A finite automaton (pronounced aw-TOM-uh-tahn) is a very simple model of a computer. Plural is “automata” (pronounced aw-TOM-uh-tah).

This model was invented by McCulloch and Pitts in 1943. They wanted to model neurons in the brain.
McCulloch and Pitts

Warren McCulloch (1898–1969) was a neurophysiologist who was interested in modeling the brain mathematically. He co-wrote the paper “A Logical Calculus of Ideas Immanent in Nervous Activity” in 1943, which is viewed as the start of automata theory.

Walter Pitts (1923–1969) co-wrote the famous paper with McCulloch. He also co-wrote “What the Frog’s Eye Tells the Frog’s Brain” in 1959, a pioneering paper in neuroscience.
An automaton consists of a finite number of states and transitions between them.

You can think of states as representing some aspect of a system, or knowledge about the input read so far.

For example, if the system is an automatic teller machine, then one state might be “awaiting user to input card”, another might be “two keystrokes of PIN are entered, waiting for third”, etc.

If the input is a string of symbols, a state might represent “I’ve seen an occurrence of aaa so far, but not bb.”
Automata

Inputs to the automaton can cause it to change state. A transition function or transition table tells the automaton what to do: for each state, and each possible input symbol, it tells the automaton which state to enter.

Example: here is the transition table for a very simple automaton of two states:

<table>
<thead>
<tr>
<th>state</th>
<th>input</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_0 )</td>
<td>( q_1 )</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_1 )</td>
<td>( q_0 )</td>
<td>( q_1 )</td>
</tr>
</tbody>
</table>

Here the states are labeled \( q_0 \) and \( q_1 \).
States and acceptance

In an automaton, there is one \textit{start state} (also called an \textit{initial state}), which specifies the state to begin the computation in, and a set of \textit{final} or \textit{accepting} states.

Usually (but not always) the start state is labeled $q_0$.

An input string $x$ is \textit{accepted} if, starting in the initial state, reading $x$ and following the transitions labeled by the symbols of $x$, causes the automaton to end up in one of the final states.
Automata

Automata are often represented by a *transition diagram*, which is a graphical way to represent them.

The initial state is denoted by an arrow without a head (only a tail) entering a state.

Accepting states are denoted by double-circles and non-accepting states by single circles.

The transition function is given by labeled arrows.

For example:
The set of all strings accepted by an automaton $M$ is called the language recognized by the automaton, and is written $L(M)$.

For example, let’s design an automaton for the set of all strings of $a$’s and $b$’s having no substring equal to $aa$.

Notice: in this course, unlike in CS 241, there is no such thing as a special kind of distinguished state called an “error state”. States are just states.
The kind of automata we have been studying are called *deterministic finite automata*, or DFA’s.

Here is the formal definition. A DFA $M$ is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite, nonempty set of states;
- $\Sigma$ is the input alphabet;
- $q_0$ is the initial state;
- $F \subseteq Q$ is the set of final states;
- and $\delta : Q \times \Sigma \to Q$ is the transition function.

Note that $\delta$ is a *total* function; this means it must be defined on all elements of its domain.
Recall the automaton for the set of all strings of $a$’s and $b$’s having no substring equal to $aa$.

Here is the formal definition for this automaton: $(Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_0, q_1, q_2\}$;
- $\Sigma = \{a, b\}$;
- $F = \{q_0, q_1\}$;
- $\delta$ is defined by the table below:

<table>
<thead>
<tr>
<th>state</th>
<th>input</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td></td>
<td>$q_1$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
<td></td>
<td>$q_2$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_2$</td>
<td></td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>
Now that we have a formal mathematical model for an automaton, we need a formal definition of acceptance.

We’d like to say something like: a string $x$ is accepted if there is a path, labeled with the symbols of $x$, from $q_0$ to a state of $F$. Otherwise, it is rejected.

In order to capture the notion of “path from one state to another”, we need to extend the domain of the transition function. Instead of a function $\delta: Q \times \Sigma \to Q$, we need a new function $\delta^*: Q \times \Sigma^* \to Q$.

The idea is that $\delta^*(q, x)$ will give us the state the DFA would be in, if we start in state $q$ and read $x$. 
Here is where a recursive definition is powerful!

We define the so-called *extended transition function* $\delta^*$ as follows:

- $\delta^*(q, \epsilon) := q$
- $\delta^*(q, xa) := \delta(\delta^*(q, x), a)$ for strings $x \in \Sigma^*$ and single letters $a \in \Sigma$.

Then we say $x$ is *accepted by* the DFA $M$ if $\delta^*(q_0, x) \in F$. Otherwise it is *rejected* by $M$.

Similarly, we define $L(M)$, the *language recognized by* the DFA $M$, to be $\{x \in \Sigma^* : \delta^*(q_0, x) \in F\}$.

We say a language $L$ is *regular* if $L = L(M)$ for some DFA $M$. 
Formal definition of acceptance

Notice that the domain of $\delta$ (which is $Q \times \Sigma$) is a subset of the domain of $\delta^*$ (which is $Q \times \Sigma^*$).

Furthermore, by the definition of $\delta^*$, we see that the two functions agree on the domain of $\delta$: $\delta^*(q, a) = \delta(\delta^*(q, \epsilon), a) = \delta(q, a)$, if $a$ is a single symbol.

For these reasons, we can consider that $\delta^*$ is just an extension of the function $\delta$, and from now on we usually just use $\delta$ for both of them.
Automata parameterized by a variable

One nice thing about the formal definition of automata is we can give a precise, abstract definition for automata that depend on a variable.

