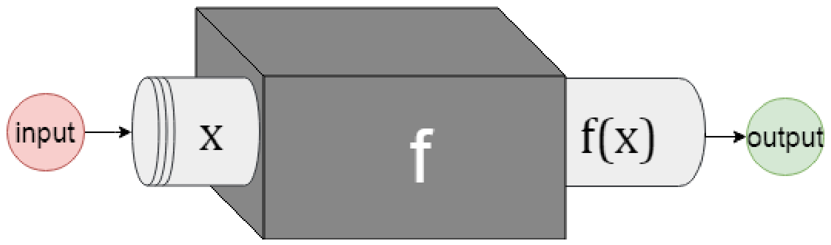
Unit 1

Computer Science as Applied Mathematics

Mathematics

Functions

A function can be thought of as a black box that accepts input, performs an operation, and produces a single output.



Typically, the notation used to express a function is called “function notation.” In function notation, a function has a name, arguments, and a definition. The notation: $f(x)$ is read aloud as “f of x.”

Consider the function: $f(x) = x^2 + 1$

f	(x)	$=$	$x^2 + 1$
Name	Arguments		Definition

Properties of Functions

- Domain – the set of all possible input values a function can receive.
- Range – the set of all possible output values a function can return

Computers allow functions to be defined in many equivalent ways. The first definition is very short and concise, while the third is cryptic and hard to understand. Becoming proficient at writing concise and efficient code is an important part of computer science.

Functions in Computing

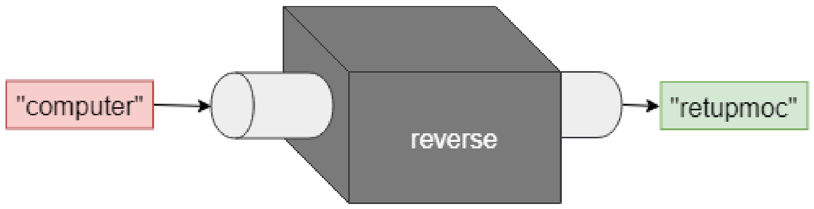
Until now, functions covered have been purely mathematical functions. That is, they describe how to process a single real number as input and produce a single real number as output. Also, we have been limited to mathematical operations like addition, exponentiation, multiplication, trigonometric functions, etc.

While these basic mathematic functions can be replicated easily in computers, functions are much more versatile and powerful.

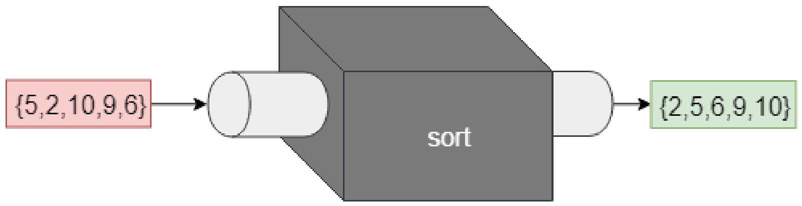
In the realm of computing, functions are typically segments of code that are called upon when they are needed. They accept input, or arguments, and produce output just like mathematical functions. Below are some notable differences between traditional mathematical functions and functions as they exist in computers.

- **Wide variety of inputs and outputs**
  Functions are no longer limited to single number input and single number output. The input and output of a function can be any type of data.

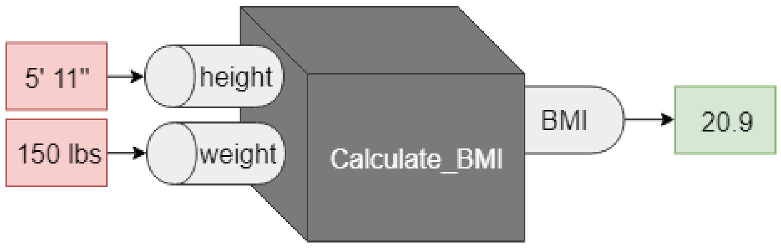
  A function can be defined in computers that accepts a string of characters as input, reverse the string of characters, and outputs the reversed string.



  A function can be defined that accepts a list of numbers, sorts them, and outputs the sorted list.



  A function can be defined that accepts height and weight as inputs, and calculates body-mass index (BMI)



Many Ways to Define a Function

When defining mathematical functions, function notation is used. Defining functions to a computer requires the use of a programming language—a standard set of instructions. There are many of these programming languages and even more equivalent ways to define a function.

