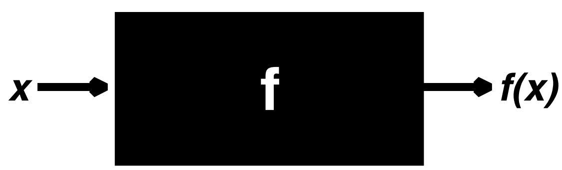
# Unit 1

Computer Science as Applied Mathematics

## Functions

In mathematics a function can be though of as a black box that takes in a set of inputs, performs some operation(s) on them, and produces a single output.




A function has a **domain**, which is a set of x-values that it can accept, and a **range**, or a set of f(x) values it can produce.

Example function: f(x) = x
domain: all real numbers
range: all real numbers
this function does nothing to its input, but it is still a function.

Example function: f(x) = x^2
domain: all real numbers:
range: [0, inf)
this function squares its input. Since you cannot square a number and get a negative result, its range is limited to 0 and positive reals.

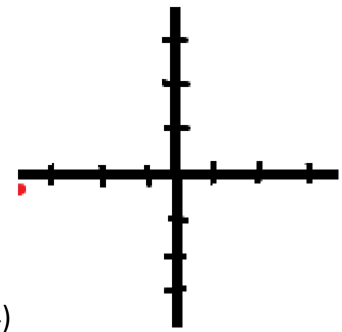
Example function: f(x) = 1/x
domain: (-inf, 0)U(0, inf)
range: all real numbers
This function's domain does not include zero because you cannot divide by zero. The function does not know how to process x = 0. We say f(0) is undefined.

**Graphing functions:**

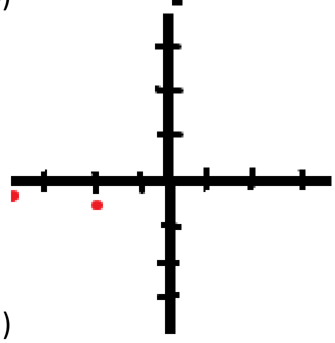
To graph a mathematical function, we pick a set of x values, find f(x) for each x value, and plot them. The set of x values we pick is TOTALLY ARBITRARY, but it will affect how the graph looks.

It is important to pick a scale that will display that function's graph well. For example: if we pick X: [-100, -99], Y: [52, 53] and we graph y=x, we won't even see the function's graph! The graph of y=x does not lie in this part of the coordinate plane.

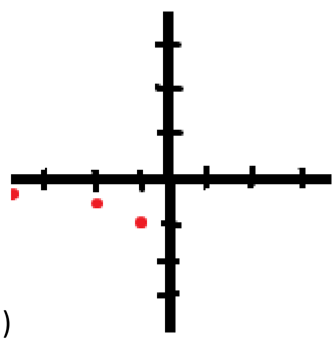
*Example*:
Let's graph y = 1/x
For our set of x-values, lets choose {-4, -2, -1, -0.5, -0.25, 0.25, 1, 2, 4}
For our scale, let's choose x:[-4, 4], y:[-4, 4]



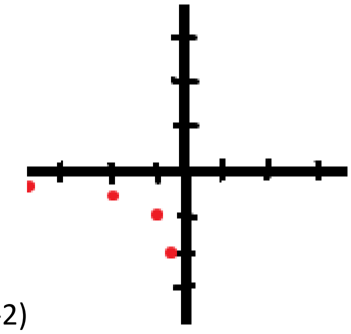
When x = -4, f(x) = -1/4 So we plot the ordered pair: (-4, -1/4)

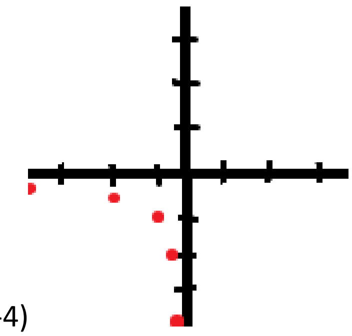


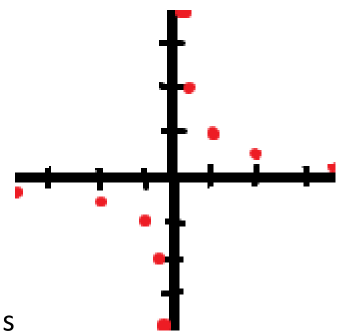
When x = -2, f(x) = -1/2 So we plot the ordered pair: (-2, -1/2)

