The current strategy of using the uploaded file's name and filesize to compare two files for similarity is brittle and easy to circumvent.

So, let's use some of the secure hashing technologies that come with Python to fix this.

Start by importing the `hashlib` library module (which comes built-in as standard):

In [1]:
```
import hashlib
```

Now, let's define some text to work with.

Note how the first three strings are all the same size, but different in content (the first two, text0 and text1, are only different from each other by a single byte: "string"->"strong"). The last string, text3, is the same as the first (text0):

In [2]:
```
text0 = 'This is a test to see what happens when I convert a string to an SHA digest.'
text1 = 'This is a test to see what happens when I convert a strong to an SHA digest.'
text2 = 'Unlike text0, this string is *very* different from the one that come before.'
text3 = 'This is a test to see what happens when I convert a string to an SHA digest.'
```

Let's now create SHA digests of each of these strings:

In [3]:
```
text0_digest = sha_hash = hashlib.sha256(bytes(text0, encoding='utf-8')).hexdigest()
text1_digest = sha_hash = hashlib.sha256(bytes(text1, encoding='utf-8')).hexdigest()
text2_digest = sha_hash = hashlib.sha256(bytes(text2, encoding='utf-8')).hexdigest()
text3_digest = sha_hash = hashlib.sha256(bytes(text3, encoding='utf-8')).hexdigest()
```

Display each of the digests - no matter the length of the strings, the SHA technology "shrinks" the original text into a string of 64 bytes of HEX:

In [4]:
```
print(text0_digest)
print(text1_digest)
print(text2_digest)
print(text3_digest)
```

```
4f2431cee611eae619a55eaaa83bfea88390bb7dad69c4cd4e24df254521cd09
4a1422ef961f91143f401931993d27da35e69dbef18ac9c668b9e54b87d3a54f
2d9e32c9fdf361e57e0e1545f304a2ad33f7423052df87a97711c51ac1a6ea46
4f2431cee611eae619a55eaaa83bfea88390bb7dad69c4cd4e24df254521cd09
```

Now we can compare the first digest to the other three to see if they are the same (note: we don't use == to do this, we use `hmac.compare_digest` as recommended by the Python docs):

In [5]:
```
import hmac

if hmac.compare_digest(text0_digest, text1_digest):
    print('text0 and text1 match.')
if hmac.compare_digest(text0_digest, text2_digest):
    print('text0 and text2 match.')
if hmac.compare_digest(text0_digest, text3_digest):
    print('text0 and text3 match.')
```

```
text0 and text3 match.
```

The SHA digests of text0 and text3 match, so they were created from the same string. Note that we do not need to store the original strings once we have the SHA digest. Even if the original string is 4GB of text, this code always "shrinks" it to a SHA digest which is 64 bytes in length.

Note that it is not possible to work out what the original string was from the SHA digest - this process only works in one direction, from text->SHA digest.