An *alphabet* is a finite set. Some examples—\{a, b, c\}, \{1, a, b\}, all characters on a keyboard. The alphabet \{0, 1\} is also called binary alphabet. We use \(\Sigma\) to denote alphabet. Each member of the alphabet is called a *symbol*.

String over an alphabet is a finite sequence of symbols from that alphabet. For example: \(abbbbc\) is a string over the alphabet \{a, b, c\}. 0000100 is a string over the alphabet \{0, 1\}. A Java program is a string over the keyboard alphabet. Often, in this course, we will view a program as a string.

Length of a string is the number of symbols in the string. We will define a special string: A string whose length is zero, is called empty string. We use \(\lambda\) to denote the empty string.

[One Reason for using empty string: Typically programs takes strings as inputs. However, some programs do not take input. We can view them as programs that take empty string as input. That way, we can say that all programs have inputs.]

Given an alphabet \(\Sigma\), the set \(\Sigma^*\) is the set of all possible strings that can be formed. By definition \(\lambda\) always belongs to \(\Sigma^*\). A language \(L\) (over an alphabet \(\Sigma\)) is a subset of \(\Sigma^*\). So \(L\) is a set of strings.

Examples of languages. Consider the alphabet \{0, 1\}. All strings of even length, all strings that end with 0, all strings that have the pattern 00111. Consider the keyboard alphabet: All valid (Syntactically correct) Java programs is a language.

A language is a set. We often use the words language and set interchangeably. When it is apparent we do not explicitly mention the underlying alphabet. It will be clear from the context.

If \(A\) and \(B\) are languages, then we can form \(A \cup B\), \(A \cap B\), and \(A - B\). \(\overline{A} = \Sigma^* - A\).

Given two set \(A\) and \(B\)

\[
AB = \{xy \mid x \in A, y \in B\}
\]

Thus \(AB\) contains strings formed by concatenating strings from \(A\) with strings from \(B\).

Given a language \(A\), \(A^1 = A\), \(A^n = AA^{n-1}\). Thus \(A^n\) is formed by taking \(n\) strings (possibly same) from \(A\) and containing them. \(A^0 = \{\lambda\}\). Given a language \(A\), \(A^* = \cup_{n=0}A^n\).

Given a set \(S\), cardinality of \(S\) is the number of elements in \(S\). We use \(|S|\) to denote cardinality of \(S\).

Given two sets \(A\) and \(B\), we would like to know which set has more elements. If the sets are finite sets, then this is easy. How do we compare cardinalities when the sets are infinite?

Consider following two sets. Let \(\mathbb{N}\) be the set of all natural numbers and let \(E\) be the set of all positive even integers. Clearly \(E\) is a proper subset of \(\mathbb{N}\). Which element has more elements? Since \(E\) is a proper subset of \(\mathbb{N}\), it is reasonable to say that \(\mathbb{N}\) has more elements.

Lets rename each element of the sets. Rename elements from \(\mathbb{N}\) as follows: We will rename 1 as “two”, 3 as “six”, 4 as “eight” and so on. Given this renaming, the set \(\mathbb{N}\) has elements “two”, “four”, “six” etc.
Rename elements from $E$ has follows: Rename 2 as “two”, 4 as “four” etc. Now this set has elements “two”, “four”, “six” etc.

Now which set has more elements? Both sets are exactly the same! If we allow renaming, then both sets look exactly the same, otherwise one set is a proper subset of the other. What is a good way to compare cardinalities of infinite sets?

We do this using functions. Given two sets $A$ and $B$ a function $f$ from $A$ to $B$ is one-one, if for every $x$ and $y$ from $A$ if $x \neq y$, then $f(x) \neq f(y)$. The function $f$ is onto if every element from $B$ has an inverse. A function $f$ is a bijection, if it is one-one and onto.

Given two sets $A$ and $B$, we say $|A| \leq |B|$, if there is a one-one function $f$ from $A$ to $B$. Recall that $f$ is one-one if for every $x \neq y$, $f(x) \neq f(y)$. Given two sets $A$ and $B$, we say $|A| = |B|$, if there is a one-one, onto function (bijection) from $A$ to $B$.

