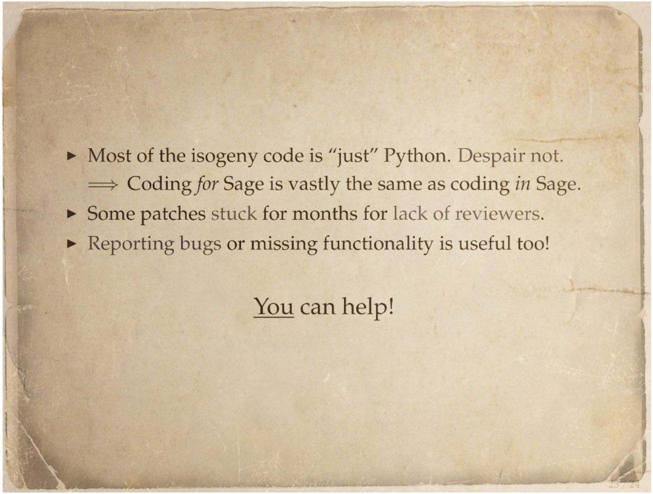
How 2 Sage Dev??

Lorenz Panny

Technische Universität München

Leuven I-sage-ny Days, Leuven, 17 January 2024

Part 4: *If it's broke, fix it!*



(Slide from my Isogeny Days 2022 talk.)

Outline

What *is* SageMath, *really?*

~~The type system~~ The three type systems

Getting your hands dirty

It's working! (That means ε% of the work is done.)

Ready? Go!

The gory internals (1)

- Sage rests on Python interfaces to mathematical libraries: They provide most of the fundamental functionality.

The gory internals (1)

- Sage rests on Python interfaces to mathematical libraries:
  They provide most of the fundamental functionality.
  - GMP & MPFR: Arbitrary-precision arithmetic.
  - PARI: Odd finite fields, univariate polynomials, power series,
    elliptic curves, quadratic forms, number fields, class groups, ...
  - NTL: $\mathbb{F}_{2^n}$, various algorithms for fast arithmetic, ...
  - Singular: Multivariate polynomials, Gröbner bases, elimination, ...
  - fplll: Lattice algorithms.
  - Linbox, M4RI: Linear algebra.
  - GAP: Abstract groups.

The gory internals (1): Example

polmodular

Return the modular polynomial of prime level L in variables x and y for the modular function specified by `inv`. If `inv` is 0 (the default), use the modular j function, if `inv` is 1 use the Weber-f function, and if `inv` is 5 use γ_2 = sqrt[3]{j}. See `polclass` for the full list of invariants. If x is given as `Mod(j, p)` or an element j of a finite field (as a `t_FFELT`), then return the modular polynomial of level L evaluated at j. If j is from a finite field and `derivs` is non-zero, then return a triple where the last two elements are the first and second derivatives of the modular polynomial evaluated at j.

```
? polmodular(3)
%1 = x^4 + (-y^3 + 2232*y^2 - 1069956*y + 36864000)*x^3 + ...
? polmodular(7, 1, , 'J)
%2 = x^8 - J^7*x^7 + 7*J^4*x^4 - 8*J*x + J^8
? polmodular(7, 5, 7*ffgen(19)^0, 'j)
%3 = j^8 + 4*j^7 + 4*j^6 + 8*j^5 + j^4 + 12*j^2 + 18*j + 18
? polmodular(7, 5, Mod(7,19), 'j)
%4 = Mod(1, 19)*j^8 + Mod(4, 19)*j^7 + Mod(4, 19)*j^6 + ...

? u = ffgen(5)^0; T = polmodular(3,0,,'j)*u;
? polmodular(3, 0, u,'j,1)
%6 = [j^4 + 3*j^2 + 4*j + 1, 3*j^2 + 2*j + 4, 3*j^3 + 4*j^2 + 4*j + 2]
? subst(T,x,u)
%7 = j^4 + 3*j^2 + 4*j + 1
? subst(T',x,u)
%8 = 3*j^2 + 2*j + 4
? subst(T'',x,u)
%9 = 3*j^3 + 4*j^2 + 4*j + 2
```

The library syntax is `GEN polmodular(long L, long inv, GEN x = NULL, long y = -1, long derivs)` where `y` is a variable number.

The gory internals (1): Example

polmodular

Return the modular polynomial of prime level L in variables x and y for the modular function specified by `inv`. If `inv` is 0 (the default), use the modular j function, if `inv` is 1 use the Weber-f function, and if `inv` is 5 use γ_2 = sqrt[3]{j}. See `polclass` for the full list of invariants. If x is given as `Mod(j, p)` or an element j of a finite field (as a `t_FFELT`), then return the modular polynomial of level L evaluated at j. If j is from a finite field and `derivs` is non-zero, then return a triple where the last two elements are the first and second derivatives of the modular polynomial evaluated at j.

```
? polmodular(3)
%1 = x^4 + (-y^3 + 2232*y^2 - 1069956*y + 36864000)*x^3 + ...
? polmodular(7, 1, , 'J)
%2 = x^8 - J^7*x^7 + 7*J^4*x^4 - 8*J*x + J^8
? polmodular(7, 5, 7*ffgen(19)^0, 'j)
%3 = j^8 + 4*j^7 + 4*j^6 + 8*j^5 + j^4 + 12*j^2 + 18*j + 18
? polmodular(7, 5, Mod(7,19), 'j)
%4 = Mod(1, 19)*j^8 + Mod(4, 19)*j^7 + Mod(4, 19)*j^6 + ...

? u = ffgen(5)^0; T = polmodular(3,0,,'j)*u;
? polmodular(3, 0, u,'j,1)
%6 = [j^4 + 3*j^2 + 4*j + 1, 3*j^2 + 2*j + 4, 3*j^3 + 4*j^2 + 4*j + 2]
? subst(T,x,u)
%7 = j^4 + 3*j^2 + 4*j + 1
? subst(T',x,u)
%8 = 3*j^2 + 2*j + 4
? subst(T'',x,u)
%9 = 3*j^3 + 4*j^2 + 4*j + 2
```

