Making Assertions

Like many programming languages, Python has an `assert` statement. Here’s how it works.

```
assert 1 + 1 == 2                                         #①
assert 1 + 1 == 3                                         #②
#Traceback (most recent call last):
# File "/usercode/__ed_file.py", line 2, in <module>
# assert 1 + 1 == 3 # \u2461
#AssertionError
```

```
assert 2 + 2 == 5, "Only for very large values of 2"   #③
#Traceback (most recent call last):
# File "/usercode/__ed_file.py", line 1, in <module>
#  assert 2 + 2 == 5, "Only for very large values of 2" #\u2462
#AssertionError: Only for very large values of 2
```

① The `assert` statement is followed by any valid Python expression. In this case, the expression `1 + 1 == 2` evaluates to `True`, so the `assert` statement does nothing.

② However, if the Python expression evaluates to `False`, the `assert` statement will raise an `AssertionError`.

③ You can also include a human-readable message that is printed if the `AssertionError` is raised.

Therefore, this line of code:

```
assert len(unique_characters) <= 10, 'Too many letters'
```

…is equivalent to this:

```
if len(unique_characters) > 10:
    raise AssertionError('Too many letters')
```

The alphametics solver uses this exact assert statement to bail out early if the puzzle contains more than ten unique letters. Since each letter is assigned a unique digit, and there are only ten digits, a puzzle with more than ten unique letters can not possibly have a solution.