Sage Reference Manual: Monoids

Release 6.3

The Sage Development Team

August 11, 2014

CONTENTS

1 Monoids 3

2 Free Monoids 5

3 Monoid Elements 9

4 Free abelian monoids 11

5 Abelian monoid elements 15

6 Indexed Monoids 17

7 String Monoid Elements 23

8 Free String Monoids 27

9 Indices and Tables 35

Bibliography 37

Sage supports free monoids and free abelian monoids in any finite number of indeterminates.

CHAPTER

ONE

MONOIDS

class sage.monoids.monoid.**Monoid_class**(*names*)

Bases: sage.structure.parent.Parent

EXAMPLES:

```
sage: from sage.monoids.monoid import Monoid_class
sage: Monoid_class(('a','b'))
<class 'sage.monoids.monoid.Monoid_class_with_category'>
```

TESTS:

```
sage: F.<a,b,c,d,e> = FreeMonoid(5)
sage: TestSuite(F).run()
```

gens()

Returns the generators for self.

EXAMPLES:

```
sage: F.<a,b,c,d,e> = FreeMonoid(5)
sage: F.gens()
(a, b, c, d, e)
```

sage.monoids.monoid.**is_Monoid**(*x*)

Returns True if x is of type Monoid_class.

EXAMPLES:

```
sage: from sage.monoids.monoid import is_Monoid
sage: is_Monoid(0)
False
sage: is_Monoid(ZZ)    # The technical math meaning of monoid has
...                    # no bearing whatsoever on the result: it's
...                    # a typecheck which is not satisfied by ZZ
...                    # since it does not inherit from Monoid_class.
False
sage: is_Monoid(sage.monoids.monoid.Monoid_class(('a','b')))
True
sage: F.<a,b,c,d,e> = FreeMonoid(5)
sage: is_Monoid(F)
True
```

CHAPTER

TWO

FREE MONOIDS

AUTHORS:

- David Kohel (2005-09)
- Simon King (2011-04): Put free monoids into the category framework

Sage supports free monoids on any prescribed finite number $n \geq 0$ of generators. Use the `FreeMonoid` function to create a free monoid, and the `gen` and `gens` functions to obtain the corresponding generators. You can print the generators as arbitrary strings using the optional `names` argument to the `FreeMonoid` function.

`sage.monoids.free_monoid.`**`FreeMonoid`**(*index_set=None*, *names=None*, *commutative=False*, ***kwds*)

Return a free monoid on n generators or with the generators indexed by a set I.

We construct free monoids by specifing either:

•the number of generators and/or the names of the generators

•the indexing set for the generators

INPUT:

•`index_set` – an indexing set for the generators; if an integer, than this becomes $\{0, 1, \ldots, n-1\}$

•`names` – names of generators

•`commutative` – (default: `False`) whether the free monoid is commutative or not

OUTPUT:

A free monoid.

EXAMPLES:

```
sage: F.<a,b,c,d,e> = FreeMonoid(); F
Free monoid on 5 generators (a, b, c, d, e)
sage: FreeMonoid(index_set=ZZ)
Free monoid indexed by Integer Ring

sage: F.<x,y,z> = FreeMonoid(abelian=True); F
Free abelian monoid on 3 generators (x, y, z)
sage: FreeMonoid(index_set=ZZ, commutative=True)
Free abelian monoid indexed by Integer Ring
```

class `sage.monoids.free_monoid.`**`FreeMonoidFactory`**

Bases: `sage.structure.factory.UniqueFactory`

Create the free monoid in n generators.

INPUT:

- n - integer
- names - names of generators

OUTPUT: free monoid

EXAMPLES:

```
sage: FreeMonoid(0,'')
Free monoid on 0 generators ()
sage: F.<a,b,c,d,e> = FreeMonoid(5); F
Free monoid on 5 generators (a, b, c, d, e)
sage: F(1)
1
sage: mul([ a, b, a, c, b, d, c, d ], F(1))
a*b*a*c*b*d*c*d
```

create_key(*n*, *names*)
x.__init__(...) initializes x; see help(type(x)) for signature

create_object(*version*, *key*, ***kwds*)
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.monoids.free_monoid.**FreeMonoid_class**(*n*, *names=None*)
Bases: sage.monoids.monoid.Monoid_class

The free monoid on n generators.

Element
alias of FreeMonoidElement

cardinality()
Return the cardinality of self, which is ∞.

EXAMPLES:

```
sage: F = FreeMonoid(2005, 'a')
sage: F.cardinality()
+Infinity
```

gen(*i=0*)
The i-th generator of the monoid.

INPUT:

- i - integer (default: 0)

EXAMPLES:

```
sage: F = FreeMonoid(3, 'a')
sage: F.gen(1)
a1
sage: F.gen(2)
a2
sage: F.gen(5)
Traceback (most recent call last):
...
IndexError: Argument i (= 5) must be between 0 and 2.
```

ngens()
The number of free generators of the monoid.

EXAMPLES:

```
sage: F = FreeMonoid(2005, 'a')
sage: F.ngens()
2005
```

one_element()
Returns the identity element in this monoid.

EXAMPLES:

```
sage: F = FreeMonoid(2005, 'a')
sage: F.one_element()
1
```

sage.monoids.free_monoid.**is_FreeMonoid**(x)
Return True if x is a free monoid.

EXAMPLES:

```
sage: from sage.monoids.free_monoid import is_FreeMonoid
sage: is_FreeMonoid(5)
False
sage: is_FreeMonoid(FreeMonoid(7,'a'))
True
sage: is_FreeMonoid(FreeAbelianMonoid(7,'a'))
False
sage: is_FreeMonoid(FreeAbelianMonoid(0,''))
False
sage: is_FreeMonoid(FreeMonoid(index_set=ZZ))
True
sage: is_FreeMonoid(FreeAbelianMonoid(index_set=ZZ))
False
```

CHAPTER

THREE

MONOID ELEMENTS

AUTHORS:

- David Kohel (2005-09-29)

Elements of free monoids are represented internally as lists of pairs of integers.

class sage.monoids.free_monoid_element.**FreeMonoidElement**(*F*, *x*, *check=True*)

Bases: sage.structure.element.MonoidElement

Element of a free monoid.

