НАСТАВА МАТЕМАТИКЕ
LXIV, 1-2 (2019), 3–12

Др Драган Машуловић

ПРИМЕНА РАЧУНАРСКОГ ПРОГРАМИРАЊА У НАСТАВИ МАТЕМАТИКЕ[1]

1. Увод

Аутоматизација пословног окружења је прича стара неколико векова. Оно са чим се данас суочавамо, и што ће много интензивније погодити популацију, јесте аутоматизација свакодневних активности коју доносе дигитални асистенти (као што су Siri компаније Apple, асистент компаније Google, и Amazon Alexa) и *Internet of Things*. Експоненцијално убрзање тренда аутоматизације има две важне последице:

- све тривијалне послове ће преузети машине (на пример, Amazon Go продавнице, аутономни аутомобили), и
- постаје скоро немогуће предвидети шта ће бити кључни/добро плаћени послови у блиској будућности.

С друге стране, изложеност ученика модерним (пре свега мобилним) технологијама не доприноси развоју дигиталних компетенција. Разлог за то је веома једноставан. Омасовљавање технологије као процес потекло је од технолошких компанија, а не од државних система. Зато је као основни циљ постављен профит компанија, а не унапређивање дигиталних компетенција популације. У циљу максимизовања профита компаније развијају производе који одговарају текућем стању дигиталних компетенција становништва. У ту сврху се анализирају најчешћи случајеви коришћења (*use cases*) и лансирају производи који раде тачно оно што је оцењено као потреба просечног корисника. Да би се уз то минимизовале потенцијалне правне заврзламе, настају корпорацијске стратегије које охрабрују корисника да производ користи само на начин који је предвидео произвођач. Корисник се обесхрабрује да буде инвентиван. Нема интелектуалних изазова – *корисник треба да буде конзумент технологије.*

[1]Пленарно излагање на Државном семинару ДМС, Београд, 2019. год.

2. Cogito, ergo sum!

У светлу тога, данас више него икада до сада уочавамо у којој мери Декартова мисао *Cogito, ergo sum!* добија на важности. Ми, наставници, треба да припремимо данашње ученике за оно што их чека у будућности. Зато је данас, на почетку 21. века, главна мисија наставника да:

- оспособи ученике да компетентно комуницирају са машинама којима ће бити окружени у сваком аспекту свог живота, и
- оснажи ученике да сами раде на развоју специфичних компетенција у складу са захтевима свог будућег радног окружења фокусирањем на фундаменталне вештине 21. века – решавање проблема и алгоритамско мишљење.

То не треба и не сме да буде мисија само наставника информатике, већ обавеза свих нас који радимо у образовању независно од тога који предмет предајемо. Већ у блиској будућности наћи ћемо се у ситуацији да ће они који не буду могли да се ухвате у коштац са захтевима дигиталне свакодневице бити гурнути на маргине друштва. (У прилог овој бојазни говори све гласнија дискусија о увођењу минималне плате за све пунолетне грађане – *universal basic income* – у богатим западним државама.) Ми, наставници, не смемо дозволити да будуће друштво оцени наш рад као пружање у основи ирелевантних вештина и способности, и да тиме себе сведемо на васпитаче у дневном бораваку за будући друштвени баласт!

3. Модеран ученик – традиционална настава

Традиционални модели наставе математике све мање дају ефекте у образовању дигиталних генерација ученика који су кроз употребу модерних технологија навикли на став

"I want it *all*, and I want it *now*".

Кључне способности и кључни унутрашњи мотиватори за учење су у паду: ученици су све мање радознали, смањује се могућност држања пажње и смањује се жеља за самосталним трагањем за решењем. Тако добијамо конфликт:

Модеран ученик – Традиционална настава.

Традиционални формалистички приступ настави, а пре свега настави математике, није значајније мењан најмање последња два века и постаје све јасније да тако организовано поучавање математике није примерено модерним дигиталним генерацијама.

Уместо тога треба прећи на вођену стратегију покушаја и грешке. Настава пре свега у основној школи не треба да буде формализована. Треба дати приоритет интуитивном усвајању концепата и инсистирати на *идеји* и *процесу* који воде ка решењу, *а не на формалном запису решења*. Оперативно, то значи да:

(1) наставник не објашњава декларативно, већ интуитивно;

(2) *ученик се одмах ставља у контекст у коме мора да решава проблеме*;

(3) наставник се обраћа ученику коректним језиком;
(4) ученик одговара како зна и уме;
(5) наставник исправља „језичке“ грешке без казне, па из почетка.

