In [2]:
```
#Data Science TP5
#Tientso Ning
import numpy as np
import math
import random
import numpy as np
```

Quantifiers of Information

a.

P(u=0) = 0.625 P(u=1) = 0.375 P(v=0) = 0.375 P(v=1) = 0.625 P(w=0) = 0.5 P(w=1) = 0.5

H(U) = 0.625*log(0.625) + 0.375*log(0.375) = **0.9544340029249649**

H(V) = **0.9544340029249649**

H(W) = **0.5*log(0.5) + 0.5*log(0.5) = 1.0**

b.

P(u=0,v=0) = 0.25 P(u=0,v=1) = 0.375 P(u=1,v=0) = 0.125 P(u=1,v=1) = 0.25

P(v=0,u=0) = 0.25 P(v=0,u=1) = 0.125 P(v=1,u=0) = 0.375 P(v=1,u=1) = 0.25

P(w=0,u=0) = 0.5 P(w=0,u=1) = 0.0 P(w=1,u=0) = 0.125 P(w=1,u=1) = 0.375

p(u=0,w=0) = 0.5 p(u=0,w=1) = 0.125 p(u=1,w=0) = 0.0 p(u=1,w=1) = 0.375

p(u=0|v=0,w=0) = 0.25 p(u=0|v=0,w=1) = 0.0 p(u=0|v=1,w=0) = 0.25 p(u=0|v=1,w=1) = 0.125 p(u=1|v=0,w=0) = 0.0 p(u=1|v=0,w=1) = 0.125 p(u=1|v=1,w=0) = 0.0 p(u=1|v=1,w=1) = 0.25

H(U|V) = $\sum \sum$ p(u,v)*log(p(u,v)) = **1.9056390622295662**

H(V|U) = **1.9056390622295662**

H(W|U) = **1.4056390622295662**

c.

I(U;V) = H(U) - H(U|V) = 0.9544340029249649-1.9056390622295662 = **0.9512050593**

I(U;W) = H(W) - H(W|U) = 1.0-1.4056390622295662 = **0.40563906223**

I(U;V,W) = I(U;W) + I(U;V|W) = I(U;W) + H(U|W) - H(U|V,W) = 0.40563906223 + 1.4056390622295662 - 2.25 = **0.43872187554**

d.

H(U,V,W) = **0.0**

Communication Through Noisy Channels

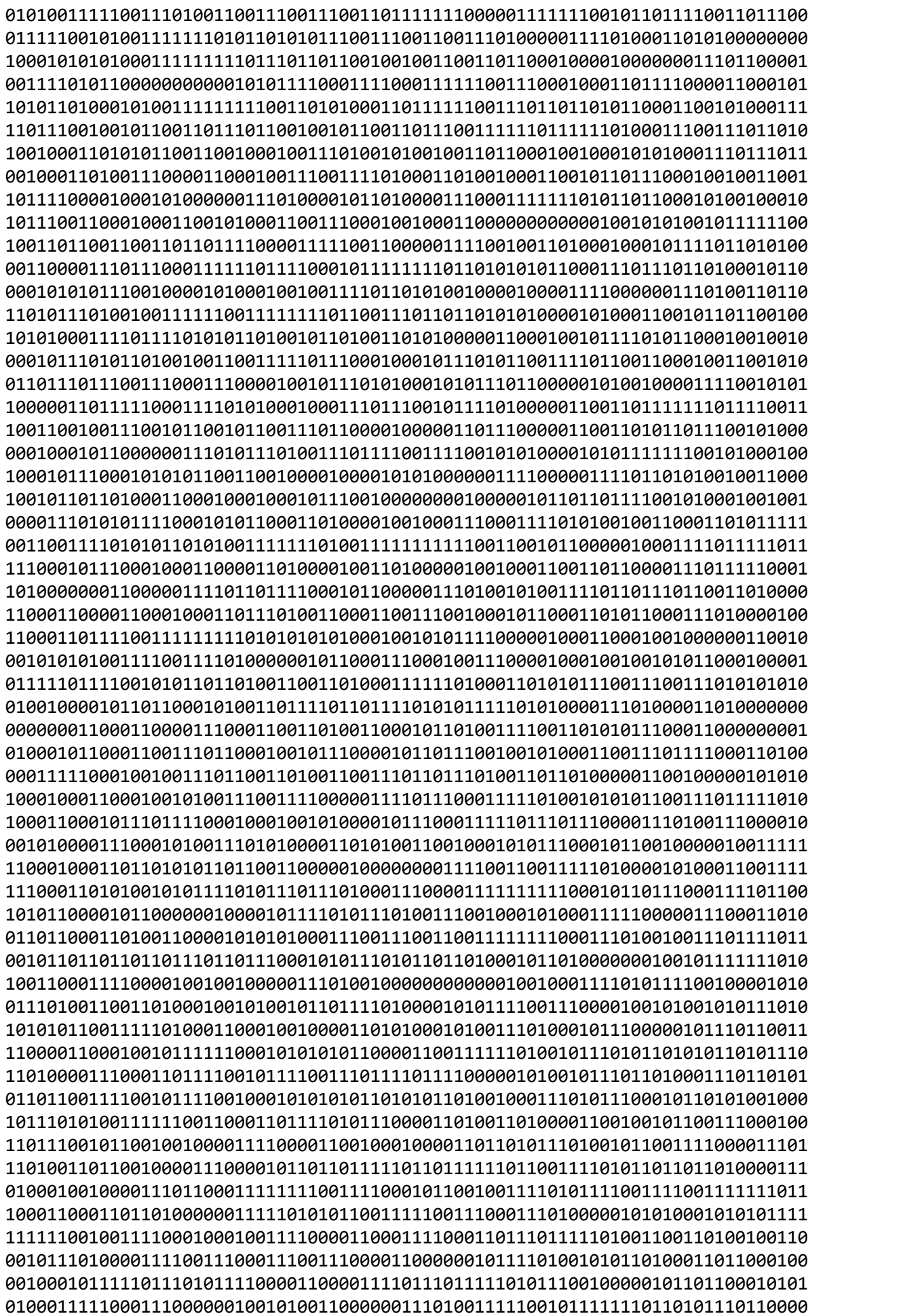
In [45]:
```
def BinarySymmetricChanel(bitstring):

    result = ""

    for ch in bitstring:
        if random.random() < 0.1: #with prob 0.1
            if ch == "0":
                result = result + "1" #flip bit
            else:
                result = result + "0" #flip bit
        else:
            result = result + ch

    return result
```