EXAMPLES:

```
sage: a = FreeMonoid(5, 'a').gens()
sage: x = a[0]*a[1]*a[4]**3
sage: x**3
a0*a1*a4^3*a0*a1*a4^3*a0*a1*a4^3
sage: x**0
1
sage: x**(-1)
Traceback (most recent call last):
...
TypeError: bad operand type for unary ~: 'FreeMonoid_class_with_category.element_class'
```

to_word(*alph=None*)

Return self as a word.

INPUT:

- alph – (optional) the alphabet which the result should be specified in

EXAMPLES:

```
sage: M.<x,y,z> = FreeMonoid(3)
sage: a = x * x * y * x
sage: w = a.to_word(); w
word: xxyx
sage: w.to_monoid_element() == a
True
```

sage.monoids.free_monoid_element.**is_FreeMonoidElement**(*x*)

x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

FOUR

FREE ABELIAN MONOIDS

AUTHORS:

- David Kohel (2005-09)

Sage supports free abelian monoids on any prescribed finite number $n \geq 0$ of generators. Use the `FreeAbelianMonoid` function to create a free abelian monoid, and the `gen` and `gens` functions to obtain the corresponding generators. You can print the generators as arbitrary strings using the optional `names` argument to the `FreeAbelianMonoid` function.

EXAMPLE 1: It is possible to create an abelian monoid in zero or more variables; the syntax T(1) creates the monoid identity element even in the rank zero case.

```
sage: T = FreeAbelianMonoid(0, '')
sage: T
Free abelian monoid on 0 generators ()
sage: T.gens()
()
sage: T(1)
1
```

EXAMPLE 2: A free abelian monoid uses a multiplicative representation of elements, but the underlying representation is lists of integer exponents.

```
sage: F = FreeAbelianMonoid(5,names='a,b,c,d,e')
sage: (a,b,c,d,e) = F.gens()
sage: a*b^2*e*d
a*b^2*d*e
sage: x = b^2*e*d*a^7
sage: x
a^7*b^2*d*e
sage: x.list()
[7, 2, 0, 1, 1]
```

sage.monoids.free_abelian_monoid.**FreeAbelianMonoid**(*index_set=None*, *names=None*, ***kwds*)

Return a free abelian monoid on n generators or with the generators indexed by a set I.

We construct free abelian monoids by specifing either:

- the number of generators and/or the names of the generators
- the indexing set for the generators (this ignores the other two inputs)

INPUT:

- `index_set` – an indexing set for the generators; if an integer, then this becomes $\{0, 1, \ldots, n-1\}$

- names – names of generators

OUTPUT:

A free abelian monoid.

EXAMPLES:

```
sage: F.<a,b,c,d,e> = FreeAbelianMonoid(); F
Free abelian monoid on 5 generators (a, b, c, d, e)
sage: FreeAbelianMonoid(index_set=ZZ)
Free abelian monoid indexed by Integer Ring
```

class sage.monoids.free_abelian_monoid.**FreeAbelianMonoidFactory**

Bases: sage.structure.factory.UniqueFactory

Create the free abelian monoid in n generators.

INPUT:

- n - integer
- names - names of generators

OUTPUT: free abelian monoid

EXAMPLES:

```
sage: FreeAbelianMonoid(0, '')
Free abelian monoid on 0 generators ()
sage: F = FreeAbelianMonoid(5,names = list("abcde"))
sage: F
Free abelian monoid on 5 generators (a, b, c, d, e)
sage: F(1)
1
sage: (a, b, c, d, e) = F.gens()
sage: mul([ a, b, a, c, b, d, c, d ], F(1))
a^2*b^2*c^2*d^2
sage: a**2 * b**3 * a**2 * b**4
a^4*b^7

sage: loads(dumps(F)) is F
True
```

create_key(*n*, *names*)

x.__init__(...) initializes x; see help(type(x)) for signature

create_object(*version*, *key*)

x.__init__(...) initializes x; see help(type(x)) for signature

class sage.monoids.free_abelian_monoid.**FreeAbelianMonoid_class**(*n*, *names*)

Bases: sage.structure.parent_gens.ParentWithGens

Free abelian monoid on n generators.

cardinality()

Return the cardinality of self, which is ∞.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(3000, 'a')
sage: F.cardinality()
+Infinity
```

gen(*i=0*)

The i-th generator of the abelian monoid.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(5,'a')
sage: F.gen(0)
a0
sage: F.gen(2)
a2
```

ngens()

The number of free generators of the abelian monoid.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(3000, 'a')
sage: F.ngens()
3000
```

sage.monoids.free_abelian_monoid.**is_FreeAbelianMonoid**(*x*)

Return True if x is a free abelian monoid.

EXAMPLES:

```
sage: from sage.monoids.free_abelian_monoid import is_FreeAbelianMonoid
sage: is_FreeAbelianMonoid(5)
False
sage: is_FreeAbelianMonoid(FreeAbelianMonoid(7,'a'))
True
sage: is_FreeAbelianMonoid(FreeMonoid(7,'a'))
False
sage: is_FreeAbelianMonoid(FreeMonoid(0,''))
False
```

CHAPTER

FIVE

ABELIAN MONOID ELEMENTS

AUTHORS:

- David Kohel (2005-09)

EXAMPLES:

Recall the example from abelian monoids.

