PRG Code Documentation

A Class named **PRG** is created incorporating all required variables and functions

```
def __init__(self,prime_no,generator,bit_length,seed):
    self.prime_no = prime_no
    self.generator = generator
    self.bit_length = bit_length
    self.seed = seed
    self.generated_random_string = ""
```

This function initializes the required values of prime number,generator,bit length, seed value
Prime number and **generator** have trivial definitions.
Seed value is the input value.
Bit length specifies the bit length of the generated number.

The generated_random_string will be used to store the generated random number.

```
def modular_exponentiation(self,a,b,mod):
    result = 1
    while(b):
        if b%2:
            result = (result%mod*a%mod)%mod
        a = ((a%mod)*(a%mod))%mod
        b = b>>1
    return result
```

Modular exponentiation computes **(a^b)** in **log(b)** using divide and conquer technique order of time and this allows us to compute exponents

instantaneously.

```
def generate_random_bit(self):
    # n = self.bit_length
    val = self.seed
    for i in range(self.bit_length):
        num = self.modular_exponentiation(self.generator,val,self.prime_no)
        if num <= (self.prime_no-1)/2:
            self.generated_random_string+="1"
        else:
            self.generated_random_string+="0"
        val = num
```

Generate_random_bit takes the value of seed and the specified bit length It then computes each bit of the generated number using the below relation.

Let x_0 be a seed, and let

$$x_{i+1} = g^{x_i} \mod p.$$

Here x_i is stored in variable **val** and **num** stores x_{i+1}

Now the ith bit is calculated using the following relation:
If **num<= (p-1)/2** then the ith bit is 1
Otherwise it is 0
This is being checked in the if-else statement and the corresponding value gets appended to the random string that is being generated.

```
if __name__=="__main__":
    prg = PRG(36389,4,100,123123)
    prg.generate_random_bit()
    print(prg.generated_random_string)
    print(prg.generated_random_string.count('0')/len(prg.generated_random_string))
```

We generate the class and call the function from **main** part of the code first the object prg is created using the required values of prime number generator seed and output length.

generate_random_bit() function is then called to generate the random number