The Rho Method
Project link: https://samsclass.info/141/proj/C106a.htm

Finding Hash Collisions with the Rho Method
Imagine you keep folding a piece of paper over and over again. Eventually, it will look the same as before because there's only so much space to fold into.

The **Rho Method** works the same way, but with numbers! You **hash** something (turn it into a special scrambled code), then hash it again, and again… Eventually, the same code will show up again. That's called a **collision**.

Step 1: The following python code will generate MD5 Hash

```
import hashlib
# Hashing the letter "a"
print(hashlib.new('md5', "a".encode('ascii')).hexdigest())
```

Step 2: Make It Simpler (Truncate the Hash)

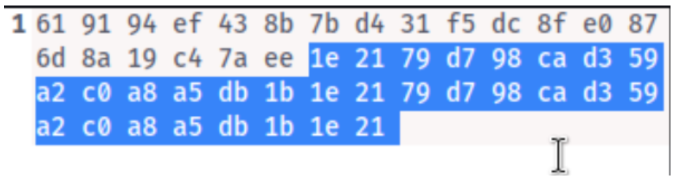
The hash is super long, but let's chop off most of it and keep only the **last two characters**.

```
import hashlib
# Only keeping the last 2 characters
print(hashlib.new('md5', "a".encode('ascii')).hexdigest()[-2:])
```

Output = 61

Step 3: Iterate Hashing (Keep repeat and finally you'll see the collision)

```
import hashlib
h = "a"  # Starting word
for i in range(50):  # Hash 50 times

   h = hashlib.new('md5', h.encode('ascii')).hexdigest()[-2:]  # Keep last 2 characters
   print(h, end=' ')
```

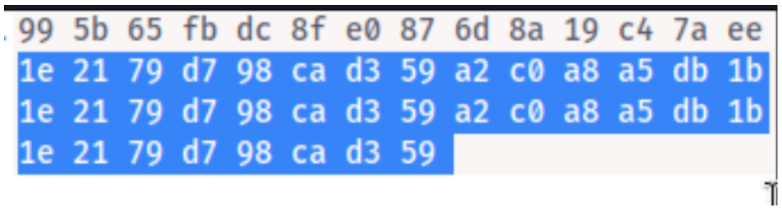


```
61 91 94 ef 43 8b 7b d4 31 f5 dc 8f e0 87
6d 8a 19 c4 7a ee 1e 21 79 d7 98 ca d3 59
a2 c0 a8 a5 db 1b 1e 21 79 d7 98 ca d3 59
a2 c0 a8 a5 db 1b 1e 21
```

C106.1: **1e**

Code used:
```
import hashlib
h = "password"
```

```
for i in range(50):
    h = hashlib.new('md5', h.encode('ascii')).hexdigest()[-2:]
    print(h, end=' ')
```

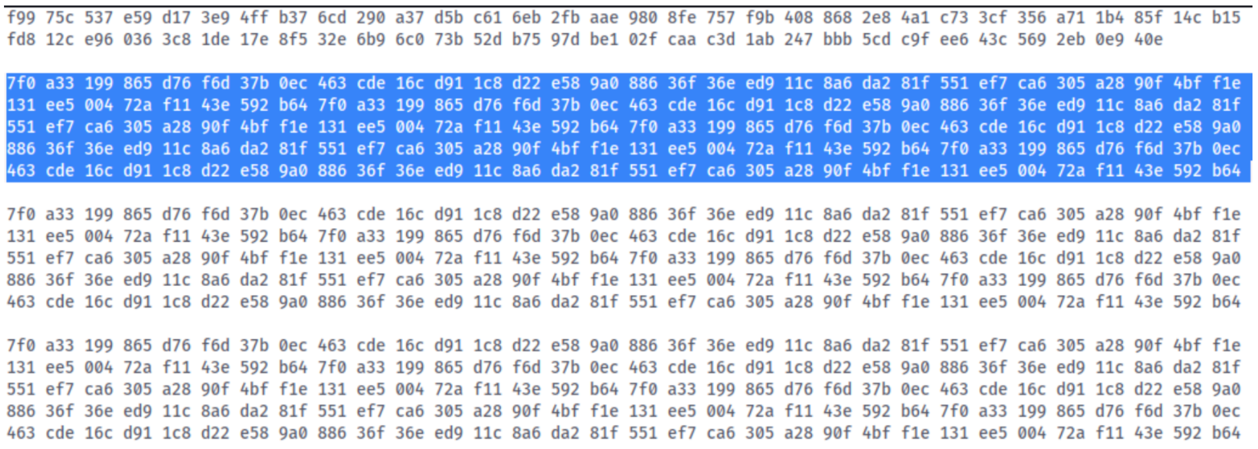


```
99 5b 65 fb dc 8f e0 87 6d 8a 19 c4 7a ee
1e 21 79 d7 98 ca d3 59 a2 c0 a8 a5 db 1b
1e 21 79 d7 98 ca d3 59 a2 c0 a8 a5 db 1b
1e 21 79 d7 98 ca d3 59
```

C106.2: **7f0**

Code used:

```
import hashlib
h = "password"

for i in range(1000):
   h = hashlib.new('md5', h.encode('ascii')).hexdigest()[-3:]
   print(h, end=' ')
```



```
f99 75c 537 e59 d17 3e9 4ff b37 6cd 290 a37 d5b c61 6eb 2fb aae 980 8fe 757 f9b 408 868 2e8 4a1 c73 3cf 356 a71 1b4 85f 14c b15
fd8 12c e96 036 3c8 1de 17e 8f5 32e 6b9 6c0 73b 52d b75 97d be1 02f caa c3d 1ab 247 bbb 5cd c9f ee6 43c 569 2eb 0e9 40e

7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e
131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f
551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0
886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec
463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64

7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e
131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f
551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0
886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec
463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64

7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e
131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f
551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec 463 cde 16c d91 1c8 d22 e58 9a0
886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64 7f0 a33 199 865 d76 f6d 37b 0ec
463 cde 16c d91 1c8 d22 e58 9a0 886 36f 36e ed9 11c 8a6 da2 81f 551 ef7 ca6 305 a28 90f 4bf f1e 131 ee5 004 72a f11 43e 592 b64
```

C106.3: **5c124e14**

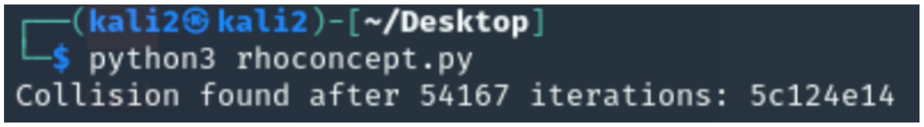
This challenge had to be automated and cannot be easily spotted.

Code used:

```
import hashlib

# Starting value
h = "password"

# Set to keep track of seen hashes
seen = set()

# Loop to find the first collision
```

```
for i in range(1000000):  # A large number to ensure we find a collision
    h = hashlib.new('md5', h.encode('ascii')).hexdigest()[-8:]  # Keep the last 8 characters
    if h in seen:
        print(f"Collision found after {i} iterations: {h}")  # First collision (flag)
        break
    seen.add(h)
```

Result:



```
┌──(kali2㉿kali2)-[~/Desktop]
└─$ python3 rhoconcept.py
Collision found after 54167 iterations: 5c124e14
```

Summary:

- **Found collisions** like a cryptographer. 🕵️
- This is how hackers test the strength of password systems. 🔐
- Practiced **Python** and **hashing**, which are super valuable coding skills! 🚀