```
sage: F = FreeAbelianMonoid(5,names = list("abcde"))
sage: (a,b,c,d,e) = F.gens()
sage: a*b^2*e*d
a*b^2*d*e
sage: x = b^2*e*d*a^7
sage: x
a^7*b^2*d*e
sage: x.list()
[7, 2, 0, 1, 1]
```

It is important to note that lists are mutable and the returned list is not a copy. As a result, reassignment of an element of the list changes the object.

```
sage: x.list()[0] = 0
sage: x
b^2*d*e
```

class sage.monoids.free_abelian_monoid_element.**FreeAbelianMonoidElement**(F, x)

Bases: sage.structure.element.MonoidElement

Create the element x of the FreeAbelianMonoid F.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(5, 'abcde')
sage: F
Free abelian monoid on 5 generators (a, b, c, d, e)
sage: F(1)
1
sage: a, b, c, d, e = F.gens()
sage: a^2 * b^3 * a^2 * b^4
a^4*b^7
sage: F = FreeAbelianMonoid(5, 'abcde')
sage: a, b, c, d, e = F.gens()
sage: a in F
True
sage: a*b in F
True
```

list()
Return (a reference to) the underlying list used to represent this element. If this is a monoid in an abelian monoid on n generators, then this is a list of nonnegative integers of length n.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(5, 'abcde')
sage: (a, b, c, d, e) = F.gens()
sage: a.list()
[1, 0, 0, 0, 0]
```

sage.monoids.free_abelian_monoid_element.**is_FreeAbelianMonoidElement**(*x*)
Queries whether x is an object of type FreeAbelianMonoidElement.

INPUT:

- x – an object.

OUTPUT:

- True if x is an object of type FreeAbelianMonoidElement; False otherwise.

CHAPTER

SIX

INDEXED MONOIDS

AUTHORS:

- Travis Scrimshaw (2013-10-15)

class sage.monoids.indexed_free_monoid.**IndexedFreeAbelianMonoid**(*indices*, *prefix*, *category=None*, *names=None*, ***kwds*)

Bases: sage.monoids.indexed_free_monoid.IndexedMonoid

Free abelian monoid with an indexed set of generators.

INPUT:

- indices – the indices for the generators

For the optional arguments that control the printing, see IndexedGenerators.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: F.gen(15)^3 * F.gen(2) * F.gen(15)
F[2]*F[15]^4
sage: F.gen(1)
F[1]
```

Now we examine some of the printing options:

```
sage: F = FreeAbelianMonoid(index_set=Partitions(), prefix='A', bracket=False, scalar_mult='%')
sage: F.gen([3,1,1]) * F.gen([2,2])
A[2, 2]%A[3, 1, 1]
```

Element

alias of IndexedFreeAbelianMonoidElement

gen(*x*)

The generator indexed by x of self.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: F.gen(0)
F[0]
sage: F.gen(2)
F[2]
```

one()

Return the identity element of self.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: F.one()
1
```

class sage.monoids.indexed_free_monoid.**IndexedFreeAbelianMonoidElement**(*F*, *x*)

Bases: sage.monoids.indexed_free_monoid.IndexedMonoidElement

An element of an indexed free abelian monoid.

dict()

Return `self` as a dictionary.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (a*c^3).dict()
{0: 1, 2: 3}
```

length()

Return the length of `self`.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: elt = a*c^3*b^2*a
sage: elt.length()
7
sage: len(elt)
7
```

class sage.monoids.indexed_free_monoid.**IndexedFreeMonoid**(*indices*, *prefix*, *category=None*, *names=None*, ***kwds*)

Bases: sage.monoids.indexed_free_monoid.IndexedMonoid

Free monoid with an indexed set of generators.

INPUT:

- `indices` – the indices for the generators

For the optional arguments that control the printing, see `IndexedGenerators`.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: F.gen(15)^3 * F.gen(2) * F.gen(15)
F[15]^3*F[2]*F[15]
sage: F.gen(1)
F[1]
```

Now we examine some of the printing options:

```
sage: F = FreeMonoid(index_set=ZZ, prefix='X', bracket=['|','>'])
sage: F.gen(2) * F.gen(12)
X|2>*X|12>
```

Element

alias of IndexedFreeMonoidElement

gen(*x*)
The generator indexed by x of self.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: F.gen(0)
F[0]
sage: F.gen(2)
F[2]
```

one()
Return the identity element of self.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: F.one()
1
```

class sage.monoids.indexed_free_monoid.**IndexedFreeMonoidElement**(*F*, *x*)
Bases: sage.monoids.indexed_free_monoid.IndexedMonoidElement

An element of an indexed free abelian monoid.

length()
Return the length of self.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: elt = a*c^3*b^2*a
sage: elt.length()
7
sage: len(elt)
7
```

class sage.monoids.indexed_free_monoid.**IndexedMonoid**(*indices*, *prefix*, *category=None*, *names=None*, ***kwds*)
Bases: sage.structure.parent.Parent, sage.structure.indexed_generators.IndexedGenerators, sage.structure.unique_representation.UniqueRepresentation

Base class for monoids with an indexed set of generators.

INPUT:

- indices – the indices for the generators

For the optional arguments that control the printing, see IndexedGenerators.

cardinality()
Return the cardinality of self, which is ∞ unless this is the trivial monoid.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: F.cardinality()
+Infinity
sage: F = FreeMonoid(index_set=())
sage: F.cardinality()
1

sage: F = FreeAbelianMonoid(index_set=ZZ)
```

```
sage: F.cardinality()
+Infinity
sage: F = FreeAbelianMonoid(index_set=())
sage: F.cardinality()
1
```

gens()
Return the monoid generators of `self`.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: F.monoid_generators()
Lazy family (Generator map from Integer Ring to
 Free abelian monoid indexed by Integer Ring(i))_{i in Integer Ring}
sage: F = FreeAbelianMonoid(index_set=tuple('abcde'))
sage: sorted(F.monoid_generators())
[F['a'], F['b'], F['c'], F['d'], F['e']]
```

monoid_generators()
Return the monoid generators of `self`.