Let $\mathbb{N}$ denote the set of natural numbers, $\{1, 2, \cdots \}$. A set $A$ is countable if either $A$ is a finite set or if $|A| = |\mathbb{N}|$. Let $E$ be the set of even positive integers. The function $f(n) = 2n$ is a bijection from $\mathbb{N}$ to $E$. Thus $|\mathbb{N}| = |E|$. Similarly, we can show that the cardinality of set positive odd numbers is same as the cardinality of natural numbers. Let $\mathbb{Z}$ denote the set of all integers. Consider the following function from $\mathbb{Z}$ to $\mathbb{N}$: $f(0) = 1$, $f(1) = 2$, $f(-1) = 3$, $f(2) = 4$, $f(-2) = 5, \cdots$. In general $f(i) = 2i$, if $i > 0$ and $f(i) = -2i + 1$, if $i < 0$. It is easy to see that $f$ is a bijection from $\mathbb{Z}$ to $\mathbb{N}$. Thus the $|\mathbb{N}| = |\mathbb{Z}|$.

We now show that the cardinality of $\mathbb{N} \times \mathbb{N}$ is same as the cardinality of $\mathbb{N}$. Instead of giving a formal proof, we give the proof idea. Let us arrange elements of $\mathbb{N} \times \mathbb{N}$ as a two dimensional matrix $M$, such that $M(i, j) = \langle i, j \rangle$. The matrix $M$ looks like:

$$
\begin{aligned}
\langle 1, 1 \rangle & \langle 1, 2 \rangle \langle 1, 3 \rangle \langle 1, 4 \rangle \cdots \\
\langle 2, 1 \rangle & \langle 2, 2 \rangle \langle 2, 3 \rangle \langle 2, 4 \rangle \cdots \\
\langle 3, 1 \rangle & \langle 3, 2 \rangle \langle 3, 3 \rangle \langle 3, 4 \rangle \cdots \\
\langle 4, 1 \rangle & \langle 4, 2 \rangle \langle 4, 3 \rangle \langle 4, 4 \rangle \cdots \\
& \cdots \cdots \cdots \cdots \cdots \\
& \cdots \cdots \cdots \cdots \cdots
\end{aligned}
$$

We traverse the matrix along the diagonals, i.e, $f(\langle 1, 1 \rangle) = 1$ $f(\langle 2, 1 \rangle) = 2$ $f(\langle 1, 2 \rangle) = 3$ $f(\langle 3, 1 \rangle) = 4$ $f(\langle 2, 2 \rangle) = 5$ $f(\langle 1, 3 \rangle) = 6$ $f(\langle 4, 1 \rangle) = 7$ $f(\langle 3, 2 \rangle) = 8$ $f(\langle 2, 3 \rangle) = 9$ $f(\langle 1, 4 \rangle) = 10$ $\cdots$. Every tuple appears in some diagonal and we will eventually reach that diagonal. Thus $f$ is a bijection.

Let $S$ be any infinite subset of $\mathbb{N}$. We can show that $|S| = |\mathbb{N}|$. For this, consider the following bijection from $\mathbb{N}$ to $S$. $f(1)$ is the smallest element of $S$, for $k > 0$, $f(k)$ is the smallest element of $S - \{f(1), \cdots f(k - 1)\}$. We can extend this argument to show that for infinite sets $S$, $|\mathbb{N}| \leq |S|$.

Let $F$ denote the set of all functions from $\mathbb{N}$ to $\{0, 1\}$. We will show that $F$ is not countable. More specifically, we show that $|F| \neq |\mathbb{N}|$. Assume that $|F| = |\mathbb{N}|$. So, there is a bijection $g$ from $\mathbb{N}$ to $B$. Thus $\langle g(0), g(1), g(2), \cdots \rangle$ is a list of all functions from $F$. We denote $g(n)$ with $g_n$.

