

## Grand Tour - part 1

# The Sage Library -- a Grand Tour -- Part 1

June 2009

William Stein

```
def ls(module): #auto
    os.system('ls $SAGE_ROOT/develop/sage/sage/%s'%module)
ls('')
```

<code>__init__.py</code>	<code>coding</code>	<code>geometry</code>	<code>logic</code>	<code>par</code>
<code>__init__.pyc</code>	<code>combinat</code>	<code>graphs</code>	<code>matrix</code>	<code>plc</code>
<code>algebras</code>	<code>crypto</code>	<code>groups</code>	<code>media</code>	<code>prc</code>
<code>all.py</code>	<code>databases</code>	<code>gsl</code>	<code>misc</code>	<code>qua</code>
<code>all_cmddline.py</code>	<code>ext</code>	<code>homology</code>	<code>modular</code>	<code>rin</code>
<code>all_notebook.py</code>	<code>finance</code>	<code>interfaces</code>	<code>modules</code>	<code>sch</code>
<code>calculus</code>	<code>functions</code>	<code>lfunctions</code>	<code>monoids</code>	<code>ser</code>
<code>categories</code>	<code>games</code>	<code>libs</code>	<code>numerical</code>	<code>set</code>

The Sage library consists of well over **350,000 lines** of new code written by Sage developers. It provides a unified interface to most existing mathematical software and fills in the many gaps in functionality of open source mathematical software.

**Major Documentation Milestone:** As of Sage-4.0.1, **77.3% of functions** in the library have docstrings that include doctests illustrating their functionality. This includes private "single underscore" methods, and represents over **94,752 lines** of input, which gets regularly tested.

The Sage library is divided up into the following **39 major modules**, and in this talk we will give a tour of the following subset of them:

### Part 1

- categories
- algebras
- monoids

- rings
- structure
- sets
- combinat
- graphs
- crypto

## Categories

A *category* is a collection of objects and morphisms, which satisfy certain axioms. They are a useful organizing principle in Sage.

The Sage categories package defines many categories: *Elements, Sequences, Objects, Sets, GSets, PointedSets, SetsWithPartialMaps, SimplicialComplexes, Semigroups, Monoids, Groupoid, Groups, AbelianSemigroups, AbelianMonoids, AbelianGroups, Rings, CommutativeRings, Fields, FiniteFields, NumberFields, Algebras, CommutativeAlgebras, MonoidAlgebras, GroupAlgebras, MatrixAlgebras, RingModules, Modules, FreeModules, VectorSpaces, HeckeModules, RingIdeals, Ideals, CommutativeRingIdeals, AlgebraModules, AlgebraIdeals, CommutativeAlgebraIdeals, ChainComplexes, Schemes, ModularAbelianVarieties*

The categories code was original written by David Kohel and William Stein. Robert Bradshaw and others actually made some limited use of it in improving the Sage coercion model.

PROJECT: An exciting recent development is that French mathematician Nicolas Thiery (director of the Sage-combinat project) has put a massive amount of work into greatly improving and *using* the categories code to usefully organize many generic algorithms. His code will soon be included in Sage.

```
ls('categories')
```

<code>__init__.py</code>	<code>category_types.py</code>	<code>map.pxd</code>
<code>action.c</code>	<code>functor.c</code>	<code>map.pyx</code>
<code>action.pxd</code>	<code>functor.pxd</code>	<code>morphism.c</code>
<code>action.pyx</code>	<code>functor.pyx</code>	<code>morphism.pxd</code>
<code>all.py</code>	<code>homset.py</code>	<code>morphism.pyx</code>
<code>category.py</code>	<code>map.c</code>	<code>pushout.py</code>

```
C = VectorSpaces(RR); C
    Category of vector spaces over Real Field with 53 bits of precision
```

```
C(QQ^3)
    Vector space of dimension 3 over Real Field with 53 bits of precision
```

```
(QQ^3).category()
    Category of vector spaces over Rational Field
```

```
Integers(7).category()
Category of rings

Primes().category()
Category of sets

sage: Hom(ZZ^3, ZZ^5)
Set of Morphisms from Ambient free module of rank 3 over the
principal ideal domain Integer Ring to Ambient free module of rank
over the principal ideal domain Integer Ring in Category of free
modules over Integer Ring
```

## Algebras

This module contains code mainly for noncommutative algebras, include free algebras, quotients of free algebras by relations, group algebras, quaternion algebras, and the Steenrod algebra.