EXAMPLES:

```
sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: F.monoid_generators()
Lazy family (Generator map from Integer Ring to
 Free abelian monoid indexed by Integer Ring(i))_{i in Integer Ring}
sage: F = FreeAbelianMonoid(index_set=tuple('abcde'))
sage: sorted(F.monoid_generators())
[F['a'], F['b'], F['c'], F['d'], F['e']]
```

class sage.monoids.indexed_free_monoid.**IndexedMonoidElement**(F, x)
Bases: sage.structure.element.MonoidElement

An element of an indexed monoid.

This is an abstract class which uses the (abstract) method `_sorted_items()` for all of its functions. So to implement an element of an indexed monoid, one just needs to implement `_sorted_items()`, which returns a list of pairs `(i, p)` where `i` is the index and `p` is the corresponding power, sorted in some order. For example, in the free monoid there is no such choice, but for the free abelian monoid, one could want lex order or have the highest powers first.

Indexed monoid elements are ordered lexicographically with respect to the result of `_sorted_items()` (which for abelian free monoids is influenced by the order on the indexing set).

leading_support()
Return the support of the leading generator of `self`.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (b*a*c^3*a).leading_support()
1

sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (b*c^3*a).leading_support()
0
```

support()

Return a list of the objects indexing `self` with non-zero exponents.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (b*a*c^3*b).support()
[0, 1, 2]

sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (a*c^3).support()
[0, 2]
```

to_word_list()

Return `self` as a word represented as a list whose entries are indices of `self`.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (b*a*c^3*a).to_word_list()
[1, 0, 2, 2, 2, 0]

sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (b*c^3*a).to_word_list()
[0, 1, 2, 2, 2]
```

trailing_support()

Return the support of the trailing generator of `self`.

EXAMPLES:

```
sage: F = FreeMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (b*a*c^3*a).trailing_support()
0

sage: F = FreeAbelianMonoid(index_set=ZZ)
sage: a,b,c,d,e = [F.gen(i) for i in range(5)]
sage: (b*c^3*a).trailing_support()
2
```

CHAPTER

SEVEN

STRING MONOID ELEMENTS

AUTHORS:

- David Kohel <kohel@maths.usyd.edu.au>, 2007-01

Elements of free string monoids, internal representation subject to change.

These are special classes of free monoid elements with distinct printing.

The internal representation of elements does not use the exponential compression of FreeMonoid elements (a feature), and could be packed into words.

class sage.monoids.string_monoid_element.**StringMonoidElement**(*S*, *x*, *check=True*)

Bases: sage.monoids.free_monoid_element.FreeMonoidElement

Element of a free string monoid.

character_count()

Return the count of each unique character.

EXAMPLES:

Count the character frequency in an object comprised of capital letters of the English alphabet:

```
sage: M = AlphabeticStrings().encoding("abcabf")
sage: sorted(M.character_count().items())
[(A, 2), (B, 2), (C, 1), (F, 1)]
```

In an object comprised of binary numbers:

```
sage: M = BinaryStrings().encoding("abcabf")
sage: sorted(M.character_count().items())
[(0, 28), (1, 20)]
```

In an object comprised of octal numbers:

```
sage: A = OctalStrings()
sage: M = A([1, 2, 3, 2, 5, 3])
sage: sorted(M.character_count().items())
[(1, 1), (2, 2), (3, 2), (5, 1)]
```

In an object comprised of hexadecimal numbers:

```
sage: A = HexadecimalStrings()
sage: M = A([1, 2, 4, 6, 2, 4, 15])
sage: sorted(M.character_count().items())
[(1, 1), (2, 2), (4, 2), (6, 1), (f, 1)]
```

In an object comprised of radix-64 characters:

```
sage: A = Radix64Strings()
sage: M = A([1, 2, 63, 45, 45, 10]); M
BC/ttK
sage: sorted(M.character_count().items())
[(B, 1), (C, 1), (K, 1), (t, 2), (/, 1)]
```

TESTS:

Empty strings return no counts of character frequency:

```
sage: M = AlphabeticStrings().encoding("")
sage: M.character_count()
{}
sage: M = BinaryStrings().encoding("")
sage: M.character_count()
{}
sage: A = OctalStrings()
sage: M = A([])
sage: M.character_count()
{}
sage: A = HexadecimalStrings()
sage: M = A([])
sage: M.character_count()
{}
sage: A = Radix64Strings()
sage: M = A([])
sage: M.character_count()
{}
```

coincidence_index(*prec=0*)

Returns the probability of two randomly chosen characters being equal.

decoding(*padic=False*)

The byte string associated to a binary or hexadecimal string monoid element.

EXAMPLES:

