In [11]:
```
#Data Science TP5
#Tientso Ning
import numpy as np
import math
import random
import numpy as np
```

Quantifiers of Information

In [23]:
```
a = [0.25, 0.375, 0.125, 0.25]
b = [0.66, 0.4, 0.33, 0.6]
total = 0.0
for i in range(len(a)):
    try:
        total += a[i] * math.log(b[i],2)
    except:
        total += 0
print(total*-1)
```

```
1.0297627105304141
```

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(u=0|v=0) = 0.25/0.375 = 0.66 P(u=0|v=1) = 0.375/0.625 = 0.6 P(u=1|v=0) = 0.125/0.375 = 0.33 P(u=1|v=1) = 0.25/0.625 = 0.4

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(v=0|u=0) = 0.25/0.625 = 0.4 P(v=0|u=1) = 0.125/0.375 = 0.33 P(v=1|u=0) = 0.375/0.625 = 0.6 P(v=1|u=1) = 0.25/0.375 = 0.66

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(w=0|u=0) = 0.5/0.625 = 0.8 P(w=0|u=1) = 0.0 P(w=1|u=0) = 0.125/0.625 = 0.2 P(w=1|u=1) = 0.375/0.375 = 1.0

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)) = **1.0297627105304141**

H(V|U) = **1.0297627105304141**

H(W|U) = **0.4512050593046014**

c.

I(U;V) = H(U) - H(U|V) = 0.9544340029249649-1.0297627105304141 = **0.0753287076**

I(U;W) = H(W) - H(W|U) = 1.0-0.4512050593046014 = **0.54879494069**

I(U;V,W) = I(U;W) + H(U|W) - H(U|V,W) = 0.54879494069 + 0.4512050593046014 - 1.0297627105304141 =**0.02976271053**

d.

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

Communication Through Noisy Channels

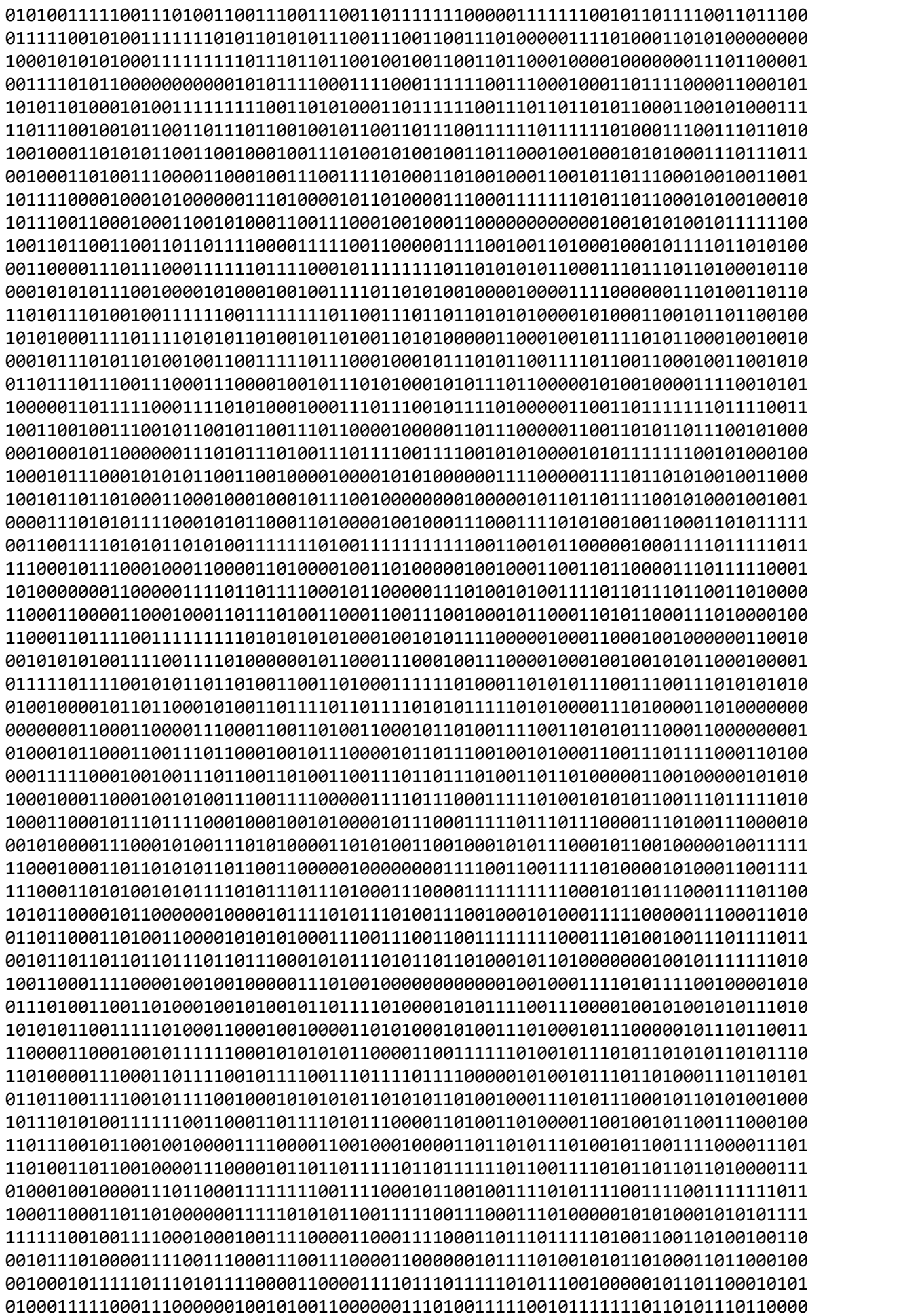
In [45]:
```
def BinarySymmetricChanel(bitstring):

    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 = "0101010101011110000100101010001"
b = BinarySymmetricChanel(a)
print(b)
```

```
0101011101011110000110011000000
```