The library syntax is `GEN polmodular(long L, long inv, GEN x = NULL, long y = -1, long derivs)` where `y` is a variable number.

```
sage: from sage.libs.pari import pari  # imported by default
sage: pari.polmodular(2)
x^3 + (-y^2 + 1488*y - 162000)*x^2
    + (1488*y^2 + 40773375*y + 8748000000)*x
    + (y^3 - 162000*y^2 + 8748000000*y - 157464000000000)
```

The gory internals (1): Example

polmodular

Return the modular polynomial of prime level L in variables x and y for the modular function specified by `inv`. If `inv` is 0 (the default), use the modular j function, if `inv` is 1 use the Weber-f function, and if `inv` is 5 use γ_2 = sqrt[3]{j}. See `polclass` for the full list of invariants. If x is given as `Mod(j, p)` or an element j of a finite field (as a `t_FFELT`), then return the modular polynomial of level L evaluated at j. If j is from a finite field and `derivs` is non-zero, then return a triple where the last two elements are the first and second derivatives of the modular polynomial evaluated at j.

```
? polmodular(3)
%1 = x^4 + (-y^3 + 2232*y^2 - 1069956*y + 36864000)*x^3 + ...
? polmodular(7, 1, , 'J)
%2 = x^8 - J^7*x^7 + 7*J^4*x^4 - 8*J*x + J^8
? polmodular(7, 5, 7*ffgen(19)^0, 'j)
%3 = j^8 + 4*j^7 + 4*j^6 + 8*j^5 + j^4 + 12*j^2 + 18*j + 18
? polmodular(7, 5, Mod(7,19), 'j)
%4 = Mod(1, 19)*j^8 + Mod(4, 19)*j^7 + Mod(4, 19)*j^6 + ...

? u = ffgen(5)^0; T = polmodular(3,0,,'j)*u;
? polmodular(3, 0, u,'j,1)
%6 = [j^4 + 3*j^2 + 4*j + 1, 3*j^2 + 2*j + 4, 3*j^3 + 4*j^2 + 4*j + 2]
? subst(T,x,u)
%7 = j^4 + 3*j^2 + 4*j + 1
? subst(T',x,u)
%8 = 3*j^2 + 2*j + 4
? subst(T'',x,u)
%9 = 3*j^3 + 4*j^2 + 4*j + 2
```

The library syntax is `GEN polmodular(long L, long inv, GEN x = NULL, long y = -1, long derivs)` where `y` is a variable number.

```
sage: from sage.libs.pari import pari  # imported by default
sage: pari.polmodular(2)
x^3 + (-y^2 + 1488*y - 162000)*x^2
    + (1488*y^2 + 40773375*y + 8748000000)*x
    + (y^3 - 162000*y^2 + 8748000000*y - 157464000000000)
sage: type(pari.polmodular(2))
<class 'cypari2.gen.Gen'>
```

The gory internals (2)

- On top of that: `sagelib`, all the “native” Sage (Python) code.
  - `src/sage/rings/finite_rings/...`
  - `src/sage/rings/polynomials/...`
  - `src/sage/algebras/quatalg/...`
  - `src/sage/schemes/elliptic_curves/...`
  - `src/sage/rings/number_field/...`
  - `src/sage/quadratic_forms/...`

The gory internals (2)

- On top of that: `sagelib`, all the "native" Sage (Python) code.
  - `src/sage/rings/finite_rings/...`
  - `src/sage/rings/polynomials/...`
  - `src/sage/algebras/quatalg/...`
  - `src/sage/schemes/elliptic_curves/...`
  - `src/sage/rings/number_field/...`
  - `src/sage/quadratic_forms/...`
- Generally higher-level functionality (but plenty of exceptions).

The gory internals (2)

- On top of that: `sagelib`, all the “native” Sage (Python) code.
  - `src/sage/rings/finite_rings/...`
  - `src/sage/rings/polynomials/...`
  - `src/sage/algebras/quatalg/...`
  - `src/sage/schemes/elliptic_curves/...`
  - `src/sage/rings/number_field/...`
  - `src/sage/quadratic_forms/...`
- Generally higher-level functionality (but plenty of exceptions).
- Sometimes uses Cython for performance (`.pyx` files).

The gory internals (2)

- On top of that: `sagelib`, all the "native" Sage (Python) code.
  - `src/sage/rings/finite_rings/...`
  - `src/sage/rings/polynomials/...`
  - `src/sage/algebras/quatalg/...`
  - `src/sage/schemes/elliptic_curves/...`
  - `src/sage/rings/number_field/...`
  - `src/sage/quadratic_forms/...`
- Generally higher-level functionality (but plenty of exceptions).
- Sometimes uses Cython for performance (`.pyx` files).
  This is an extremely hacky Python dialect that is compiled for speed.

The gory internals (2)

- On top of that: `sagelib`, all the "native" Sage (Python) code.
  - `src/sage/rings/finite_rings/...`
  - `src/sage/rings/polynomials/...`
  - `src/sage/algebras/quatalg/...`
  - `src/sage/schemes/elliptic_curves/...`
  - `src/sage/rings/number_field/...`
  - `src/sage/quadratic_forms/...`
- Generally higher-level functionality (but plenty of exceptions).
- Sometimes uses Cython for performance (`.pyx` files).
  This is an extremely hacky Python dialect that is compiled for speed.
  (I don't *love* it.)