The **free algebras** functionality was implemented by David Kohel in 2005, and has been hardly touched since. It could use a major overhaul.

David Loeffler recently wrote the **group algebras** functionality, which allows one to compute in a natural ring associated to a group.

Jon Bober, William Stein (me), and a few other people wrote the **quaternion algebras** functionality from scratch in February and March of 2009. It's highly optimized code, and there are some natural ways to extend it to more general cases.

The **Steenrod algebra** code is a relatively recent major new package by John Palmieri that provides powerful tools for the study of algebraic topology.

PROJECT: It would be good if the algebras directory had much better or comprehensive support for a wider range of noncommutative algebras. For example, Singular -- a component of Sage -- has powerful specialized code ("plural") for working with certain classes of noncommutative rings. This should all be made nicely available in Sage.

```
ls('algebras')
__init__.py          quatalg
algebra.py           quaternion_algebra.py
algebra_element.py  quaternion_algebra_element.py
all.py               steenrod_algebra.py
free_algebra.py      steenrod_algebra_bases.py
free_algebra_element.py steenrod_algebra_element.py
free_algebra_quotient.py steenrod_milnor_multiplicat
```

```
free_algebra_quotient_element.py  steenrod_milnor_multiplicat
py
group_algebra.py

K.<x,y,z> = FreeAlgebra(QQ,3); K
Free Algebra on 3 generators (x, y, z) over Rational Field
z^2*x*y*x*z + (2/3)*z*y*y
2/3*z*y^2 + z^2*x*y*x*z
```

### A group ring

```
S = SymmetricGroup(3); R=GroupAlgebra(S, QQ)

R
Group algebra of group "SymmetricGroup(3)" over base ring Rational
Field

S.gens()
[(1,2,3), (1,2)]

a = R(S.0) + (2/3)*R(S.1); a
2/3*(1,2) + (1,2,3)

a*a
4/9*( ) + 2/3*(2,3) + (1,3,2) + 2/3*(1,3)
```

### A quaternion algebra

```
A.<i,j,k> = QuaternionAlgebra(93938,39292902020); A
Quaternion Algebra (93938, 39292902020) with base ring Rational
Field

i^2
93938

j^2
39292902020

(1+i+j)*(j-2*i)
39292714144 - 2*i + j + 3*k

A.ramified_primes()
[5, 13, 3613, 1964645101]
```

```
i*j
k
a = 1+9393*i+392903*j-393923298304823904*k
a*a
-572768004828494291666114999674938501059466982679817 + 18786*i +
785806*j - 787846596609647808*k
timeit('a*a')
625 loops, best of 3: 3.41 µs per loop
```

## Monoids

A monoid is just like a group, but *without* the axiom that every element has an inverse. Sage has code that David Kohel wrote several years ago for working with monoids.

PROJECT: This code has not been touched in years, so it would likely be easy to improve, optimize, and extend it.

```
ls('monoids')
__init__.py          free_monoid_element.py
all.py              monoid.py
free_abelian_monoid.py string_monoid.py
free_abelian_monoid_element.py string_monoid_element.py
free_monoid.py      string_ops.py
S = HexadecimalStrings(); S
Free hexadecimal string monoid
x = S.gen(0); y = S.gen(10); z = S.gen(15)
x*y^3*z
0aaaf
```

## Rings

The huge rings module implements much of the underlying arithmetic in Sage. It is most complicated and mature module in Sage.

**Algebraic rings:** All of the standard rings, such as  $\mathbf{Z}$ ,  $\mathbf{Q}$ , finite fields  $\mathbf{F}_{p^n}$ , and polynomial and power series rings over any other ring in Sage. Substantial code for number fields, and three models of  $p$ -adic numbers: capped relative, capped absolute, fixed modulus. The algebraic closure of  $\mathbf{Q}$  and its maximal totally real subfield are also implemented, using intervals.

