Definitions:

- A language L is **recognizable** iff there exists a Turing Machine (TM) M such that L(M) = L. Here, M is called a **recognizer**.
  Note: M will halt and accept only the strings in that language. For strings not in that language, M will either reject it, in which it will halt, or M will loop. With recognizers, there is no requirement to halt.
  Note: On a given language L, a recognizer will either
    a. Halt and accept L or
    b. Halt and reject L or
    c. Loop on L
  Note: If a TM, M, doesn't accept a language, L, it doesn't mean that it rejects L. It could reject L or it could loop on L
- A language is **decidable** iff there exists a TM M such that L(M) = L and M halts on every input. Here, M is called a **decider**.
- U, the universal language, = {⟨M, x⟩ | M accepts x}.
  U is recognizable but not decidable.
- **M_u**, the universal TM, takes ⟨M, x⟩ as input and simulates M on x.
- The halting problem, denoted as **H**, is H = {⟨M, x⟩ | M halts on x}.
  H is recognizable but not decidable.

General Turing Reduction:

- Reduction allows us to easily prove more languages are undecidable or unrecognizable.
- Let P and Q be languages.
- P **Turing-reduces** to Q, denoted as $P \leq_T Q$, if there exists an algorithm for P that uses an algorithm for Q as a "black box".
- Here are the general steps to prove that L is undecidable by using reduction:
    1. Assume L is decidable.
    2. Therefore, there is a TM M1 that decides it.
    3. Show that we can construct a TM M2 that uses M to decide U or some other undecidable problem.
    4. Since this contradicts that U is undecidable, L is undecidable.

Mapping Reduction:

- **Definition:** Let P and Q $\subseteq \Sigma^*$ be languages. P is **mapping-reducible** to Q, denoted as $P \leq_m Q$, iff there exists a computable function, $f : \Sigma^* \rightarrow \Sigma^*$, such that $x \in P$ iff $f(x) \in Q$. The function f is called the reduction of P to Q.
  Note: The function, f, does not have to be, and is usually not, onto.
  Note: The function, f, must be computable.
  To demonstrate a computable function, we will typically write a little program or describe in English how to perform the transformation that f is supposed to do.
  Note: f maps yes-instances of P to yes-instances of Q and no-instances of P to no-instances of Q.
- In general, when we are mapping-reducing language P to language Q, f should take an input of P as an input and output something that is an input of Q.
- **Theorems:**
  Suppose that $P \leq_m Q$
    1. **If Q is decidable, then P is decidable.**
       This is because we need to use a given solution to Q to solve P. If Q is decidable, then that means it halts on every input. Since P uses the output of Q on the input, P must halt on every input, too. Hence, P is decidable.

2. **If P is undecidable, then Q is undecidable.**
   This is because we need Q to solve P. If P isn't solvable, then neither is Q.
3. **If Q is recognizable, then P is recognizable.**
   This is because we need to use a given solution to Q to solve P. If Q is recognizable, then that means it either accepts, rejects or loops on every input. Since P uses the output of Q on the input, P must accept, reject or loop on every input, too. Hence, P is recognizable.
4. **If P is unrecognizable, then Q is unrecognizable.**
   This is because we need Q to solve P. If P isn't solvable, then neither is Q.
5. **If P $\leq_m$ Q, then !P $\leq_m$!Q, where !P is the complement of P and !Q is the complement of Q.**

Note: If P $\leq_m$ Q and Q is unrecognizable, it doesn't tell us if P is recognizable or not.
Note: If P $\leq_m$ Q and Q is undecidable, it doesn't tell us if P is decidable or not.

- To prove that a language P is unrecognizable or undecidable, it suffices to prove that U $\leq_m$ P, for undecidable, and !U $\leq_m$ P, for unrecognizable.