The set of all real numbers (numbers on the number line) is denoted $\mathbb{R}$. The (set of all) integers (whole numbers including negatives, zero, and positives) is denoted $\mathbb{Z}$. The set of all (strictly) positive integers is denoted $\mathbb{Z}^+$. The set of all natural numbers is the set of all nonnegative integers and is denoted $\mathbb{N}$. **Note**: we use the convention that 0 is a natural number.

Recursive Definitions of Sets: The set S (pick a name) is defined by:

Basis Step:	Specify finitely many elements of S
Recursive Step:	Give rule(s) for creating a new element of S from known values existing in S, and potentially other values.

The set S then consists of all and only elements that are put in S by finitely many (a nonnegative integer number) of applications of the recursive step after the basis step.

Let X and Y be sets. The **Cartesian product** of X and Y, denoted $X \times Y$, is the set of all ordered pairs (x, y) where $x \in X$ and $y \in Y$ $X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}$ Conventions: (1) Cartesian products can be chained together to result in sets of n-tuples and (2) When we form the Cartesian product of a set with itself $X \times X$ we can denote that set as X^2, or X^n for the Cartesian product of a set with itself n times for a positive integer n.

Let X and Y be sets of strings over the same alphabet. The **set-wise concatenation** of X and Y, denoted $X \circ Y$, is the set of all results of string concatenation xy where $x \in X$ and $y \in Y$ $X \circ Y = \{xy \mid x \in X \text{ and } y \in Y\}$

Union: When A and B are sets, $A \cup B = \{x \mid x \in A \vee x \in B\}$

Intersection: When A and B are sets, $A \cap B = \{x \mid x \in A \wedge x \in B\}$

Set difference: When A and B are sets, $A - B = \{x \mid x \in A \wedge x \notin B\}$

Disjoint sets: sets A and B are disjoint means $A \cap B = \emptyset$

Power set: When U is a set, $\mathcal{P}(U) = \{X \mid X \subseteq U\}$

RNA is made up of strands of four different bases that encode genomic information in specific ways. The bases are elements of the set $B = \{\mathtt{A}, \mathtt{C}, \mathtt{U}, \mathtt{G}\}$. The set of RNA strands S is defined (recursively) by:

Basis Step:	$\mathtt{A} \in S, \mathtt{C} \in S, \mathtt{U} \in S, \mathtt{G} \in S$
Recursive Step:	If $s \in S$ and $b \in B$, then $sb \in S$

where sb is string concatenation.

The function *rnalen* that computes the length of RNA strands in S is defined recursively by:

		$rnalen : S$	$\to \mathbb{Z}^+$
Basis Step:	If $b \in B$ then	$rnalen(b)$	$= 1$
Recursive Step:	If $s \in S$ and $b \in B$, then	$rnalen(sb)$	$= 1 + rnalen(s)$

The function *basecount* that computes the number of a given base b appearing in a RNA strand s is defined recursively by:

		$basecount : S \times B$	$\to \mathbb{N}$	
Basis Step:	If $b_1 \in B, b_2 \in B$	$basecount(\ (b_1, b_2)\ )$	$= \begin{cases} 1 & \text{when } b_1 = b_2 \\ 0 & \text{when } b_1 \neq b_2 \end{cases}$	
Recursive Step:	If $s \in S, b_1 \in B, b_2 \in B$	$basecount(\ (sb_1, b_2)\ )$	$= \begin{cases} 1 + basecount(\ (s, b_2)\ ) & \text{when } b_1 = b_2 \\ basecount(\ (s, b_2)\ ) & \text{when } b_1 \neq b_2 \end{cases}$	

Calculating integer part of base b logarithm

```
1   procedure logb(b,n: positive integers with b>1)
2   i := 0
3   while n > b-1
4     i := i+1
5     n := n div b
6   return i {i holds the integer part of the base b logarithm of n}
```

Let n be an integer and d a positive integer. There are unique integers q and r, with $0 \leq r < d$, such that $n = dq + r$. In this case, d is called the divisor, n is called the **dividend**, q is called the **quotient**, and r is called the **remainder**. Because these numbers are guaranteed to exist, the following functions are well-defined:

div $: \mathbb{Z} \times \mathbb{Z}^+ \to \mathbb{Z}$ given by **div** $(\ (n, d)\ )$ is the quotient when n is the dividend and d is the divisor. We use the notation n **div** $d =$ **div** $(\ (n, d)\ )$

mod $: \mathbb{Z} \times \mathbb{Z}^+ \to \mathbb{Z}$ given by **mod** $(\ (n, d)\ )$ is the remainder when n is the dividend and d is the divisor.We use the notation n **mod** $d =$ **mod** $(\ (n, d)\ )$.

