1. Let \( \langle M \rangle \) denote the encoding of a Turing machine \( M \) (or if you prefer, the Python source code for the executable code \( M \)). Recall that \( w^R \) denotes the reversal of string \( w \). Prove that the following language is undecidable.

\[
\text{SelfRevAccept} := \{ \langle M \rangle \mid M \text{ accepts the string } \langle M \rangle^R \}
\]

Note that Rice’s theorem does not apply to this language.

**Solution (diagonalization):** For the sake of argument, suppose that there is a Turing machine \( \overline{SRA} \) that decides \( \text{SelfRevAccept} \). For any Turing machine \( M \), we have

\[
\overline{SRA} \text{ accepts } \langle M \rangle \iff M \text{ accepts } \langle M \rangle^R.
\]

Let \( \overline{SRA} \) be the Turing machine obtained from \( SRA \) by swapping its accept and reject states. For any Turing machine \( M \), we have

\[
\overline{SRA} \text{ rejects } \langle M \rangle \iff M \text{ accepts } \langle M \rangle^R
\]

Finally, let \( \overline{SRA}^* \) be the Turing machine that reverses its input string and then passes control to \( \overline{SRA} \). For any Turing machine \( M \), we have

\[
\overline{SRA}^* \text{ rejects } \langle M \rangle^R \iff M \text{ accepts } \langle M \rangle^R
\]

In particular, if we set \( M = \overline{SRA}^* \), we have

\[
\overline{SRA}^* \text{ rejects } \langle \overline{SRA}^* \rangle^R \iff \overline{SRA}^* \text{ accepts } \langle \overline{SRA}^* \rangle^R
\]

But that’s impossible! Our original assumption must be incorrect; \( SRA \) does not exist.

**Rubric:** Standard diagonalization rubric.
Solution (reduction from Halt): For the sake of argument, suppose there is an algorithm `DecideSelfRevAccept` that correctly decides the language `SelfRevAccept`. Then we can solve the halting problem as follows:

\[
\text{DecideHalt}(\langle M, w \rangle):
\begin{align*}
\text{Encode the following Turing machine } M': \\
\quad M'(x): & \text{ run } M \text{ on input } w \\
& \text{return True} \\
\end{align*}
\]

We prove this reduction correct as follows:

\[\implies\] Suppose \( M \) halts on input \( w \).
   Then \( M' \) accepts every input string \( x \).
   In particular, \( M' \) accepts the string \( \langle M \rangle^R \).
   So `DecideSelfRevAccept` must accept the encoding \( \langle M' \rangle \).
   We conclude that `DecideHalt` correctly accepts the encoding \( \langle M, w \rangle \).

\[\iff\] Suppose \( M \) does not halt on input \( w \).
   Then \( M' \) diverges on every input string \( x \).
   In particular, \( M' \) does not accept the string \( \langle M \rangle^R \).
   So `DecideSelfRevAccept` must reject the encoding \( \langle M' \rangle \).
   We conclude that `DecideHalt` correctly rejects the encoding \( \langle M, w \rangle \).

In both cases, `DecideHalt` is correct. But that's impossible, because `Halt` is undecidable. We conclude that the algorithm `DecideSelfRevAccept` does not exist. ■

Rubric: Standard undecidability reduction rubric. This is not the only correct reduction.
2. Let $M$ be a Turing machine, let $w$ be an arbitrary input string, and let $s$ be an integer. We say that $M$ accepts $w$ in space $s$ if, given $w$ as input, $M$ accesses only the first $s$ (or fewer) cells on its tape and eventually accepts.

Prove that the following language is undecidable:

$$\text{SomeSquareSpace} = \{ \langle M \rangle \mid M \text{ accepts at least one string } w \text{ in space } |w|^2 \}$$

**Solution (reduction from Halt):** For the sake of argument, suppose there is an algorithm $\text{DecideLowSpace}$ that correctly decides the stated language. Then we can solve the halting problem as follows:

**$\text{DecideHalt}(\langle M, w \rangle)$:**

Encode the following Turing machine $M'$:

- $M'(x)$: run $M$ on input $w$
- return True

return $\text{DecideLowSpace}(\langle M' \rangle)$

(We seem to use this reduction a lot, don't we?) We prove this reduction correct as follows. Without loss of generality, assume that the input alphabet contains the symbol $1$.

$\implies$ Suppose $M$ halts on input $w$.

Let $s$ be the number of cells that $M$ accesses on its tape given input $w$.

Then $M'$ accepts every input string $x$ using space $s$.