How to find stuff (1)

```
sage: E = EllipticCurve(j=42)
sage: E.isogeny?
```

How to find stuff (1)

```
sage: E = EllipticCurve(j=42)
sage: E.isogeny?
Signature:
E.isogeny(
    kernel,
    codomain=None,
    degree=None,
    model=None,
    check=True,
    algorithm=None,
)
Docstring:
    Return an elliptic-curve isogeny from this elliptic curve.

# ...

Init docstring: Initialize self.  See help(type(self)) for accurate signature.
File:           ~sage/src/sage/schemes/elliptic_curves/ell_field.py
Type:           method
```

How to find stuff (1)

```
sage: E = EllipticCurve(j=42)
sage: E.isogeny??
```

How to find stuff (1)

```
sage: E = EllipticCurve(j=42)
sage: E.isogeny??
Signature:
E.isogeny(
# ...
)

# ...

  if algorithm == "velusqrt":
    from sage.schemes.elliptic_curves.hom_velusqrt import EllipticCurveHom_velusqrt
    return EllipticCurveHom_velusqrt(self, kernel, codomain=codomain, model=model)
  if algorithm == "factored":
    from sage.schemes.elliptic_curves.hom_composite import EllipticCurveHom_composite
    return EllipticCurveHom_composite(self, kernel, codomain=codomain, model=model)
  try:
    return EllipticCurveIsogeny(self, kernel, codomain, degree, model, check=check)
  except AttributeError as e:
    raise RuntimeError("Unable to construct isogeny: %s" % e)
File:      ~sage/src/sage/schemes/elliptic_curves/ell_field.py
Type:      method
```

How to find stuff (1)

```
sage: E = EllipticCurve(j=42)
sage: E.isogeny???
```

How to find stuff (1)

```
sage: E = EllipticCurve(j=42)
sage: E.isogeny???
  Cell In[1], line 2
    E.isogeny???
             ^
SyntaxError: invalid syntax
```

How to find stuff (2)

```
sage: E = EllipticCurve(GF(419^2), [1,0])
sage: pi = E.frobenius_isogeny()
sage: type(pi)
<class 'sage.schemes.elliptic_curves.hom_frobenius.EllipticCurveHom_frobenius'>
```

How to find stuff (2)

```
sage: E = EllipticCurve(GF(419^2), [1,0])
sage: pi = E.frobenius_isogeny()
sage: type(pi)
<class 'sage.schemes.elliptic_curves.hom_frobenius.EllipticCurveHom_frobenius'>
```

```
sage: import_statements(order_from_multiple)
# ...
NameError: name 'order_from_multiple' is not defined
sage: import_statements('order_from_multiple')
from sage.groups.generic import order_from_multiple
```

Common Sage vs. Python pitfalls

Sage	**Python**
(nothing)	`from sage.somewhere import Something`
`^`	`**`
`^^`	`^`
`42`	`Integer(42)`
`R.<x> = Thing`	`R,x = Thing.objgen()`

Outline

What *is* SageMath, *really?*

~~The type system~~ The three type systems

Getting your hands dirty

It's working! (That means ε % of the work is done.)

Ready? Go!

Type system #1: Python

- ...is just Python's: Every `object` has a `type`.

```
sage: type('Hello, world!')
<class 'str'>
```

Type system #1: Python

- ...is just Python's: Every `object` has a `type`.

```
sage: type('Hello, world!')
<class 'str'>
```

- "Ground truth" of what code really gets executed.
  You write `a + b`, it calls `type(a).__add__(a,b)`. (Et cetera.)

Type system #1: Python

- ...is just Python's: Every `object` has a `type`.

```
sage: type('Hello, world!')
<class 'str'>
```

- "Ground truth" of what code really gets executed.
  You write `a + b`, it calls `type(a).__add__(a,b)`. (Et cetera.)

- Creating new types: `class` keyword.

Type system #1: Python

- ...is just Python's: Every `object` has a `type`.

```
sage: type('Hello, world!')
<class 'str'>
```

- "Ground truth" of what code really gets executed.
  You write `a + b`, it calls `type(a).__add__(a,b)`. (Et cetera.)

- Creating new types: `class` keyword.
- Inheritance: Copy code and data from an ancestor class.
  For example, `EllipticCurve_finite_field` inherits from `EllipticCurve_field`.

```
class EllipticCurve_finite_field(EllipticCurve_field,
                                 HyperellipticCurve_finite_field):
    # ...
```

Type system #2: `SageObject`s and `Parent`s

- Every "thing" in Sage is a `SageObject` (which is a Python type).

Type system #2: `SageObjects` and `Parents`

- Every "thing" in Sage is a `SageObject` (which is a Python type).
- "Things" are typically `Elements` or `Morphisms` or `Parents`.
  - Example: `EllipticCurvePoint`, `EllipticCurveHom`, `EllipticCurve`.

Type system #2: SageObjects and Parents

- Every "thing" in Sage is a `SageObject` (which is a Python type).
- "Things" are typically `Elements` or `Morphisms` or `Parents`.
  - Example: `EllipticCurvePoint`, `EllipticCurveHom`, `EllipticCurve`.
- We can get the parent of any object `x` using... `x.parent()`.

```
sage: 5.parent()
Integer Ring
```

Type system #2: SageObjects and Parents

- Every "thing" in Sage is a `SageObject` (which is a Python type).
- "Things" are typically `Elements` or `Morphisms` or `Parents`.
  - Example: `EllipticCurvePoint`, `EllipticCurveHom`, `EllipticCurve`.
- We can get the parent of any object `x` using... `x.parent()`.

```
sage: 5.parent()
Integer Ring
```

- Lots of conventions for how these "things" behave.
  - Particular `SageObjects` should not override (say) `.__add__()`.
  - Instead, "things" first defer to the coercion system for type checks and conversions, which *then* calls your (say) `._add_()`.

```
# from sage.quadratic_forms.bqf_class_group.BQFClassGroup_element
def _add_(self, other):
    r""" ... """
    F = self._form * other._form
    return BQFClassGroup_element(F, parent=self.parent())
```

Type system #3: Categories

Previous slide:

- Each kind of "thing" is split into three seemingly independent types: `Elements`, `Morphisms`, `Parents`.