In [47]:
```
a = "01010101010111100001001010100 1"
b = BinarySymmetricChanel(a)
print(b)
```

```
0101011101011110000110011000 00
```

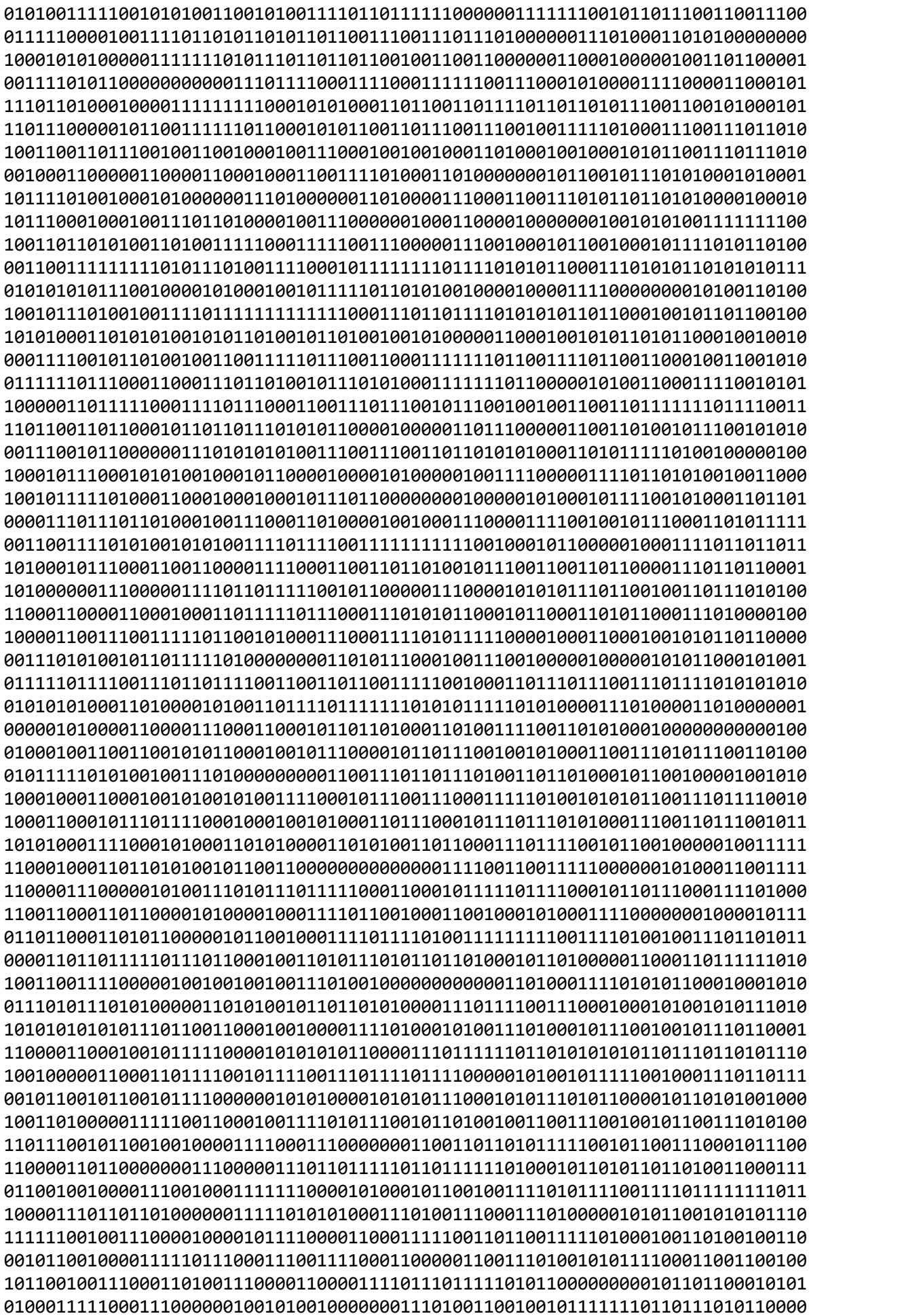
In [56]:

```
a = ""
for i in range(5000):
    a = a + str(random.randint(0,1))
print(a)
```