**Numerical:** Real and complex numbers of any fixed precision. Double precision reals and complex (for speed). Rings that model  $\mathbf{R}$  and  $\mathbf{C}$  with intervals (interval arithmetic).

At any given time, some parts of the code are very mature, and others are being actively improved.

PROJECT: The  $p$ -adics and numbers fields are currently actively being improved by David Roe, John Cremona, Kiran Kedlaya, and others.

PROJECT: Function fields are not sufficiently well supported yet.

PROJECT: Substantial work needs to be put into optimizing arithmetic with relative numbers fields. (Nick Alexander has worked some on this recently).

PROJECT: Groebner basis functionality is in the rings/polynomial directory, and currently mainly uses Singular. There is a huge need in Sage for an optimized F4 implementation, and we hope that this will eventually get developed by somebody (the EU has just funded Singular, which is part of Sage, to do this).

```
ls('rings')
__init__.py          integral_domain_element.py
all.py              laurent_series_ring.py
arith.py           laurent_series_ring_element
bernm             laurent_series_ring_element
bernm.cpp         laurent_series_ring_element
bernm.pyx        memory.c
bernoulli_mod_p.cpp memory.pyx
bernoulli_mod_p.pyx misc.py
big_oh.py         monomials.py
coerce_python.py morphism.c
commutative_algebra.py morphism.pxd
commutative_algebra_element.py morphism.pyx
commutative_ring.py mpfi.pxi
commutative_ring_element.py noetherian_ring.py
complex_double.c  notes
complex_double.h  number_field
complex_double.pxd padics
complex_double.pyx pari_ring.py
complex_double_api.h polynomial
```

```

complex_field.py
complex_interval.c
complex_interval.pxd
complex_interval.pyx
complex_interval_field.py
complex_number.c
complex_number.pxd
complex_number.pyx
contfrac.py
dedekind_domain.py
dedekind_domain_element.py
euclidean_domain.py
euclidean_domain_element.py
fast_arith.c
fast_arith.pxd
fast_arith.pyx
field.py
field_element.py
finite_field.py
finite_field_element.py
finite_field_ext_pari.py
finite_field_givaro.cpp
finite_field_givaro.pxd
finite_field_givaro.pyx
finite_field_morphism.py
finite_field_ntl_gf2e.cpp
finite_field_ntl_gf2e.pxd
finite_field_ntl_gf2e.pyx
finite_field_prime_modn.py
fraction_field.py
fraction_field_element.c
fraction_field_element.pyx
homset.py
ideal.py
ideal_monoid.py
infinity.py
integer.c
integer.h
integer.pxd
integer.pyx
integer_mod.c
integer_mod.pxd
integer_mod.pyx
integer_mod_ring.py
integer_ring.c
integer_ring.pxd
integer_ring.pyx
integer_ring_python.py
integral_domain.py

```

```
ls('rings/number_field')
```

```
__init__.py
```

```

power_series_mpoly.c
power_series_mpoly.pxd
power_series_mpoly.pyx
power_series_poly.c
power_series_poly.pxd
power_series_poly.pyx
power_series_ring.py
power_series_ring_element.c
power_series_ring_element.py
power_series_ring_element.pyx
principal_ideal_domain.py
principal_ideal_domain_element.py
qgbar.py
quotient_ring.py
quotient_ring_element.py
rational.c
rational.h
rational.pxd
rational.pxi
rational.pyx
rational_field.py
real_double.c
real_double.pxd
real_double.pxi
real_double.pyx
real_interval_field.py
real_lazy.c
real_lazy.pxd
real_lazy.pyx
real_mphi.c
real_mphi.pxd
real_mphi.pyx
real_mpfr.c
real_mpfr.pxd
real_mpfr.pyx
real_rqdf.cpp
real_rqdf.pxd
real_rqdf.pyx
residue_field.c
residue_field.pxd
residue_field.pyx
ring.c
ring.pxd
ring.pyx
ring_element.py
rqdf_fix.h
solaris_fix.h
stdint.h
tests.py