The following is the function $f(x) = x^2$ defined in the python programming language in 3 different ways:

```
#first way
def f(x):
        return x * x

#second way
def f(x):
        return math.pow(x,2)

#third way
def f(x):
        result = 0
        for i in range(0, x):
                result += x
        return result
```

If the goal were to simply calculate $x^2 + 2x + 1$ for $x = 3$, then it would make sense to state the problem in terms of functions.
Given $f(x) = x^2 + 2x + 1$, what is $f(3)$?

Equations

Equations are useful tools for describing how two or more quantities are related. Equations are very similar to functions. The important difference between equations and functions is that equations explain how two quantities are related, while functions explain how to calculate an output given an input.

Operations in Algorithm (8, 16, 17)- Alg, Flow Chart, Seq, Sel, Ite, Flow Chart for each

Algorithms are descriptions of how to solve problems. In some ways they're like a recipe for making food: they're just a list of steps to follow.

For example, whenever you multiply two numbers together, you follow an algorithm taught to you in elementary school. First, you multiply the column on the right, writing down the ones digit below and carrying the tens place for the next column. You repeat this for every column.

Computers solve problems by performing algorithms that their programmers specified. Whenever you search for something on Google, for example, you cause computers to follow a set of steps that eventually leads to them giving you a list of things on the internet that matched your search.

Operations

When designing an algorithm, one must think about what input is involved in the algorithm, what problem the algorithm will solve, and how the algorithm should output a solution. In the following examples of algorithms, we will include a brief description of what it does, define the input and output, and then write the algorithm in **pseudocode** with each line numbered. Pseudocode is a way to write algorithms in a more human-readable way without worrying about syntax.

Sequencing

Sequential operations are operations that do a single well-defined task.

Example: Add 3 cups of water to the mixture

Increase the value of x by 1

Example Algorithm (Average Grade)

Description: This algorithm calculates the average grade of a student taking 6 classes.

Input: $G_1, G_2, \ldots, G_6$ The set of numeric grades of the student's 6 classes

Output: avg, the numeric average grade of the student

```
1    avg($G_1$, $G_2$, …, $G_6$)
2        sum = $G_1$
3        sum = sum + $G_2$
4        sum = sum + $G_3$
5        sum = sum + $G_4$
6        sum = sum + $G_5$
7        sum = sum + $G_6$
8        average = sum / 6
9        return average
```

The example uses only sequencing to calculate the average grade of a student with 6 classes. If a student's grades were 0.98, 0.76, 0.75, 0.80, 0.90, and 0.88, then

*avg(*0.98, 0.76, 0.75, 0.80, 0.90, 0.88)

solves the problem of determining their average grade. With these values as input, we see that at line 8, the value of the variable *average* will be *0.845*. The next instruction is to *return average*, or rather, signify that the current value of *average* is the answer to the problem.

Selection

Selection, or conditional operations are operations in which you select an operation based on a condition. These conditions are typically a Boolean expression. Recall that Boolean expressions are logical expressions which evaluate to either true or false.

Example: If the mixture is too thick, then add ½ cup of water.

```
If x > 10 then
    print "x is larger than 10"
else
    print "x is not larger than 10"
```

Example Algorithm 2 (maximum of two numbers)

Description:	This algorithm calculates the maximum of two numbers
Input:	*X* and *Y*, two real numbers
Output:	The maximum of X and Y. If X and Y are equal, then the value of both X and Y is returned.

```
1     MAX(X, Y)
2           if (X > Y)
3                 return X
4           else
5                 return Y
```

This example uses a selection to choose the greater value between two numbers. The Boolean expression, “X > Y” is the condition to the conditional operation on line 2. If the value of “X > Y” is true, then X must be the maximum, and thus line 3 is performed (and line 5 is not performed). If the value of “X > Y” is false, then X is either less than or equal to Y—in which case line 3 is skipped and line 5 is performed. Again, we see the word “return,” which means that the algorithm should report a value as the answer to the problem.

As an example, the value of MAX(5, 10) is 10, and the value of MAX(55, 10) is 55. As an exercise, try to apply our definition of the MAX algorithm while substituting these example input values for X and Y.

Iteration

Iterative operations are operations that repeat a certain task or set of tasks until a certain condition is met. This condition, is again, often a Boolean expression.