Calculating base b expansion, from left

```
1  procedure basebmost(n, b: positive integers with b > 1)
2  v := n
3  k := 1+ output of logb algorithm with inputs b and n
4  for i := 1 to k
5     a_{k-i} := 0
6     while v ≥ b^{k-i}
7        a_{k-i} := a_{k-i} + 1
8        v := v - b^{k-i}
9  return (a_{k-1}, ..., a_0){(a_{k-1} ... a_0)_b is the base b expansion of n}
```

Calculating base b expansion, from right

```
1  procedure basebleast(n, b: positive integers with b > 1)
2  k := 0
3  q := n
4  while q ≠ 0
5     a_k := q mod b
6     k := k + 1
7     q := q div b
8  return (a_{k-1}, ..., a_0){(a_{k-1} ... a_0)_b is the base b expansion of n}
```

For b an integer greater than 1 and n a positive integer, the **base b expansion of** n is $(a_{k-1}\cdots a_1a_0)_b$ where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are (symbols for) nonnegative integers less than b, $a_{k-1} \neq 0$, and $n = \sum_{i=0}^{k-1} a_i b^i$

Base b	Collection of possible coefficients in base b expansion of a positive integer
Binary ($b = 2$)	$\{0, 1\}$
Ternary ($b = 3$)	$\{0, 1, 2\}$
Octal ($b = 8$)	$\{0, 1, 2, 3, 4, 5, 6, 7\}$
Decimal ($b = 10$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Hexadecimal ($b = 16$)	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ letter coefficient symbols represent numerical values $(A)_{16} = (10)_{10}$ $(B)_{16} = (11)_{10}$ $(C)_{16} = (12)_{10}$ $(D)_{16} = (13)_{10}$ $(E)_{16} = (14)_{10}$ $(F)_{16} = (15)_{10}$

For b an integer greater than 1, w a positive integer, and n a nonnegative integer the **base b fixed-width w expansion of** n is $(a_{w-1}\cdots a_1a_0)_{b,w}$ where $a_0, a_1, \ldots, a_{w-1}$ are nonnegative integers less than b and $n = \sum_{i=0}^{w-1} a_i b^i$

For n a positive integer and width w a positive integer greater than 1

- To represent n in sign-magnitude representation with width w, use $[0a_{w-2}\cdots a_0]_{s,w}$ where $n = (a_{w-2}\cdots a_0)_{2,w-1}$
- To represent n in 2s complement representation with width w, use $[0a_{w-2}\cdots a_0]_{2c,w}$ where $n = (a_{w-2}\cdots a_0)_{2,w-1}$
- To represent $-n$ in sign-magnitude representation with width w, use $[1a_{w-2}\cdots a_0]_{s,w}$ where $n = (a_{w-2}\cdots a_0)_{2,w-1}$
- To represent n in 2s complement representation with width w, use $[1a_{w-2}\cdots a_0]_{2c,w}$ where $2^{w-1} - n = (a_{w-2}\cdots a_0)_{2,w-1}$.

For b an integer greater than 1, w a positive integer, w' a positive integer, and x a real number the **base b fixed-width expansion of x with integer part width w and fractional part width w'** is $(a_{w-1}\cdots a_1a_0.c_1\cdots c_{w'})_{b,w,w'}$ where $a_0, a_1, \ldots, a_{w-1}, c_1, \ldots, c_{w'}$ are nonnegative integers less than b and $x \geq \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j}$ and $x < \sum_{i=0}^{w-1} a_i b^i + \sum_{j=1}^{w'} c_j b^{-j} + b^{-w'}$

A **hex color** is a nonnegative integer, n, that has a base 16 fixed-width 6 expansion $n = (r_1r_2g_1g_2b_1b_2)_{16,6}$ where $(r_1r_2)_{16,2}$ is the red component, $(g_1g_2)_{16,2}$ is the green component, and $(b_1b_2)_{16,2}$ is the blue.

In a **combinatorial circuit** (also known as a **logic circuit**), we have **logic gates** connected by **wires**. The inputs to the circuits are the values set on the input wires: possible values are 0 (low) or 1 (high). The values flow along the wires from left to right. A wire may be split into two or more wires, indicated with a filled-in circle (representing solder). Values stay the same along a wire. When one or more wires flow into a gate, the output value of that gate is computed from the input values based on the gate's definition table. Outputs of gates may become inputs to other gates.

