, али и јасније позиционирање улоге програмирања у образовању. Мишљење аутора је да је потребно извршити много чвршћу интеграцију наставних програма ових предмета, на свим нивоима, као и реорганизацију редоследа неких наставних тема, у складу са потребама оних других („математика је краљица, али и слушкиња наука“). На пример, елементарна аналитичка геометрија која обухвата рад са координатама и векторима отвара простор за креирање анимација и симулација које могу бити корисне у настави многих природних наука и има смисла размотрити раније и ојачано изучавање ових наставних тема. Јачу интеграцију ових предмета могуће је постићи формирањем јединствене радне групе за израду наставних планова ових предмета (што је по мишљењу аутора много боље у односу на традиционални приступ у коме се за сваки предмет формирају засебне радне групе и доводи до неусаглашених програма наставе).

Теме приказане у овом излагању су често непопуларне и заједнице наставника не показују велико интересовање за њих: информатичари их доживљавају као превише математичке и стога их избегавају (са образложењем да је то деци претешко и да деца не воле математику, па ће се та ,,не љубав'' пренети и на информатику), док их математичари и физичари доживљавају као превише програмерске и обично ни не разматрају могућност било каквог коришћења про-

грамирања у настави. Мишљење аутора је да промена мора доћи одозго – информатичко и математичко образовање наставника мора јаче да се интегрише унутар факултета и осавремени, да би се у будућности промене пренеле наниже.

ЛИТЕРАТУРА

[1] Н. Васиљевић, *Модернизација улоге рачунарског програмирања у образовању*, Настава математике, LX, 3–4 (2015), 19–31.

[2] Ф. Марић и други, *Програмирање у Пајтону, приручник за шести разред*, Фондација „Петља“, 2018.

[3] Ф. Марић, М. Вугделија и други, *Програмирање графике помоћу Pygame, приручник за седми разред*. Фондација „Петља“, 2019.

[4] М. Чабаркапа, *Увод у програмирање коришћењем Python-а*, Настава математике LXIV, 1–2 (2019), 31–40.

[5] М. Чабаркапа, *Python – основни типови података*, Настава математике LXIV, 3–4 (2019), 101–111.

Математички факултет, Београд

E-mail: filip@matf.bg.ac.rs