"that's how it was done in Sage before 2009, and there are still many traces of this"
— `https://doc.sagemath.org/html/en/reference/categories/sage/categories/primer.html`

Type system #3: Categories

<u>Previous slide:</u>

- Each kind of "thing" is split into three seemingly independent types: `Elements`, `Morphisms`, `Parents`.

  "that's how it was done in Sage before 2009, and there are still many traces of this"
  — `https://doc.sagemath.org/html/en/reference/categories/sage/categories/primer.html`

$\Longrightarrow$ "New" type system ($\sim$2009?): Categories.

...unifies inheritance of `Elements`, `Morphisms`, `Parents` into one type.

Type system #3: Categories

<u>Previous slide:</u>

- Each kind of "thing" is split into three seemingly independent types: `Elements`, `Morphisms`, `Parents`.

  "that's how it was done in Sage before 2009, and there are still many traces of this"
  — `https://doc.sagemath.org/html/en/reference/categories/sage/categories/primer.html`

$\Longrightarrow$ "New" type system ($\sim$2009?): Categories.
...unifies inheritance of `Elements`, `Morphisms`, `Parents` into one type.

- As far as I can tell, this is still not widely used.

Type system #3: Categories

Previous slide:

- Each kind of "thing" is split into three seemingly independent types: `Elements`, `Morphisms`, `Parents`.

  "that's how it was done in Sage before 2009, and there are still many traces of this"
  — `https://doc.sagemath.org/html/en/reference/categories/sage/categories/primer.html`

$\Longrightarrow$ "New" type system ($\sim$2009?): Categories.
...unifies inheritance of `Elements`, `Morphisms`, `Parents` into one type.

- As far as I can tell, this is still not widely used.
- General advice: Mimic existing, similar code.

Outline

What *is* SageMath, *really?*

~~The type system~~ The three type systems

Getting your hands dirty

It's working! (That means $\varepsilon\,\%$ of the work is done.)

Ready? Go!

How 2 Sage Dev!! (for lazy people)

1. Just write your code in a `.sage` file as usual.

How 2 Sage Dev!! (for lazy people)

1. Just write your code in a `.sage` file as usual.

2. Wait for someone else to put it *into* Sage.

How 2 Sage Dev!! (for the laziest of people)

1. Found a bug? Report it.

How 2 Sage Dev!! (for the laziest of people)

1. Found a bug? Report it.
2. Something's missing? Report it.

How 2 Sage Dev!! (for the laziest of people)

1. Found a bug? Report it.
2. Something's missing? Report it.
3. Something's unreasonably slow? Report it.

How 2 Sage Dev!!

1. Edit the code in whatever way you think is right.

How 2 Sage Dev!!

1. Edit the code in whatever way you think is right.

2. Possibly rebuild Sage: `./sage -b` or perhaps `make build`.
   (Nowadays, this should only be necessary after editing a `.pyx` file — but it can't hurt.)
   Careful: Running `make` without `build` can take ≈ infinitely longer.

How 2 Sage Dev!!

1. Edit the code in whatever way you think is right.

2. Possibly rebuild Sage: `./sage -b` or perhaps `make build`.
   (Nowadays, this should only be necessary after editing a `.pyx` file — but it can't hurt.)
   Careful: Running `make` without `build` can take ≈ infinitely longer.

3. Check the results!
   - In all cases: Try one or a few basic examples first.
   - In all cases: Run old (and possibly new) tests. (More later.)
   - To check performance changes: Use `%timeit`, `%prun`.

How 2 Sage Dev!!

1. Edit the code in whatever way you think is right.

2. Possibly rebuild Sage: `./sage -b` or perhaps `make build`.
   (Nowadays, this should only be necessary after editing a `.pyx` file — but it can't hurt.)
   Careful: Running `make` without `build` can take ≈ infinitely longer.

3. Check the results!
   - In all cases: Try one or a few basic examples first.
   - In all cases: Run old (and possibly new) tests. (More later.)
   - To check performance changes: Use `%timeit`, `%prun`.

4. Unless satisfied, go back to Step 1 and iterate.

Assertions

Recommendation: `assert` anything that *could* go wrong.

Assertions

Recommendation: `assert` anything that *could* go wrong.

```
sage: B = QuaternionAlgebra(-1, -419)
sage: I = B.maximal_order().unit_ideal()
sage: a = I.random_element()
sage: assert a in I
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[1], line 1
----> 1 assert a in I

AssertionError:
```

Assertions

Recommendation: `assert` anything that *could* go wrong.

```
sage: B = QuaternionAlgebra(-1, -419)
sage: I = B.maximal_order().unit_ideal()
sage: a = I.random_element()
sage: assert a in I
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[1], line 1
----> 1 assert a in I

AssertionError:
```

- Assertions are free*: can be skipped in production code.

Assertions

Recommendation: `assert` anything that *could* go wrong.

```
sage: B = QuaternionAlgebra(-1, -419)
sage: I = B.maximal_order().unit_ideal()
sage: a = I.random_element()
sage: assert a in I
-----------------------------------------------------------
AssertionError                          Traceback (most recent call last)
Cell In[1], line 1
----> 1 assert a in I

AssertionError:
```

- Assertions are free*: can be skipped in production code.