101101010100011000010101100010001010111100110000010100101010101001111010110000
101100000001000001101100000111000000101001011110011000111000110100110001100111
111110000100011010010010100010000000100101000000011111111010000110001111010000
101110001000101110010010110011110001100101111011101101001100110111111000010001
010001000111100011000000001010011001001111110111100110100100001101000010010111
110011101010100100010010010000111100111100011011010110101111110010111000110000
010010011100000111111001110000101111000110100110110000001111000010100011011100
111110100001000000010100000000100001010000010010101001110110010100011101

In [57]:
```
b = BinarySymmetricChanel(a)
print(b)
```



```
11110101001001100001010110101100101011010011000001110010001000110111101011100
10110010000100000110110000011100100110010101111011100011100011010011000110011
11111000010001111001001010001000000110010100000111110111101010011000111111100
10111000100011111001001010001111000111010111101110111011100010011110010010001
01000100011111001100000100101001100110111101011110010010010000111100001010011
11000100001110011001001101000010100011110001101111010010101111001001100011000
01001001110000111110100011000010111100010000011011000000101100001010001101110
11111010000110100001010000000010000101100001001010100111011001010100111111
```

In [76]:
```
#compare a and b
a_list = []
b_list = []

for ch in a:
    a_list.append(int(ch))
for ch in b:
    b_list.append(int(ch))

erroneous = 0
matches = np.logical_xor(a_list, b_list)
for i in matches:
    if i:
        erroneous += 1
print(erroneous/5000) #should be around 10%
```

