Contents

1 Classes **2**

1.1 factor.util – utilities for factorization 2

1.1.1 FactoringInteger – keeping track of factorization 3

1.1.1.1 getNextTarget – next target 4

1.1.1.2 getResult – result of factorization 4

1.1.1.3 register – register a new factor 4

1.1.1.4 sortFactors – sort factors 4

1.1.2 FactoringMethod – method of factorization 6

1.1.2.1 factor – do factorization 7

1.1.2.2 †continue_factor – continue factorization 7

1.1.2.3 †find – find a factor 7

1.1.2.4 †generate – generate prime factors 7

Chapter 1

Classes

1.1 factor.util – utilities for factorization

- **Classes**
  - **FactoringInteger**
  - **FactoringMethod**

This module uses following type:

factorlist :
`factorlist` is a list which consists of pairs `(base, index)`. Each pair means $base^{index}$. The product of those terms expresses whole prime factorization.

1.1.1 FactoringInteger – keeping track of factorization

Initialize (Constructor)

FactoringInteger(`number`: *integer*) → ***FactoringInteger***

This is the base class for factoring integers.

`number` is stored in the attribute **number**. The factors will be stored in the attribute **factors**, and primality of factors will be tracked in the attribute **primality**.

The given `number` must be a composite number.

Attribute

number :
The composite number.

factors :
Factors known at the time being referred.

primality :
A dictionary of primality information of known factors. `True` if the factor is prime, `False` composite, or `None` undetermined.

Methods

1.1.1.1 getNextTarget – next target

getNextTarget(`self`, `cond`: *function*=`None`) → *integer*

Return the next target which meets `cond`.

If `cond` is not specified, then the next target is a composite (or undetermined) factor of **number**.

`cond` should be a binary predicate whose arguments are base and index.
If there is no target factor, LookupError will be raised.

1.1.1.2 getResult – result of factorization

getResult(`self`) → **factors**

Return the currently known factorization of the **number**.

1.1.1.3 register – register a new factor

register(`self`, `divisor`: *integer*, `isprime`: *bool*=`None`)
→

Register a `divisor` of the **number** if the `divisor` is a true divisor of the number.

The number is divided by the `divisor` as many times as possible.

The optional argument `isprime` tells the primality of the `divisor` (default to undetermined).

1.1.1.4 sortFactors – sort factors

sortFactors(`self`) →
Sort factors list.

This affects the result of **getResult**.

Examples

```
>>> A = factor.util.FactoringInteger(100)
>>> A.getNextTarget()
100
>>> A.getResult()
[(100, 1)]
>>> A.register(5, True)
>>> A.getResult()
[(5, 2), (4, 1)]
>>> A.sortFactors()
>>> A.getResult()
[(4, 1), (5, 2)
>>> A.primality
{4: None, 5: True}
>>> A.getNextTarget()
4
```

1.1.2 FactoringMethod – method of factorization

Initialize (Constructor)

FactoringMethod() → *FactoringMethod*

Base class of factoring methods.

All methods defined in **factor.methods** are implemented as derived classes of this class. The method which users may call is **factor** only. Other methods are explained for future implementers of a new factoring method.

Methods

1.1.2.1 factor – do factorization

factor(`self`, `number`: *integer*, `return_type`: *str*='list', `need_sort`: *bool*=False)
→ **factorlist**

Return the factorization of the given positive integer `number`.
The default returned type is a **factorlist**.
A keyword option `return_type` can be as the following:

1. `'list'` for default type (**factorlist**).
2. `'tracker'` for **FactoringInteger**.

Another keyword option `need_sort` is Boolean: `True` to sort the result. This should be specified with `return_type='list'`.

1.1.2.2 †continue_factor – continue factorization

continue_factor(`self`, `tracker`: **FactoringInteger**, `return_type`: *str*='tracker', `primeq`: *func*=**primeq**)
→ **FactoringInteger**

Continue factoring of the given `tracker` and return the result of factorization.

The default returned type is **FactoringInteger**, but if `return_type` is specified as `'list'` then it returns **factorlist**. The primality is judged by a function specified in `primeq` optional keyword argument, which default is **primeq**.

1.1.2.3 †find – find a factor

find(`self`, `target`: *integer*, **`options`) → *integer*

Find a factor from the `target` number.

This method has to be overridden, or **factor** method should be overridden not to call this method.

1.1.2.4 †generate – generate prime factors

generate(`self`, `target`: *integer*, **`options`) → *integer*

Generate prime factors of the `target` number with their valuations.

The method may terminate with yielding `(1, 1)` to indicate the factorization is incomplete.
This method has to be overridden, or **factor** method should be overridden not to call this method.

Bibliography