The "6k ± 1 rule" is a prime number optimization strategy that states that with the exception of 2 and 3, all prime numbers are either one less or one more than a multiple of 6. In other words, prime numbers can be expressed as 6k ± 1, where k is a non-negative integer.

Here's a brief explanation:

6k: All numbers divisible by 6 can be expressed as 6k, where k is an integer.

6k ± 1: Numbers of the form 6k ± 1 cover two cases: 6k + 1 and 6k - 1. These are the potential locations of prime numbers, as numbers that are not multiples of 2 or 3.

For example:

5 is a prime number, and it can be expressed as 6k - 1 (where k = 1). 7 is a prime number, and it can be expressed as 6k + 1 (where k = 1).

k	6k-1	6k	6k+1
1	5	6	7
2	11	12	13
3	17	18	19
4	23	24	25
5	29	30	31
6	35	36	37
7	41	42	43
8	47	48	49

use this rule to optimize finding prime numbers.

Method 1

In [7]:
```
iteration_count = 0
list1 = [2,3]
def pri(n):
    global iteration_count
    if n%2==0 or n%3==0:
        return 0
    for i in range(5, int((n**0.5) + 1), 6):
        iteration_count += 1
        if n%i==0 or n%(i+2)==0:
            return 0
    return 1

b = int(input("Enter Number: "))+ 1
for i in range(5,b):
    iteration_count += 1
    if pri(i):
        list1.append(i)

print(list1)
print("No of primes : ", len(list1))
print("No. of iteration : " , iteration_count)
```

```
Enter Number:  1000

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 7
1, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 1
51, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499,
503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601,
607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691,
701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907,
911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
No of primes :  168
No. of iteration :  1759
```

Method 2

In [4]:
```
iteration_count = 0
def is_prime(n):
  global iteration_count
  for i in range(5, int((n**0.5) + 1), 6):
      iteration_count += 1
      if n%i==0 or n%(i+2)==0:
        return 0
  return True

def generate_primes(n):
  global iteration_count
  primes_list = [2, 3]
  for i in range(5, n + 1, 6):
      iteration_count += 1
      if is_prime(i):
          primes_list.append(i)
      if is_prime(i + 2):
          primes_list.append(i + 2)
  return primes_list, iteration_count

n = int(input("Enter Number: "))
prime_numbers, iteration_count = generate_primes(n)

print(prime_numbers)
print("No of primes:", len(prime_numbers))
print("No. of iteration:", iteration_count)
```

```
Enter Number:  1000

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 7
1, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 1
51, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499,
503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601,
607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691,
701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907,
911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
No of primes: 168
No. of iteration: 929
```

Method 3

In [3]:
```
iteration_count = 0
primes_list = [2, 3]

def is_prime(n):
  global iteration_count,primes_list
  flag = [True, True]
  k = n + 2
  for i in range(5, int((n**0.5)) + 2, 6):
      iteration_count += 1
      if flag[0] and ( n%i==0 or n%(i+2)==0 ):flag[0] = False
      if flag[1] and ( k%i==0 or k%(i+2)==0 ): flag[1] = False
      if flag[0] == False and flag[1] == False: return

  if flag[0] : primes_list.append(n)
  if flag[1] : primes_list.append(k)

def generate_primes(n):
  global iteration_count
  for i in range(5, n + 1, 6):
      iteration_count += 1
      is_prime(i)

n = int(input("Enter Number: "))
generate_primes(n)

print(primes_list)
print("No of primes:", len(primes_list))
print("No. of iteration:", iteration_count)
```

```
Enter Number:  1000

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 7
1, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 1
51, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499,
503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601,
607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691,
701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907,
911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
No of primes: 168
No. of iteration: 678
```

Optimizing Finding Factors of A Number

In [11]:
```
iteration_count = 0
n = int(input("Enter Number: "))
factors = [1,n]
rt = int(n ** 0.5)
for i in range (2,rt):
    iteration_count += 1
    if n % i == 0:
        factors.append(i)
        factors.append(n//i)
if rt*rt == n:
    factors.append(rt)
factors.sort()
print(f"Total Iterations : ",iteration_count)
print(f"Factors of {n} are ",factors)
```

```
Enter Number:  10000
Total Iterations :  98
Factors of 10000 are  [1, 2, 4, 5, 8, 10, 16, 20, 25, 40, 50, 80, 100, 12
5, 200, 250, 400, 500, 625, 1000, 1250, 2000, 2500, 5000, 10000]
```