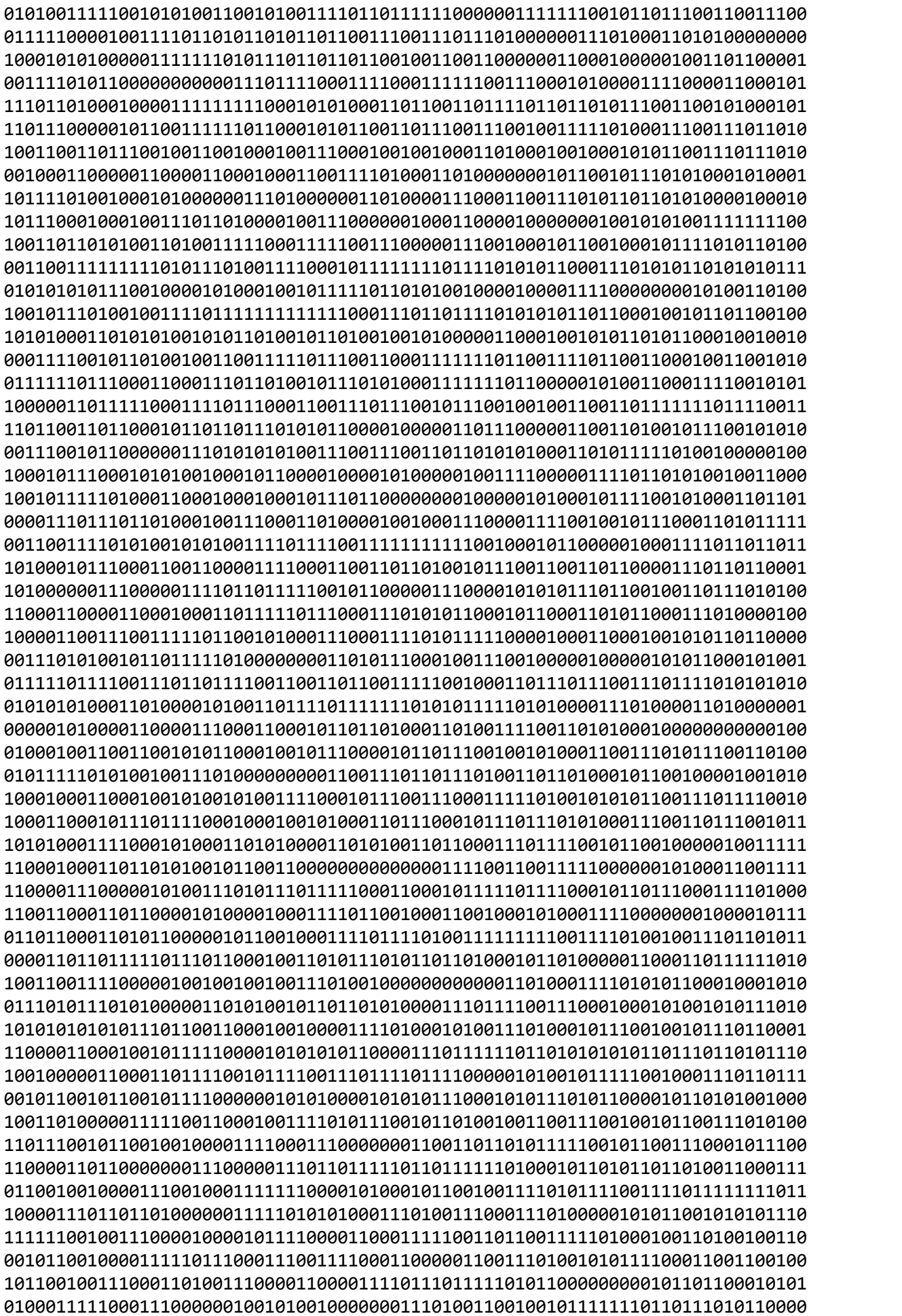
In [56]:

```
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%
```