For example, let’s define an automaton $M_n$ that accepts exactly one string, namely the string $a^n$.

Our automaton has $n + 2$ states, $Q = \{q_0, q_1, \ldots, q_{n+1}\}$.

It has one final state: $F = \{q_n\}$.

The transitions are: $\delta(q_i, a) = q_{i+1}$ for $0 \leq i \leq n$, and $\delta(q_{n+1}, a) = q_{n+1}$.

A state like $q_{n+1}$ is sometimes called a “dead state”, because starting from state $q_{n+1}$, it is not possible to reach any accepting state.
Let’s look at a famous example of an automaton, the person-goat-wolf-cabbage problem.

(This problem appeared in the 1st season of the TV show *Fargo*, Episode 9. It’s a great show!)

A person, wolf, goat, and cabbage are on one side of a river. There’s a rowboat that can take at most two of them across at one time. If left alone with the cabbage, the goat will eat it, and if left alone with the goat, the wolf will eat it. How can the person get all three across the river? Only the person can steer the boat.
Solution to the problem

Let’s make an automaton out of the possibilities, with a state like pwc-g meaning the person, wolf, and cabbage are on one side of the river, and the goat is on the other. Let’s draw transitions between the states if a trip across the river can take one state to the other. (Not all states and transitions are shown.)

The person-wolf-goat-cabbage problem (after Hopcroft and Ullman).

A person, wolf, goat, and cabbage are on one side of a river. There is a rowboat that can take at most two of them across at a time. However, wolf and goat. How can all four get to the other side?

If left alone with the cabbage, the goat will eat it, and the same for the wolf.

Note: not all transitions are shown.
If you are asked to construct a DFA for a language, first consider what pieces of information you will need about a string in order to decide whether to accept it.

Then make a state for each possible set of pieces of information.

States represent knowledge about the string you’ve seen so far.

Add transitions from states to states as appropriate.
Draw transition diagrams for the following languages. (Solutions are given in the following slides.)

1. $L$ is the language of strings over the alphabet \{a, b\} having an odd number of occurrences of the substring \textit{aa} (which are allowed to overlap).

2. $L$ is the language of strings over the alphabet \{a, b\} having either \textit{aa} or \textit{bb} as a substring.

3. $L$ is the language of binary strings representing integers in base 2 that are divisible by 3. Here leading zeroes are allowed.
1. $L$ is the language of strings over the alphabet $\{a, b\}$ having an odd number of occurrences of the substring $aa$ (which are allowed to overlap).

What does the automaton need to know?

- the previous symbol read (so it can tell if there's an $aa$ ending at the current symbol) and
- whether the number of occurrences of $aa$ seen so far is odd or even.
1. $L$ is the language of strings over the alphabet $\{a, b\}$ having an odd number of occurrences of the substring $aa$ (which are allowed to overlap).

So build a DFA to record these two pieces of information. States are pairs $[c, d]$ where

- $c$ is the previous symbol read (or $b$ if nothing)
- $d$ is the parity of the number of $aa$'s seen so far.

This gives the following DFA:
2. $L$ is the language of strings over the alphabet \( \{a, b\} \) having either $aa$ or $bb$ as a substring.

What does the automaton need to know?

- What the previous symbol seen is (so it can know whether the current symbol forms an $aa$ or $bb$)
- Whether an $aa$ or $bb$ has been seen so far.
2. $L$ is the language of strings over the alphabet $\{a, b\}$ having either $aa$ or $bb$ as a substring.

This gives the following DFA:
3. $L$ is the language of binary strings representing integers in base 2 that are divisible by 3. Here leading zeroes are allowed.

Here the needed insight is that states should represent residue classes modulo 3.

That is, if we reach state $q_i$ on input $x$, then $x$ in base 2 should leave a remainder of $i$ when divided by 3.
3. $L$ is the language of binary strings representing integers in base 2 that are divisible by 3. Here leading zeroes are allowed.

Next we have to figure out the effect of reading a 0 and 1 from the input.

If we are in the state representing a number $n$ in base 2, modulo 3, then reading a zero is like replacing $n$ with $2n \pmod{3}$.

If we are in the state representing a number $n$ in base 2, modulo 3, then reading a one is like replacing $n$ with $2n + 1 \pmod{3}$. 
3. $L$ is the language of binary strings representing integers in base 2 that are divisible by 3. Here leading zeroes are allowed.

This gives the following DFA:
Some misconceptions about DFA’s

Here are some of the most common student misconceptions about DFA’s:

▶ The transition function $\delta$ can be defined by saying how it behaves on all strings (instead of single letters).
  ▶ Not so. By definition the domain of $\delta$ is $Q \times \Sigma$, not $Q \times \Sigma^*$. If you define $\delta$ on $Q \times \Sigma^*$ you have no guarantee it will behave “consistently”. For example, what if you define $\delta(q, ab) = r$ and $\delta(r, cd) = s$ and $\delta(q, abcd) = t$ where $s \neq t$? Then it is not a DFA.

▶ The transition diagram has to be “connected”.
  ▶ Not so. The only requirement is that the transition function $\delta$ map $Q \times \Sigma$ to $Q$. 
More misconceptions about DFA’s

- The DFA recognizing a language $L$ has to be minimal (have the smallest possible number of states for $L$)

  - Not necessarily. Obviously, a smaller number of states is desirable, but sometimes a general construction will result in unneeded states.