```

```
number_field_ideal.py
```

```

all.py
class_group.py
galois_group.py
maps.py
morphism.py
number_field.py
number_field_base.c
number_field_base.pxd
number_field_base.pyx
number_field_element.cpp
number_field_element.pxd
number_field_element.pyx
number_field_element_quadratic.cpp
number_field_element_quadratic.pxd
number_field_element_quadratic.pyx

```

```
ls('rings/polynomial')
```

```

__init__.py
all.py
complex_roots.py
convolution.py
cyclotomic.c
cyclotomic.pyx
groebner_fan.py
infinite_polynomial_element.py
infinite_polynomial_ring.py
laurent_polynomial.c
laurent_polynomial.pxd
laurent_polynomial.pyx
laurent_polynomial_ring.py
multi_polynomial.c
multi_polynomial.pxd
multi_polynomial.pyx
multi_polynomial_element.py
multi_polynomial_ideal.py
multi_polynomial_ideal_libsingular.cpp
x
multi_polynomial_ideal_libsingular.pyx
multi_polynomial_libsingular.cpp
multi_polynomial_libsingular.pxd
multi_polynomial_libsingular.pyx
multi_polynomial_ring.py
multi_polynomial_ring_generic.c
multi_polynomial_ring_generic.pxd
multi_polynomial_ring_generic.pyx
padics
pbori.cpp
pbori.pxd
pbori.pyx
polydict.c
polydict.pxd
polydict.pyx

```

```

number_field_ideal_rel.py
number_field_morphisms.c
number_field_morphisms.pyx
number_field_rel.py
order.py
small_primes_of_degree_one.
totallyreal.c
totallyreal.pyx
totallyreal_data.c
totallyreal_data.pxd
totallyreal_data.pyx
totallyreal_dsage.py
totallyreal_phc.py
totallyreal_rel.py
unit_group.py

```

```

polynomial_element.pyx
polynomial_element_generic.
polynomial_fateman.py
polynomial_gf2x.cpp
polynomial_gf2x.pxd
polynomial_gf2x.pyx
polynomial_integer_dense_fl
polynomial_integer_dense_fl
polynomial_integer_dense_fl
polynomial_integer_dense_nt
polynomial_integer_dense_nt
polynomial_integer_dense_nt
polynomial_integer_dense_nt.c
polynomial_modn_dense_ntl.c
polynomial_modn_dense_ntl.py
polynomial_modn_dense_ntl.pyx
polynomial_quotient_ring.py
polynomial_quotient_ring_el
polynomial_real_mpfr_dense.
polynomial_real_mpfr_dense.
polynomial_ring.py
polynomial_ring_constructor
polynomial_singular_interfa
polynomial_template.pxi
polynomial_template_header.
polynomial_zmod_flint.c
polynomial_zmod_flint.pxd
polynomial_zmod_flint.pyx
real_roots.c
real_roots.pxd
real_roots.pyx
symmetric_ideal.py
symmetric_reduction.c
symmetric_reduction.pxd
symmetric_reduction.pyx

```

```

polynomial_compiled.c
polynomial_compiled.pxd
polynomial_compiled.pyx
polynomial_element.c
polynomial_element.pxd

```

```

term_order.py
toy_buchberger.py
toy_d_basis.py
toy_variety.py

```

```
parent(2/3)
```

```
Rational Field
```

```
var('x')
```

```
K.<a,b> = NumberField([x^3+x+1, x^4+13*x^2-2]); K
```

```
Number Field in a with defining polynomial x^3 + x + 1 over its base field
```

```
RDF
```

```
Real Double Field
```

```
@interact
```

```
def _(number=(2..20)):
    html('<h2>%s Random Rings</h2>' % number)
    i=0
    for R in sage.rings.tests.random_rings(1):
        print R
        #show(R)
        i += 1
        if i > number: break
```

```
number
```