Example: Stir the rice <u>until all water is absorbed</u>.

```
While x < 10
      increase the value of x by 2
```

Example Algorithm 3 (Linear Search)

Description: This algorithm searches a list of integers for a given integer X. *(This is like checking every item in a list until we find what we are looking for.)*

Input: A list of Integers *L*, an Integer *N* that is the size of the list, and an Integer *X*

Output: The Algorithm returns true if *X* is present in the list, *L*. It returns false otherwise.

```
1     Search(L, N, X)
2           i = 1
3           while (i ≤ N)
4                 if( L_i == X )
5                       return true
6                 else
7                       i = i + 1
8
9
10          return false
```

Example algorithm 3 is a famous algorithm called **Linear Search** that uses iteration, sequencing, and selection to search a list for an occurrence of an integer. It is very important to note that the entire conditional operation of lines 4-7 is repeated as part of the iterative operation on line 3.

This algorithm will initially set $i = 1$. This variable i will be used as an index into the list we are searching through. The algorithm instructs us to keep checking the next spot (or rather, increasing values for the index i) until the item at the current spot in the list is equal to what we are looking for. If i exceeds N then we have checked every spot without success, so the algorithm exits out of the iterative operation on line 3 and instead performs line 10, which says to report that the item was not in the list.

Flow Charts

Flowcharts are another way of representing algorithms. They are useful in showing the “flow” of an algorithm. Table 1 shows some basic shapes that are used when creating a flowchart. It is important to use the correct shape to avoid confusion and an incorrect representation of an algorithm.

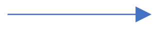
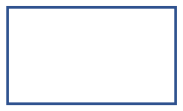
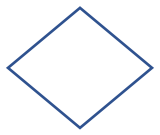
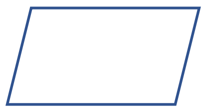
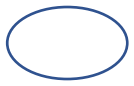
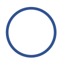
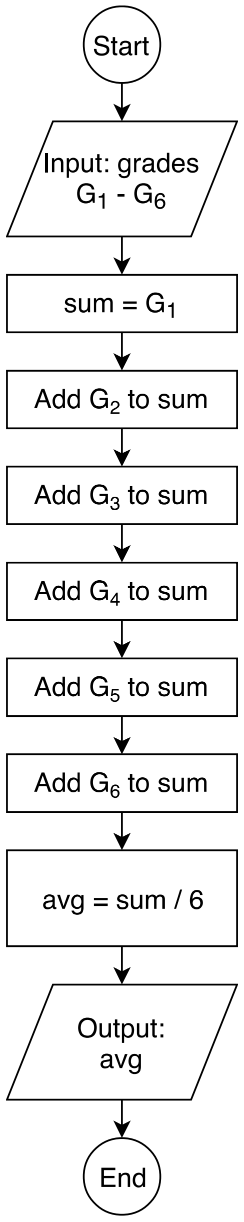
Basic Shapes for a Flowchart		
Shape	**Name**	**Meaning**
	Flow line	An arrow pointing from one symbol to another represents the direction of the flow of the algorithm
	Process	A rectangle represents an action that is performed.
	Decision	A diamond represents a Boolean test. There will be two branches, an arrow pointing to what happens when the expression evaluates to true and an arrow pointing to what happens when the expression evaluates to false
	Input / Output	A parallelogram is used when receiving input or displaying output
	Terminal	Circles or ovals are used for the start of a program and the end of a program
	Connector	When several arrows point to the same shape, a circular connector can be placed so that the arrows intersect at the connector, and a single arrow points from the connector to the shape.

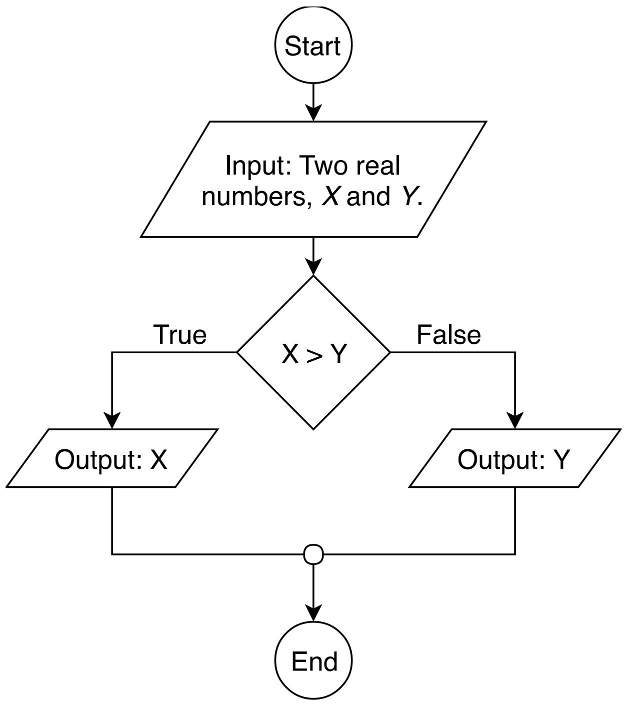
Table 1. shows the basic shapes used in making flowcharts and what they represent.

