Asian MathSci League, Inc (AMSLI)
Partner: National Olympiad in Informatics Philippines (NOI.PH)
Email address: amsliphil@yahoo.com / ask@noi.ph
Contact Nos: +632-9254526 +63906-1186067

Student Copy
AIEP Scratch 2 Session 7B

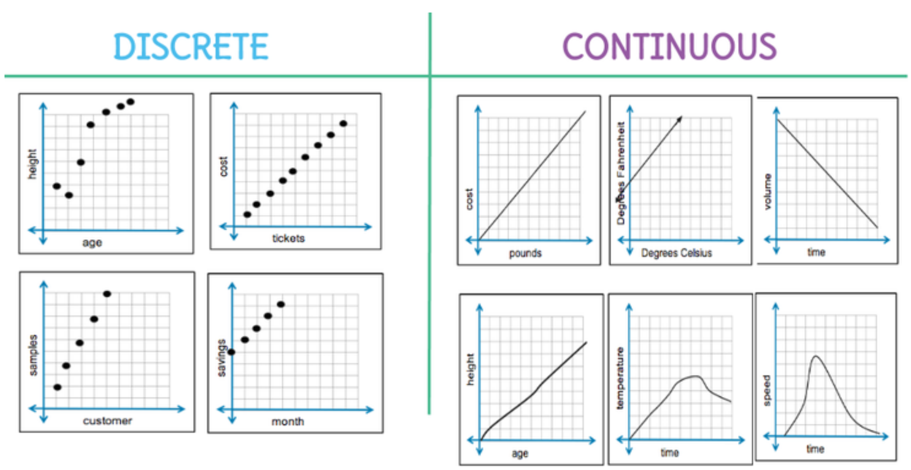
BISECTION METHOD (Divide and Conquer Part 2)

"The function of good software is to make the complex appear to be simple."

~ Grady Booch

In the previous handout, we discussed **mergesort** and **binary search**; both of these are applications of the **divide-and-conquer** paradigm on DISCRETE spaces. "Discrete" means that the items are can be "chunked" individually and separately. Examples include the set of integers (1, 2, 3, 4, 5, 6, and so on) or your favorite fruits (apple, kiwi, orange, and so on).

The opposite of "discrete" is CONTINUOUS. The range of possible heights is one such example. It is more complicated than just 100 cm, 200 cm, 300 cm, and so on because a person's height can be a decimal number[1]. For instance, in between 100 cm and 200 cm, we can have 100.5 cm. In between 100 cm and 100.5 cm, we can have 100.25 cm, *ad infinitum.*



https://www.doingdata.org/blog/difference-between-discrete-and-continuous-variables-in-tableau

Interestingly, the idea of binary seaching can be extended even to continuous spaces. This technique is known as the BISECTION METHOD, and the goal of this discussion is to provide you with a hands-on appreciation of the ingenuity of this powerful strategy.

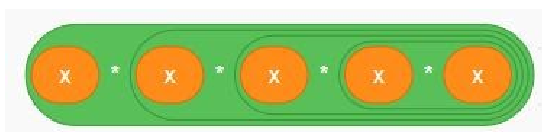
[1] Technically, a non-integer rational number

Before we go to bisection method per se, we have to learn some prerequisites. Although we may encounter some math and a little bit of algebra along the way — recall that the root word of "computer science" is "compute" — do not let them intimidate you. We will go through each idea step by step and build our intuition **from scratch and through Scratch**!

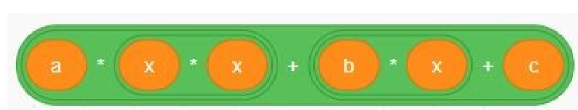
CONVERTING EXPRESSIONS TO SCRATCH CODE

Suppose we want to multiply the value of the variable *x* by itself 5 times. The naive way of writing this would be *x* * *x* * *x* * *x* * *x*. But this is quite cumbersome and can get unwieldy in the long run. Therefore, we usually shorten this into x^5, read as "x raised to 5." The raised (small) number is called the exponent. It tells us how many times is multiplied by itself.

Unfortunately, unlike some programming languages (like Python or Perl), Scratch does not have a built-in operator for exponentiation[2]. We have to take advantage of the `repeat` block and write our own My Block for exponents. **Nevertheless, in the interest of reducing complexity in this discussion, we can freely convert x^5 to the following Scratch code:**



This is a literal translation of *x* * *x* * *x* * *x* * *x*. Let us try some more examples as we explore the different built-in operators in Scratch:

Equation	Scratch Code
$a * x^2 + b * x + c$	
((abs of *a*) + *b*) * sin *x* [3]	

[2] In the two mentioned languages (Perl and Python), x^5 is concisely translated as `x ** 5`.
[3] sin stands for sine, a ratio in trigonometry. abs stands for absolute value, the nonnegative value that is left when the sign of a number is removed.