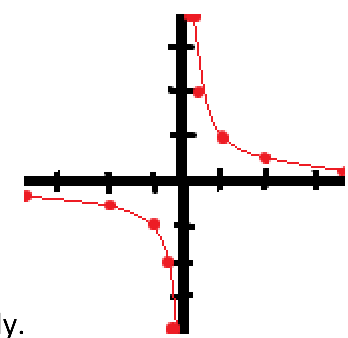


When x = -1, f(x) = -1/1 So we plot the ordered pair: (-1, -1)

When x = -0.5, f(x) = -2 So we plot the ordered pair: (-1/2, -2) 

When x = -0.25, f(x) = -4 So we plot the ordered pair: (-1/4, -4) 

Doing a similar process with x = {0.25, 0.5, 1, 2, 4} yields 

You can draw a line to connect the dots to see the shape more clearly. 

This process can be formalized in an algorithm that is the basis for the functionality of all graphing calculators

*Example Algorithm 3 (Function Grapher)*

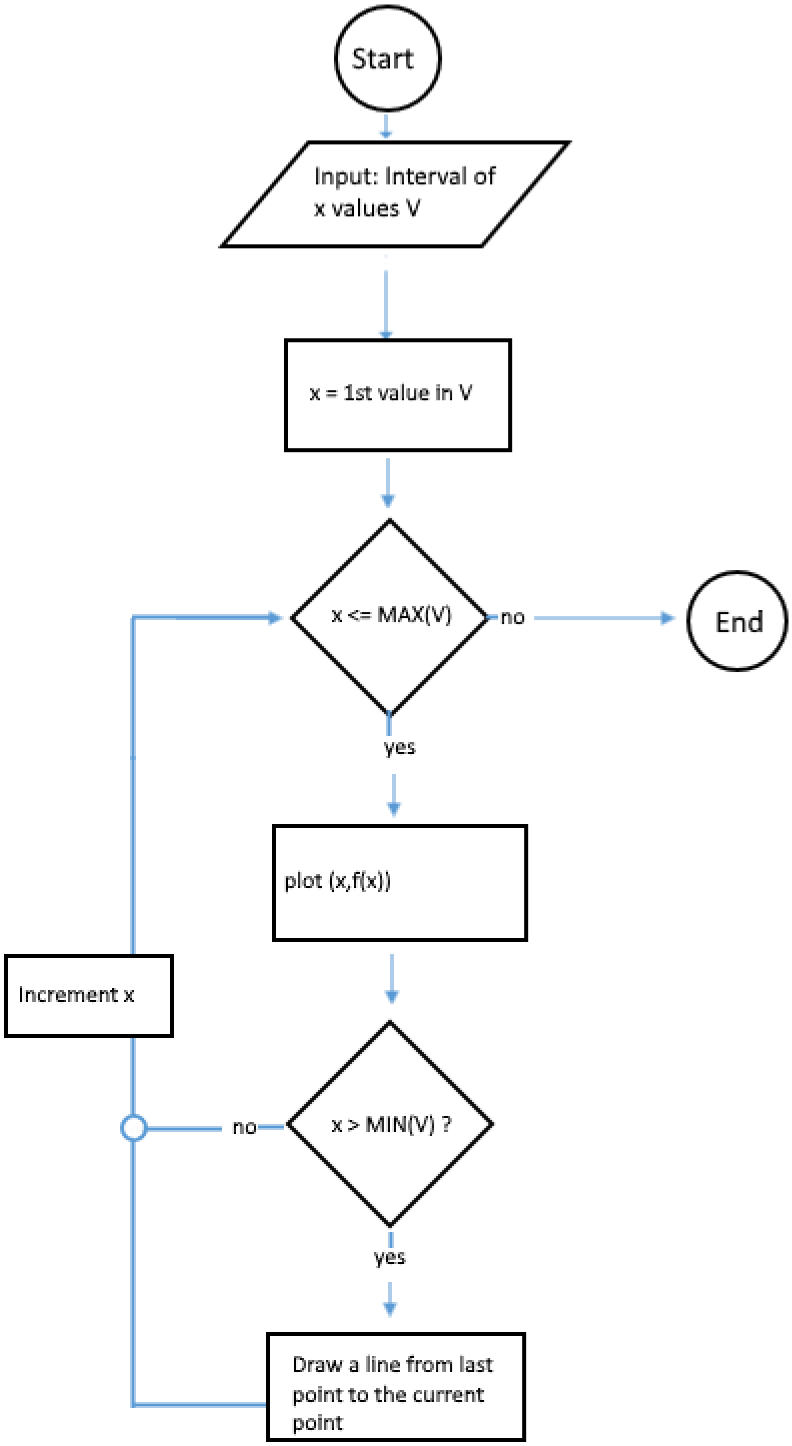
```
        Description:    This algorithm graphs a function f given the range of x-Values V
        Input:          A function, f, a range x-values V
        Output:         The graph of f
1       Graph(f, V)
2               x = the first value in V
3               while (x <= the largest value in V)
4                       y = f(x)
5                       plot (x,y)
6                       if ( x > the first value in V )
7                               draw a line from (x-1, f(x-1)) to (x,y)
8                       increment x by 1
```

