

# Reinforcing Mathematical Skills in Introductory Programming Courses

Jeffrey L. Duffany<sup>1</sup>, Ph.D.

Universidad del Turabo, Puerto Rico, U.S.A., [jeduffany@suagm.edu](mailto:jeduffany@suagm.edu)

*Abstract— Introductory programming classes are usually part of the core curriculum taken in the first two years by all engineering students. In these classes students learn programming skills by writing, compiling and running computer programs in a chosen computer language. By carefully selecting the programming exercises that are assigned and used in the classroom a variety of outcomes can be achieved depending on what is perceived to be relevant or important to an engineering education. This paper provides an overview of how introductory programming can be used to reinforce basic mathematical skills and gives several specific examples of how this might be done. For example students might be asked to write a computer program to solve the quadratic formula or multiply two square matrices. Not only does this teach the programming language but it also reinforces mathematical skills that are required in more advanced engineering classes.*

*Keywords-- programming, active learning, mathematics.*

Digital Object Identifier (DOI):

<http://dx.doi.org/10.18687/LACCEI2017.1.1.463>

ISBN: 978-0-9993443-0-9

ISSN: 2414-6390

# Reinforcing Mathematical Skills in Introductory Programming Courses

Jeffrey L. Duffany<sup>1</sup>, Ph.D.

Universidad del Turabo, Puerto Rico, U.S.A., [jeduffany@suagm.edu](mailto:jeduffany@suagm.edu)

**Abstract**– *Introductory programming classes are usually part of the core curriculum taken in the first two years by all engineering students. In these classes students learn programming skills by writing, compiling and running computer programs in a chosen computer language. By carefully selecting the programming exercises that are assigned and used in the classroom a variety of outcomes can be achieved depending on what is perceived to be relevant or important to an engineering education. This paper provides an overview of how introductory programming can be used to reinforce basic mathematical skills and gives several specific examples of how this might be done. For example students might be asked to write a computer program to solve the quadratic formula or multiply two square matrices. Not only does this teach the programming language but it also reinforces mathematical skills that are required in more advanced engineering classes.*

**Keywords**-- *programming, active learning, mathematics.*

## I. INTRODUCTION

In introductory programming classes students learn to write computer programs. So why not leverage off of this existing infrastructure to take advantage of an opportunity to reinforce mathematical skills and concepts at the same time? Mathematical concepts can be reinforced in many ways. For example a *for loop* can be used to reinforce the concept of a mathematical summation (e.g., adding the integers from 1 to 100) whereas drawing a flowchart helps to develop logical thinking. In addition, many programming exercises can be viewed as a type of puzzle that needs to be solved by writing the code needed to get the solution. Critical thinking should be a byproduct of the overall engineering education and each course should contribute to its development. This should occur naturally without having to do anything however there is always room for improvement.

Computers have been used in teaching of mathematics since the 1960s and 1970s[1] with various levels of success[2][3]. The fundamental problem is that educational software programs tend to be passive in nature with the student being presented information on a computer screen and later asked to demonstrate knowledge of what was previously presented[3][4]. As a result the benefits of computer-based learning have never reached the potential that was originally envisioned[1][2]. In an attempt to increase the effectiveness of computer-aided learning there has been a trend towards "gamification" of the learning experience[5][6]. Make it more

Digital Object Identifier (DOI): <http://dx.doi.org/10.18687/LACCEI2017.1.1.463>  
ISBN: 978-0-9993443-0-9

ISSN: 2414-6390

**15<sup>th</sup> LACCEI International Multi-Conference for Engineering, Education, and Technology:** "Global Partnerships for Development and Engineering Education", 19-21 July 2017, Boca Raton FL, United States.

entertaining, the theory goes, similar in spirit to the 1970s children's television program "Sesame Street"[10] which worked mainly on the principle of repetition[11]. More recently there has been a trend towards what is called "active learning"[7] where the passive teaching techniques are reduced to a minimum while the student is more "actively" engaged in the learning process through a variety of techniques[7][8]. One of these active learning techniques is called "learning by teaching" and is well documented even outside of the active learning community[9] and that is one of the principles that support using computer programming class to reinforce mathematical concepts and skills in engineering students. The idea is that by programming a computer to solve a problem you are effectively "teaching" that computer how to solve a problem and by extension the student is learning and reinforcing those concepts and skills. In addition to learning by teaching the student is learning by repetition[11]. Being exposed to the mathematical concepts in a variety of contexts is a form of repetition as the student will be exposed through mathematics classes, programming classes and later on in various engineering classes.