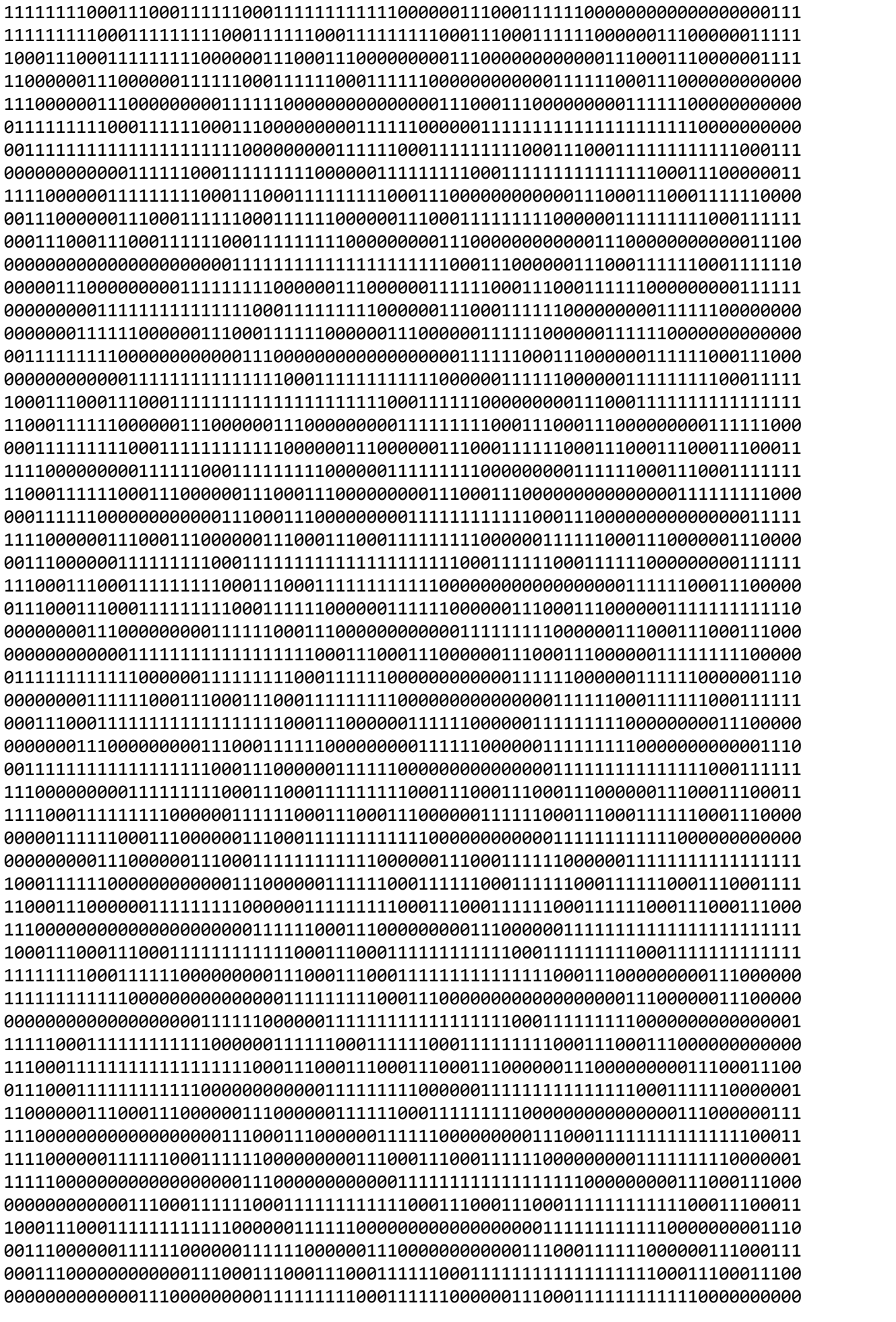
```
0.0994
```

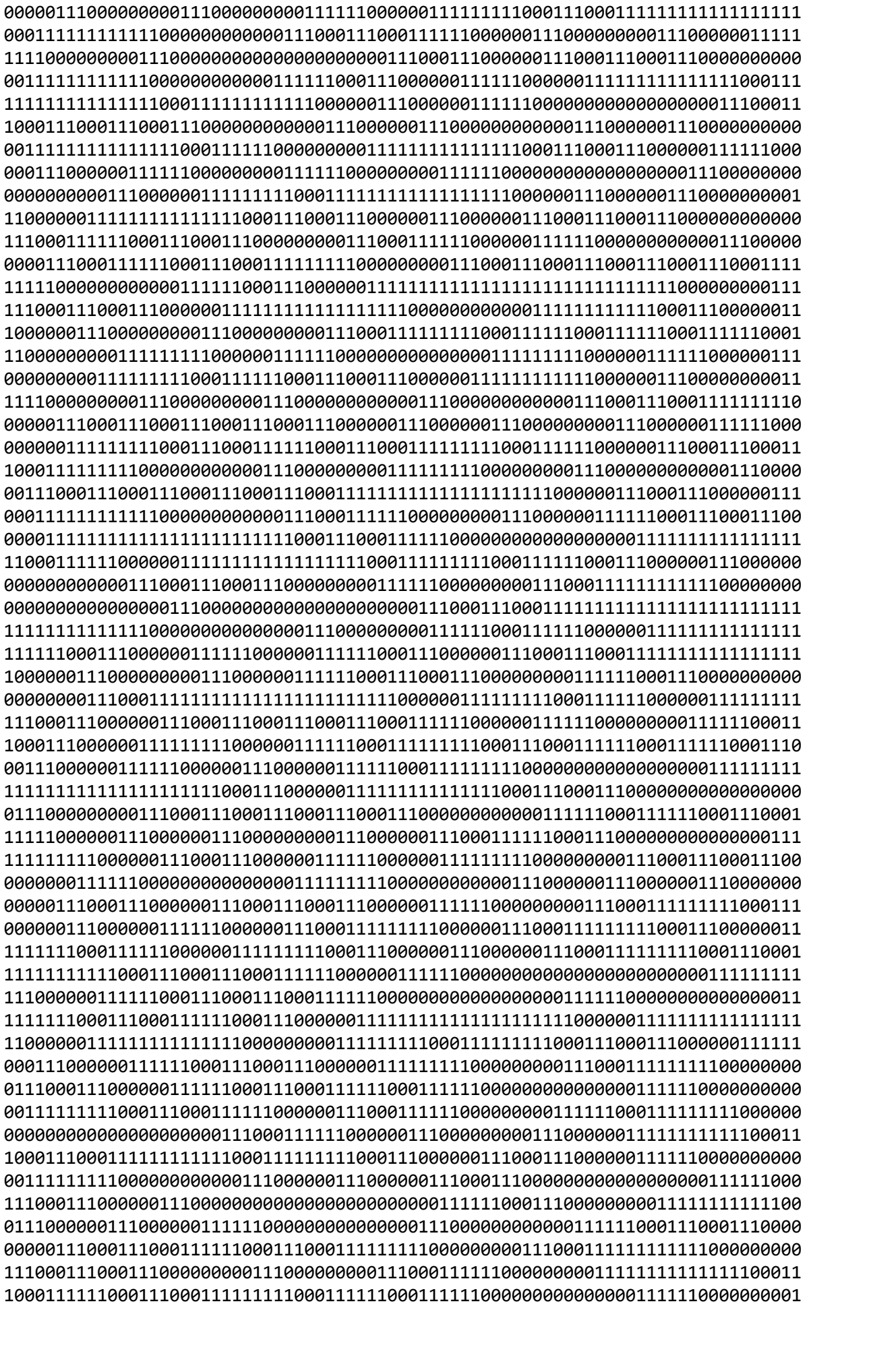
In [102]:
```
def calc_erroneous(a,b):
    a_list = []
    b_list = []

    for ch in a:
        a_list.append(int(ch))
    for ch in b:
        b_list.append(int(ch))

    erroneous = 0
    matches = np.logical_xor(a_list, b_list)
    for i in matches:
        if i:
            erroneous +=1
    return erroneous/5000
```

