Problem. Consider a fair coin, which lands heads and tails each with a 50% probability. How many times should I flip the coin in order be 50% sure that there are at least n consecutive heads in my flips?

Solution. This is equivalent to asking, "How long a string of random bits should be taken to be 50% sure that there are no n consecutive 0s?

Define this class of $n-$long bitstrings with no $p-$consecutive 0s as S_p. Then we define the functional equation

$$B_p(z) = (1 + z + \cdots + z^{p-1})(1 + zB_p(z))$$

Solving for $B_p(z)$, we rewrite this as

$$B_p(z) = \frac{\frac{1-z^p}{1-z}}{1 - z\frac{1-z^p}{1-z}}$$

$$B_p(z) = \frac{1 - z^p}{1 - 2z + z^{p+1}}$$

One can verify this by simply expanding the series for this ordinary generating function; for instance, with $p = 3$:

$$\frac{1 - z^3}{1 - 2z + z^4}$$
$$= 1 + 2z + 4z^2 + 7z^3 + 13z^4 + 24z^5$$

(The exponent gives the length of bitstring n, and its coefficient provides the number of bitstrings of that length with no run of p consecutive 0s.)

We finally use the transfer theorem for rational generating functions to compute the values of constants $c_p \cdot \beta_p^N$. You can find more information about this computation at this link.

```
from mpmath import *
import functools
def calc(n):
    mp.dps=int(n*0.8) #This is more than enough digits of precision needed to work while being fast enough to run in a minute or so
    beta = lambda k: mp.fdiv(1,mp.findroot(lambda x: 1-2*x+x**(k+1), 0.5))
    c = lambda p: mp.fdiv(functools.reduce(lambda a, b: mp.fadd(a,b), [mp.power(beta(p),(i+1)) for i in range(p)]),functools.reduce(l
    return(str(int(ceil(mp.log(mp.fdiv(0.5,c(n)),b=mp.fdiv(beta(n),2))))))
print(int(calc(4)))
#22
```

Note that the ceil operation is used because this computation gives because we require "at **least** 50%". mpmath is used to increase the amount of precision needed in this computation.