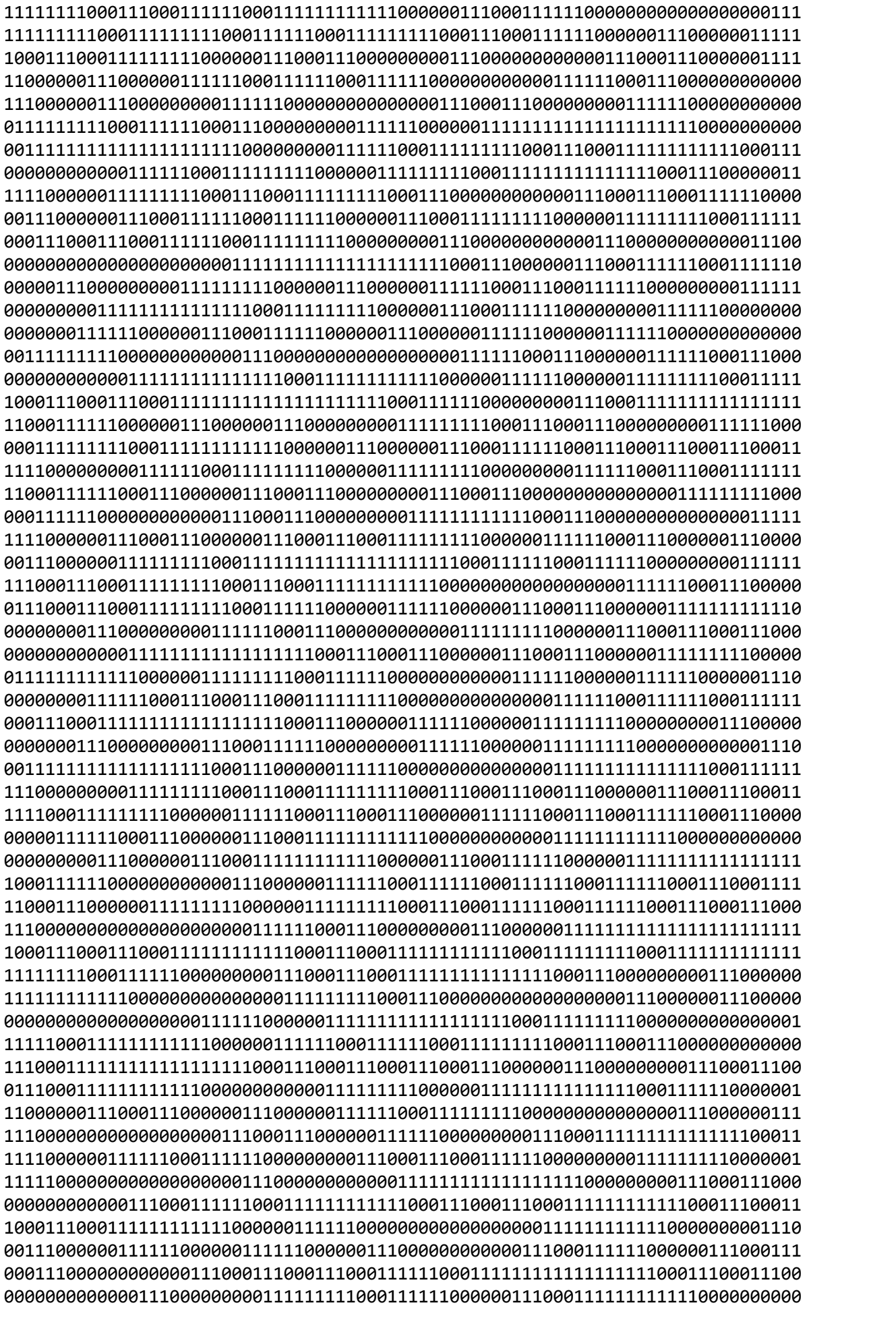
```
0.0994
```

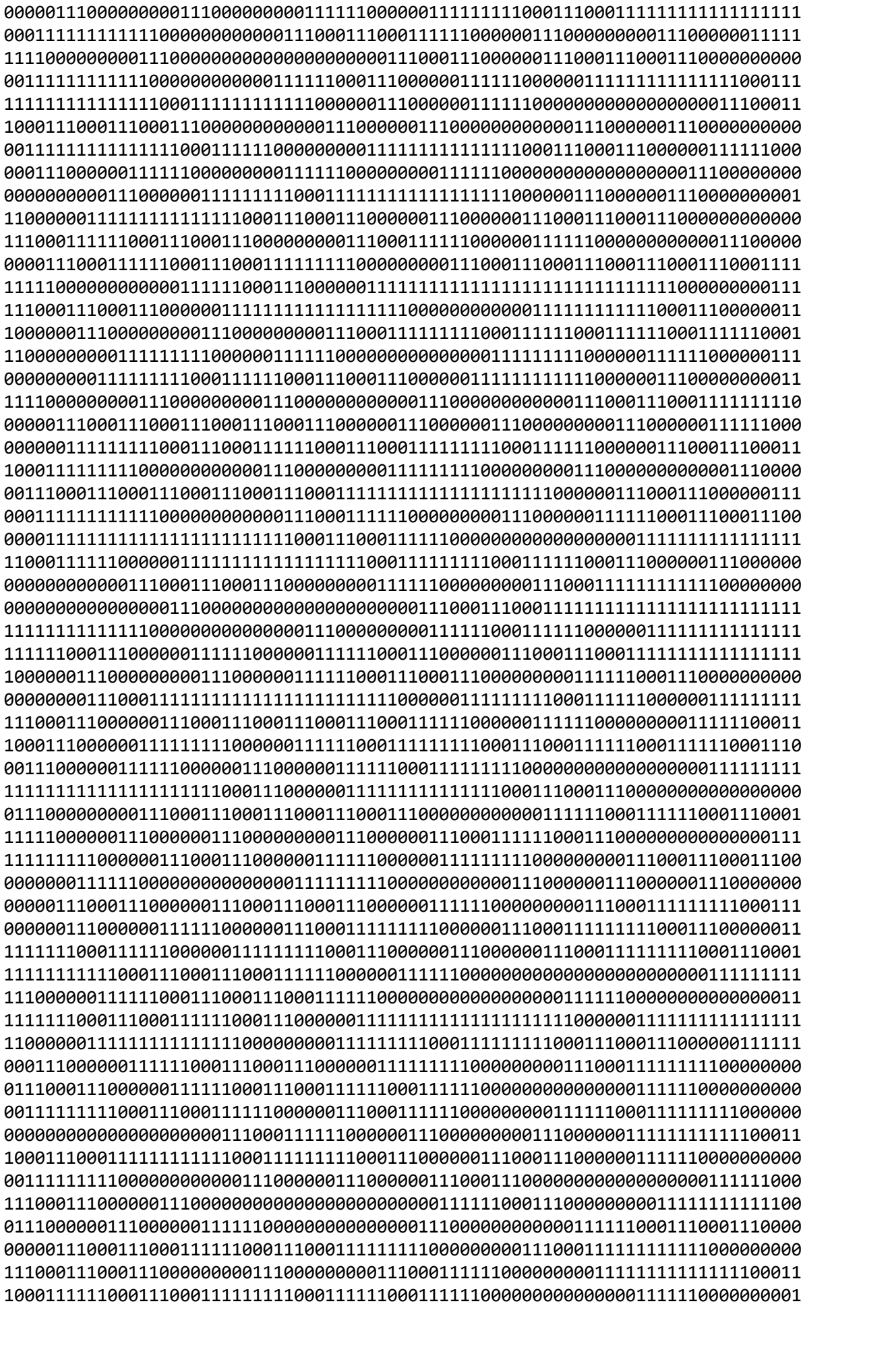
In [102]:
```
def calc_erroneous(a,b):
    a_list = []
    b_list = []

    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
```