Зато чак и у настави математике постаје све актуелнији став да на нивоу општег образовања знање треба стицати експериментисањем, из искуства и мноштва примера (проблемски/пројектно оријентисана настава)! Формални аспект математичког знања, који је незаобилазан и од кога нипошто не смемо одустати, мора, међутим, да постане предмет *васпитног, а не образовног* сегмента у настави математике!

4. Зашто наставник треба да предаје уз рачунар?

Развој и приступачност савремених рачунарских технологија је омогућио да се професионални математичари све више у свом раду ослањају на рачунаре. На тај начин се ефикасно проверавају хипотезе (нпр. Риманова хипотеза, и до недавно Велика Фермаова хипотеза) и све чешће се у научним радовима јављају докази који се ослањају на хиљаде сати рада рачунара. Овај тренд је кренуо са доказом Проблема четири боје кога су 1976. приказали Апел и Хакен, а данас се сматра потпуно равноправном стратегијом. Нема никаквог разлога да се овај тренд искључује из савремене наставе математике! У складу са тим, један од начина да се настава математике приближи савременим генерацијама видимо у томе да се у наставу математике уведе експеримент.

Експеримент у настави математике може да послужи као средство за уочавање феномена и као средство за испитивање феномена, док непотпуност експерименталне методе („проверио сам неколико примера; ко ми гарантује да то ради увек?“) може да послужи као кључни мотив за неопходност строгог математичког резоновања. Од недавно у основним школама у Републици Србији је настава информатике постала обавезна, а као основни програмски језик одабран је програмски језик Python као једноставан програмски језик са великом базом бесплатно доступних алата који се редовно одржава и унапређује. Јавно доступни и бесплатни материјали за програмски језик Python постоје чак и на српском језику. Фондација *Петља* је на сајту `www.petlja.org` за програмски језик Python обезбедила комплетну подршку на српском језику која обухвата упутство за инсталацију, упутство за наставнике, синтаксни подсетник за програмски језик, приручник за програмирање, методичку збирку алгоритамских задатака и још много тога.

Уз врло мало почетног улагања већ у нижим разредима основне школе Python може да се укључи у извођење наставе математике просто као калкулатор. Тако ученици стичу основне вештине (покретање програмског окружења, синтакса алгебарских израза) које ће им касније у великој мери олакшати прве кораке у програмирању. Покажимо три примера којима се решавају задаци на нивоу 4. разреда основне школе.

Пример. Милош има 850 динара, Милица 420 више од њега, а Зоран 270 динара мање од Милоша. Колико новца имају заједно?

Решење (папир). $850 + (850 + 420) + (850 - 279)$

Решење (рачунар).

```
> 850 + (850 + 420) + (850 - 279)
```

Пример. Ана је купила 16 бомбона, Бојана за 7 мање од Ане, а Весна два пута више него Ана и Бојана заједно. Колико бомбона су купиле све три заједно?

Решење (папир).

$$\begin{aligned} A &= 16 \\ B &= A - 7 = 9 \\ V &= 2(A + B) = 2(16 + 9) = 50 \\ A + B + V &= 16 + 9 + 50 = 75. \end{aligned}$$

Решење (рачунар).

```
> Ana = 16
> Bojana = Ana - 7
> Vesna = 2  (Ana + Bojana)
> print(Ana + Bojana + Vesna)
```

Пример. Између неких од цифара уписати неки од симбола + или − тако да се добије тачна једнакост: 1 2 3 4 5 6 7 8 9 = 100

Решење (папир). Уз мало мозгања долазимо до решења 1+2+3−4+5+6+78+9 = 100.

Решење (рачунар). На нивоу 4. разреда основне школе овај задатак није лако решити помоћу рачунара.

Тако већ у нижим разредима основне школе видимо да је могуће и *потребно* дискутовати са ученицима о могућности решавања проблема помоћу рачунара. Неки проблеми се, просто, не могу лако решити помоћу рачунара. То наставнику даје одличан мотив да са ученицима продискутује неколико важних тема:

- постоје проблеми који се лако могу решити помоћу рачунара,
- постоје проблеми који се *не могу лако* решити помоћу рачунара,
- па чак и они који се *уопште* не могу решити помоћу рачунара (машине нису свемогуће).

