Def.: A language $A$ is (Turing-)recognizable if there is a deterministic TM $M$ s.t. $A = L(M)$.

Synonyms:
- Turing-acceptable (safer W.HO)
- recursively enumerable (r.e.) Historical term.
- computably enumerable (c.e.) modern alternative.
- "partially decidable" (on the "yes" side)

Def.: A DTM $M$ is total, synonym halts for all inputs if for all $x \in \Sigma^*$, $M(x) \downarrow = "halts."

Def.: The language $A$ is decidable if it equals $L(M)$ for some total DTM.

Synonym: recursive Historical: concept original, defined for formal systems of recursion.

Def.: The class of decidable languages is denoted by $\text{DEC}$ or $\text{REC}$.
The class of Turing-recognizable languages is only called $\text{RE}$.

Def.: A function $f: \Sigma^* \rightarrow \Sigma^*$ is computable if there is a Turing machine transducer (i.e., a DTM $M$ that produces output) s.t. for all $x$, $M(x) \downarrow = f(x)$.

Synonym: $f$ is recursive or total recursive.

If we allow $\text{Dom}(f)$ to not be all of $\Sigma^*$, and allow $M(x)$ might not halt for $x \notin \text{Dom}(f)$, then $f$ is partial, computable synonym: partial recursive.

We will now see that these def's apply not just to TMs but to every other high-level model or programming language (HLL).
Some basic capabilities of TMs that we’ve seen (in demos): 

- Simple arithmetic, like $3 + 1$. With a 3-type TM we can directly execute standard addition in binary. Can do multiplication either directly ("CISC") or by repeated addition ("RISC-like")

  - Mut by 2 and div. by 2 are done by shifting.

- Compare two strings char by char, forward or backward.
  - A second tape helps speed this up. Can also keep count, pointers.
  - This capability can be used to search for a label on a tape.

- Copy strings from tape to tape.

- If not enough room to copy w/betw. two `[' ]` delimiters, then we can automatically enter a "Shift-Over" routine on reading the `]`

Can use same routine $I_0(r)$ for TMs whose tapes have a fixed left edge.

These capabilities suffice to emulate a "Mini Assembly" language, actually built in - even with

| Load Indirect $i$ | Look up the value in register $i$, treat as another address and load the value $v'$ into ALU from there.
| Store Indirect $i$ | Look up $v$ at reg $i$, then copy ALU into $v$. |

Handout: What it shows is:

**Theorem:** If a function $f$ is (partially) computable in any HLL, then it is (partially) computable on a TM. Likewise for (partially) deciding language.

This is the largest evidence for the Church-Turing Thesis: Every machine in Nature, any human decision criterion, will be no stronger than the TM-based criteria.
Theorem: There is a Universal TM, i.e., a TM $U$ that takes input of the form $W = M\#x$ and such that $U(W)$ simulates $M(x)$.

Proof: We can give input $W$ already to the Turing Kit.

Compile Turing Kit to mini-assembly program $P$.

Use $U =$ the UTM Simulator with $P, M, x$ hard-coded.

$W = M\#x$ as its input. \( \square \)

Note: $L(U) = \{ <M,x> : M \text{ accepts } x \}$ is the language of the TM Acceptance Problem. Hence $A_m$ is a c.e. language. Is it decidable? We'll see -- not!

Theorem: If $A$ is decidable then so is its complement $\overline{A}$.

Fact: Not all OPDA are total, but every OPDA $D$ can be converted to an equivalent $D'$ that is total. (6.2.4)

Theorem: Every $K$-tape TM $M$ can be converted into an equivalent 1-tape TM $M'$, so $M'$ is total iff $M$ is.

(proof skipped)

Footnote: We should include as a basic capability on page 2 the idea of testing a cell for 0 or Op blank). This was exemplified in the "3n+1" machine example by the final test for $n=1$. We can fact a whole string in a register for being all-0s. This implements JMP = jump code to label & if ALU = 0.