Now, we define a new function $f_{new}$ as follows.

$$
f_{new}(k) = 1 - g_k(k),
$$

We claim that $f_{new}$ is not in the list $\langle g_0, g_1, \cdots \rangle$. Assume that it is in the list. Assume that, it is the $k$th member in the list. This means that $f_{new}$ is same as $g_k$. So for every number $n$,
\( f_{\text{new}}(n) = g_k(n) \). However, by the definition of \( f_{\text{new}} \),

\[ f_{\text{new}}(k) = 1 - g_k(k). \]

Thus \( f_{\text{new}}(k) \neq g_k(k) \). Thus \( f_{\text{new}} \) is not the same as \( g_k \). This is a contradiction. So \( F \) is not countable.

We will now show that the set of real numbers between 0 and 1 is also not countable.

Let \( R \) denote the set of real numbers between 0 and 1. The cardinality of natural numbers is less than or equal to the cardinality of \( R \). This is because, the function \( f(n) = 0.n \) is a one-one function from \( \mathbb{N} \) to \( R \). We now show that \( R \) is not countably infinite. We show this by contradiction. Assume \( |\mathbb{N}| = |R| \). Thus there is a bijection \( f \) from \( \mathbb{N} \) to \( R \). Note that some real numbers have two equivalent representations. For example 0.2 = 0.199999\ldots. In such cases we consider the later version.

We now construct a new real number \( r_{\text{new}} \) such that \( r_{\text{new}} \notin \text{range}(f) \). We define \( r_{\text{new}} \) as follows: we make sure that the \( k \)-th bit of \( r_{\text{new}} \) is different from the \( k \)-th bit of the number \( f(k) \). Let \( a_{kk} \) denote the \( k \)-th bit of \( f(k) \). The \( k \)-th bit of \( r_{\text{new}} = 5 \) if \( a_{kk} \neq 5 \), and the \( k \)-th bit of \( r_{\text{new}} = 6 \) if \( a_{kk} = 5 \). It is obvious that \( r_{\text{new}} \) is a real number.

We now claim that there is no \( m \) such that \( f(m) = r_{\text{new}} \). Assume there is a natural number \( m \) such that \( f(m) = r_{\text{new}} \). Let \( a_{mm} \) be the \( m \)-th bit of \( f(m) \). By definition of \( r_{\text{new}} \), the \( m \)-th bit of \( r_{\text{new}} \) is different from \( a_{mm} \). Thus \( r_{\text{new}} \neq f(m) \). Thus \( f \) is not a bijection from \( \mathbb{N} \) to \( R \). Thus \( R \) is not countably infinite. The technique is called Diagonalization.

Let \( \Sigma \) be any finite alphabet. We can show that \( \Sigma^\ast \) is countable. The idea is as follows: Let \( s \) be the cardinality of \( \Sigma \). Note that \( \Sigma^s \) has exactly \( s^s \) elements. \( \sigma^\ast \) is nothing but the union of \( \sigma^0, \sigma^1, \sigma^2 \cdots \). The function \( f \) maps \( \sigma^0 \) to 1, maps elements from \( \sigma^1 \) to next \( s \) numbers, maps elements from \( \sigma^2 \) to next \( s^2 \) numbers and so on. In general the elements from \( \sigma^s \) are mapped to numbers between \( s^s \) and \( s^{s^2} + 1 \).

What if \( \sigma \) is an infinite set? is it the case that \( \sigma^\ast \) is countable?

Now consider the alphabet \( \Sigma \) consisting of all characters from the keyboard. \( \Sigma^\ast \) is countable. Since the set of all Java programs is a subset of \( \Sigma^\ast \), the set of all Java programs is also countable. As seen before, the set of all functions from \( \mathbb{N} \) to \( \{0,1\} \) is not countable. This means that there are more functions than Java programs. So there are functions for which no Java programs exist.

This raises several questions: Above is an existential argument, It just shows that there are functions that cannot be computed by Java programs. Can we given an explicit example of such functions? What if we consider \( C \) programs instead of Java programs? Can we design a programming languages in future that can compute all functions? Or more generally, are there computational devices (quantum computers, cloud computes, DNA computers etc), that can be build in future and they have the power to compute all functions? What is a computational device? What is a program? What does it mean for a program to compute a function?

First part of this course will answer these questions.