DRAWING GRAPHS

A GRAPH is a way to visualize a function by plotting points corresponding to its *x*- and *y*-values on a coordinate system, such as the Cartesian plane (recall our session 1 lesson). Our goal is to give Scratch an expression[4] and write a My Block for automatically drawing the graph for us.

For example, take the expression x^2. First, we translate it to a Scratch code:

In order to create the graph, we follow the steps outlined below. The implementation of this algorithm in Scratch is also shown (note that we have to import the Pen tool):

a. Make sure that we have two variables: x for the *x*-value and y for the *y*-value.

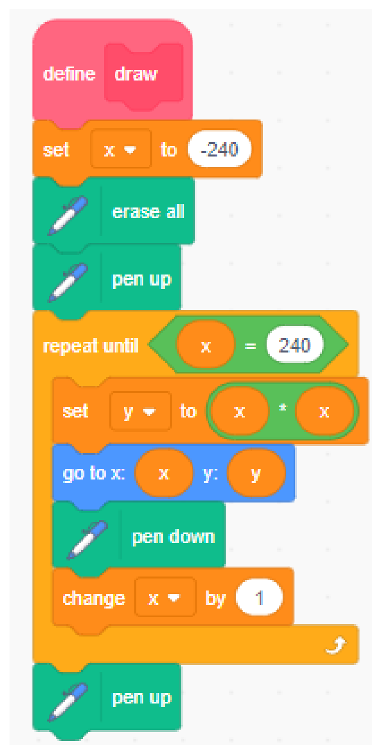
b. Initialize x to the leftmost edge of the stage: -240.

c. Set y to the Scratch code for the expression.

d. Draw the point.

e. Change x by a small value (for example, 1).

f. Repeat (c) to (e) until x becomes the rightmost edge of the stage: 240.



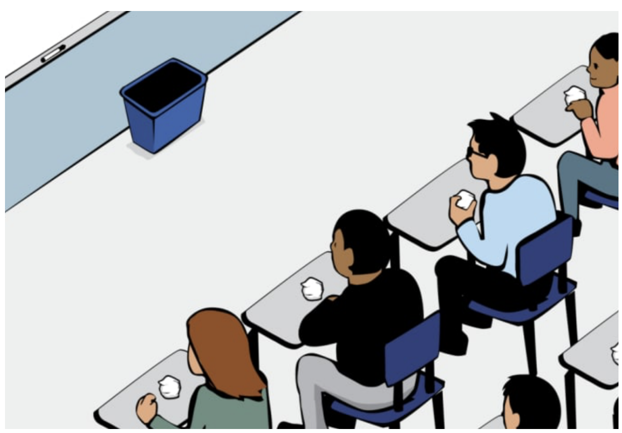
[4] We are taking the liberty of omitting some formal distinctions among terminologies for pedagogical purposes. Technically, we are not graphing the expression. Instead, we are graphing the function $f(x)$ = expression.

SOLVING EQUATIONS

The crux of the discussion is using bisection method in order to solve a certain class of functions known as MONOTONIC functions. They are functions whose graphs are either solely going up (increasing) or solely going down (decreasing). Remember that the prefix "mono-" means one, and a monotonic function has a graph that goes in ONE direction.

In line with the example earlier, suppose that we are searching for a nonnegative number x such that $x^2 = 49$. In English, this translates to looking for a number such that when this number is multiplied by itself, the result is 49. Two (fictional) students Jane and Jade turn this scenario into a guessing game:

Jade: Is it 100?
Jane: Too high!
Jade: Is it 50?
Jane: Too high!
Jade: Is it 25?
Jane: Too high!
Jade: Is it 12?
Jane: Too high!
Jade: Is it 6?
Jane: Too low!
Jade: Is it 9?
Jane: Too high!
Jade: Is it 7?
Jane: YES!