  *Sad news: Sage currently does not disable assertions even in release builds.

Assertions

Recommendation: `assert` anything that *could* go wrong.

```
sage: B = QuaternionAlgebra(-1, -419)
sage: I = B.maximal_order().unit_ideal()
sage: a = I.random_element()
sage: assert a in I
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[1], line 1
----> 1 assert a in I

AssertionError:
```

- Assertions are free*: can be skipped in production code.
  *Sad news: Sage currently does not disable assertions even in release builds.
  - You can skip assertions by setting `PYTHONOPTIMIZE=1` in your environment.

Assertions

Recommendation: `assert` anything that *could* go wrong.

```
sage: B = QuaternionAlgebra(-1, -419)
sage: I = B.maximal_order().unit_ideal()
sage: a = I.random_element()
sage: assert a in I
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[1], line 1
----> 1 assert a in I

AssertionError:
```

- Assertions are free*: can be skipped in production code.

  *Sad news: Sage currently does not disable assertions even in release builds.

  - You can skip assertions by setting `PYTHONOPTIMIZE=1` in your environment.
  - Almost all users won't do this and complain about *your* slow code.

Assertions

Recommendation: `assert` anything that *could* go wrong.

```
sage: B = QuaternionAlgebra(-1, -419)
sage: I = B.maximal_order().unit_ideal()
sage: a = I.random_element()
sage: assert a in I
---------------------------------------------------------------
AssertionError                Traceback (most recent call last)
Cell In[1], line 1
----> 1 assert a in I

AssertionError:
```

- Assertions are free*: can be skipped in production code.

  *Sad news: Sage currently does not disable assertions even in release builds.

  - You can skip assertions by setting `PYTHONOPTIMIZE=1` in your environment.
  - Almost all users won't do this and complain about *your* slow code.

The next best thing™:

- After debugging, comment out `asserts` (instead of deleting them).

  It aids later debugging sessions, and it helps convey your intentions behind the code.

Outline

What *is* SageMath, *really?*

~~The type system~~ The three type systems

Getting your hands dirty

It's working! (That means $\varepsilon\,\%$ of the work is done.)

Ready? Go!

Coding style

- People have strong opinions on how to format code.

Coding style

- People have strong opinions on how to format code.
- Those people have automated check scripts, too.

Coding style

- People have strong opinions on how to format code.
- Those people have automated check scripts, too.

$\rightsquigarrow$ Make sure your Sage patch adheres to the rules.

https://doc.sagemath.org/html/en/developer/coding_basics.html#python-code-style

Docstrings

Example from `EllipticCurveHom_velusqrt` (part 1/4):

```
r"""
This class implements separable odd-degree isogenies of elliptic
curves over finite fields using the square-root Vélu algorithm.

The complexity is `\tilde O(\sqrt{\ell})` base-field operations,
where `\ell` is the degree.

REFERENCES: [BDLS2020]_

INPUT:

- ``E`` -- an elliptic curve over a finite field
- ``P`` -- a point on `E` of odd order `\geq 9`
- ``codomain`` -- codomain elliptic curve (optional)
- ``model`` -- string (optional); input to
  :meth:`~sage.schemes.elliptic_curves.ell_field.compute_model`
- ``Q`` -- a point on `E` outside `\langle P\rangle`, or ``None``

# ...
```

Docstrings

Example from `EllipticCurveHom_velusqrt` (part 2/4):

```
r"""
# ...

EXAMPLES::

    sage: from sage.schemes.elliptic_curves.hom_velusqrt import
                                                EllipticCurveHom_velusqrt
    sage: F.<t> = GF(10009^3)
    sage: E = EllipticCurve(F, [t,t])
    sage: K = E(2154*t^2 + 5711*t + 2899, 7340*t^2 + 4653*t + 6935)
    sage: phi = EllipticCurveHom_velusqrt(E, K); phi
    Elliptic-curve isogeny (using square-root Vélu) of degree 601:
      From: Elliptic Curve defined by y^2 = x^3 + t*x + t
             over Finite Field in t of size 10009^3
      To:   Elliptic Curve defined by y^2 = x^3 + (263*t^2+3173*t+4759)*x
             + (3898*t^2+6111*t+9443) over Finite Field in t of size 10009^3
    sage: phi(K)
    (0 : 1 : 0)
    sage: P = E(2, 3163*t^2 + 7293*t + 5999)
    sage: phi(P)
    (6085*t^2 + 855*t + 8720 : 8078*t^2 + 9889*t + 6030 : 1)
    sage: Q = E(6, 5575*t^2 + 6607*t + 9991)
    sage: phi(Q)
    (626*t^2 + 9749*t + 1291 : 5931*t^2 + 8549*t + 3111 : 1)
    sage: phi(P + Q)
    (983*t^2 + 4894*t + 4072 : 5047*t^2 + 9325*t + 336 : 1)
    sage: phi(P) + phi(Q)
    (983*t^2 + 4894*t + 4072 : 5047*t^2 + 9325*t + 336 : 1)

# ...
```

Docstrings

Example from `EllipticCurveHom_velusqrt` (part 3/4):

```
r"""
# ...

TESTS:

Check on a random example that the isogeny is a well-defined
group homomorphism with the correct kernel::

    sage: from sage.schemes.elliptic_curves.hom_velusqrt import
                                                _random_example_for_testing
    sage: E, K = _random_example_for_testing()
    sage: phi = EllipticCurveHom_velusqrt(E, K)
    sage: not phi(K)
    True
    sage: not phi(randrange(2^99) * K)
    True
    sage: P = E.random_point()
    sage: phi(P) in phi.codomain()
    True
    sage: Q = E.random_point()
    sage: phi(Q) in phi.codomain()
    True
    sage: phi(P + Q) == phi(P) + phi(Q)
    True

# ...
```