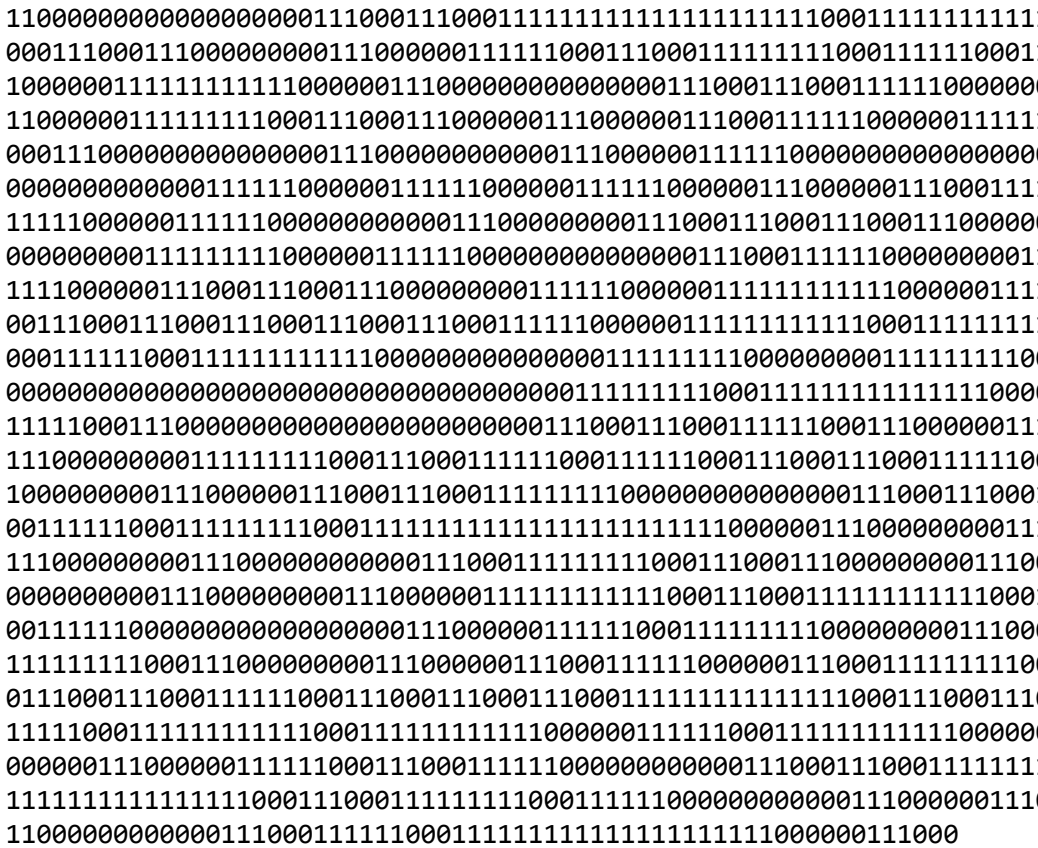
In [156]:

```
a = ""
for i in range(1000):

    #create a word
    b = ""
    for j in range(5):
        b = b + str(random.randint(0,1))

    #for each word, repeat each character
    c = ""
    for x in range(len(b)):
        c = c + b[x]+b[x]+b[x]

    a = a + c
print(len(a))
print(a)
```