## II. MATHEMATICAL CONCEPTS

There is a high degree of synergy between programming and various mathematical topics as can be seen by opening just about any book on just about any programming language[12][13][15]. Books have even been written on using programming languages to teach certain specific branches of mathematics such as statistics[14]. Even books that are not specifically written about programming, such as books on computer algorithms, can be a good source of mathematical concepts that can be reinforced with engineering students[16]. This idea will be further developed with some illustrative examples. For example take multiplication of two square matrices. For many students it is difficult to visualize the multiplication of two matrices however by actually programming it using array structures it will help them to solidify and reinforce their understanding of the concept. The idea is if the students have to program it by hand then they will learn it better and remember it better[11]. This can help them in many different ways. Take for example take Boolean logic. Increased exposure and practice with Boolean Logic can help the students later on in courses such as Digital Logic Design and Computer Architecture. As another example take

imaginary and complex numbers which could help the students later on with Electric Circuits class. Another good example is the quadratic equation that the student learns in mathematics class. As will be seen the solution of the quadratic equation can be used to illustrate several important programming concepts such as Boolean logic and complex numbers.

The following is a representative list of mathematical concepts that frequently show up in engineering courses. This is not intended to be a comprehensive list only representative. The examples were chosen to be easy to program and at the same time illustrate a concept from mathematics that will benefit the student later on.

- A. Boolean Logic
- B. Prime Numbers
- C. Limits and Infinity
- D. Imaginary Numbers
- E. Quadratic Formula
- F. Vectors and Matrices
- G. Probability and Statistics
- H. Chaos Theory

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Figure 1. Truth Table for AND, OR and NOT

#### A. BOOLEAN LOGIC

Boolean logic is a branch of mathematics where the values of the variables are *true* and *false*, usually denoted 1 and 0 respectively. The main operations of Boolean algebra are the conjunction *AND* the disjunction *OR* and the negation *NOT*. In Figure 1 A and B are logic variables which can be either true or false. Figure 1 shows the truth table for the AND and OR and NOT operators. The concept should be familiar to anyone who uses a search engine. AND represents both while OR represents either. Also the value of NOT A is shown. The following program illustrates the concept by printing out a message appropriate to the time of day. For example to print out "good morning" the time must be after 6AM AND before 12PM noon.

```
H = Hour(today)
If (H >= 6 And H <= 11) Then
    MsgBox("good morning")
ElseIf (H >= 12 And H <= 18) Then
```

```
MsgBox("good afternoon")
ElseIf (H > 18 And H <= 23) Then
    MsgBox("good night")
ElseIf (H > 0 And H < 6) Then
    MsgBox("it's past your bedtime ")
End If
```

Figure 2. Program to illustrate Boolean Logic

#### B. PRIME NUMBERS

Prime numbers are very important in mathematics. A prime number is defined to be any integer that is divisible only by itself and 1. Prime numbers include 2,3,5,7,11,13,17, 19, 23... etc. as shown in Figure 3. There is no largest prime number. One famous technique of identifying a set of prime numbers is called the sieve of Eratosthenes first expounded by Eratosthenes in 200 BC[16]. Starting with a set of integers from 1 to n say for example 1 to 100. Now eliminate 1 from that set as it is not considered a prime number even if it meets the definition of a prime number. The next number left is 2. All numbers less than 2 have been eliminated so 2 must be prime by definition. To find the remaining prime numbers make use of the fact that all numbers that are multiples of a prime number can be eliminated. That leaves 3 which is prime. The integer 4 has been eliminated as a multiple of 2 so 5 must be prime. Now eliminate all multiples of 5. Repeat until you reach the square root of n. All remaining numbers that have not been eliminated up until this point must be prime.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Figure 3. Prime numbers less than or equal to 100