```
sage: S = HexadecimalStrings()
sage: s = S.encoding("A..Za..z"); s
412e2e5a612e2e7a
sage: s.decoding()
'A..Za..z'
sage: s = S.encoding("A..Za..z",padic=True); s
14e2e2a516e2e2a7
sage: s.decoding()
'\x14\xe2\xe2\xa5\x16\xe2\xe2\xa7'
sage: s.decoding(padic=True)
'A..Za..z'
sage: S = BinaryStrings()
sage: s = S.encoding("A..Za..z"); s
01000001001011100010111001011010011000010010111000101111001111010
sage: s.decoding()
'A..Za..z'
sage: s = S.encoding("A..Za..z",padic=True); s
10000010011101000111010001011010100001100111010001110100010111110
sage: s.decoding()
'\x82ttZ\x86tt^'
sage: s.decoding(padic=True)
'A..Za..z'
```

frequency_distribution (*length=1*, *prec=0*)

Returns the probability space of character frequencies. The output of this method is different from that of the method `characteristic_frequency()`. One can think of the characteristic frequency probability of an element in an alphabet A as the expected probability of that element occurring. Let S be a string encoded using elements of A. The frequency probability distribution corresponding to S provides us with the frequency probability of each element of A as observed occurring in S. Thus one distribution provides expected probabilities, while the other provides observed probabilities.

INPUT:

- `length` – (default `1`) if `length=1` then consider the probability space of monogram frequency, i.e. probability distribution of single characters. If `length=2` then consider the probability space of digram frequency, i.e. probability distribution of pairs of characters. This method currently supports the generation of probability spaces for monogram frequency (`length=1`) and digram frequency (`length=2`).
- `prec` – (default `0`) a non-negative integer representing the precision (in number of bits) of a floating-point number. The default value `prec=0` means that we use 53 bits to represent the mantissa of a floating-point number. For more information on the precision of floating-point numbers, see the function `RealField()` or refer to the module `real_mpfr`.

EXAMPLES:

Capital letters of the English alphabet:

```
sage: M = AlphabeticStrings().encoding("abcd")
sage: L = M.frequency_distribution().function()
sage: sorted(L.items())

[(A, 0.250000000000000),
(B, 0.250000000000000),
(C, 0.250000000000000),
(D, 0.250000000000000)]
```

The binary number system:

```
sage: M = BinaryStrings().encoding("abcd")
sage: L = M.frequency_distribution().function()
sage: sorted(L.items())
[(0, 0.593750000000000), (1, 0.406250000000000)]
```

The hexadecimal number system:

```
sage: M = HexadecimalStrings().encoding("abcd")
sage: L = M.frequency_distribution().function()
sage: sorted(L.items())

[(1, 0.125000000000000),
(2, 0.125000000000000),
(3, 0.125000000000000),
(4, 0.125000000000000),
(6, 0.500000000000000)]
```

Get the observed frequency probability distribution of digrams in the string "ABCD". This string consists of the following digrams: "AB", "BC", and "CD". Now find out the frequency probability of each of these digrams as they occur in the string "ABCD":

```
sage: M = AlphabeticStrings().encoding("abcd")
sage: D = M.frequency_distribution(length=2).function()
sage: sorted(D.items())
[(AB, 0.333333333333333), (BC, 0.333333333333333), (CD, 0.333333333333333)]
```

sage.monoids.string_monoid_element.**is_AlphabeticStringMonoidElement**(*x*)

sage.monoids.string_monoid_element.**is_BinaryStringMonoidElement**(*x*)

sage.monoids.string_monoid_element.**is_HexadecimalStringMonoidElement**(*x*)

sage.monoids.string_monoid_element.**is_OctalStringMonoidElement**(*x*)

sage.monoids.string_monoid_element.**is_Radix64StringMonoidElement**(*x*)

sage.monoids.string_monoid_element.**is_StringMonoidElement**(*x*)

CHAPTER

EIGHT

FREE STRING MONOIDS

AUTHORS:

- David Kohel <kohel@maths.usyd.edu.au>, 2007-01

Sage supports a wide range of specific free string monoids.

class sage.monoids.string_monoid.**AlphabeticStringMonoid**

Bases: sage.monoids.string_monoid.StringMonoid_class

The free alphabetic string monoid on generators A-Z.

EXAMPLES:

```
sage: S = AlphabeticStrings(); S
Free alphabetic string monoid on A-Z
sage: S.gen(0)
A
sage: S.gen(25)
Z
sage: S([ i for i in range(26) ])
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

characteristic_frequency(*table_name='beker_piper'*)

Return a table of the characteristic frequency probability distribution of the English alphabet. In written English, various letters of the English alphabet occur more frequently than others. For example, the letter "E" appears more often than other vowels such as "A", "I", "O", and "U". In long works of written English such as books, the probability of a letter occurring tends to stabilize around a value. We call this value the characteristic frequency probability of the letter under consideration. When this probability is considered for each letter of the English alphabet, the resulting probabilities for all letters of this alphabet is referred to as the characteristic frequency probability distribution. Various studies report slightly different values for the characteristic frequency probability of an English letter. For instance, [Lew00] reports that "E" has a characteristic frequency probability of 0.12702, while [BekPip82] reports this value as 0.127. The concepts of characteristic frequency probability and characteristic frequency probability distribution can also be applied to non-empty alphabets other than the English alphabet.

The output of this method is different from that of the method frequency_distribution(). One can think of the characteristic frequency probability of an element in an alphabet A as the expected probability of that element occurring. Let S be a string encoded using elements of A. The frequency probability distribution corresponding to S provides us with the frequency probability of each element of A as observed occurring in S. Thus one distribution provides expected probabilities, while the other provides observed probabilities.

INPUT:

- `table_name` – (default `"beker_piper"`) the table of characteristic frequency probability distribution to use. The following tables are supported:
  - `"beker_piper"` – the table of characteristic frequency probability distribution by Beker and Piper [BekPip82]. This is the default table to use.
  - `"lewand"` – the table of characteristic frequency probability distribution by Lewand as described on page 36 of [Lew00].

OUTPUT:

- A table of the characteristic frequency probability distribution of the English alphabet. This is a dictionary of letter/probability pairs.