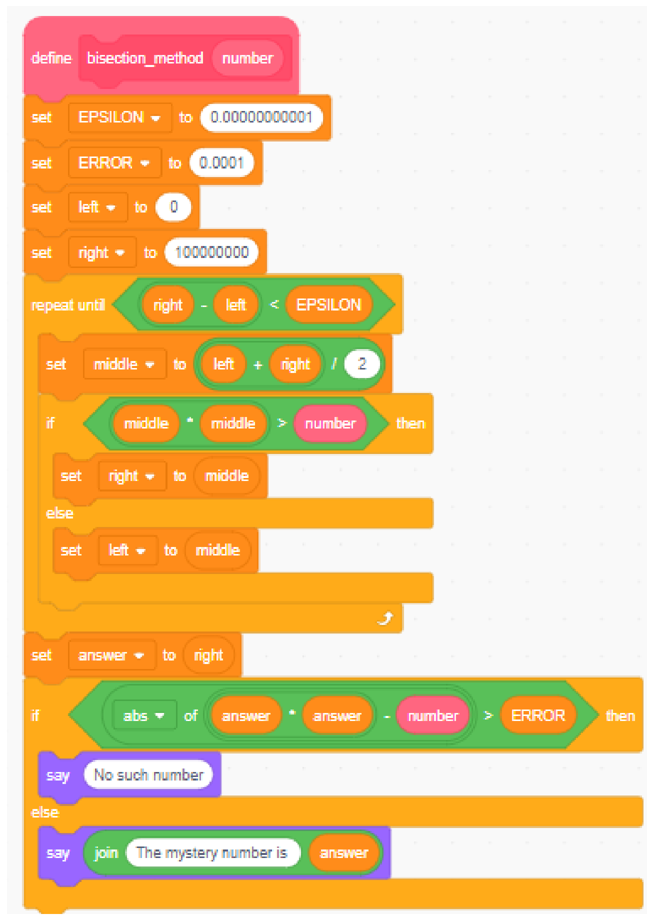


https://smallactbigimpact.com/for-educators/middle-school-resources-grades-6-8

Real-World Analogy: Let us say that a student is trying to shoot a crumpled paper in the trash bin located at the middle of their classroom. Of course, the student first tries to position herself at the middle. If her shot lands to the left of the trash bin, then she moves to the right. But, if her shot lands to the right of the trash bin, then she moves to the left.

These scenarios show the intuition behind the bisection method. Bisecting is dividing into two halves; in this case, we are dividing a graph into two halves, then we adjust (by going left or right) depending on how far our guess is from the correct answer. To put this idea in algorithmic terms[5]:

a. Set `EPSILON` to a very small positive number.
b. Set `ERROR` to a very small positive number.
c. Set `left` to the lowest possible value of our guess.
d. Set `right` to the largest possible value of our guess.
e. Set middle to the "midpoint" or average of left and right. This is our guess.
f. Check if our guess is correct.
   - If the result given by our guess is greater than the expected, then we should guess lower.
   - If the result given by our guess is less than the expected, then we should guess higher.
g. Repeat steps (e) to (f) until the difference between `right` and `left` becomes lower than `EPSILON`.
h. Check if our answer makes sense.



Observe that the procedure is almost the same as binary search. There are `left`, `right`, and `middle` variables. The key change is in the addition of two more variables, `EPSILON` and `ERROR`, which are explained on the next page.

[5] The algorithm presented is for monotonically increasing functions (those whose graphs are going up), like the example given in the previous page.

Asian MathSci League, Inc (AMSLI)
Partner: National Olympiad in Informatics Philippines (NOI.PH)
Email address: amsliphil@yahoo.com / ask@noi.ph
Contact Nos: +632-9254526 +63906-1186067

a. `EPSILON` - Since we are searching a continuous space, we have to set a "stop point"; otherwise, our algorithm will keep running forever. This is the function of `EPSILON`. Epsilon is actually the fifth letter in the Greek alphabet, and it is used in math to denote extremely small quantities.

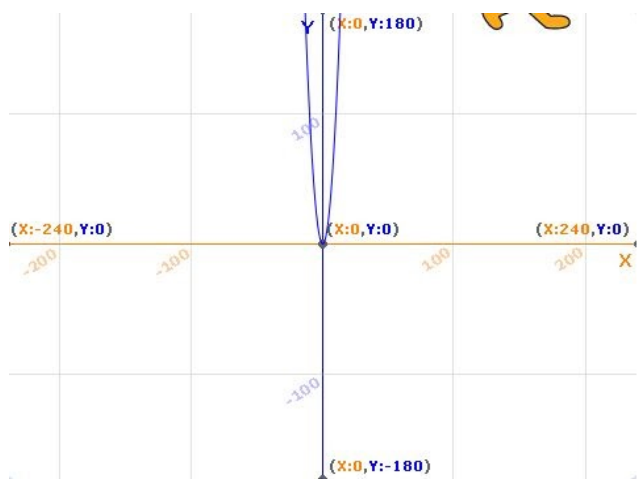
    The smaller `EPSILON` is, the more precise our answer will be. However, it will also take longer for our code to run because the number of times the `repeat` block gets executed will be increased.

b. `ERROR` - The bisection method only gives an estimate, and `ERROR` measures how far are we allowing our answer to "stray away" from the actual answer. Thus, the smaller `ERROR` is, the "stricter" our program will be.