We will now represent the previous sequencing, selection, and iteration examples with flowcharts.

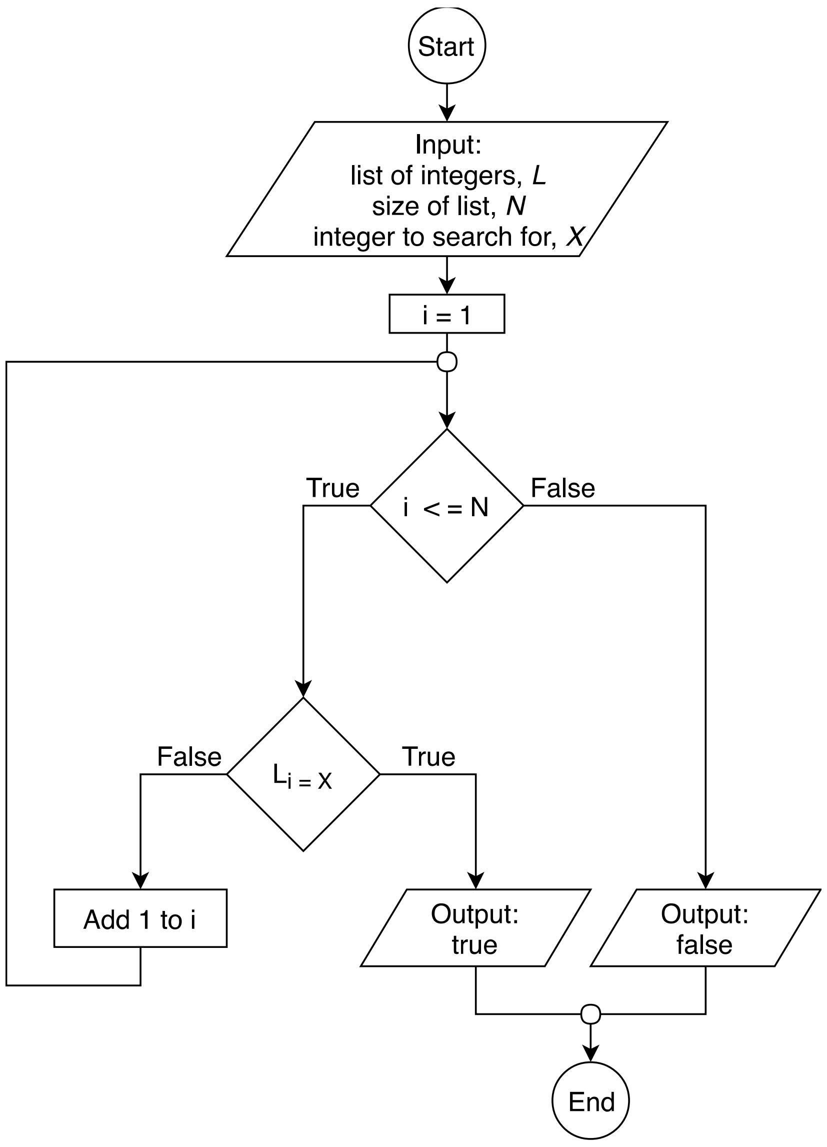
Sequencing


Start
Input: grades G_1 - G_6
sum = G_1
Add G_2 to sum
Add G_3 to sum
Add G_4 to sum
Add G_5 to sum
Add G_6 to sum
avg = sum / 6
Output: avg
End

Selection


Start
Input: Two real numbers, X and Y.
X > Y
True
False
Output: X
Output: Y
End

Iteration


Start
Input:
list of integers, *L*
size of list, *N*
integer to search for, *X*
i = 1
i <= N
True
False
$L_{i = X}$
False
True
Add 1 to i
Output: true
Output: false
End