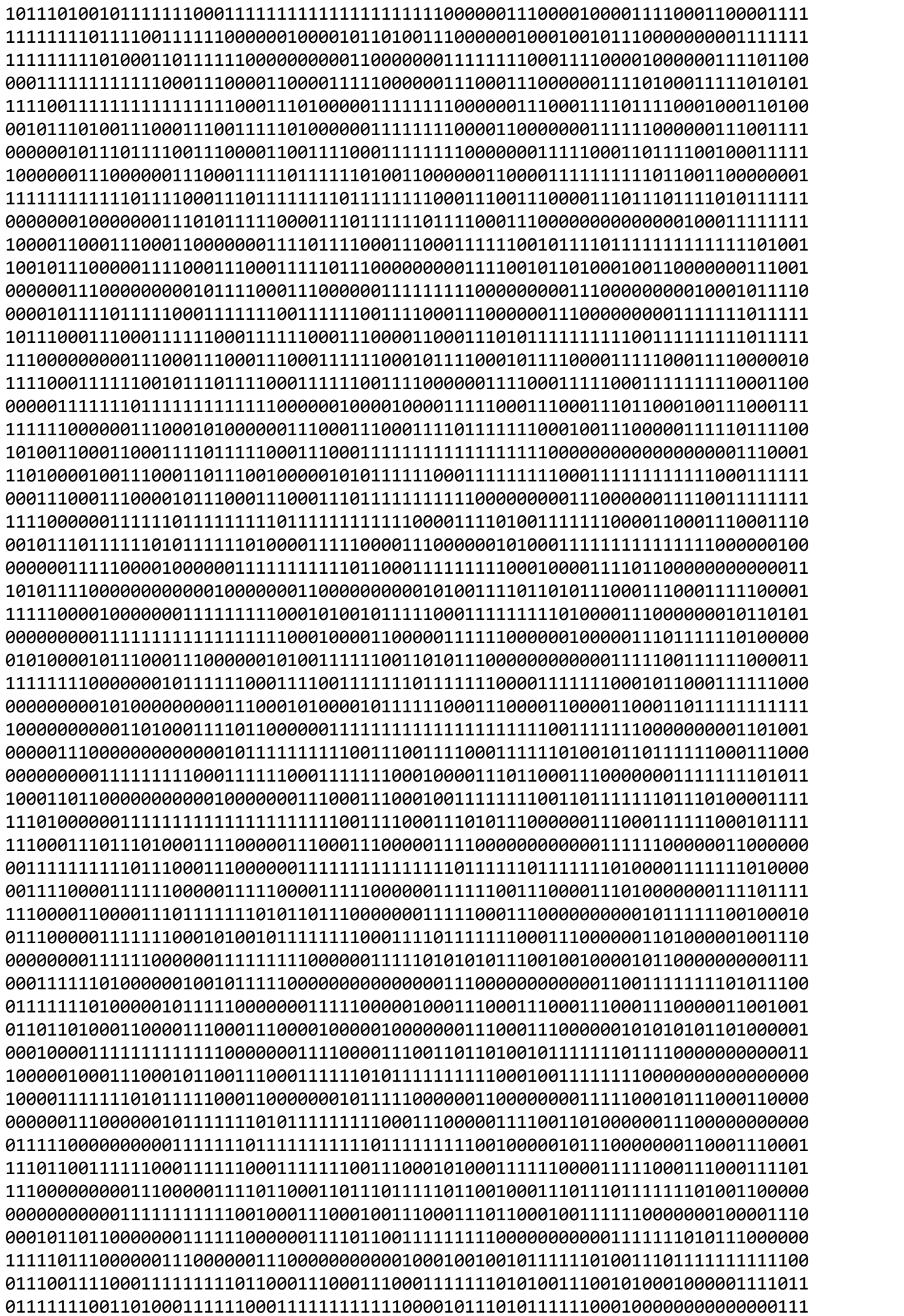
15000
111111000111111111000111111111111111111111000000111000111000111000111100011111
111111000111000111111000000000111111000111000000000000000111000000000111111
111111110000001111111110000000001110000001111111110001110000000000000111111000
000111111111111000111000111000111111000000111000111000000111111100011100011100
011100000011111111111000111000000111111111000000111000111111111000000111000000
000001110001110001110001111110000001111111110000000000000111111000000111000111
000000111111111000111000001111110001111111110000001111111111111100000011111111
100000011100000011100011111111111110001110000001110001111111111110001110000001
111111111111111110001111111111111111111111000111000000000111111111111000111111
000000000000000111000111111000111111111111111000111000000000000000000011111111
100011100011100011100000111111111000111000111111000111111111111111111110001111
110001110000001110001110001111111110000000001110001111110000001110000000111000
000000111000000000111111000111000000111111111000000000111000000000000000011100
000011111111111100011111100011111111111100011100000011100000000011111111111111
111110001110001111110001111110001110001110001110001111111110001111111111111111
111000000000111000111000111000111111000111110001111110001111100011100000011111
111100011111100000011111100011111100011100000111000111111100011111111110001110
000001111111111111111111111000000000000000111111000111000111111000000111000111
111111000000111000111000000111000111000111111111111000000111000111111111111100
000000000011100011111111100011100011111111111111111100000000000000000001110001
110000000001110001111110000000000011111100011111111100011111111111000111111111
000111000111000000111000111000111111111111111100000000011100000011100011111111
111100000011111111111111111111111111111100011111100011111100000000011100111110
001111111111110001111110000001111110001110000001110001111111111111111000000000
000000111000000000001111111111100000011111111000001111111100000011111111100000
011100000011100011100000011111111111111100011100000000000011111111111110000001
111111100000000011111000111000111111111111100011111111000111000111111000111111
111000111111100011111100011111100011100011100011111100011111100011100011111111
111000000000111000000111111000111111111111111000000111111111111111000111000000
000000000000011111111100000011100000011100011111100011111111100000000000001110
000001111110000001111110000001111111111111111100000000000011111100011100000000
111111111000000111000000111000000000000000000001111111110001111111111111100000
011100011100011111111111000111000111000111111111000111000111000000000111111111