EXAMPLES:

The characteristic frequency probability distribution table of Beker and Piper [BekPip82]:

```
sage: A = AlphabeticStrings()
sage: table = A.characteristic_frequency(table_name="beker_piper")
sage: sorted(table.items())

[('A', 0.0820000000000000),
('B', 0.0150000000000000),
('C', 0.0280000000000000),
('D', 0.0430000000000000),
('E', 0.127000000000000),
('F', 0.0220000000000000),
('G', 0.0200000000000000),
('H', 0.0610000000000000),
('I', 0.0700000000000000),
('J', 0.00200000000000000),
('K', 0.00800000000000000),
('L', 0.0400000000000000),
('M', 0.0240000000000000),
('N', 0.0670000000000000),
('O', 0.0750000000000000),
('P', 0.0190000000000000),
('Q', 0.00100000000000000),
('R', 0.0600000000000000),
('S', 0.0630000000000000),
('T', 0.0910000000000000),
('U', 0.0280000000000000),
('V', 0.0100000000000000),
('W', 0.0230000000000000),
('X', 0.00100000000000000),
('Y', 0.0200000000000000),
('Z', 0.00100000000000000)]
```

The characteristic frequency probability distribution table of Lewand [Lew00]:

```
sage: table = A.characteristic_frequency(table_name="lewand")
sage: sorted(table.items())

[('A', 0.0816700000000000),
('B', 0.0149200000000000),
('C', 0.0278200000000000),
('D', 0.0425300000000000),
('E', 0.127020000000000),
('F', 0.0222800000000000),
('G', 0.0201500000000000),
('H', 0.0609400000000000),
```

```
('I', 0.0696600000000000),
('J', 0.00153000000000000),
('K', 0.00772000000000000),
('L', 0.0402500000000000),
('M', 0.0240600000000000),
('N', 0.0674900000000000),
('O', 0.0750700000000000),
('P', 0.0192900000000000),
('Q', 0.000950000000000000),
('R', 0.0598700000000000),
('S', 0.0632700000000000),
('T', 0.0905600000000000),
('U', 0.0275800000000000),
('V', 0.00978000000000000),
('W', 0.0236000000000000),
('X', 0.00150000000000000),
('Y', 0.0197400000000000),
('Z', 0.000740000000000000)]
```

Illustrating the difference between `characteristic_frequency()` and `frequency_distribution()`:

```
sage: A = AlphabeticStrings()
sage: M = A.encoding("abcd")
sage: FD = M.frequency_distribution().function()
sage: sorted(FD.items())

[(A, 0.250000000000000),
(B, 0.250000000000000),
(C, 0.250000000000000),
(D, 0.250000000000000)]
sage: CF = A.characteristic_frequency()
sage: sorted(CF.items())

[('A', 0.0820000000000000),
('B', 0.0150000000000000),
('C', 0.0280000000000000),
('D', 0.0430000000000000),
('E', 0.127000000000000),
('F', 0.0220000000000000),
('G', 0.0200000000000000),
('H', 0.0610000000000000),
('I', 0.0700000000000000),
('J', 0.00200000000000000),
('K', 0.00800000000000000),
('L', 0.0400000000000000),
('M', 0.0240000000000000),
('N', 0.0670000000000000),
('O', 0.0750000000000000),
('P', 0.0190000000000000),
('Q', 0.00100000000000000),
('R', 0.0600000000000000),
('S', 0.0630000000000000),
('T', 0.0910000000000000),
('U', 0.0280000000000000),
('V', 0.0100000000000000),
('W', 0.0230000000000000),
('X', 0.00100000000000000),
```

```
('Y', 0.0200000000000000),
('Z', 0.00100000000000000)]
```

TESTS:

The table name must be either "beker_piper" or "lewand":

```
sage: table = A.characteristic_frequency(table_name="")
Traceback (most recent call last):
...
ValueError: Table name must be either 'beker_piper' or 'lewand'.
sage: table = A.characteristic_frequency(table_name="none")
Traceback (most recent call last):
...
ValueError: Table name must be either 'beker_piper' or 'lewand'.
```

REFERENCES:

encoding(*S*)

The encoding of the string `S` in the alphabetic string monoid, obtained by the monoid homomorphism

```
A -> A, ..., Z -> Z, a -> A, ..., z -> Z
```

and stripping away all other characters. It should be noted that this is a non-injective monoid homomorphism.

EXAMPLES:

```
sage: S = AlphabeticStrings()
sage: s = S.encoding("The cat in the hat."); s
THECATINTHEHAT
sage: s.decoding()
'THECATINTHEHAT'
```

sage.monoids.string_monoid.**AlphabeticStrings**()

Returns the string monoid on generators A-Z: $\{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\}$.

OUTPUT:

- Free alphabetic string monoid on A-Z.

EXAMPLES:

```
sage: S = AlphabeticStrings(); S
Free alphabetic string monoid on A-Z
sage: x = S.gens()
sage: x[0]
A
sage: x[25]
Z
```

class sage.monoids.string_monoid.**BinaryStringMonoid**

Bases: sage.monoids.string_monoid.StringMonoid_class

The free binary string monoid on generators $\{0, 1\}$.

encoding(*S*, *padic=False*)

The binary encoding of the string `S`, as a binary string element.

The default is to keep the standard ASCII byte encoding, e.g.

```
A = 65 -> 01000001
B = 66 -> 01000010
```

.
.
.
Z = 90 -> 01001110

rather than a 2-adic representation 65 -> 10000010.

Set `padic=True` to reverse the bit string.

EXAMPLES:

```
sage: S = BinaryStrings()
sage: S.encoding('A')
01000001
sage: S.encoding('A',padic=True)
10000010
sage: S.encoding(' ',padic=True)
00000100
```

sage.monoids.string_monoid.**BinaryStrings**()

Returns the free binary string monoid on generators $\{0, 1\}$.

OUTPUT:

- Free binary string monoid.

EXAMPLES:

```
sage: S = BinaryStrings(); S
Free binary string monoid
sage: u = S('')
sage: u

sage: x = S('0')
sage: x
0
sage: y = S('1')
sage: y
1
sage: z = S('01110')
sage: z
01110
sage: x*y^3*x == z
True
sage: u*x == x*u
True
```

class sage.monoids.string_monoid.**HexadecimalStringMonoid**

Bases: sage.monoids.string_monoid.StringMonoid_class

The free hexadecimal string monoid on generators $\{0, 1, \ldots, 9, a, b, c, d, e, f\}$.

encoding(*S*, *padic=False*)

The encoding of the string `S` as a hexadecimal string element.