Docstrings

Example from `EllipticCurveHom_velusqrt` (part 4/4):

```
r"""
# ...

Check that the isogeny preserves the field of definition::

    sage: Sequence(K).universe() == phi.domain().base_field()
    True
    sage: phi.codomain().base_field() == phi.domain().base_field()
    True

Check that the isogeny affects the Weil pairing in the correct way::

    sage: m = lcm(P.order(), Q.order())
    sage: e1 = P.weil_pairing(Q, m)
    sage: e2 = phi(P).weil_pairing(phi(Q), m)
    sage: e2 == e1^phi.degree()
    True

Check that the isogeny matches (up to isomorphism) the one from
:class:`~sage.schemes.elliptic_curves.ell_curve_isogeny.EllipticCurveIsogeny`::

    sage: psi = EllipticCurveIsogeny(E, K)
    sage: check = lambda iso: all(iso(psi(Q)) == phi(Q) for Q in E.gens())
    sage: any(map(check, psi.codomain().isomorphisms(phi.codomain())))
    True

.. SEEALSO::

    :class:`~sage.schemes.elliptic_curves.ell_curve_isogeny.EllipticCurveIsogeny`
"""
```

Meaningful documentation

- Whenever some object in a docstring is not universally well-known and uniquely defined, clarify the definition.

Meaningful documentation

- Whenever some object in a docstring is not universally well-known and uniquely defined, <u>clarify</u> the definition.

```
scaling_factor()
```

Return the Weierstrass scaling factor associated to this elliptic-curve morphism.

The scaling factor is the constant u (in the base field) such that $\varphi^*\omega_2 = u\omega_1$, where $\varphi : E_1 \to E_2$ is this morphism and ω_i are the standard Weierstrass differentials on E_i defined by $\mathrm{d}x/(2y + a_1x + a_3)$.

Meaningful documentation

- Whenever some object in a docstring is not universally well-known and uniquely defined, clarify the definition.

`matrix_on_subgroup(domain_gens, codomain_gens=None)`

Return the matrix by which this isogeny acts on the n-torsion subgroup with respect to the given bases.

INPUT:

- `domain_gens` – basis (P, Q) of some n-torsion subgroup on the domain of this elliptic-curve morphism
- `codomain_gens` – basis (R, S) of the n-torsion on the codomain of this morphism, or (default) `None` if `self` is an endomorphism

OUTPUT:

A 2×2 matrix M over $\mathbf{Z}/n$, such that the image of any point $[a]P + [b]Q$ under this morphism equals $[c]R + [d]S$ where $(c\ d)^T = (a\ b)M$.

Examples & tests

Examples & tests

- Examples are for users.
  ⇝ Easy to read and understand; should demonstrate the core ideas.

Examples & tests

- Examples are for users.
  $\rightsquigarrow$ Easy to read and understand; should demonstrate the core ideas.
- Tests are for the machine.
  $\rightsquigarrow$ Should cover as many inputs as possible, in particular edge cases.

Examples & tests

- Examples are for users.
  ⇝ Easy to read and understand; should demonstrate the core ideas.
- Tests are for the machine.
  ⇝ Should cover as many inputs as possible, in particular edge cases.
- Sage can run all examples & tests to verify the output:

```
$ ./sage -t src/sage/schemes/elliptic_curves/hom_velusqrt.py
# ...
Doctesting 1 file.
sage -t --random-seed=331181331784926877031128059220586214893
                        src/sage/schemes/elliptic_curves/hom_velusqrt.py
    [259 tests, 1.37 s]
----------------------------------------------------------------------
All tests passed!
----------------------------------------------------------------------
Total time for all tests: 1.4 seconds
 cpu time: 1.3 seconds
 cumulative wall time: 1.4 seconds
```

Writing *good* tests

- Things can go wrong in so many different ways.

Writing *good* tests

- Things can go wrong in so many different ways.
- Most important part for Sage: Check the actual math(s).

Writing *good* tests

- Things can go wrong in so many different ways.
- Most important part for Sage: Check the actual math(s).
  Example: If some φ should be a group morphism, check $\varphi(a+b) = \varphi(a) + \varphi(b)$.

Writing *good* tests

- Things can go wrong in so many different ways.
- Most important part for Sage: Check the actual math(s).
  Example: If some φ should be a group morphism, check $\varphi(a+b) = \varphi(a) + \varphi(b)$.
  Example: If something can be computed in two ways, do it both ways and compare!

Writing *good* tests

- Things can go wrong in so many different ways.
- Most important part for Sage: Check the actual math(s).

  Example: If some φ should be a group morphism, check $\varphi(a + b) = \varphi(a) + \varphi(b)$.
  Example: If something can be computed in two ways, do it both ways and compare!
  Example: Add the use case that motivated you to write the code as an example or test!

Writing *good* tests

- Things can go wrong in so many different ways.
- Most important part for Sage: Check the actual math(s).
  Example: If some φ should be a group morphism, check $\varphi(a+b) = \varphi(a) + \varphi(b)$.
  Example: If something can be computed in two ways, do it both ways and compare!
  Example: Add the use case that motivated you to write the code as an example or test!
- Amazing tool: Randomized tests.
  Sage's tests are run over and over again by many different people.
  Eventually <u>someone</u> will get (un)lucky enough to find rare edge cases.