С друге стране, то не значи да треба да одустанемо од употребе рачунара као средства у свакој прилици у којој рачунар може да помогне. На тај начин наставник прави прве кораке у *изградњи здравог односа ученика према рачунарској технологији.*

У петом разреду ученици почињу систематски да уче програмирање користећи визуелно програмско окружење Scratch. У том тренутку је погодно у наставу математике, пре свега када се раде наставне теме везане за геометрију, увести неки систем за динамичку геометрију као што је GeoGebra. Системи за динамичку

геометрију би требало да буду кључни алат наставника математике све док се настава геометрије ослања на демонстрацију феномена, пре него на формално доказивање. Рецимо, ученицима се прво демонстрира чињеница да се праве које садрже висине троугла секу у једној тачки, а доказ овог става ученици можда и неће видети у основној школи. Уколико ову демонстрацију организујемо као експеримент у коме се очекује да се ученици, који су још увек релативно невешти у употреби геометријског прибора, сами уверe да се праве које садрже висине троугла секу у једној тачки можемо лако себе довести у ситуацију да бар један ученик, услед лоше конструкције, не дође до жељеног закључка. Тиме је педагошка функција експеримента изгубљена. Наставник ће, у најбољем случају пресећи даљу дискусију, догматизовати овај феномен, и изгубити интерес још једног ученика. У оном другом случају наставник ће покушати да разреши дилему тако што ће ученику рећи (највероватније гласно и пред целим разредом) да је погрешио, и да је грешка последица његовог невештог коришћења геометријског прибора, и тако сигурно изгубити још једног ученика.

Да би експеримент успео потребно је да га изведе наставник уз употребу неког софтверског алата за прецизне конструкције како би ученици видели да је то заиста тако, а онда може затражити од ученика да понове експеримент у својој свесци уз употребу шестара и лењира. У овом другом сценарију, уколико се код неког ученика три праве не пресеку у једној тачки наставник може да делује брзо и умирујуће: „Видели смо да се праве заиста секу када слику црта машина. Зато данас занимања као што су инжењери и архитекте искључиво користе рачунаре за прецизне конструкције.“

У шестом разреду (и касније) ученици већ стичу систематизовано познавање основа програмирања и пред њих се могу поставити проблеми који су много више усмерени ка увођењу експеримента у наставу математике. Навешћемо сада два везана примера. У првом примеру ученик може лако експериментом да дође до одговора. Други пример поставља ученика у ситуацију где математички модел проблема представља ефикаснији пут ка решењу.

Проблем. Милош у првом тромесечју из музичког има две петице и једну двојку. Колико петица још Милош треба да добије да би наставник морао да му закључи пет на полугодишту?

Решење (рачунар-експеримент). Напишемо функцију која рачуна просек листе:

```
def prosek(L):
  return sum(L)/len(L)
```

и онда пустимо ученике да се поиграју:

```
print(prosek([5,5,2]))
print(prosek([5,5,2,5]))
print(prosek([5,5,2,5,5]))
## itd
```

Решење (папир-егзактно). Решити неједначину

$$(5+5+2+5n) : (n+3) \geqslant 4{,}50.$$

Проблем. У једној генерацији има 150 ученика. На првом писменом задатку из математике у тој генерацији нико није добио оцену 1, 42 ученика је добило оцену 2, а 48 ученика оцену 3. Које оцене треба да добију остали ученици у генерацији како би просек генерације на том писменом био барем 3,50?

Решење (рачунар-експеримент). У овом случају није лако решити задатак без дубљег познавања посебности програмског језика Python.

Решење (папир-егзактно). С друге стране решење на папиру и даље прати исту логику. Потребно је решити неједначину

$$(42 \cdot 2 + 48 \cdot 3 + 4 \cdot n + 5 \cdot (60 - n)) : 150 \geqslant 3,50.$$

Кроз експеримент ученик стиче осећај о томе како се неки феномен (рецимо, просек) понаша. Играње на рачунару може да помогне у решавању једноставнијих проблема, али не може (без много додатног знања из програмирања) да помогне у решавању сложених проблема. Тако експеримент у настави математике можемо да употребимо да истовремено разумемо појам, али и да мотивишемо ригорозно математичко резоновање, односно, потребу да се феномени описују математичким моделима. Док експеримент помаже у стицању интуиције у вези са феноменом, тек формирање модела омогућује да се понашање феномена у потпуности разуме.