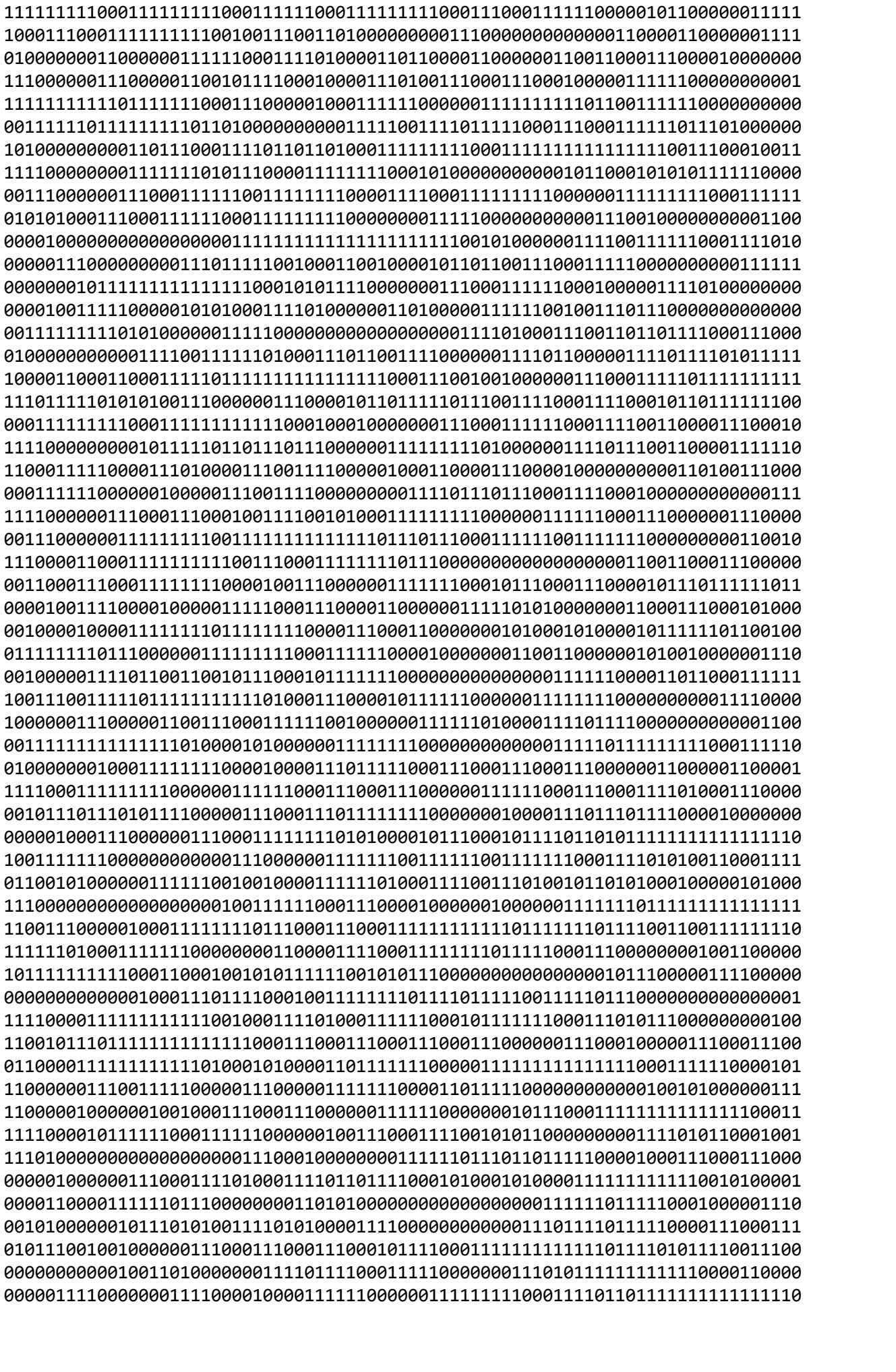


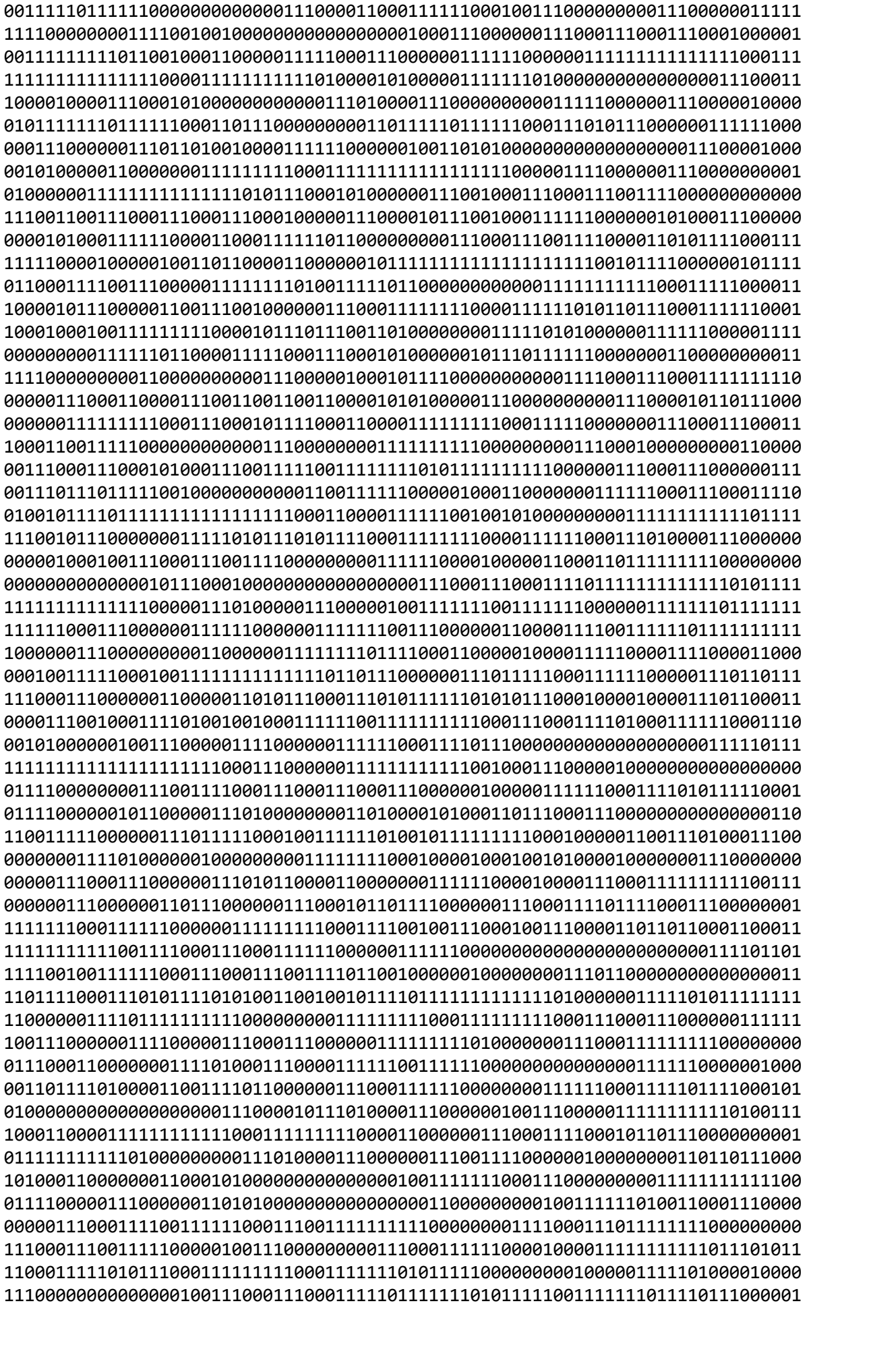




In [157]:
```
b = BinarySymmetricChanel(a)
print(b)
```







```
00011100011100000000011100000011111100011100001000111100011111100010010011110
10001001111111111110000001110000000110000000110000110000111010100010100001111
10000000111111110100111000111000100101100000111010111111000000010111111011000
00011100100000000000010100100100010011100010010111100000000000100000101111101
00000000000001110110001001111100000001101110001001110100011110001101110001111
11011000000111111000000000000111100000000011100010100011101001100100010001001O
00010000001111111110000001101110100000000000000111000010111100000100111111111
01010000000111010111000111000001000111101010100101111111111000000101111110000
00110000111000111000111000101000111111000000011111110111000111111101111011000
00011011100011111101111000000001000000011111111010001000111111111000011111100
00000000100101000000100000000010000001111110110001111111101111111000000101111
11111100111001010000000000001000000111000111000011101000111010000110000111011
11110000000101111111010001110001011110101111110000110011110001111110001111111
10000000001111000000111000101000011111110000000000000000110010101000101111110
00111101000110011111000111111111110110111101110000001100100100001101111111010
11100000010001110000000000000110000001111111000110000011000000000110000010100000
00000000011100001000001110000000111111111110001110001111111111110001111010000
01111111100000000000000000001110000011111010001111110110000000000111000111111000
11111111100011101100110001100000011100011111100000001110001100100010001110000O
01110001110101101110001110001100001110001111111111111110011100011011011111001
11111000101111011111000111111111110000001111100001111110011110000000110001111
10000011100000011111000011100011111100010000000011100011100011111111111111101
11011111111111110001110001011111110011111110000100000000111000011101010110111
10000000000010111100011111000111111111111111111111100010011100O