Figure 4 is an R language program that implements the sieve of Eratosthenes [16] for the set of integers from 1 to 100. It can easily be modified for any value of n.

```
n<-100
x<-1:n
x[1]<-0
for(i in 2:n){
```

```

if(x[i]!=0){
j<-x[i]+x[i]
while(j<=n){
x[j]<-0
j<-j+x[i]
}}
print(x[x!=0])
}

```

Figure. 4 Program for the Sieve of Eratosthenes in R Language

### C. LIMITS AND INFINITY

Computer programming is well suited for teaching the concept of infinity. If you continually increase the size of a number it will eventually exceed the largest number that can be represented internally. There are many ways to explore the idea of infinity however perhaps the simplest is to use the factorial function. Write a program that calculates 1!, 2!, 3!, etc. until it exceeds the programming language's ability to represent the value of the result. Another way is to calculate the limit of  $1/x$  as  $x$  approaches zero. Start with  $x=1$  then divide it by 10 and calculate  $1/x$ . Eventually this will exceed either the upper or lower limits of the machine. The following computer program calculates  $x!$  for values of  $x$  from 1 to 1000. However at 171! the capability of the language is exceeded and is printed out as Inf (infinity).

```

xfact=1
for (i in 1:1000) {
xfact=i*xfact
print(c(i,xfact))}

170! = 7.257416e+306
171! = Inf
172! = Inf
173! = Inf

```

Figure. 5 Program for Calculating  $x!$  in R language and Output

The largest number that can be represented in R language is  $10^{308}$  and the smallest number is  $10^{-308}$ . Since 171! is greater than  $10^{308}$  R language simply prints out "Inf" which stands for infinity. A similar program can be used as the limit as  $x \rightarrow 0$ . It is the same program except  $x$  is multiplied by 10 and then the reciprocal is calculated. When  $i=308$  R language is able to represent the value of  $x=10^{308}$  and  $1/x=10^{-308}$ . However once the value of  $i=309$   $x$  is represented as infinity and  $1/x$  is represented as 0 by R.

### D. IMAGINARY NUMBERS

Imaginary numbers arise from attempting to take the square root of negative 1. Mathematicians have effectively solved this problem by postulating the existence of an imaginary number called "i". To calculate the square root of a negative number

you simply factor out the -1 and replace it by the imaginary number  $i$ . So for example the square root of -4 is the same as the square root of 4 except it is multiplied by "i". When working with complex numbers and computer programs you have to format the output with the real and imaginary parts. In visual basic you can use MsgBox (x & "i"). Where "&" is the concatenation operator which appends the suffix "i" onto the string or number  $x$ .

### E. THE QUADRATIC FORMULA

The quadratic formula is used to solve the equation  $ax^2 + bx + c = 0$ . Written as computer code it would look something like what is shown in Figure 6. However as shown in Figure 7 there are several possible outcomes depending on the values of  $a$ ,  $b$  and  $c$ . Since a square root is involved you can end up with imaginary roots if the discriminant  $b^2 - 4ac$  is negative. This is not necessarily a serious problem you just have to append the imaginary part of the output with "i" as shown in the MSMsgBox commands at the bottom of Figure 6. This will be a good learning experience as the program will work for some inputs and not others and they will have to figure out why. This will then reinforce concepts of imaginary numbers. Students should create a flowchart before attempting to write the code for this exercise.

```

If (b ^ 2 - 4 * a * c > 0) Then
    r1 = (-b + (b ^ 2 - 4 * a * c) ^ 0.5) / (2 * a)
    r2 = (-b - (b ^ 2 - 4 * a * c) ^ 0.5) / (2 * a)

If (b ^ 2 - 4 * a * c < 0) Then
    r1r = -b / (2 * a)
    r1i = (4 * a * c - b ^ 2) ^ 0.5 / (2 * a)
    r2r = -b / (2 * a)
    r2i = (4 * a * c - b ^ 2) ^ 0.5 / (2 * a)
    MsgBox(r1r & "+" & r1i & "i")
    MsgBox(r2r & "+" & r2i & "i")