Пред крај основне школе, или почетком средње школе, математички проблеми чије решење помоћу рачунара је било изван домета програмерских способности ученика сада могу постати добар мотив за демонстрацију потребе за алгоритамским размишљањем у настави математике и за савладавањем неких програмерских техника. Ево неколико примера.

Пример. Дешифровати сабирање:

$$\mathrm{BOR} + \cdots + \mathrm{BOR} = \check{\mathrm{S}}\mathrm{UMA},$$

где различитим словима одговарају различите цифре. У шуми треба да буде бар два бора, а може их бити произвољно много.

Решење. Овај задатак није лако решити на папиру зато што има 1260 решења. Сва решења се помоћу рачунара могу наћи овако (тако и знамо да их има 1260):

```
for bor in range(100, 999):
  if razlCifre(bor):
    for k in range(2, 9999 // bor + 1):
      suma = k * bor
      if razlCifre(1000 * suma + bor):
        print(k, "*", bor, "=", suma)
```

Функција `razlCifre` проверава да ли су све цифре нког броја различите и може се имплементирати, рецимо, овако:

```
def razlCifre(n):
  c = [0,0,0,0,0,0,0,0,0,0]
  while n > 0:
```

```
    k = n % 10
    n = n // 10
    c[k] += 1
  for i in range(len(c)):
    if c[i] >= 2:  return False
  return True
```

Пример. Посматрајмо полином $f(n) = n^2 - 79n + 1601$. Лако се проверава да су $f(0) = 1601$, $f(1) = 1523$, $f(2) = 1447$ и $f(3) = 1373$ прости бројеви.

(а) Наћи број n са особином да $f(n)$ није прост број.

(б) Наћи најмањи број n са особином да $f(n)$ није прост број.

Решење (папир). (а) Очито је $f(1601) = 1601^2 - 79 \cdot 1601 + 1601 = 1601 \cdot (1601 - 79 + 1)$.

(б) Овај задатак није лако решити без употребе рачунара.

Решење (рачунар-експеримент). Проверићемо да ли је то тачно за првих неколико вредности, рецимо овако:

```
for n in range(100):
  print(n, prost(n*n - 79*n + 1601))
```

Тако лако добијамо да већ за $n = 80$ број $f(n)$ није прост. Функцију `prost` која проверава да ли је број прост може написати наставник као помоћ приликом извођења експеримента, или може дати ученицима да је напишу уколико се ради о посебно мотивисаној групи ученика:

```
def prost(n):
if n <= 1:  return False
  if n == 2 or n == 3:  return True
  if n % 2 == 0:  return False
  d = 3
  while d < n:
    if n % d == 0:  return False
    d += 2
  return True
```

Претходни пример може бити интересантан и када се обрађује математичка индукција, јер показује да иако је неко тврђење тачно за $n = 0$, $n = 1$, $n = 2$, $\ldots$, $n = 79$, то и даље не значи да је то тврђење тачно за свако n.

У последња два примера користимо напредније програмерске вештине за генерисање комбинаторних конфигурација, варијација са понављањем у првом, односно, пермутација у другом примеру. Уколико се одлучи да ова два примера покаже у одељењу у коме ученици још увек нису стекли одговарајуће програмерске вештине, наставник може ученицима дати помоћну функцију која генерише

наредну конфигурацију и показати како се она користи да бисмо проверили све могућности.

Пример. Између неких од цифара уписати неки од симбола + или − тако да се добије тачна једнакост: 1 2 3 4 5 6 7 8 9 = 100.