### 13 Random Rings

```

Multivariate Polynomial Ring in x0, x1, x2, x3, x4 over Finite Field
of size 39277416837830680409
Finite Field of size 8976059674155500579
Finite Field of size 72808295338992658777
Multivariate Polynomial Ring in x0, x1, x2, x3, x4, x5 over Number
Field in a with defining polynomial x^2 + 42418
Finite Field of size 64019
Finite Field in a of size 839651^8
Rational Field
Integer Ring
Univariate Polynomial Ring in x over Rational Field
Finite Field in a of size 347003^2
Finite Field in a of size 989231^16
Number Field in a with defining polynomial x^2 + 87142
Multivariate Polynomial Ring in x0, x1, x2 over Integer Ring
Number Field in a with defining polynomial x^5 - 42*x^4 - 95*x^3 -
x^2 + 94*x + 83

```

Working in  $\overline{\mathbb{Q}}$ .

```
R.<X> = QQbar[]; R
```

```
Univariate Polynomial Ring in X over Algebraic Field
```

```
f = X^3 + 2*X - 17; r = f.roots(); r
```

```
[(2.312973501144772?, 1), (-1.156486750572386? -
2.452016478890065?*I, 1), (-1.156486750572386? +
2.452016478890065?*I, 1)]
```

```
g = X^2 + 13; s = g.roots(); s
```

```
[(-3.605551275463990?*I, 1), (3.605551275463990?*I, 1)]
```

```
t = r[0][0] + s[0][0]; t
```

```
2.312973501144772? - 3.605551275463990?*I
```

```
t.minpoly()
```

```
x^6 + 43*x^4 - 34*x^3 + 511*x^2 + 1258*x + 1862
```

## Structure

The structure module

- defines the **SageObject** base class for most objects in Sage
- provides the base classes for all **elements** and **parent** structures
- implements much of the **coercion model**
- formal sums, sequences, and factorizations
- the **proof** object, which allows you to determine whether or not certain classes of algorithms are only required to succeed with high probability or can assume conjectures (such as GRH),
- .. and much more.

After having undergone at least 3 major and painful revisions, the structure module is *fairly mature* at this point. It will gradually improve as the coercion model in Sage is slightly tweaked, as categories are improved, etc.

```
ls('structure')
```

```

__init__.py
all.py
category_object.c
category_object.pxd
category_object.pyx
coerce.c
coerce.pxd

```

```

generators.pyx
gens_py.py
mutability.c
mutability.pxd
mutability.pyx
mutability_py.py
nonexact.py

```

```

coerce.pxi          parent.c
coerce.pyx          parent.pxd
coerce_actions.c   parent.pyx
coerce_actions.pxd parent_base.c
coerce_actions.pyx parent_base.pxd
coerce_dict.c      parent_base.pyx
coerce_dict.pxd   parent_gens.c
coerce_dict.pyx   parent_gens.pxd
coerce_maps.c     parent_gens.pyx
coerce_maps.pxd   parent_old.c
coerce_maps.pyx   parent_old.pxd
element.c         parent_old.pyx
element.pxd       proof
element.pyx       sage_object.c
element_py.py     sage_object.pxd
element_verify.py sage_object.pyx
factorization.py  sequence.py
factory.c         test.sobj
factory.pyx       unique_representation.py
formal_sum.py     wrapper_parent.c
generators.c      wrapper_parent.pxd
generators.pxd    wrapper_parent.pyx

```

```

sage.structure.element.Element
<type 'sage.structure.element.Element'>
sage.structure.parent.Parent
<type 'sage.structure.parent.Parent'>
Sequence([2/3, 3, 8]).universe()
Rational Field
FormalSum([(3, Mod(5,3)), (-2, 'hello')])
-2*hello + 3*2
proof.all(False)
proof.all()
{'polynomial': False, 'other': False, 'elliptic_curve': False,
'number_field': False, 'linear_algebra': False, 'arithmetic': False}
proof.all(True); proof.all()
{'polynomial': True, 'other': True, 'elliptic_curve': True,
'number_field': True, 'linear_algebra': True, 'arithmetic': True}

```

**A coercion example:**

Notice that the sum below is in the parent of neither the left or right hand ring. These natural nontrivial coercions are done systematically throughout Sage. Getting this right was very hard work of Robert Bradshaw, David Roe, Craig Citro, etc.

```

R.<x> = ZZ[]
R