```

Figure. 6 Program for Solving the Quadratic Formula

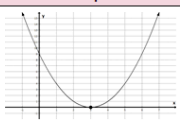
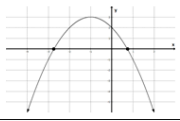
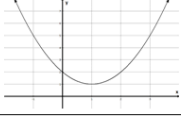
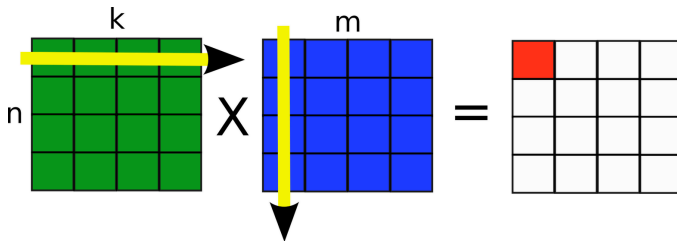
Discriminant	Number of roots	Example	Graph
$b^2 - 4ac = 0$	<b>1 real root</b> Touches x axis once	$y = x^2 - 6x + 9$ $b^2 - 4ac =$ $(-6)^2 - 4(1)(9) =$ $36 - 36 = 0$	
$b^2 - 4ac > 0$	<b>2 real roots</b> Touches x axis twice	$y = -x^2 - 2x + 2$ $b^2 - 4ac =$ $(-2)^2 - 4(-1)(2) =$ $4 + 8 = 12$	
$b^2 - 4ac < 0$	<b>No real roots</b> Doesn't touch x axis - no x-intercepts	$y = x^2 - 2x + 2$ $b^2 - 4ac =$ $(-2)^2 - 4(1)(2) =$ $4 - 8 = -4$	

Figure. 7 Possible Outcomes for Solving the Quadratic Formula

## F. Matrix Multiplication

Multiplication of two square matrices is illustrated in Figure 8. The  $(i,j)$  element in the product is the vector product of row  $i$  of the first matrix with column  $j$  of the second matrix. This can easily be implemented with repetition loops as shown in Figure 8. This also provides an introduction to nested repetition loops.



```
function(a,b){
  n<-ncol(a)
  for(i in 1:n){
    for (j in 1:n){
      c[i,j]=a[i,]*b[,j]
    }
  }
}
```

Figure. 8 R Language code to Multiply Two  $n \times n$  Square Matrices

## G. PROBABILITY AND STATISTICS

The next example illustrates a simulation of rolling a die however it can easily be modified to simulate tossing a coin. It involves random numbers and array storage. A random number between 0 and 1 is generated using `Rnd()`. This is converted into an integer between 1 and 6 which is used to increment an array "a" to keep track of the statistics of how many times that number came up. The `Randomize()` function ensures the simulation result will be different each time. The student runs the simulation for several sample sizes and records the results in the table.

```
Sub Main()
  Dim i, dice, a(6) As Integer
  Randomize()
  For i = 1 To 6
    dice = Int(Rnd() * 6) + 1
    a(dice) = a(dice) + 1
  Next i

  For i = 1 To 6
    Console.Write(a(i) & " ")
  
```

Next  
End Sub

Figure. 9 Code for Simulation of Die Throwing

rolls	1's	2's	3's	4's	5's	6's
6	0	1	1	2	2	0
60						
600						
6000						
60000						

flips	Heads	Tails
10	6	4
100		
1000		
10000		
100000		
1000000		

Figure. 10 Tables to Store Result of the Simulation of Die Throwing and Coin Flipping

It is always beneficial to use examples that the student can relate to in one way or another as is the case with flipping coins or rolling dice. This could be further reinforced by having the student run the experiment using an actual dice or coin and comparing the results of the computer simulation. A lot can be learned by this type of exercise that cannot simply be taught through traditional classroom lecture.