    The final `if-else` block is related to this notion. If the difference between the expected result (value of `number`) and the result given by our estimate is greater than `ERROR` — the maximum error that we are tolerating — then we are forced to conclude that our estimate does not make sense and a solution is not found ☹

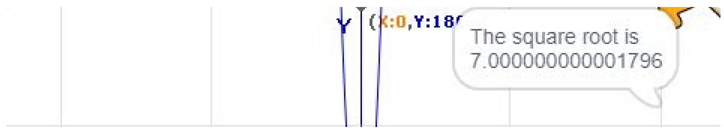
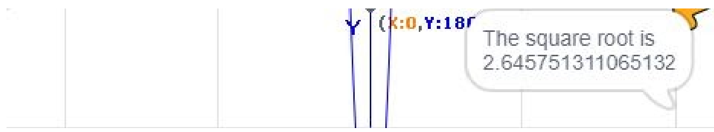
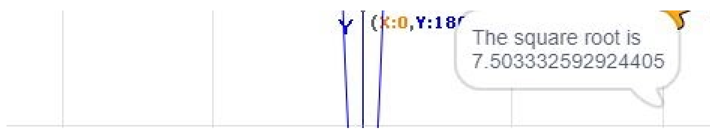
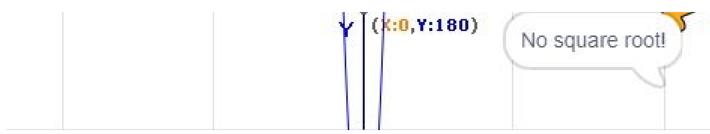
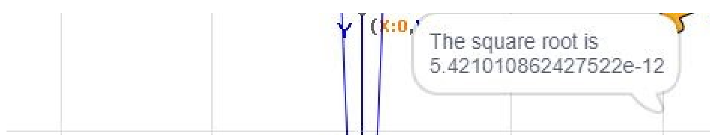
ACTIVITY: Solve It

Load the following Scratch file: Solve It_For Students.sb3. Complete the skeleton code, and accomplish the following tasks with the discussion as reference:



a. Graph the function related to the expression x^2. The sample output is shown on the right.
b. Complete the following statement: **"The left half of the graph is _________ while the right half of the graph is _________."**
c. Use the bisection method to estimate the square root of any number. Note that x is the square root of a number if x^2 is equal to that number.

    If the number has no square root, say "`No square root!`"

Type of Test Case	Input	Output
Perfect square	49	
Non-perfect square	7	
Decimal	56.3	
Negative number	-100	
Zero	0	

☆ BOSS-LEVEL CHALLENGE: Solve It (Part 2) ☆

Using the same skeleton code Solve It_For Students.sb3, try solving the following problem taken from the University of Valladolid (UVa) Online Judge. The problem is UVa 10341: Solve It and is as follows[6]:

"Solve the equation:

$$p * e^{-x} + q * \sin(x * 180/3.141592653589) + r * \cos(x * 180/3.141592653589)$$
$$+ s * \tan(x * 180/3.141592653589) + t * x^2 + u = 0,$$

where $0 \leq x \leq 1$."

[6] The original problem description can be accessed from https://onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=1282.

Since arguments in Scratch are in degrees, the appropriate conversion from radians has been incorporated into the equation. The input specifications have also been slightly modified to temper the difficulty of the problem.

To temper the difficulty, you are guaranteed that the given values of *p, q, r, s, t,* and *u* will always result in a valid *x*. **This means that we can safely remove step (h) from our bisection method discussion** (page 5).

For now, it is perfectly fine not to understand the math behind sin, cos, and tan. The skill that we are trying to build here is <u>**translating a long equation into a Scratch code**</u>. Here are some hints to help you get started with this challenge:

a. **Translate the equation first.** It can be done using only built-in blocks.
b. **Graph the function, and play with it!**
c. Unlike the (right half of the) square root activity earlier, the graph is going **DOWN**.
d. If the graph is going down, what small change do we have to do to step (f) from our bisection method discussion (page 5)?
e. What is the value of `left` (lowest possible value of *x*)?
f. What is the value of `right` (highest possible value of *x*)?

p	*q*	*r*	*s*	*t*	*u*	Output
12	-15	18	-9	-4	-19	

-- return 0; --