Contents

Chapter 1

Functions

1.1 prime – primality test , prime generation

1.1.1 trialDivision – trial division test

trialDivision(n: *integer*, bound: *integer/float*=0) → *True/False*

Trial division primality test for an odd natural number.

bound is a search bound of primes. If it returns 1 under the condition that bound is given and less than the square root of n, it only means there is no prime factor less than bound.

1.1.2 spsp – strong pseudo-prime test

spsp(n: *integer*, base: *integer*, s: *integer*=None, t: *integer*=None) → *True/False*

Strong Pseudo-Prime test on base base.

s and t are the numbers such that $n - 1 = 2^s t$ and t is odd.

1.2 smallSpsp – strong pseudo-prime test for small number

smallSpsp(n: *integer*) → *True/False*

Strong Pseudo-Prime test for integer n less than 10^{12}.

4 spsp tests are sufficient to determine whether an integer less than 10^{12} is prime or not.

1.2.1 miller – Miller's primality test

miller(`n`: *integer*) → ***True/False***

Miller's primality test.

This test is valid under GRH. See **config**.

1.2.2 millerRabin – Miller-Rabin primality test

miller(`n`: *integer*, `times`: *integer*=**20**) → ***True/False***

Miller's primality test.

The difference from **miller** is that the method uses (fast) probabilistic algorithm, on the other hand, **miller** employs deterministic algorithm under GRH.

`times` (default to 20) is the number of repetition. The error probability is at most $4^{-\texttt{times}}$.

1.3 lpsp – Lucas test

lpsp(`n`: *integer*, `a`: *integer*, `b`: *integer*) → ***True/False***

Lucas test.

Return True if `n` is a Lucas pseudoprime of parameters `a`, `b`, i.e. with respect to $x^2 - \texttt{a}x + \texttt{b}$.

1.4 fpsp – Frobenius test

fpsp(`n`: *integer*, `a`: *integer*, `b`: *integer*) → ***True/False***

Frobenius test.

Return True if `n` is a Frobenius pseudoprime of parameters `a`, `b`, i.e. with respect to $x^2 - \texttt{a}x + \texttt{b}$.

1.5 apr – Jacobi sum test

apr(n: *integer*) → ***True/False***

APR (Adleman-Pomerance-Rumery) primality test or the Jacobi sum test.

Assuming n has no prime factors less than 32. Assuming n is spsp (strong pseudo-prime) for several bases.

1.6 primeq – primality test automatically

primeq(n: *integer*) → ***True/False***

A convenient function for primality test.

It uses one of **trialDivision**, **smallSpsp** or **apr** depending on the size of n.

1.7 prime – n-th prime number

prime(n: *integer*) → *integer*

Return the n-th prime number.

1.8 nextPrime – generate next prime

nextPrime(n: *integer*) → *integer*

Return the smallest prime bigger than the given integer n.

1.9 randPrime – generate random prime

randPrime(n: *integer*) → *integer*

Return a random n-digits prime.

1.10 generator – generate primes

generator((None)) → *generator*

Generate primes from 2 to ∞ (as generator).

1.11 generator_eratosthenes – generate primes using Eratosthenes sieve

generator_eratosthenes(n: *integer*) → *generator*

Generate primes up to `n` using Eratosthenes sieve.

1.12 primonial – product of primes

primonial(p: *integer*) → *integer*

Return the product

$$\prod_{q \in \mathbb{P}_{\leq \mathtt{p}}} q = 2 \cdot 3 \cdot 5 \cdots \mathtt{p} \ .$$

1.13 properDivisors – proper divisors

properDivisors(n: *integer*) → *list*

Return proper divisors of `n` (all divisors of `n` excluding 1 and `n`).

It is only useful for a product of small primes.

The output is the list of all proper divisors.

1.14 primitive_root – primitive root

primitive_root(p: *integer*) → *integer*

Return a primitive root of `p`.

`p` must be an odd prime.

1.15 Lucas_chain – Lucas sequence

Lucas_chain(`n`: *integer*, `f`: *function*, `g`: *function*, `x_0`: *integer*, `x_1`: *integer*)
→ (*integer*, *integer*)

Return the value of (x_n, x_{n+1}) for the sequnce $\{x_i\}$ defined as:

$$x_{2i} = \mathtt{f}(x_i)$$
$$x_{2i+1} = \mathtt{g}(x_i, x_{i+1}) \ ,$$

where the initial values `x_0`, `x_1`.

`f` is the function which can be input as 1-ary integer. `g` is the function which can be input as 2-ary integer.

Examples

```
>>> prime.primeq(131)
True
>>> prime.primeq(133)
False
>>> g = prime.generator()
>>> g.next()
2
>>> g.next()
3
>>> prime.prime(10)
29
>>> prime.nextPrime(100)
101
>>> prime.properDivisors(2 * 3 * 5 * 7)
[2, 3, 5, 6, 7, 10, 14, 15, 21, 30, 35, 42, 70, 105]
>>> prime.primitive_root(23)
5
```

Bibliography