Another possible exercise could reinforce what was learned about matrices and storage arrays and combine that with random number generation (Figures 11 and 12). Figure 11 shows the computer code for generating a 10x10 square of random digits as shown in Figure 12.

```
Dim i, j, a(11,11) as Integer
For i = 1 To 10
  For j = 1 To 10
    a(i,j)= 1
    Console.Write(Int(9.999*Rnd()))
  Next i
  Console.WriteLine("")
Next j
```

Figure 11. Program to write a "square" of 100 random numbers

```

2073046197
5921745034
0638719328
7334912576
4109527435
8980071342
5371029468
6835742109
1753648902
2407398561

```

Figure 12. A 10x10 "square" of 100 random digits

At some point earlier in the course it is assumed that the student was given active learning exercises on random number generators (e.g., the Rnd() command). To generate the square of random digits as illustrated in Figure 12 the student can use the Console.Write() statement in Figure 11 to print a random digit (0-9). This is done by first creating a random number between 0 and 9.999 and then taking the integer part of that number for example: Console.Write(Int(9.999\*Rnd())).

#### H. CHAOS THEORY

Although chaos theory does not show up very often in undergraduate engineering education it can be used to reinforce several mathematical concepts including complex numbers and divergence of a series. The Mandelbrot set is easy to calculate and is a good example of something that can stimulate an interest in the student through visualization. Figure 13 shows a simple R language program that computes an instance of the Mandelbrot set and creates a graphical output shown in Figure 14.

```

dx <- 400
dy <- 400
C <- complex( real=rep(seq(-2.2, 1.0, length.out=dx), each=dy ),
              imag=rep(seq(-1.2, 1.2, length.out=dy), dx ) )
C <- matrix(C,dy,dx)
Z <- 0
X <- array(0, c(dy,dx,20))
for (k in 1:20) {
  Z <- Z^2+C
  X[,k] <- exp(-abs(Z))
}
write.gif(X, "Mandelbrot.gif", col=jet.colors, delay=900)

```

Figure 13. Program to calculate and display the Mandelbrot set

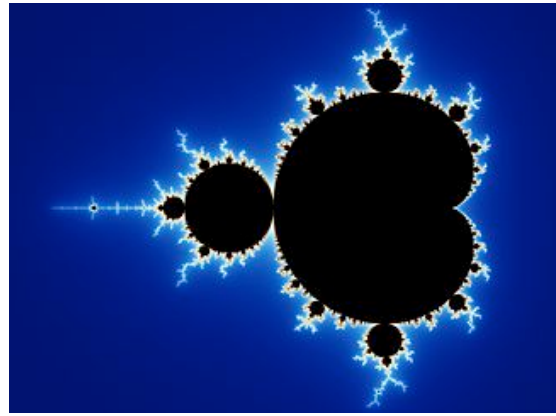


Figure 14. The Mandelbrot Set

#### III. DISCUSSION

Mathematics courses do a good job to prepare students for engineering courses. However even though students are exposed to the concepts in mathematics courses it may be the case that some of this learning is superficial without a deep understanding of the concept. The result is that the student can still have difficulty when required to use these concepts in engineering courses.

Programming can take the drudgery out of the mathematics leaving the fun of exploring the topic deeper especially with a highly interactive language such as R Language[14]. Usually deep learning occurs after the student is exposed to a concept several times in different ways and is called upon to apply the concept to a specific situation. This process could be accelerated if the programming classes intentionally reinforce key mathematical concepts a number of which have already been discussed. A good source would be the actual books that the students will be using in later engineering courses. For example for industrial engineering students it might be beneficial to use more examples from statistics or simulation. Some other possibilities are listed below.

- Irrational Numbers
- Finding Roots of Polynomials
- Summation Formulae

For irrational numbers the value of pi can be approximated and there are many techniques for doing this[16]. This can work out particularly well if the course is given in the spring semester and one of the classes happens to fall on March 14th (Pi day). Finding roots of polynomials is an integral part of the solution to a wide variety of engineering problems. Summation formulae such as adding up the numbers from 1 to n arise naturally from repetition loops and can be used for illustrating for example Taylor's Series.

