To figure out which languages are recognised by a Turing machine we need to introduce the notion of a confirmation. As a Turing machine goes through its computations, changes take place in:

1. The state of the machine
2. The tape contents
3. The tape head location

A setting of these three items is called a **configuration**.

Configurations

Representing Configurations:
We represent a configuration as U S_i V, where the U, V are strings in the tape alphabet Ã and S_i is the current state of the machine. The tape contents are then the string UV and the current location of the tape head is on the first symbol of V. The assumption here is that the tape contains only blanks after the last symbol in V.

Example: εi001 is the configuration [0][0][1][_][… Tape Head
as we start examining the string 001 in our previous example of a Turing machine.

Definition:
Let C_1, C_2 be two configurations of a given Turing machine. We say that the configuration C_1 **yields** the configuration C_2 if the Turing machine can go from C_1 to C_2 in one step.

Example: If s_i,s_j are states, u and v are strings in the tape alphabet Ã, and a,b,c ∈ Ã.

A configuration C_1 = us_ibv yields a configuration C_2 = us_jacv if the transition mapping t specifies a transition t(s_i, b) = (s_j,c, L).
In other words, the Turing machine is in the state s_i, it reads character b, writes character c in its place, enters state s_j, and its head moves left.

Types of Configurations:

1. **Initial Configuration** with input u is iu, which indicates that the machine is in the initial state i with its head at the leftmost position on the tape (which is the reason why this configuration has no strings left of the state).
2. **Accepting Configuration** $us_{acc}v$ for u,v ∈ Ã* (u,v string in Ã), namely the machine in the accept state.
3. **Rejecting Configuration** $us_{rej}v$ for u,v ∈ Ã*, namely the machine in the reject state.
4. **Halting Configurations** yield no further configurations - no transitions are defined out of their states. Accepting and rejecting configurations are examples of halting configurations.

Definition:
A Turing machine M accepts input w ∈ A* (string over the input alphabet A) if ∃ a sequence of configurations C_1, C_2, …, C_k such that:

1. C_1 is the start configuration with input w.
2. Each C_i yields C_{i+1} for i = 1, ... , k-1.
3. C_k is an accepting configuration,

Definition:
Let M be a Turing machine. L(M) = {w ∈ A* | M accepts w} is the language recognised by M.

Definition:
A language $L \subset A^*$ is called **Turing-recognisable** if $\exists$ a Turing machine that recognises L, i.e. L = L(M).

NB: Some textbooks use the terminology **recursively enumerable language** (RE language) instead of Turing-recognisable.

Turing recognisable is not necessarily as strong a notion as we might need because a Turing machine can:

1. Accept
2. Reject
3. Loop

Looping is any simple or complex behaviour that does not lead to a halting state. The problem with looping is that the user does not have infinite time. It can be difficult to distinguish between looping or taking a very long time to compute. We thus prefer deciders.

Definition:
A **decider** is a Turing machine that enters either an accept state or a reject state for every input in A*.

Definition:
A decider that recognises some language $L \subset A^*$ is said to **decide** that language.

Definition:
A language $L \subset A^*$ is called **Turing-decidable** if $\exists$ a Turing machine M that decides L.

NB: Some textbooks use the terminology **recursive language** instead of Turing-decidable.

Example:
$L = \{0^m1^m \mid m \in N, m \geq 1\}$ is Turing-decidable because the Turing machine we built that recognises it was in fact a decider. (check again to convince yourself that the machine did not loop.)
Turing-decidable $\Rightarrow$ Turing-recognisable, but the converse is not true.
Turing-recognisable $\not\Rightarrow$ Turing-decidable.
We will hopefully have time to cover an example of a language that is Turing-recognisable, but **not** Turing-decidable before the end of the term.

Variants of Turing Machines

Task: Explore variants of the original setup of a Turing machine and show they do not enlarge the set of Turing-recognisable languages.

A) Add "stay put" to the list of allowable directions
Say instead of allowing just {L, R} (The tape head moves left or right) we also allow the "stay put" option (no change in the position of the tape head).
Thus, the transition mapping is defined as $t : S \times Ã \rightarrow S \times Ã \times \{L, R, N\}$ where N is for "no movement" or "stay put" instead of $t : S \times Ã \rightarrow S \times Ã \times \{L, R\}$.
We realise N is the same as R+L or L+R (move the tape head left then right and vice versa)
⇒ variant A) yields no increase in computational power

B) Multiple Turing machines
We allow the Turing machine to have several tapes, each with its own tape head for reading and writing. Initially, the input is on tape 1, and the others are blank. The transition mapping then must allow for reading, writing and moving the tape head on some or all of the tapes simultaneously. If k is the number of tapes, then the transition mapping is defined as
$t : S \times Ã^k \rightarrow S \times Ã^k \times \{L, R, N\}^k$ where:

- $Ã^k$ = k-fold Cartesian product, i.e. (Ã x Ã x … x Ã) k times
- $\{L, R, N\}^k$ = k-fold Cartesian product, i.e. ({L, R, N} x {L, R, N} x … x {L, R, N}) k times

Since one of the tape heads or more might not move for some transitions, we make use of the option N ("no movement") besides left and right.
Multiple Turing machines seem more powerful than ordinary simple-tape ones, but that is **not** the case.

Definition:
We call the two Turing machines M_1 and M_2 **equivalent** if $L(M_1) = L(M_2)$, namely if they recognise the same language.

Theorem: Every multi-tape Turing machine has an equivalent single-tape Turing machine.

Sketch of Proof:
Let M^k be a Turing machine with k tapes. We will simulate it with a simple-tape Turing machine M^1 constructed as follows:
We add # to the tape alphabet Ã and use it to separate the contents of the different tapes. M^1 also needs to keep track of the locations of the tape heads of M^k. It does so by adding a dot to the character to which a tape head is pointing. We thus only need to enlarge the tape alphabet Ã by allowing a version with a dot above for every character in Ã apart from # and the blank symbol _.

Corollary:
A language L is Turing-recognisable ⇔ some multi-tape Turing machine recognises L.

Proof:
"⇒" A language L is Turing-recognisable if ∃ M a single-tape Turing machine that recognises it. A single-tape Turing machine is a special type of a multi-tape Turing machine, so we are done.
"⇐" follows from the previous theorem. (*q.e.d*)