```

```

Univariate Polynomial Ring in x over Integer Ring
1/2 + (x+4)
x + 9/2
parent(1/2), parent(x+4)
(Rational Field, Univariate Polynomial Ring in x over Integer Ring)
parent((1/2) + (x+4))
Univariate Polynomial Ring in x over Rational Field
sage.structure.element.get_coercion_model().explain(1/2, x+4)
Action discovered.
Left scalar multiplication by Rational Field on Univariate
Polynomial Ring in x over Integer Ring
Result lives in Univariate Polynomial Ring in x over Rational Field
Univariate Polynomial Ring in x over Rational Field

```

Surprisingly, even Magma's coercion model is way behind Sage's:

```

% magma
R<x> := PolynomialRing(IntegerRing());
1/2 + (x+4)
Traceback (click to the left for traceback)
...
Argument types given: FldRatElt, RngUPolElt[RngInt]

```

**Sets**

The sets module implements basic notions of sets of Python objects, and operations on them. It's similar to what's in the standard Python library (the `set` class) but with more mathematical semantics, and support for infinite sets.

*Example:* `primes.py` defines the set of prime numbers, as a sort of example of how to define an infinite set.

PROJECT: Add more examples of interesting sets to this module.

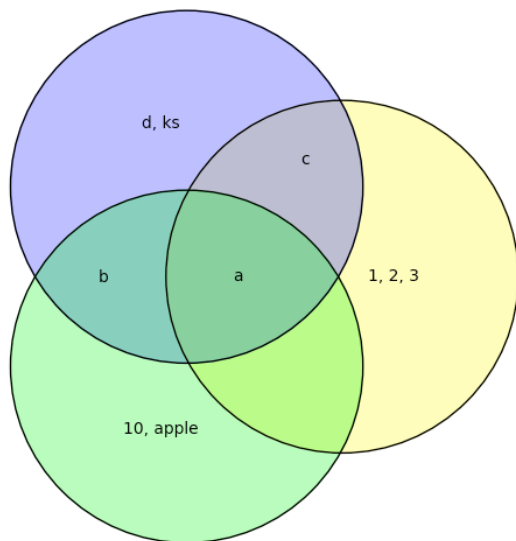
```

ls('sets')

```

	<code>__init__.py</code>	<code>all.py</code>	<code>family.py</code>	<code>primes.py</code>	<code>set</code>
X	1,2,3,a				
Y	2,a,3,4,apple				
Z	a,b,10,apple				

$$\begin{aligned}
 X \cap Y &= \{a, c\} \\
 X \cap Z &= \{a\} \\
 Y \cap Z &= \{a, b\} \\
 X \cap Y \cap Z &= \{a\}
 \end{aligned}$$



```
X = Set([1,2,5, 'orange']); X
```

```
{'orange', 1, 2, 5}
```

```
Y = Set([2, 'apple']); Y
```

```
{2, 'apple'}
```

```
X.union(Y)
```

```
{'orange', 1, 2, 5, 'apple'}
```

```
X.intersection(Y)
```

```
{2}
```

```
set(X)
```

```
set(['orange', 1, 2, 5])
```

```
Primes()
```

```
Set of all prime numbers: 2, 3, 5, 7, ...
```

This seems dumb, since clearly the result should just be a finite set with all elements enumerated.

```
Z = X.intersection(Primes()); Z
```

```
Set-theoretic intersection of {'orange', 1, 2, 5} and Set of all prime numbers: 2, 3, 5, 7, ...
```

```
2 in Z
```

```
True
```

```
3 in Z
```

```
False
```

## Combinat

The **combinat** directory is well over 50,000 lines of code that implements a wide range of algebraic combinatorics functionality in Sage. It is under very active development.

- Started in 2007, when **Mike Hansen** (then an undergrad) decided to single-handedly port as much as possible of the MuPAD-combinat project (a major effort of nearly a dozen mathematicians) over to Sage. He got about halfway through.
- In 2008, the MuPAD combinat group decided to **switch to Sage**... just in time as it turns out, right before MuPAD was bought out by MATLAB.
- They have been working on porting all of MuPAD-combinat to Sage during the last year, and greatly cleaning up and improved much of the design and code. **(Doing**

**things right.)**

- Major **new functionality**: ode for combinatorics of words, representation theory of Lie groups, etc.