Writing *good* tests

- Things can go wrong in so many different ways.
- Most important part for Sage: Check the actual math(s).
  Example: If some φ should be a group morphism, check $\varphi(a+b) = \varphi(a) + \varphi(b)$.
  Example: If something can be computed in two ways, do it both ways and compare!
  Example: Add the use case that motivated you to write the code as an example or test!
- Amazing tool: Randomized tests.
  Sage's tests are run over and over again by many different people.
  Eventually someone will get (un)lucky enough to find rare edge cases.
- Code coverage should ideally be 100 %.
  (This implies we should also test invalid inputs and error cases.)

Randomized testing will find you funny examples

- For $\sqrt{\text{élu}}$ we need a point outside the kernel.

Randomized testing will find you funny examples

- For $\sqrt{\text{élu}}$ we need a point outside the kernel.
- If the kernel is all of $E(\mathbb{F}_q)$: Easy, just extend to $\mathbb{F}_{q^2}$.

Randomized testing will find you funny examples

- For $\sqrt{\text{élu}}$ we need a point outside the kernel.
- If the kernel is all of $E(\mathbb{F}_q)$: Easy, just extend to $\mathbb{F}_{q^2}$.
- *Right?*

Randomized testing will find you funny examples

- For $\sqrt{\text{élu}}$ we need a point outside the kernel.
- If the kernel is all of $E(\mathbb{F}_q)$: Easy, just extend to $\mathbb{F}_{q^2}$.
- *Right?*
- Debugging random test failures
  $\rightsquigarrow$ There exists exactly one example where $E(\mathbb{F}_{q^2}) = E(\mathbb{F}_q)$.

Randomized testing will find you funny examples

- For $\sqrt{\text{élu}}$ we need a point outside the kernel.
- If the kernel is all of $E(\mathbb{F}_q)$: Easy, just extend to $\mathbb{F}_{q^2}$.
- *Right?*
- Debugging random test failures
  $\rightsquigarrow$ There exists exactly one example where $E(\mathbb{F}_{q^2}) = E(\mathbb{F}_q)$.
- The curve $y^2 = x^3 - x + 1$ has 7 points over both $\mathbb{F}_3$ and $\mathbb{F}_9$.

[See `https://github.com/sagemath/sage/issues/34467`.]

Changing existing behaviour: Deprecation warnings

- Since Sage is very reliable and stable,
  we have to warn users when something is about to change.

Changing existing behaviour: Deprecation warnings

- Since Sage is *aspiring to be* very reliable and stable,
  we have to warn users when something is about to change.

Changing existing behaviour: Deprecation warnings

- Since Sage is *aspiring to be* very reliable and stable,
  we have to warn users when something is about to change.

```
sage: E = EllipticCurve(j=42)
sage: E.multiplication_by_m_isogeny(-1)
<ipython-input-3-4fcd9ad225e6>:1: DeprecationWarning:
 The .multiplication_by_m_isogeny() method is superseded by .scalar_multiplication().
See https://github.com/sagemath/sage/issues/32826 for details.
  E.multiplication_by_m_isogeny(-Integer(1))
Isogeny of degree 1
 from Elliptic Curve defined by y^2 = x^3 + 5901*x + 1105454 over Rational Field
 to Elliptic Curve defined by y^2 = x^3 + 5901*x + 1105454 over Rational Field
```

Changing existing behaviour: Deprecation warnings

- Since Sage is *aspiring to be* very reliable and stable, we have to warn users when something is about to change.

```
sage: E = EllipticCurve(j=42)
sage: E.multiplication_by_m_isogeny(-1)
<ipython-input-3-4fcd9ad225e6>:1: DeprecationWarning:
 The .multiplication_by_m_isogeny() method is superseded by .scalar_multiplication().
See https://github.com/sagemath/sage/issues/32826 for details.
  E.multiplication_by_m_isogeny(-Integer(1))
Isogeny of degree 1
 from Elliptic Curve defined by y^2 = x^3 + 5901*x + 1105454 over Rational Field
 to Elliptic Curve defined by y^2 = x^3 + 5901*x + 1105454 over Rational Field
```

- This warning has to remain in place for at least one year before we are allowed to break old code that used to work.

Outline

What *is* SageMath, *really?*

~~The type system~~ The three type systems

Getting your hands dirty

It's working! (That means ε % of the work is done.)

Ready? Go!

How 2 Sage Dev!! (continued)

1. Edit the code in whatever way you think is right.
2. Possibly rebuild Sage: `make build`.
3. Check the results!
4. Unless satisfied, go back to Step 1 and iterate.

How 2 Sage Dev!! (continued)

1. Edit the code in whatever way you think is right.
2. Possibly rebuild Sage: `make build`.
3. Check the results!
4. Unless satisfied, go back to Step 1 and iterate.
5. Commit your changes — ideally in small meaningful units.
   (As far as I can tell, the "small meaningful commits" principle seems to be entirely unenforced in Sage.)

How 2 Sage Dev!! (continued)

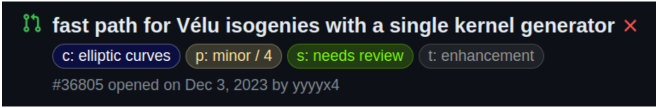
1. Edit the code in whatever way you think is right.
2. Possibly rebuild Sage: `make build`.
3. Check the results!
4. Unless satisfied, go back to Step 1 and <u>iterate</u>.
5. Commit your changes — ideally in small meaningful units.
   (As far as I can tell, the "small meaningful commits" principle seems to be entirely unenforced in Sage.)
6. Re-check everything once more. (People sometimes omit this step...)

How 2 Sage Dev!! (continued)

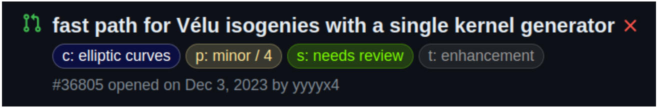
1. Edit the code in whatever way you think is right.
2. Possibly rebuild Sage: `make build`.
3. Check the results!
4. Unless satisfied, go back to Step 1 and <u>iterate</u>.

