BaseHash Documentation

Release 1.0.6

Nathan Lucas

Sep 27, 2017

Contents

BaseHash is available on PyPi, to install simply do `pip install basehash`. The source is availabile at the GitHub repository python-basehash.

Contents:

CHAPTER 1

The heart of BaseHash

BaseHash Constants

The two constants of BaseHash are `HASH_LENGTH` and `GENERATOR`.

`HASH_LENGTH`, default set to `6`, is used as a default hashing length, which can be overridden in `baseN.hash()`.

`GENERATOR` uses the Golden Ratio, `1.618033988749894848`, to determine the next highest prime, which is based on `base ^ length - 1`. `GENERATOR` can either be overridden globally or can be overridden within `base_hash` or `base_unhash`.

prime

prime(*base*, *n*, *gen*)
Returns next highest prime. using base^n * gen.

base_encode

base_encode(*num*, *alphabet*)
Encodes *int* `num` to *base* `alphabet`. Returns `string`

base_decode

base_decode(*key*, *alphabet*)
Decodes *string* `key` from *string* `alphabet` (or `base`). Returns `int`

base_hash

base_hash(*num*, *length*, *alphabet*[, *gen=GENERATOR*])

Hashes *int* `num` to *string* `alphabet` (or `base`), *int* `length` digits long using the built in `base.GENERATOR`, which can be overridden. Returns `string`

base_unhash

base_unhash(*key*, *alphabet*[, *gen=GENERATOR*])

Unhashes *string* `key` from *string* `alphabet` (or `base`) using the built in `base.GENERATOR`, which can be overridden.

base_maximum

base_maximum(*base*, *length*)

Returns maximum `int` that *int* `base` ^ *int* `length` can take. Returns `int`

CHAPTER 2

Built-in `BaseN`

BaseHash comes with a few built-in bases, `Base36`, `Base52`, `Base56`, `Base58`, `Base62`, and `Base94`.

BaseN.BASEN

```
BASE36 = 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
BASE52 = 0123456789BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyz
BASE56 = 23456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnpqrstuvwxyz
BASE58 = 123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
BASE62 = 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
BASE94 = !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

encode

`baseN.`**`encode`**`(`*num*`)`

Encodes *int* `num` to `baseN`. Returns `base_encode(num, BASEN)`. Returns `string`

decode

`baseN.`**`decode`**`(`*key*`)`

Decodes *string* key from `baseN`. Returns `base_decode(key, BASEN)`. Returns `int`

hash

baseN.**hash**(*num*[, *length=HASH_LENGTH*])

Hashes *int* num to baseN at *int* length characters. Returns base_hash(num, length, BASEN). Returns string

unhash

baseN.**unhash**(*key*)

Unhashes *string* key from baseN. Returns base_unhash(key, BASEN). Returns int

maximum

baseN.**maximum**([*length=HASH_LENGTH*])

Returns maximum value for a hash of given *int* length. Returns base_maximum(len(BASEN), length) Returns int

CHAPTER 3

Extending to `BaseX`

Much work was put into generating prime numbers on the fly, allowing BaseHash to be extended to `BaseX` with ease. To extend the library, you just need to import `basehash.base` and call a few mthods.

```
from basehash.base import *

# ALPHA must be a tuple
ALPHA = tuple('24680ACEGIKMOQSUWYbdfhjlnprtvxz')


# hash `num` to `ALPHA` at `length` characters
def hash(num, length=HASH_LENGTH):
    return base_hash(num, length, ALPHA)


# unhash `key` from `ALPHA`
def unhash(key):
    return base_unhash(key, ALPHA)

## optional methods:


# encode `num` to `ALPHA`
def encode(num):
    return base_encode(num, ALPHA)


# decode `key` from `ALPHA`
def decode(key):
    return base_decode(key, ALPHA)


# return maximum value for `hash` at `length`
def maximum(length=HASH_LENGTH):
    return base_maximum(len(ALPHA), length)
```

CHAPTER 4

Indices and tables

- genindex
- search

Index

B

P