PROJECT: Completely finish the port of MuPAD-combinat.

PROJECT: Dan Bump has been adding substantial new functionality for root systems.

```
ls('combinat')
__init__.py          partition_algebra.py
all.py               partitions.cpp
alternating_sign_matrix.py  partitions.pyx
backtrack.py         partitions_c.cc
cartesian_product.py partitions_c.h
choose_nk.py         permutation.py
combinat.py          permutation_nk.py
combination.py       posets
combinatorial_algebra.py  q_analogues.py
composition.py        ranker.py
composition_signed.py  restricted_growth.py
crystals             ribbon.py
designs              ribbon_tableau.py
dlx.py              root_system
dyck_word.py        schubert_polynomial.py
expnums.c           set_partition.py
expnums.pyx         set_partition_ordered.py
family.py           sf
finite_class.py     skew_partition.py
free_module.py      skew_tableau.py
generator.py        sloane_functions.py
graph_path.py       species
integer_list.py     split_nk.py
integer_vector.py   subset.py
integer_vector_weighted.py  subword.py
lyndon_word.py      symmetric_group_algebra.py
matrices           tableau.py
misc.py             tools.py
multichoose_nk.py   tuple.py
necklace.py         words
output.py           yamanouchi.py
partition.py
```

**Cartesian Products**

```
S = CartesianProduct([1,2,3],['x','y']); S
Cartesian product of [1, 2, 3], ['x', 'y']
S.list()
[[1, 'x'], [1, 'y'], [2, 'x'], [2, 'y'], [3, 'x'], [3, 'y']]
```

**Root Systems**

```
S = RootSystem(['B',3]); S
Root system of type ['B', 3]
S.root_lattice()
Root lattice of the Root system of type ['B', 3]
```

**Partitions of an integer:**

```
P = Partitions(5); P
Partitions of the integer 5
P.cardinality()
7
P.list()
[[5], [4, 1], [3, 2], [3, 1, 1], [2, 2, 1], [2, 1, 1, 1], [1, 1, 1, 1, 1]]
time Partitions(10^6).cardinality()
1471684986358223398631004760609895943484030484439142125334612747351
6611741891861827633014887398359755584201537413060028809592938734712
2322703278495780019327843960720642286590487130201709718407610256764
9860846908142829356706929785991290519899445490672219997823452874982
7402228822985013676756629478188749468787900382469998819772920063206
6687359966622738167982662134824172084466310274280019181321981771806
6511234542595026728424452592296781193448139994664730105742564359154
9498918148528535137055139947671998169145902201559910195960141747407
7154307500221848958152093390124817344694483193232801506653840429940
4179587751761294916248142479998802936507195257074485047571662771763
0339144249511382329819526300833648982604583771220245530499638214460
0285318320045190465919683027875374181184860006120168525935427419802
5046267245473237321845833427512524227465399130174076941280847400831
4221799928607110833630331629828910244464969680539541679187548001085
6367740220231284676469197750223485625207477418433436578015341307047
1975530375169707999287040285677841619347472368171772154046664303121
15630003467104673818
Time: CPU 0.04 s, Wall: 0.08 s
```

Dyck words are the expressions containing  $n$  pairs of correctly matched parentheses.

```
@interact
```

```
def _(n=(1..6)):
    for w in DyckWords(n):
        print w
```

```
n
()
```

## Graphs

The Sage graphs module implements a huge amount of functionality for working with graphs (mostly by Robert Miller, Emily Kirkman, and Jason Grout, and built partly on NetworkX).

- Complete new implementation of computation of **automorphism groups of graphs**, determining isomorphism between graphs, and canonical labelings.
- Extensive code for **plotting** graphs.
- Queryable **relational database** of all graphs on at most 7 vertices.
- New **highly optimized data structure** for implementing algorithms that manipulate
- **Enumerating** certain types of graphs efficiently (e.g., trees).