Notice the conditional operation on lines 6 and 7. In short, this conditional statement draws a line from the last point we plotted to the current one we just plotted.

It tests x to see if it is greater than the first value in V. If it is not, then the iterative operation on line 3 has not yet completed its first operation. (i.e. we have only plotted one point.) We don't want to draw a line between the point we plotted and the last point we plotted if we've only plotted 1. However, if x is greater than the first Value in V, then the iterative operation on line 3 must have completed at least once, and we must have drawn at least one point! This means we can draw a line between the current point and the previous point.

Also: Notice that line 7 is very vague. It simply says "draw a line from … to …" If you typed that line into your computer, it will have no idea what to do, but that is ok. When writing algorithms, we have the freedom to be as abstract or vague as we want. That is to say, at this point in developing an algorithm, we don't care how exactly to draw a line from point A to B on a computer screen. We just know that it must be done. We can expand on line number 7 later.

Here is an equivalent flow chart for the algorithm:


Start
Input: Interval of x values V
x = 1st value in V
x <= MAX(V)
no
End
yes
plot (x,f(x))
Increment x
no
x > MIN(V) ?
yes
Draw a line from last point to the current point

## Equations

Functions and equations are very similar. The difference can be thought of as “Functions tell you how to map a domain to a range, while equations tell you how two or more variables are related.”

Typically we use equations to solve for unknown values (e.g. solve for x), while we use functions to calculate some useful value given some input(e.g. calculate y, given x).

In most cases, algorithms can be developed to solve a mathematical equation. We will focus on the quadratic equation.

Recall: The quadratic equation gives us the values for the roots a degree-2 polynomial.

More formally stated, if $y = ax^2 + bx + c$ then the solutions to y = 0 are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

From the definition of this equation we can gather several things:

(1) The solutions to this equation are entirely dependent on the coefficients a, b, and c of the standard-form equation: $x = ax^2 + bx + c$.

(2) There may not be any real solutions to the quadratic equation. (What if the thing under the square root (the **discriminant**) is negative?)

(3) This equation may have 0, 1, or 2 unique solutions. (What if the discriminant is 0?)

Since we have three possible outcomes, we cannot use a typical if-else statement. In this scenario, we will continue evaluating the logic by placing an additional if statement inside of the previous else statement. This is known as a nested if statement.

From these three truths and the definition of the quadratic equation, we can come up with the following algorithm:

*Example Algorithm 3 (Quadratic solver)*

Description: This algorithm solves for the roots of a degree-2 polynomial if they exist, given the coefficients of the standard-form equation

Input: a, b, and c—real numbers

Output: The solutions to x = 0 if x = ax^2 + bx + c

```
Solve_QDR(a, b, c)
        discriminant = b^2 − 4ac

if ( discriminant < 0 )
        print(“There are no real solutions”)
```

else

    if(discriminant = 1)

        sol. 1 = $\frac{-b+\sqrt{b^2-4ac}}{2a}$

        print(sol. 1)

    else

        sol. 1 = $\frac{-b+\sqrt{b^2-4ac}}{2a}$

        sol. 2 = $\frac{-b-\sqrt{b^2-4ac}}{2a}$

        print(sol. 1, sol. 2)

Initially, the discriminate is calculated. If this number is less than zero, there are no real solutions. If the number is greater than zero, we must evaluate two more conditions. If the discriminant is equal to one, there is one solution and that solution is displayed. If the discriminant is not equal to one, there are two solutions and both are displayed to the user.