In particular, $M'$ accepts the string $1^s$ using space $s \leq |1^s|^2$.

So $\text{DecideLowSpace}$ must accept the encoding $\langle M' \rangle$.

We conclude that $\text{DecideHalt}$ correctly accepts the encoding $\langle M, w \rangle$.

$\Longleftarrow$ Suppose $M$ does not halt on input $w$.

Then $M'$ diverges on every input string $x$.

In particular, $M'$ does not accept any string $w$ (in space $|w|^2$ or otherwise).

So $\text{DecideLowSpace}$ must reject the encoding $\langle M' \rangle$.

We conclude that $\text{DecideHalt}$ correctly rejects the encoding $\langle M, w \rangle$.

In both cases, $\text{DecideHalt}$ is correct. But that’s impossible, because $\text{Halt}$ is undecidable. We conclude that the algorithm $\text{DecideLowSpace}$ does not exist.

$\blacksquare$

**Rubric:** 5 points: standard undecidability reduction rubric (scaled). This is not the only correct solution. Notice that Rice’s Theorem cannot be used here.
3. Consider the following language:

\[
Picky = \left\{ \langle M \rangle \mid \text{ } M \text{ accepts at least one input string and } M \text{ rejects at least one input string} \right\}
\]

(a) Prove that Picky is undecidable.

**Solution (reduction from Halt):** We can reduce the standard halting problem to Picky as follows:

\[
\text{DecideHalt}(\langle M \rangle, w) :=
\]

Encode the following Turing machine \( M' \):

\[
M'(x):
\]

if \( x = w \)

run \( M \) on input \( w \)

return TRUE

else

return FALSE

return DecidePicky(\( \langle M' \rangle \))

We prove this reduction correct as follows:

\[\Rightarrow\] Suppose \( M \) halts on input \( w \).

Then \( M' \) accepts \( w \) but rejects every other input string.

So \( \langle M' \rangle \in \text{Picky} \).

So \( \text{DecidePicky} \) accepts \( \langle M' \rangle \).

We conclude that \( \text{DecideHalt} \) correctly accepts \( \langle M \rangle, w \).

\[\Leftarrow\] Suppose \( M \) does not halt on input \( w \).

Then \( M' \) diverges on \( w \) but rejects every other input string.

So \( \langle M' \rangle \notin \text{Picky} \).

So \( \text{DecidePicky} \) rejects \( \langle M' \rangle \).

We conclude that \( \text{DecideHalt} \) correctly rejects \( \langle M \rangle, w \).

In both cases, \( \text{DecideHalt} \) is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm \( \text{DecidePicky} \) does not exist. ■

**Rubric:** 5 points: standard undecidability rubric (scaled). These are not the only correct solutions. Notice that Rice’s Theorem cannot be used here.
(b) Sketch an algorithm that accepts Picky.

**Solution:** The following algorithm uses a universal Turing machine with a timer, to simulate the encoded Turing machine $M$.

```plaintext
AcceptPicky((M)):
    accepted ← False
    rejected ← False
    for $L ← 1$ to $∞$
        for all strings $w ∈ Σ^*$ with $|w| ≤ L$
            simulate $M$ on input $w$ for $L$ steps
            if $M(w)$ accepts before step $L$
                accepted ← True
            if $M(w)$ rejects before step $L$
                rejected ← True
            if accepted ∧ rejected
                return True
```

Each iteration of the outer loop executes in finite time, because there are only finitely many strings of length at most $L$ and we simulate $M$ on each of those input strings for a finite number of steps. Suppose $M$ accepts input string $w$ after $T$ steps, and rejects some string $w'$ after $T'$ steps. Then $\text{AcceptPicky}$ will halt and return True after at most $\max\{T, T'\}$ iterations of the outer loop.

---

**Rubric:** 5 points = 1 for using universal TM (or other TM simulation) + 1 for timer + 2 for dovetailing details + 1 for correctness argument. This is not the only correct solution.

---

**Standard rubrics for undecidability proofs.** For problems out of 10 points:

- **Diagonalization:**
  + 4 for correct wrapper Turing machine
  + 6 for self-contradiction proof (= 3 for $⇔$ + 3 for $⇒$)

- **Reduction:**
  + 4 for correct reduction
  + 3 for “if” proof
  + 3 for “only if” proof

- **Rice’s Theorem:**
  + 4 for positive Turing machine
  + 4 for negative Turing machine
  + 2 for other details (including using the correct variant of Rice’s Theorem)