Решење (рачунар). Низ `op` садржи низ операција које треба уметнути између цифара. Операције + и − имају стандардно значење, док празан стринг означава да ће одговарајуће цифре бити „слепљене“ како би формирале вишецифрени број. Низ `op` се током рада програма мења систематски тако да се испробају све могућности. Операција ће прво из стања + прећи у стање −, а онда ће из стања − прећи у празан стринг. Главни програм се врти у `while` петљи испробавајући све опције док не дође у ситуацију да су све операције сведене на празан стринг. Тада се рад програма завршава. Програм ради тако што између цифара умеће операције и потом позивом функције `eval` рачуна вредност тако добијеног израза.

```
op = ["+", "+", "+", "+", "+", "+", "+", "+"]
while op != ["", "", "", "", "", "", "", ""]:
  s = "1" + op[0] + "2" + op[1] + "3" + op[2]
    + "4" + op[3] + "5" + op[4] + "6" + op[5]
    + "7" + op[6] + "8" + op[7] + "9"
  if eval(s) == 100:  print(s, "= 100")
  sledeci_niz(op)
```

Помоћна функција `sledeci_niz` од датог низа операција генерише наредни у лексикографском поретку.

```
def sledeci_niz(op):
  i = 0
  while op[i] == "":
    op[i] = "+"
    i += 1
  if op[i] == "+":  op[i] = "-"
  elif op[i] == "-":  op[i] = ""
```

Пример (В. Андрић). Распоредити цифре 1, 2, 3, 4, 5, 6, 7, 8, 9 у кућице у изразу испод тако да се добије низ тачних једнакости:

$$\square - \square = \square + \square = \square \cdot \square = \square\square : \square$$

Решење (рачунар). Решење задатка се своди на то да генеришемо све пермутације скупа $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ и да испробамо сваку.

```
p = [1,2,3,4,5,6,7,8,9]
while p != [9,8,7,6,5,4,3,2,1]:
  if p[0] > p[1]:
```

```
    a = p[0] - p[1]
    b = p[2] + p[3]
    c = p[4] * p[5]
    d = (10 * p[6] + p[7]) / p[8]
    if a == b and b == c and c == d:
      print(p)
  sledeca_perm(p)
```

Генерисање пермутација лексикографски је захтеван програмерски задатак и у овом случају мислимо да би било примерено да наставник ученицима представи помоћну функцију која то ради за њих.

```
def sledeca_perm(p):
  n = len(p)
  i = n-2
  while p[i] > p[i+1]:  i -= 1
  j = i + 1
  k = n - 1
  while j < k:
    p[j], p[k] = p[k], p[j]
    j += 1
    k -= 1
    j = i + 1
    while p[i] > p[j]:  j += 1
    p[i], p[j] = p[j], p[i]
    return p
```

5. Закључак

Развој дигиталних компетенција модерних генерација ученика не треба и не сме да буде мисија само наставника информатике, већ обавеза свих нас који радимо у образовању независно од тога који предмет предајемо. Математика пружа једно од првих и најважнијих извориштa идеја које омогућују фокус на развој вештина потребних за решавање апстрактних проблема и алгоритамско мишљење. Традиционални (формалистички) приступ настави математике све мање погодује модерним дигиталним генерацијама. Чак и у настави математике постаје све актуелнији став да на нивоу општег образовања знање треба стицати експериментисањем, из искуства и мноштва примера (проблемска/пројектна настава). Формални аспект математичког знања, који је незаобилазан и од кога нипошто не смемо одустати, мора, међутим, да постане предмет *васпитног, а не образовног* сегмента у настави математике!

Експеримент у настави математике можемо да употребимо да истовремено разумемо појам, али и да мотивишемо ригорозно математичко резоновање, односно, потребу да се феномени описују математичким моделима. Док експеримент помаже у стицању интуиције у вези са феноменом, тек формирање модела омогућује да се понашање феномена у потпуности разуме. Пред крај основне школе, или почетком средње школе, математички проблеми чије решење помоћу рачунара је било изван домета програмерских способности ученика сада може постати добар мотив за демонстрацију потребе за алгоритамским размишљањем у настави математике.

Уграђивањем у наставни процес реалности у којој ученици живе и делећи своје вредносне ставове са ученицима подстичемо развој дигиталних компетенција својих ученика, помажемо ученицима да изграде здрав однос према технологији и оснажујемо ученике да сами развију специфичне компетенције у складу са захтевима друштва будућности фокусирањем на кључне вештине 21. века – решавање проблема и алгоритамско мишљење.

Департман за математику и информатику, Природно-математички факултет, Нови Сад
E-mail: dragan.masulovic@dmi.uns.ac.rs