## Modeling and Simulations

One of the important roles played by computers is simulating the natural world. For example, a weather forecast is generated by observing current conditions and letting a computer algorithm simulate, based on the patterns we have already observed, how the weather will change. Modelling something like climate is incredibly complicated; we do not (and may never) have a mathematical description of climate that accounts for all possible variables. The consequence is that computer simulation of weather and climate will never be perfect (we have all seen a wrong weather forecast). However, weather forecasts remain reliable enough to be useful despite their imperfections. It is important to understand that the quality of a simulation depends on how well you can model the thing you are simulating.

A very simple (and accurate) model of a physical system is the acceleration of an object due to gravity. Near the surface of Earth, an object released from height *h* will fall towards Earth with an acceleration of $g = 9.8\ m/s^2$.

The time, *t*, that it takes for the object to reach the ground is:

$$t = \sqrt{\frac{2h}{g}}$$

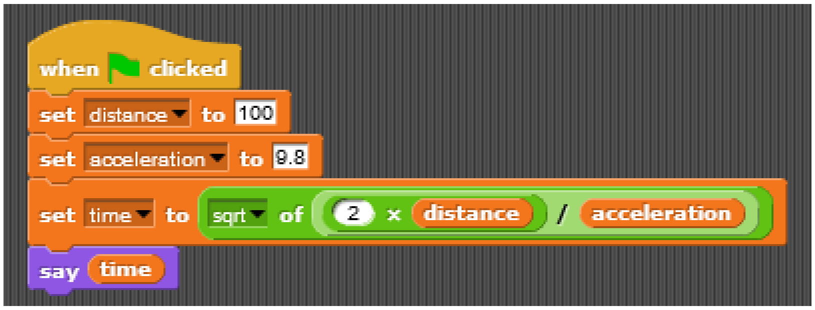
So, on earth (when ignoring air resistance), an object released from *100 meters* above the ground will take:

$$t = \sqrt{\frac{2 * 100}{9.8}} \approx 4.5 seconds$$

to reach the ground.

Although this "simulation" only involves plugging in values for *g* and *h*, it demonstrates that computers are good at giving us answers to our questions once we can describe the question mathematically.

A SNAP program that could model an object falling from any height at any value for *any* value for acceleration due to gravity would look something like the following:




You may want to ask your students to change this model so that it simulates the time taken for the objects to fall on planets other than Earth. Here is a table for values of *g* that they can use.

| **Acceleration Due to Gravity Comparison** | | | | |
|---|---|---|---|---|
| **Body** | **Mass [kg]** | **Radius [m]** | **Acceleration Due to Gravity, "g" [m/s²]** | **g / g-Earth** |
| Sun | 1.99 x $10^{30}$ | 6.96 x $10^{8}$ | 274.13 | 27.95 |
| Mercury | 3.18 x $10^{23}$ | 2.43 x $10^{6}$ | 3.59 | 0.37 |
| Venus | 4.88 x $10^{24}$ | 6.06 x $10^{6}$ | 8.87 | 0.90 |
| Earth | 5.98 x $10^{24}$ | 6.38 x $10^{6}$ | 9.81 | 1.00 |
| Moon | 7.36 x $10^{22}$ | 1.74 x $10^{6}$ | 1.62 | 0.17 |
| Mars | 6.42 x $10^{23}$ | 3.37 x $10^{6}$ | 3.77 | 0.38 |
| Jupiter | 1.90 x $10^{27}$ | 6.99 x $10^{7}$ | 25.95 | 2.65 |
| Saturn | 5.68 x $10^{26}$ | 5.85 x $10^{7}$ | 11.08 | 1.13 |
| Uranus | 8.68 x $10^{25}$ | 2.33 x $10^{7}$ | 10.67 | 1.09 |
| Neptune | 1.03 x $10^{26}$ | 2.21 x $10^{7}$ | 14.07 | 1.43 |
| Pluto | 1.40 x $10^{22}$ | 1.50 x $10^{6}$ | 0.42 | 0.04 |

( From http://www.aerospaceweb.org/question/astronomy/q0227.shtml)