

```

1 from random import randrange
2
3
4
5 def run(n: int, t: int) -> bool:
6     """
7     n: number to be evaluated
8     t: number of test iterations
9     Function for running the Miller-Rabin primality test.
10    Returns a boolean showing that n is either composite or probably prime.
11    """
12
13    if n == 2 or n == 3:
14        return True
15
16    if n > 2 and n % 2 == 0:
17        return False # n is even
18
19    # we will halve m iteratively until we achieve the equality:
20    # n - 1 = (2^k)m
21    k = 0
22    m = n - 1
23
24    # this loop halves m
25    while m % 2 == 0:
26        k += 1
27        m //= 2 # floor div
28
29    # we now have k, m such that m is an odd integer
30
31    for _ in range(t):
32
33        # determining random test value in range [2, n - 1]
34        a = randrange(2, n - 1)
35
36        # getting initial value for b
37        b = pow(a, m, n)
38
39        # initial check for primality
40        if b == 1 or b == n - 1:
41            continue
42
43        # iterate until a result is found
44        for _ in range(k - 1):
45
46            # raising b**2 and getting remainder from modulo n
47            b = pow(b, 2, n)
48
49            # checking value of b
50            if b == n - 1:
51                break
52
53        # if inner loop ends, check reason for ending
54        # loop was broken
55        if b == n - 1:
56            continue
57        # loop ran out
58        else:
59            return False
60
61        # if all iterations were completed without throwing False, then n is probably prime
62    return True

```

Python module for running the Miller-Rabin primality test. Will return boolean value to indicate if the number being testing

is composite or *probably* prime. Optimal t-value is 60 with probability of failure being approximately 2^{-128} .

```

1
2 from math import log
3
4
5 def run(primeList: list) -> list:
6     """
7     Chebyshev's Theta Function.
8     Returns a sorted list containing the log transformed product of the primorial at each prime in the
9     given list.
10    """
11
12    # list for storing the product at each nth prime
13    products = list()
14
15    # initiating var to store current Theta(x)
16    lastVal = 0
17
18    # iterating through all primes in the list
19    for prime in primeList:
20
21        # getting sum of logs
22        # using log laws, we know log(n) + log(m) == log(nm)
23        current = log(prime) + lastVal
24
25        # storing to list
26        products.append(current)
27
28        # updating product
29        lastVal = current
30
31    return products

```

Python code to calculate $\theta(x)$, also known as Chebyshev's First Function.

```

1
2 def run(primes: list) -> dict:
3     """
4     Returns a dict of all Germain Prime sequences identified in the given list
5     """
6
7     # dict for storing results
8     sequences = dict()
9
10    # building set of primes of O(1) checking
11    primeSet = set(primes)
12
13    # iterating through all primes in given list
14    for prime in primes:
15
16        # list for storing the current sequence achieved
17        seq = list()
18
19        # assigning the first prime to check as the current prime in the given list of primes
20        gt = prime
21
22        # checking that gt is Germain, and if so, adding to sequence and updating gt
23        while (gt * 2) + 1 in primeSet:
24            seq.append(gt)
25
26            gt = (gt * 2) + 1
27
28        # if the seq variable is not empty, meaning at least one Germain prime was identified, it gets added
29        # to results
30        if seq:
31            sequences[prime] = {"sequence": seq, "length": len(seq)}
32
33    # returning the results
34    return sequences
35
36 content...

```

Python function to generate sequences of Germain Primes.