5. Commit your changes — ideally in small meaningful units.
   (As far as I can tell, the "small meaningful commits" principle seems to be entirely unenforced in Sage.)
6. Re-check everything once more. (People sometimes omit this step...)
7. Push to a branch on your fork and make a pull request.

The review pipeline

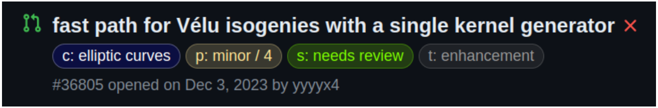


The review pipeline



- Leftover habit from earlier times: Labels.
  Typically one label each to indicate the **c**omponent, **p**riority, **t**ype, and **s**tatus.

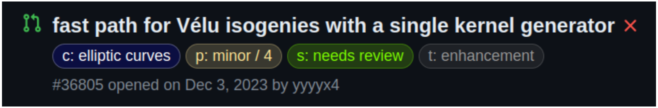
The review pipeline



- Leftover habit from earlier times: Labels.
  Typically one label each to indicate the **c**omponent, **p**riority, **t**ype, and **s**tatus.

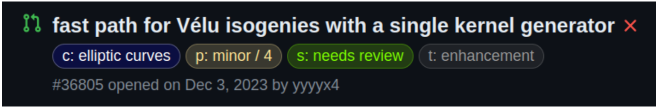
- The infrastructure automatically runs various checks.
  Indicated by × or ✓. <u>Note:</u> These checks often fail even when the patch is good... ☹

The review pipeline



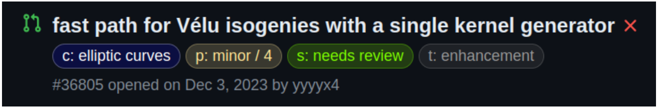
- Leftover habit from earlier times: Labels.
  Typically one label each to indicate the **c**omponent, **p**riority, **t**ype, and **s**tatus.
- The infrastructure automatically runs various checks.
  Indicated by × or ✓. Note: These checks often fail even when the patch is good... ☹
- When the stars are right, a real person will volunteer to review the proposed changes.

The review pipeline



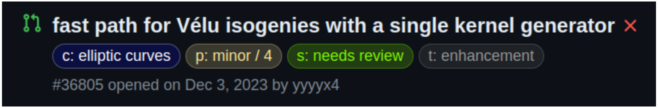
- Leftover habit from earlier times: Labels.
  Typically one label each to indicate the **c**omponent, **p**riority, **t**ype, and **s**tatus.
- The infrastructure automatically runs various checks.
  Indicated by × or ✓. Note: These checks often fail even when the patch is good... ☹
- When the stars are right, a real person will volunteer to review the proposed changes. Possible outcomes:
  - "LGTM" (**L**ooks **G**ood **T**o **M**e)

The review pipeline



- Leftover habit from earlier times: Labels.
  Typically one label each to indicate the **c**omponent, **p**riority, **t**ype, and **s**tatus.
- The infrastructure automatically runs various checks.
  Indicated by × or ✓. Note: These checks often fail even when the patch is good... ☹
- When the stars are right, a real person will volunteer to review the proposed changes. Possible outcomes:
  - "LGTM" (**L**ooks **G**ood **T**o **M**e)
  - Modifications are requested (at varying levels of insistence)

The review pipeline

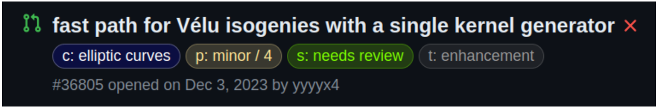


- Leftover habit from earlier times: Labels.
  Typically one label each to indicate the **c**omponent, **p**riority, **t**ype, and **s**tatus.

- The infrastructure automatically runs various checks.
  Indicated by × or ✓. <u>Note:</u> These checks often fail even when the patch is good... ☹

- When the stars are right, a real person will volunteer to review the proposed changes. Possible outcomes:
  - "LGTM" (**L**ooks **G**ood **T**o **M**e)
  - Modifications are requested (at varying levels of insistence)
  - An intense fundamental debate about *something* goes into its next round and unfortunately it's happening on your pull request this time — `glhf`

The review pipeline



- Leftover habit from earlier times: Labels.
  Typically one label each to indicate the **c**omponent, **p**riority, **t**ype, and **s**tatus.
- The infrastructure automatically runs various checks.
  Indicated by × or ✓. Note: These checks often fail even when the patch is good… ☹
- When the stars are right, a real person will volunteer to review the proposed changes. Possible outcomes:
  - "LGTM" (**L**ooks **G**ood **T**o **M**e)
  - Modifications are requested (at varying levels of insistence)
  - An intense fundamental debate about *something* goes into its next round and unfortunately it's happening on your pull request this time — `glhf`
- Finally (a little while after positive review): It gets **merged!**
  Congratulations, you're now ~~rich and~~ famous. 🎉

Two more remarks

- GitHub supports multiple names on a commit.
  Keyword to search for: `Co-authored-by`.

Two more remarks

- GitHub supports multiple names on a commit.
  Keyword to search for: `Co-authored-by`.

- Don't forget to mention your changes in the release tour, assuming users would benefit from learning about them.

~~Two~~ Three more remarks

- GitHub supports multiple names on a commit.
  Keyword to search for: `Co-authored-by`.

- Don't forget to mention your changes in the release tour, assuming users would benefit from learning about them.

- You, too, can be a reviewer! ☺

Hype!!!1!11

`https://doc.sagemath.org/html/en/developer`