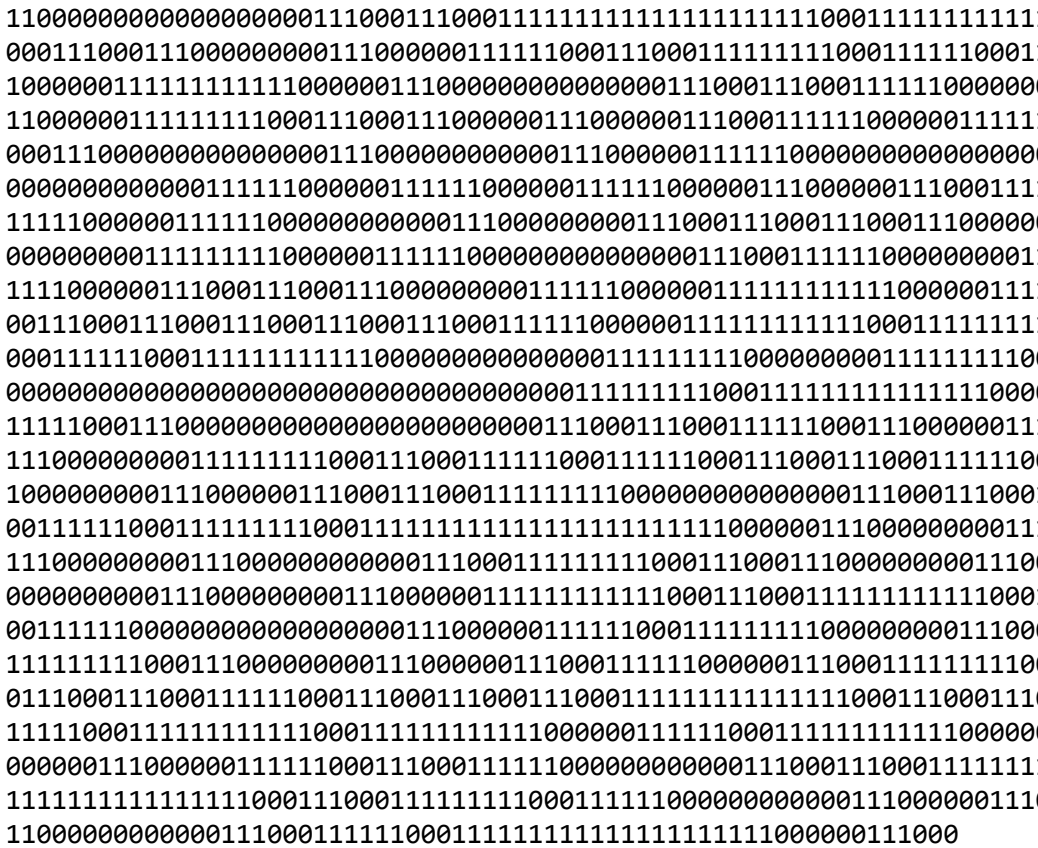
In [156]:

```
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)
```