#### IV. SUMMARY AND CONCLUSIONS

There is a high degree of synergy between computer programming and various mathematical topics. This can be exploited to stimulate a student's interest in mathematics. Normally the student learns the quadratic equation in mathematics class. However it can be used to illustrate programming concepts such as Boolean Logic and complex numbers. Programming a mathematical concept can help to visualize the mechanics of the solution and therefore help the student to visualize it better and therefore to understand it better. Also repeated exposure to a topic in different ways can help to improve understanding. This has been shown with some illustrative examples, These examples were chosen to be easy to program and at the same time practice a concept from mathematics. The idea is if the students have to program something by hand then they will learn it better. Reinforcing mathematical concepts in an introductory programming course can contribute to what is being called deep learning[7][8] and can benefit the student in later courses. A programming course has many benefits but can also be used to stimulate interest in mathematics and reinforce mathematical concepts.

#### REFERENCES

- [1] Thomas, M. O. J. (2006), "Teachers Using Computers in Mathematics: A Longitudinal Study. International Group for the Psychology of Mathematics Education, Vol. 5, pp. 265-272.
- [2] Ruthven, K. & Hennessy, S. (2002). "A practitioner model of the use of computer-based tools and resources to support mathematics learning and teaching", Educational Studies in Mathematics, 49, 47-88.
- [3] Afzal, M. T. , Gondal, M. B. and Fatima, N. 2014. "The effect of computer based instructional technique for the learning of elementary level mathematics among high, average and low achievers" International Journal of Education and Development using Information and Communication Technology (IJEDICT), 2014, Vol. 10, Issue 4, pp. 47-59.
- [4] Fernández-Alemán, J. L., Palmer-Brown, D. & Jayne, C. 2011. "Effects of Response-Driven Feedback in Computer Science Learning", IEEE Transactions on Education, 54, 501-508.
- [5] Domínguez, A., Saenz-de-Navarrete, J., de-Marcos, L., Fernández-Sanz, L., Pagés, C., Martínez-Herráiz, J.J. "Gamifying learning experiences: Practical implications and outcomes". Computers & Education. 63. pp. 380-392. <http://dx.doi.org/10.1016/j.compedu.2012.12.020>.
- [6] de Freitas, S.&Oliver, M., 2006. "How can exploratory learning with games and simulations within the curriculum be most effectively evaluated?", Computers & Education, 46(3), 249-264.
- [7] R.M. Felder and R. Brent, "Active Learning: An Introduction." ASQ Higher Education Brief, 2(4), August 2009.
- [8] Duffany, J.L. "Active Learning Applied to Introductory Programming", LACCEI 2015 Conference, Santo Domingo, Dominican Republic.
- [9] A. Gartner, M. Kohler, F. Riessman: "Children teach children. Learning by teaching". Harper & Row, New York u.a. 1971, ISBN 0-06-013553-0.
- [10] Fisch, Shalom M. and Rosemarie T. Truglio, Eds. (2001). "G" is for Growing: Thirty Years of Research on Children and Sesame Street. Mahwah, New Jersey: Lawrence Erlbaum Publishers. ISBN 0-8058-3395-1
- [11]Thorndike, Edward L. (1898-1901) 1911 Animal Intelligence: Experimental Studies. New York: Macmillan.
- [12]Duffany, J.L. "Choice of Language for an Introductory Programming Course", LACCEI 2014 Conference, Guayaquil, Ecuador.
- [13]Bronson, G. (2010). *C++ For Scientists and Engineers*, 3<sup>rd</sup> edition, Course Technology/CENGAGE Learning.
- [14]Crawley, Michael J. *Statistics: An Introduction using R*. Wiley, 2nd edition, 2014. ISBN 978-1-118-94109-6.
- [15]Zak, D. (2013) *Programming with Microsoft Visual Basic 2012*, Course Technology/CENGAGE Learning.
- [16]Levitin, Anany, *Introduction to the Design and Analysis of Algorithms*", 2002, Addison-Wesley, ISBN: 0-201-74395-7.