PROJECT: Robert Miller's vision for **graphs** is to have **graphs/base** completely production level (under active development now!), so that if someone wants to write some functions in Cython, then this is the clear graph theory library to use. We need a couple submodules:

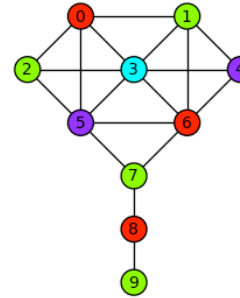
- **sage/graphs/coloring/**
- **sage/graphs/structured\_graphs/** (or something similar, for bipartite graphs, bundles, etc.)
- **sage/graphs/generators/**

**graph.py** needs to get split into several levels like the matrix code, and mostly Cythonized.

```
ls('graphs')
__init__.py      graph_database.py  linearextensions.py
all.py          graph_fast.c       planarity
base            graph_fast.pyx    planarity.c
bipartite_graph.py graph_generators.py planarity.pyx
chrompoly.c     graph_isom.c       print_graphs.py
chrompoly.pyx   graph_isom.pxd    schnyder.py
graph.py        graph_isom.pyx     trees.c
graph_bundle.py graph_list.py      trees.pxd
graph_coloring.py graph_plot.py      trees.pyx

G = graphs.KrackhardtKiteGraph()
```

```
H = G.coloring(hex_colors=True)
G.plot(vertex_colors=H).show(figsize=3)
```



```
G.automorphism_group()
Permutation Group with generators [(1,10)(2,4)(5,6)]

G.chromatic_polynomial()
x^10 - 18*x^9 + 142*x^8 - 643*x^7 + 1837*x^6 - 3424*x^5 + 4152*x^4
3151*x^3 + 1356*x^2 - 252*x
```

## Crypto

The cryptography module is under current active development (mainly by Martin Albrecht and Minh Nguyen), with the goal of making Sage the standard tool of choice for analysing symmetric ciphers using algebraic and numerical methods. The crypto module has two purposes:

- Teaching of fundamental building blocks of cryptography (stream ciphers and e.g. LFSRs, block ciphers and their S-Boxes).
- Aids research in cryptography by providing common building blocks.

The crypto module

- provides **elementary cryptographic primitives** for the purposes of teaching and

cryptanalysis,

- includes **scaled-down versions** of standard cryptographic algorithms, and
- provides interfaces to **standard cryptographic algorithms**, and
- toolboxes for encryption, decryption, signatures, and hashing.

PROJECT: Minh Nguyen is working on expanding the teaching aspect right now by adding a MiniAES to sage.crypto and Martin Albrecht is expanding its usefulness for algebraic cryptanalysis by (a) adding more ciphers (e.g. DES and PRESENT) and (b) e.g. a SAT-Solver interface.

PROJECT: Create a number of modules that wrap common number theoretic building blocks related to public-key cryptography, and package them up in a form useful for performing public-key cryptographic operations. All the building blocks are there (optimized arithmetic over the integers, support for basic and advanced number theory).

```
ls('crypto')
__init__.py      classical_cipher.py  stream.py
all.py           cryptosystem.py     stream_cipher.py
cipher.py        lfsr.py
classical.py     mq
```

**Teaching crypto** (there is a free undergrad cryptography book by David Kohel and Sage includes all the code he wrote for that book):

```
M = AlphabeticStrings()
E = SubstitutionCryptosystem(M); E
Substitution cryptosystem on Free alphabetic string monoid on A-Z
K = M([ 25-i for i in range(26) ]); K
ZYXWVUTSRQPONMLKJIHGFEDCBA
e = E(K)
m = M("THECATINTHEHAT")
secret = e(m); secret
GSVXZGRMGSVSZG
e.inverse()(secret)
THECATINTHEHAT
```

**Small Scale Variants of the AES (SR) Polynomial System Generator.** Used to experiment with algebraic attacks on block ciphers.

```
sr = mq.SR(1, 1, 1, 4)
sr
```

```
SR(1,1,1,4)
print sr.R.repr_long()
Polynomial Ring
Base Ring : Finite Field in a of size 2^4
Size : 20 Variables
Block 0 : Ordering : degrevlex
Names : k100, k101, k102, k103, x100, x101, x102,
x103, w100, w101, w102, w103, s000, s001, s002, s003, k000, k001,
k002, k003
```