Gate operations

- **AND**: Output is 1 if both inputs are 1, otherwise 0 
- **OR**: Output is 1 if at least one input is 1, otherwise 0 
- **XOR**: Output is 1 if inputs are different, otherwise 0 
- **NOT**: Output is the opposite of the input (0→1, 1→0) 

An expression built of variables and logical operators is in **disjunctive normal form** (DNF) means that it is an OR of ANDs of variables and their negations.

An expression built of variables and logical operators is in **conjunctive normal form** (CNF) means that it is an AND of ORs of variables and their negations.

Input		Output				
		Conjunction	Exclusive or	Disjunction	Conditional	Biconditional
p	q	$p \wedge q$	$p \oplus q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	T	F	T	T	T
T	F	F	T	T	F	F
F	T	F	T	T	T	F
F	F	F	F	F	T	T
		"p and q"	"p xor q"	"p or q"	"if p then q"	"p if and only if q"

Logical equivalence: Two compound propositions are **logically equivalent** means that they have the same truth values for all settings of truth values to their propositional variables.

- **Tautology**: A compound proposition that evaluates to true for all settings of truth values to its propositional variables; it is abbreviated T.
- **Contradiction**: A compound proposition that evaluates to false for all settings of truth values to its propositional variables; it is abbreviated F.
- **Contingency**: A compound proposition that is neither a tautology nor a contradiction.

A collection of compound propositions is called **consistent** if there is an assignment of truth values to the propositional variables that makes each of the compound propositions true.

A **predicate** is a function from a given set (domain) to $\{T, F\}$.

A predicate can be applied, or **evaluated** at, an element of the domain.

Usually, a predicate *describes a property* that domain elements may or may not have.

Two predicates over the same domain are **equivalent** means they evaluate to the same truth values for all possible assignments of domain elements to the input. In other words, they are equivalent means that they are equal as functions.

To define a predicate, we must specify its domain and its value at each domain element. The rule assigning truth values to domain elements can be specified using a formula, English description, in a table (if the domain is finite), or recursively (if the domain is recursively defined).

Note: Since logical operations are functions, they can be part of the definitions of predicates.

The **universal quantification** of predicate $P(x)$ over domain U is the statement "$P(x)$ for all values of x in the domain U" and is written $\forall x P(x)$ or $\forall x \in U\ P(x)$. When the domain is finite, universal quantification over the domain is equivalent to iterated *conjunction* (ands).

The **existential quantification** of predicate $P(x)$ over domain U is the statement "There exists an element x in the domain U such that $P(x)$" and is written $\exists x P(x)$ for $\exists x \in U\ P(x)$. When the domain is finite, existential quantification over the domain is equivalent to iterated *disjunction* (ors).

A **set** is an unordered collection of elements. When A and B are sets, $A = B$ (set equality) means $\forall x(x \in A \leftrightarrow x \in B)$

When A and B are sets, $A \subseteq B$ ("A is a **subset** of B") means $\forall x(x \in A \rightarrow x \in B)$

When A and B are sets, $A \subsetneq B$ ("A is a **proper subset** of B") means $(A \subseteq B) \wedge (A \neq B)$

When a and b are integers and a is nonzero, a **divides** b means there is an integer c such that $b = ac$.

Symbolically, $F(\ (a, b)\ ) = \exists c \in \mathbb{Z}\ (b = ac)$ and is a predicate over the domain $\mathbb{Z}^{\neq 0} \times \mathbb{Z}$. Other (synonymous) ways to say that $F(\ (a, b)\ )$ is true are "a is a **factor** of b" and "a is a **divisor** of b" and "b is a **multiple** of a" and "$a|b$".

When a is a positive integer and b is any integer, $a|b$ exactly when b **mod** $a = 0$

When a is a positive integer and b is any integer, $a|b$ exactly when $b = a \cdot (b$ **div** $a)$

An integer p greater than 1 is called **prime** means the only positive factors of p are 1 and p. A positive integer that is greater than 1 and is not prime is called composite.