```

```python
def decode(bitstring, step):
    decoded = ""
    for i in range(0, len(bitstring)-2, step):
        if sum([int(x) for x in bitstring[i:i+step]]) > (step//2): #bit is a 1
            decoded = decoded + "1"
        else:
            decoded = decoded + "0"
    return decoded
```

```python
c = decode(a,3)
d = decode(b,3)
print(calc_erroneous(c,d))
```

```
0.0296
```

In [139]:
```
a = ""
for i in range(1000):

    #create a word
    b = ""
    for j in range(5):
        b = b + str(random.randint(0,1))

    #for each word, repeat each character
    c = ""
    for x in range(len(b)):
        c = c + b[x]*5

    a = a + c

b = BinarySymmetricChanel(a)
c = decode(a,5)
d = decode(b,5)
print(calc_erroneous(c,d))
```

```
0.0086
```

In [146]:
```
a = ""
for i in range(1000):

    #create a word
    b = ""
    for j in range(5):
        b = b + str(random.randint(0,1))

    #for each word, repeat each character
    c = ""
    for x in range(len(b)):
        c = c + b[x]*11

    a = a + c

b = BinarySymmetricChanel(a)
c = decode(a,11)
d = decode(b,11)
print(calc_erroneous(c,d))
```

```
0.0002
```

In [148]:
```
a = ""
for i in range(1000):

    #create a word
    b = ""
    for j in range(5):
        b = b + str(random.randint(0,1))

    #for each word, repeat each character
    c = ""
    for x in range(len(b)):
        c = c + b[x]*23

    a = a + c

b = BinarySymmetricChanel(a)
c = decode(a,23)
d = decode(b,23)
print(calc_erroneous(c,d))
```

```
0.0
```

In [161]:
```
def com_rate(size, n):
    return math.log(size,2)/n
```

In [163]:
```
print(com_rate(3,15000))
print(com_rate(5,25000))
print(com_rate(11,55000))
print(com_rate(23,115000))
```

```
0.00010566416671474376
9.287712379549448e-05
6.289875670249632e-05
3.93353213570175e-05
```

As we can see, increasing the number of repetitions decreases the rate of communication (speed reduces). But as we can also see from above, the probability of errors goes down as well.