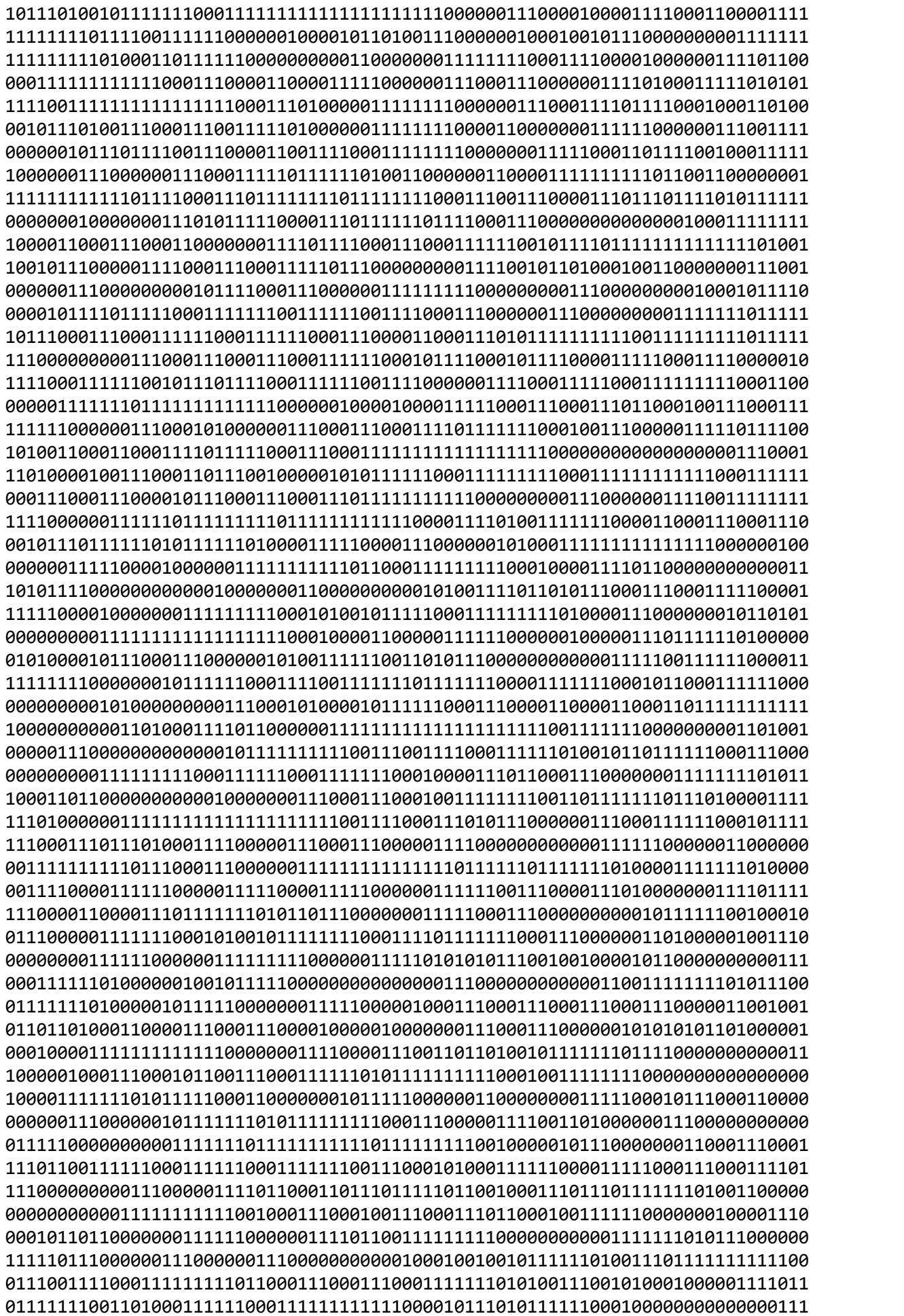
```
15000
1111110001111111110001111111111111111111110000001110001110001110001110001111
```