The default is to keep the standard right-to-left byte encoding, e.g.

```
A = '\x41' -> 41
B = '\x42' -> 42
.
.
.
Z = '\x5a' -> 5a
```

rather than a left-to-right representation A = 65 -> 14. Although standard (e.g., in the Python constructor 'xhh'), this can be confusing when the string reads left-to-right.

Set `padic=True` to reverse the character encoding.

EXAMPLES:

```
sage: S = HexadecimalStrings()
sage: S.encoding('A')
41
sage: S.encoding('A',padic=True)
14
sage: S.encoding(' ',padic=False)
20
sage: S.encoding(' ',padic=True)
02
```

sage.monoids.string_monoid.**HexadecimalStrings**()

Returns the free hexadecimal string monoid on generators $\{0, 1, \ldots, 9, a, b, c, d, e, f\}$.

OUTPUT:

- Free hexadecimal string monoid.

EXAMPLES:

```
sage: S = HexadecimalStrings(); S
Free hexadecimal string monoid
sage: x = S.gen(0)
sage: y = S.gen(10)
sage: z = S.gen(15)
sage: z
f
sage: x*y^3*z
0aaaf
```

class sage.monoids.string_monoid.**OctalStringMonoid**

Bases: sage.monoids.string_monoid.StringMonoid_class

The free octal string monoid on generators $\{0, 1, \ldots, 7\}$.

sage.monoids.string_monoid.**OctalStrings**()

Returns the free octal string monoid on generators $\{0, 1, \ldots, 7\}$.

OUTPUT:

- Free octal string monoid.

EXAMPLES:

```
sage: S = OctalStrings(); S
Free octal string monoid
sage: x = S.gens()
sage: x[0]
0
sage: x[7]
7
sage: x[0] * x[3]^3 * x[5]^4 * x[6]
033355556
```

class sage.monoids.string_monoid.**Radix64StringMonoid**

Bases: sage.monoids.string_monoid.StringMonoid_class

The free radix 64 string monoid on 64 generators.

sage.monoids.string_monoid.**Radix64Strings**()

Returns the free radix 64 string monoid on 64 generators

```
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,
0,1,2,3,4,5,6,7,8,9,+,/
```

OUTPUT:

- Free radix 64 string monoid.

EXAMPLES:

```
sage: S = Radix64Strings(); S
Free radix 64 string monoid
sage: x = S.gens()
sage: x[0]
A
sage: x[62]
+
sage: x[63]
/
```

class sage.monoids.string_monoid.**StringMonoid_class**(*n*, *alphabet=()*)

Bases: sage.monoids.free_monoid.FreeMonoid_class

A free string monoid on n generators.

alphabet()

x.__init__(...) initializes x; see help(type(x)) for signature

gen(*i=0*)

The i-th generator of the monoid.

INPUT:

- i – integer (default: 0)

EXAMPLES:

```
sage: S = BinaryStrings()
sage: S.gen(0)
0
sage: S.gen(1)
1
sage: S.gen(2)
Traceback (most recent call last):
...
IndexError: Argument i (= 2) must be between 0 and 1.
sage: S = HexadecimalStrings()
sage: S.gen(0)
0
sage: S.gen(12)
c
sage: S.gen(16)
Traceback (most recent call last):
...
IndexError: Argument i (= 16) must be between 0 and 15.
```

CHAPTER

NINE

INDICES AND TABLES

- Index
- Module Index
- Search Page

BIBLIOGRAPHY

[BekPip82] H. Beker and F. Piper. *Cipher Systems: The Protection of Communications*. John Wiley and Sons, 1982.

[Lew00] Robert Edward Lewand. *Cryptological Mathematics*. The Mathematical Association of America, 2000.

PYTHON MODULE INDEX

m

sage.monoids.free_abelian_monoid, 11
sage.monoids.free_abelian_monoid_element, 15
sage.monoids.free_monoid, 5
sage.monoids.free_monoid_element, 9
sage.monoids.indexed_free_monoid, 17
sage.monoids.monoid, 3
sage.monoids.string_monoid, 27
sage.monoids.string_monoid_element, 23

INDEX

A

alphabet() (sage.monoids.string_monoid.StringMonoid_class method), 33
AlphabeticStringMonoid (class in sage.monoids.string_monoid), 27
AlphabeticStrings() (in module sage.monoids.string_monoid), 30

B

BinaryStringMonoid (class in sage.monoids.string_monoid), 30
BinaryStrings() (in module sage.monoids.string_monoid), 31

C

cardinality() (sage.monoids.free_abelian_monoid.FreeAbelianMonoid_class method), 12
cardinality() (sage.monoids.free_monoid.FreeMonoid_class method), 6
cardinality() (sage.monoids.indexed_free_monoid.IndexedMonoid method), 19
character_count() (sage.monoids.string_monoid_element.StringMonoidElement method), 23
characteristic_frequency() (sage.monoids.string_monoid.AlphabeticStringMonoid method), 27
coincidence_index() (sage.monoids.string_monoid_element.StringMonoidElement method), 24
create_key() (sage.monoids.free_abelian_monoid.FreeAbelianMonoidFactory method), 12
create_key() (sage.monoids.free_monoid.FreeMonoidFactory method), 6
create_object() (sage.monoids.free_abelian_monoid.FreeAbelianMonoidFactory method), 12
create_object() (sage.monoids.free_monoid.FreeMonoidFactory method), 6

D

decoding() (sage.monoids.string_monoid_element.StringMonoidElement method), 24
dict() (sage.monoids.indexed_free_monoid.IndexedFreeAbelianMonoidElement method), 18

