We do not know too much about whether P = NP. Most people who have pondered the question believe that P ≠ NP.

However, we know of a very interesting phenomenon - a host of natural problems that are complete for NP.

A decision problem X is said to be NP-Complete if

(a) X ∈ NP

(b) For any Y ∈ NP,

\[ Y \leq_p X. \]

If only (b) holds, X is said to be NP-hard.
What is striking about this definition is the requirement (b), which says that every problem in $\text{NP}$ must be poly-time reducible to $X$. That brings up the question of whether there are any natural problems that are $\text{NP}$-Complete. Here on this shortly. First:

**Fact** Suppose $X$ is an $\text{NP}$-complete problem. Then $X \in \text{P}$ if and only if $\text{P} = \text{NP}$.

**Proof:** Suppose $\text{P} = \text{NP}$. Then since $X \in \text{NP}$, $X \in \text{P}$.

For other direction, suppose $X \in \text{P}$. Let $Y$ be any problem in $\text{NP}$. Since $X$ is $\text{NP}$-complete, we know $Y \leq_p X$. Since $X \in \text{P}$
and \( Y \leq_p X \), \( Y \in \mathbf{NP} \). We conclude that \( \mathbf{NP} \subseteq \mathbf{P} \), and so \( \mathbf{P} = \mathbf{NP} \).

The fact implies that if \( \mathbf{P} \neq \mathbf{NP} \), that is, if any problem in \( \mathbf{NP} \) can't be solved in poly-time, then no \( \mathbf{NP} \)-Complete problem can be solved in poly-time.

We now introduce the circuit satisfiability problem, which will be our first \( \mathbf{NP} \)-Complete problem. Here, we are given a combinatorial circuit involving and/or/not gates, and we want to know if there is an assignment to input nodes that causes output to evaluate to 1.
First some examples, then the formal details.

\[ \text{Input nodes} \]

\( \wedge \) - and
\( \lor \) - or
\( \neg \) - not

The above is a circuit. It is satisfiable - the assignment 1, 1, 0, 1 to input nodes is a satisfying assignment.
Here is a trivial circuit that is not satisfiable:

```
\[
\begin{array}{c}
\text{Input} \\
\rightarrow \\
\downarrow \\
\rightarrow \\
\text{Output}
\end{array}
\]
```

Formally, a circuit is a directed acyclic graph. The nodes with no incoming edges are called input nodes. Every other node is labelled either $\land$, $\lor$, or $\neg$. Nodes labelled $\land$ or $\lor$ have two incoming edges. Nodes labelled $\neg$ have one incoming edge. There is exactly one node with no outgoing edge. This is called the output node.
Since a circuit is a DAG, it can be topologically sorted. We may assume that input nodes occur first in this order. Given an assignment of T/F (or 1/0) values to input nodes, the other nodes can be evaluated as follows. We go thru the non-input nodes in order - when we arrive at node \( v \), all nodes \( u \) such that \((u, v)\) is an edge have been evaluated. We evaluate \( v \) as follows. Suppose \( v \) is an and labelled \( \land \) and has incoming edges from \( u_1 \) and \( u_2 \). Then \( v \) evaluates to the "and" of whatever \( u_1 \) and \( u_2 \) evaluated to. The evaluation of \( v \) if it is labelled \( \lor \) or \( \forall \) is defined similarly.
Finally, the circuit evaluates to whatever the output node evaluates to.

The Circuit-satisfiability problem then is to determine, given a circuit $C$, whether there is an assignment to input nodes of $C$ that causes the circuit to evaluate to true.

Theorem Circuit-Satisfiability is NP-Complete.

Proof Sketch:
It is easy to construct an efficient verifier for circuit-satisfiability. Thus the problem is in NP.
Let $Y$ be any problem in $NP$. To show $Y \leq_p \text{Circuit-Satisfiability}$, we describe a poly-time algorithm that takes an input an instance $x$ of $Y$ and outputs a circuit $D(x)$ so that

$x$ is a yes-instance of $Y$ iff $D(x)$ is satisfiable.

Our description of this algorithm will be very sketchy.

Since $Y \in NP$, it has an efficient verifier $B$. 
We know there is a polynomial $p$, so that if $x$ is a yes-instance of $Y$, there is a $t$ with $1t1 \leq p(1x1)$ so that $B(x, t)$ outputs yes.

Algorithm constructs a circuit $C$ that has "two" inputs $x$ sets of input nodes: $x$ and $t'$, where $1t'1 = p(1x1)$. Think of $t'$ as $t$ followed by "end of shing" pattern.
The Circuit C "simulate" B on x and t. Since B's running time is polynomial in \(|x| + |t|\), and \(|t| \leq p(|x|)\), B runs in only for a polynomial number of steps in x. So C needs to simulate B only for a polynomial number of steps, \((\text{polynomial in } |x|)\). So C will have size polynomial in \(|x|\).

Output node of C corresponds to output of B(x, t).
Finally, algo hard-codes $x$ by adding one input node $z$. For example, if $x_i = 0$, it adds

![Diagram](image1)

and if $x_i = 1$, it adds

![Diagram](image2)

Call resulting circuit $D(x)$. 
$D(x)$ has form

\[
\begin{array}{c}
\vdash \\
1 \\
\hline
D \\
1 \\
\end{array}
\]

Since $D(x)$ simulates $B$ on $x$ and $t$, $D(x)$ is satisfiable iff $\exists t$ with $1t1 \leq p(1x1)$, such that $B(x,t)$ outputs 1.

In other words, $D(x)$ is satisfiable iff $x$ is yes-instance of $Y$.

End of Proof Sketch.

Transforming $x$ to $D(x)$ is done in poly-time
Having one NP-complete problem makes it much easier to show other problems NP-complete. This is because of the following fact, which is easy to prove.

Fact. Suppose $Y$ is NP-complete. Suppose $X \in \text{NP}$, and $Y \leq_p X$. Then $X$ is NP-complete.

We will show Circuit-Sat's hardness \( \leq_p \text{3CNF-SAT} \). Using Since \( \text{3CNF-SAT} \in \text{NP} \), we conclude \( \text{3CNF-SAT} \) is NP-complete.
Since $3\text{CNF-SAT} \leq_p \text{IND-SET}$,
$3\text{CNF-SAT} \leq_p \text{Vertex-Cover}$,
$3\text{CNF-SAT} \leq_p \text{Set-Cover}$,
and all these problems on $\mathcal{RHS}$
are in $\mathcal{NP}$, we conclude:

$\text{IND-SET}$, $\text{Vertex-Cover}$, $\text{Set-Cover}$
are $\mathcal{NP}$-complete.

To show some new problem $Z$
to be $\mathcal{NP}$-complete, we show $Z \in \mathcal{NP}$. This is usually easy.
We pick a problem $Y$ known to
be $\mathcal{NP}$-complete, we show $Y \leq_p Z$.
This can be harder, but choice of
$Y$ can greatly help us. The homework
gives you some experience with this process.