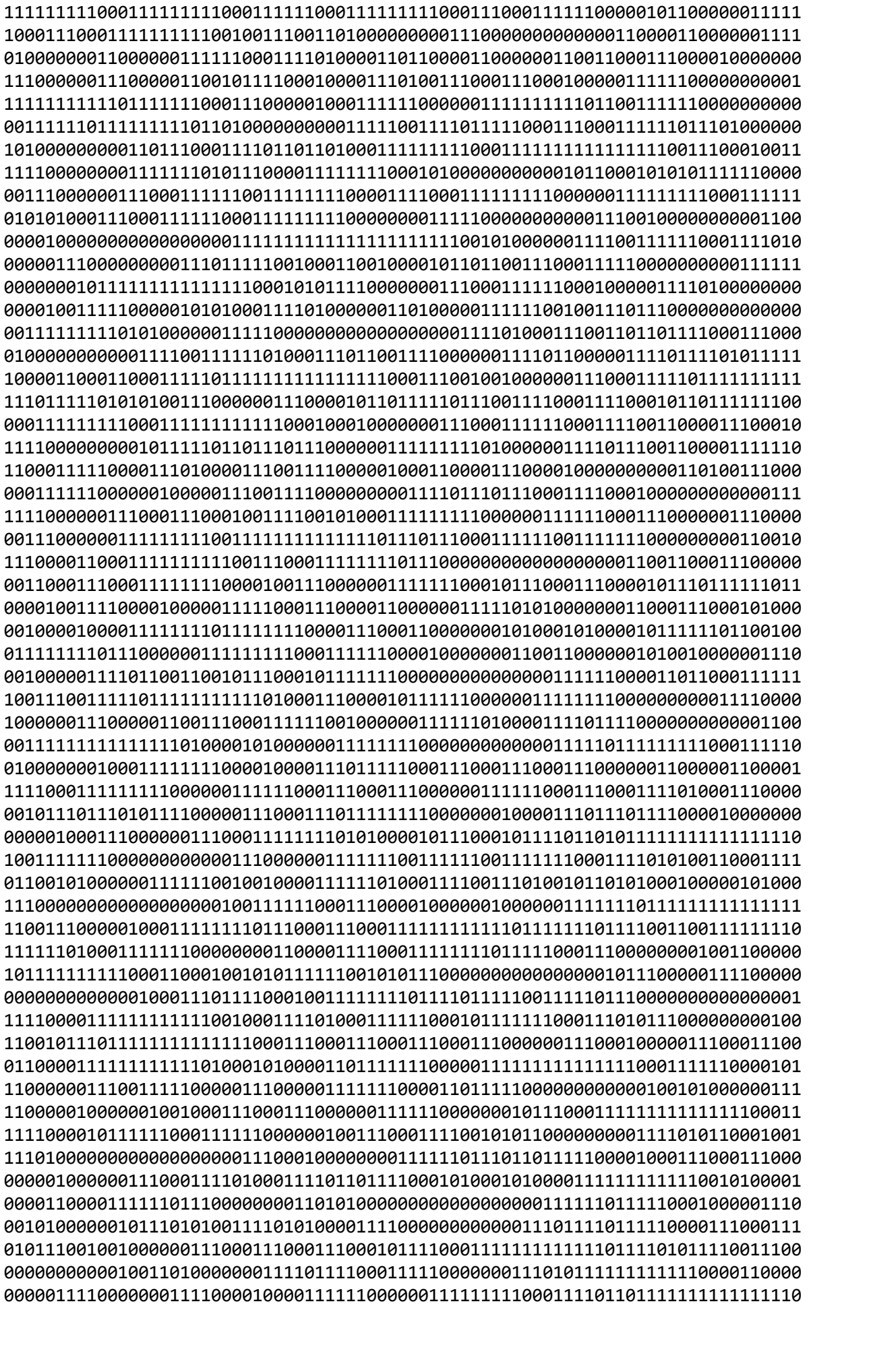


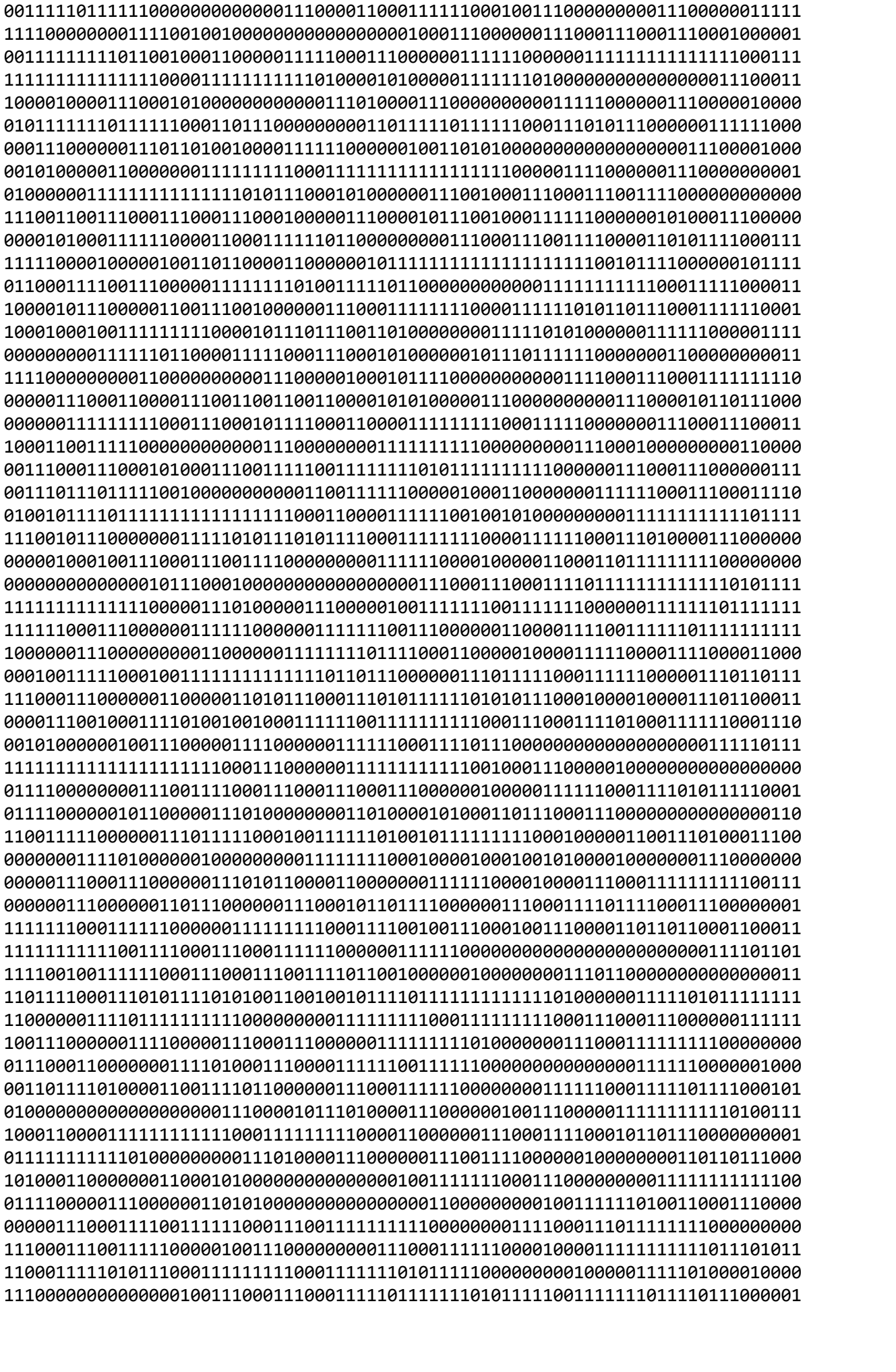




In [157]:
```
b = BinarySymmetricChanel(a)
print(b)
```







```
0001110001110000000001110000001111110001110000100011110001111110001001001110
1000100111111111111000000111000000011000000011000011000011101010001010000111
1000000011111111010011100011100010010110000011101011111100000001011111011000
0001110010000000000001010010010001001110001001011110000000000001000001011111
0000000000000111011000100111111000000011011100010011101000111100011011100011
1101100000011111100000000000011110000000000111000101000111010011001000100010
0001000000111111111000000110111010000000000000011100001011110000010011111111
0101000000111010111000111000001000111101010100101111111111000000101111111000
0011000011100011100011100010100011111100000011111110111000111111101111101100
0001101110001111111011110000000100000000111111111010001000111111111000011111
0000000010010100000001000000000100000011111101100011111111101111110000001011
1111110011100101000000000000100000011100011100001110100011101000011000011101
1111000000101111111010001110001011110101111110000110011110001111110001111111
1000000000111100000011100010100001111111000000000000000110010101000101111111
0011110100011001111100011111111111101101111011100000011001001000011011111101
1110000001000111000000000000110000001111110001100000110000000001100001010000
0000000001110000100000111000000111111111111000111000111111111111100011110100
0111111110000000000000000011100000111110100011111101100000000011100011111100
1111111110001110110011000110000001110001111110000000111000110010001000111000
0111000111010110111000111000110000111000111111111111111100111000110110111100
1111100010111101111110001111111111110000001111100001111110011110000000110001
1000001110000000111110000111000111111000100000000111000111000111111111111111
1101111111111111000111000101111111001111111000010000000011100001110101011101
100000000000101111000111110001111111111111111111111000100111000
```

In [158]:
```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
```

In [159]:
```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.