E

Element (sage.monoids.free_monoid.FreeMonoid_class attribute), 6
Element (sage.monoids.indexed_free_monoid.IndexedFreeAbelianMonoid attribute), 17
Element (sage.monoids.indexed_free_monoid.IndexedFreeMonoid attribute), 18
encoding() (sage.monoids.string_monoid.AlphabeticStringMonoid method), 30
encoding() (sage.monoids.string_monoid.BinaryStringMonoid method), 30
encoding() (sage.monoids.string_monoid.HexadecimalStringMonoid method), 31

F

FreeAbelianMonoid() (in module sage.monoids.free_abelian_monoid), 11
FreeAbelianMonoid_class (class in sage.monoids.free_abelian_monoid), 12

FreeAbelianMonoidElement (class in sage.monoids.free_abelian_monoid_element), 15
FreeAbelianMonoidFactory (class in sage.monoids.free_abelian_monoid), 12
FreeMonoid() (in module sage.monoids.free_monoid), 5
FreeMonoid_class (class in sage.monoids.free_monoid), 6
FreeMonoidElement (class in sage.monoids.free_monoid_element), 9
FreeMonoidFactory (class in sage.monoids.free_monoid), 5
frequency_distribution() (sage.monoids.string_monoid_element.StringMonoidElement method), 24

G

gen() (sage.monoids.free_abelian_monoid.FreeAbelianMonoid_class method), 12
gen() (sage.monoids.free_monoid.FreeMonoid_class method), 6
gen() (sage.monoids.indexed_free_monoid.IndexedFreeAbelianMonoid method), 17
gen() (sage.monoids.indexed_free_monoid.IndexedFreeMonoid method), 18
gen() (sage.monoids.string_monoid.StringMonoid_class method), 33
gens() (sage.monoids.indexed_free_monoid.IndexedMonoid method), 20
gens() (sage.monoids.monoid.Monoid_class method), 3

H

HexadecimalStringMonoid (class in sage.monoids.string_monoid), 31
HexadecimalStrings() (in module sage.monoids.string_monoid), 32

I

IndexedFreeAbelianMonoid (class in sage.monoids.indexed_free_monoid), 17
IndexedFreeAbelianMonoidElement (class in sage.monoids.indexed_free_monoid), 18
IndexedFreeMonoid (class in sage.monoids.indexed_free_monoid), 18
IndexedFreeMonoidElement (class in sage.monoids.indexed_free_monoid), 19
IndexedMonoid (class in sage.monoids.indexed_free_monoid), 19
IndexedMonoidElement (class in sage.monoids.indexed_free_monoid), 20
is_AlphabeticStringMonoidElement() (in module sage.monoids.string_monoid_element), 25
is_BinaryStringMonoidElement() (in module sage.monoids.string_monoid_element), 26
is_FreeAbelianMonoid() (in module sage.monoids.free_abelian_monoid), 13
is_FreeAbelianMonoidElement() (in module sage.monoids.free_abelian_monoid_element), 16
is_FreeMonoid() (in module sage.monoids.free_monoid), 7
is_FreeMonoidElement() (in module sage.monoids.free_monoid_element), 9
is_HexadecimalStringMonoidElement() (in module sage.monoids.string_monoid_element), 26
is_Monoid() (in module sage.monoids.monoid), 3
is_OctalStringMonoidElement() (in module sage.monoids.string_monoid_element), 26
is_Radix64StringMonoidElement() (in module sage.monoids.string_monoid_element), 26
is_StringMonoidElement() (in module sage.monoids.string_monoid_element), 26

L

leading_support() (sage.monoids.indexed_free_monoid.IndexedMonoidElement method), 20
length() (sage.monoids.indexed_free_monoid.IndexedFreeAbelianMonoidElement method), 18
length() (sage.monoids.indexed_free_monoid.IndexedFreeMonoidElement method), 19
list() (sage.monoids.free_abelian_monoid_element.FreeAbelianMonoidElement method), 15

M

Monoid_class (class in sage.monoids.monoid), 3
monoid_generators() (sage.monoids.indexed_free_monoid.IndexedMonoid method), 20

N

ngens() (sage.monoids.free_abelian_monoid.FreeAbelianMonoid_class method), 13
ngens() (sage.monoids.free_monoid.FreeMonoid_class method), 6

O

OctalStringMonoid (class in sage.monoids.string_monoid), 32
OctalStrings() (in module sage.monoids.string_monoid), 32
one() (sage.monoids.indexed_free_monoid.IndexedFreeAbelianMonoid method), 17
one() (sage.monoids.indexed_free_monoid.IndexedFreeMonoid method), 19
one_element() (sage.monoids.free_monoid.FreeMonoid_class method), 7

R

Radix64StringMonoid (class in sage.monoids.string_monoid), 32
Radix64Strings() (in module sage.monoids.string_monoid), 33

S

sage.monoids.free_abelian_monoid (module), 11
sage.monoids.free_abelian_monoid_element (module), 15
sage.monoids.free_monoid (module), 5
sage.monoids.free_monoid_element (module), 9
sage.monoids.indexed_free_monoid (module), 17
sage.monoids.monoid (module), 3
sage.monoids.string_monoid (module), 27
sage.monoids.string_monoid_element (module), 23
StringMonoid_class (class in sage.monoids.string_monoid), 33
StringMonoidElement (class in sage.monoids.string_monoid_element), 23
support() (sage.monoids.indexed_free_monoid.IndexedMonoidElement method), 20

T

to_word() (sage.monoids.free_monoid_element.FreeMonoidElement method), 9
to_word_list() (sage.monoids.indexed_free_monoid.IndexedMonoidElement method), 21
trailing_support() (sage.monoids.indexed_free_monoid.IndexedMonoidElement method), 21