The set of linked lists of natural numbers L is defined recursively by

Basis Step:	$[] \in L$
Recursive Step:	If $l \in L$ and $n \in \mathbb{N}$, then $(n, l) \in L$

The function $append : L \times \mathbb{N} \rightarrow L$ that adds an element at the end of a linked list is defined by

Basis Step:	If $m \in \mathbb{N}$ then	$append(\ ([], m)\ )$	$= (m, [])$
Recursive Step:	If $l \in L$ and $n \in \mathbb{N}$ and $m \in \mathbb{N}$, then	$append(\ (\ (n, l), m\ )\ )$	$= (n, append(\ (l, m)\ )\ )$

The function $increment : L \rightarrow L$ that adds one to each data value in the nodes of the list is defined by:

		$increment : L$	$\rightarrow L$
Basis Step:		$increment([])$	$= []$
Recursive Step:	If $l \in L, n \in \mathbb{N}$	$increment((n, l))$	$= (1 + n, increment(l))$

The length of a linked list of natural numbers L, $length : L \rightarrow \mathbb{N}$ is defined by

Basis Step:		$length(\ []\ )$	$= 0$
Recursive Step:	If $l \in L$ and $n \in \mathbb{N}$, then	$length(\ (n, l)\ )$	$= 1 + length(l)$

The function $prepend : L \times \mathbb{N} \rightarrow L$ that adds an element at the front of a linked list is defined by $prepend((l, n)) = (n, l)$

The function $sum : L \rightarrow \mathbb{N}$ that adds together all the data in nodes of the list is defined by:

		$sum : L$	$\rightarrow \mathbb{N}$
Basis Step:		$sum([])$	$= 0$
Recursive Step:	If $l \in L, n \in \mathbb{N}$	$sum((n, l))$	$= n + sum(l)$

Euclidean algorithm for calculating greatest common divisor

```
1    procedure Euclidean(a: a positive integer, b: a positive integer)
2    x := a
3    y := b
4    while y ≠ 0
5      r := x mod y
6      x := y
7      y := r
8    return x {the result of gcd( (a, b) )}
```

Greatest common divisor Let a and b be integers, not both zero. The largest integer d such that d is a factor of a and d is a factor of b is called the greatest common divisor of a and b and is denoted by $gcd(\ (a, b)\ )$.

Proof strategies discussed in this class are as follows:

- A **counterexample** can be used to prove that $\forall x P(x)$ is *false*.
- A **witness** can be used to prove that $\exists x P(x)$ is *true*.
- Proof by **universal generalization**: To prove that $\forall x\, P(x)$ is *true*, we can take an arbitrary element e from the domain and show that $P(e)$ is true, without making any assumptions about e other than that it comes from the domain.
- To prove that $\exists x P(x)$ is *false*, write the universal statement that is logically equivalent to its negation and then prove it true using universal generalization.
- To prove that $p \wedge q$ is true, have two subgoals: subgoal (1) prove p is true; and, subgoal (2) prove q is true. To prove that $p \wedge q$ is false, it's enough to prove that p is false. To prove that $p \wedge q$ is false, it's enough to prove that q is false.
- Proof of Conditional by **Direct Proof**: To prove that the implication $p \to q$ is *true*, we can assume p is true and use that assumption to show q is true.
- Proof of Conditional by **Contrapositive Proof**: To prove that the implication $p \to q$ is *true*, we can assume $\neg q$ is true and use that assumption to show $\neg p$ is true.
- **Proof by Cases**: To prove q when we know $p_1 \vee p_2$, show that $p_1 \to q$ and $p_2 \to q$.
- Proof by **Structural Induction**: To prove that $\forall x \in X\, P(x)$ where X is a recursively defined set, prove two cases:
  - Basis Step: Show the statement holds for elements specified in the basis step of the definition.
  - Recursive Step: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.
- Proof by **Mathematical Induction**: To prove a universal quantification over the set of all integers greater than or equal to some base integer b:
  - Basis Step: Show the statement holds for b.
  - Recursive Step: Consider an arbitrary integer n greater than or equal to b, assume (as the **induction hypothesis**) that the property holds for n, and use this and other facts to prove that the property holds for $n+1$.
- Proof by **Strong Induction**: To prove that a universal quantification over the set of all integers greater than or equal to some base integer b holds, pick a fixed nonnegative integer j and then:
  - Basis Step: Show the statement holds for $b, b+1, \ldots, b+j$.
  - Recursive Step: Consider an arbitrary integer n greater than or equal to $b+j$, assume (as the **strong induction hypothesis**) that the property holds for *each of* $b, b+1, \ldots, n$, and use this and other facts to prove that the property holds for $n+1$.
- **Proof by Contradiction**: To prove that a statement p is true, pick another statement r and once we show that $\neg p \to (r \wedge \neg r)$ then we can conclude that p is true.

The **set of rational numbers**, $\mathbb{Q}$, is defined as $\left\{ \frac{p}{q} \mid p \in \mathbb{Z} \text{ and } q \in \mathbb{Z} \text{ and } q \neq 0 \right\}$.

