```python

from random import randrange


def run(n: int, t: int) -> bool:
  """
  n: number to be evaluated
  t: number of test iterations
  Function for running the Miller-Rabin primality test.
  Returns a boolean showing that n is either composite or probably prime.
  """

  if n == 2 or n == 3:
    return True

  if n > 2 and n % 2 == 0:
    return False  # n is even

  # we will halve m iteratively until we achieve the equality:
  # n - 1 = (2^k)m
  k = 0
  m = n - 1

  # this loop halves m
  while m % 2 == 0:
    k += 1
    m //= 2  # floor div

  # we now have k, m such that m is an odd integer

  for _ in range(t):

    # determining random test value in range [2, n - 1]
    a = randrange(2, n - 1)

    # getting initial value for b
    b = pow(a, m, n)

    # initial check for primality
    if b == 1 or b == n - 1:
      continue

  # iterate until a result is found
  for _ in range(k - 1):

  # raising b**2 and getting remainder from modulo n
  b = pow(b, 2, n)

  # checking value of b
  if b == n - 1:
    break

  # if inner loop ends, check reason for ending
  # loop was broken
  if b == n - 1:
    continue
  # loop ran out
  else:
    return False

  # if all iterations were completed without throwing False, then n is probably prime
  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}.

```python

from math import log


def run(primeList: list) -> list:
  """
  Chebyshev's Theta Function.
  Returns a sorted list containing the log transformed product of the primorial at each prime in the
    given list.
  """

  # list for storing the product at each nth prime
  products = list()

  # initiating var to store current Theta(x)
  lastVal = 0

  # iterating through all primes in the list
  for prime in primeList:

    # getting sum of logs
    # using log laws, we know log(n) + log(m) == log(nm)
    current = log(prime) + lastVal

    # storing to list
    products.append(current)

    # updating product
    lastVal = current

  return products
```

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

```python

def run(primes: list) -> dict:
  """
  Returns a dict of all Germain Prime sequences identified in the given list
  """

  # dict for storing results
  sequences = dict()

  # building set of primes of O(1) checking
  primeSet = set(primes)

  # iterating through all primes in given list
  for prime in primes:

    # list for storing the current sequence achieved
    seq = list()

    # assigning the first prime to check as the current prime in the given list of primes
    gt = prime

    # checking that gt is Germain, and if so, adding to sequence and updating gt
    while (gt * 2) + 1 in primeSet:
    seq.append(gt)

    gt = (gt * 2) + 1

    # if the seq variable is not empty, meaning at least one Germain prime was identified, it gets added to results
    if seq:
    sequences[prime] = {"sequence": seq, "length": len(seq)}

  # returning the results
  return sequences
```

Python function to generate sequences of Germain Primes.

```python

def run(prime: int) -> bool:
  """
  Checks if the given prime is a palindrome arithmetically.
  Returns boolean.
  """

  if prime < 10:
    return True

  # saving prime to variable n to check if n == reverse
  n = prime

  # variable to track reversed prime value
  rev = 0

  # since we are using floor division for prime, we iterate until prime <= 0
  while prime > 0:

    # the currect digit being worked on is the remainder from mod 10, giving us the last digit
    dig = prime % 10

    # multiply current reverse by 10 to allow for addition of dig
    rev = (rev * 10) + dig

    # floor division on prime to truncate last digit
    prime = prime // 10

  # if n, the starting prime, is equal to the reverse then it is a palindrome
  if n == rev:
    return True
  else:
    return False
```

Python function to identify palindromic prime numbers.