A function $f : D \to C$ is a **bijection** means that it is both one-to-one and onto. The **inverse** of a bijection $f : D \to C$ is the function $g : C \to D$ such that $g(b) = a$ iff $f(a) = b$.

A function $f : D \to C$ is **one-to-one** (or injective) means for every a, b in the domain D, if $f(a) = f(b)$ then $a = b$. Formally, $f : D \to C$ is one-to-one means $\forall a \in D \forall b \in D\ (f(a) = f(b) \to a = b)$.

A function $f : D \to C$ is **onto** (or surjective) means for every b in the codomain, there is an element a in the domain with $f(a) = b$. Formally, $f : D \to C$ is onto means $\forall b \in C \exists a \in D(f(a) = b)$.

For nonempty sets A, B we say

$|A| \leq |B|$ means there is a one-to-one function with domain A, codomain B
$|A| \geq |B|$ means there is an onto function with domain A, codomain B
$|A| = |B|$ means there is a bijection with domain A, codomain B

For all sets A, we say $|A| = |\emptyset|$, $|\emptyset| = |A|$ if and only if $A = \emptyset$.

For nonempty sets A, B, we say that **the cardinality of A is no bigger than the cardinality of B**, and write $|A| \leq |B|$, to mean there is a one-to-one function with domain A and codomain B. Also, we define $|\emptyset| \leq |B|$ for all sets B.

For nonempty sets A, B, we say that **the cardinality of A is no smaller than the cardinality of B**, and write $|A| \geq |B|$, to mean there is an onto function with domain A and codomain B. Also, we define $|A| \geq |\emptyset|$ for all sets A.

A set A is **countably infinite** means it is the same size as $\mathbb{N}$.

A **finite** set is one whose distinct elements can be counted by a natural number.

When A and B are sets, we say any subset of $A \times B$ is a **binary relation**. A relation R can also be represented as

- A function $f_{TF} : A \times B \to \{T, F\}$ where, for $a \in A$ and $b \in B$, $f_{TF}(\ (a, b)\ ) = \begin{cases} T & \text{when } (a, b) \in R \\ F & \text{when } (a, b) \notin R \end{cases}$
- A function $f_{\mathcal{P}} : A \to \mathcal{P}(B)$ where, for $a \in A$, $f_{\mathcal{P}}(a) = \{b \in B \mid (a, b) \in R\}$

When A is a set, we say any subset of $A \times A$ is a (binary) **relation** on A.

A relation R on a set A is called **antisymmetric** means $\forall a \in A\ \forall b \in A\ (\ (\ (a, b) \in R \wedge (b, a) \in R\ ) \to a = b\ )$

A relation R on a set A is called **reflexive** means $(a, a) \in R$ for every element $a \in A$.

A relation R on a set A is called **symmetric** means $(b, a) \in R$ whenever $(a, b) \in R$, for all $a, b \in A$.

A relation R on a set A is called **transitive** means whenever $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$, for all $a, b, c \in A$.

An **equivalence class** of an element $a \in A$ with respect to an equivalence relation R on the set A is the set $\{s \in A \mid (a, s) \in R\}$. We write $[a]_R$ for this set, which is the equivalence class of a with respect to R.

A relation is an **equivalence relation** means it is reflexive, symmetric, and transitive.

A **partition** of a set A is a set of non-empty, disjoint subsets $A_1, A_2, \cdots, A_n$ such that $A = \bigcup_{i=1}^{n} A_i = \{x \mid \exists i(x \in A_i)\}$

For a partial ordering, its **Hasse diagram** is a graph representing the relationship between elements in the ordering. The nodes (vertices) of the graph are the elements of the domain of the binary relation. The edges do not have arrow heads. The directionality of the partial order is indicated by the arrangements of the nodes. The nodes are arranged so that nodes connected to nodes above them by edges indicate that the relation holds between the lower node and the higher node. Moreover, the diagram omits self-loops and omits edges that are guaranteed by transitivity.

A relation is a **partial ordering** (or partial order) means it is reflexive, antisymmetric, and transitive.

Modular Exponentation

```
1  procedure modular exponentiation(b: integer;
2                n = (a_{k-1}a_{k-2}...a_1a_0)_2, m: positive integers)
3  x := 1
4  power := b mod m
5  for i:= 0 to k-1
6     if a_i = 1 then x:= (x · power) mod m
7     power := (power · power) mod m
8  return x {x equals b^n mod m}
```