Parent, élément, coercion

Author: Thierry Monteil
Author: Vincent Delecroix

Voici une courte introduction à propos de la notion de parent et d'élement dans Sage, vous trouverez plus de détails dans ce tutoriel (sagenb live / jupyter live)

Les éléments (nombres, matrices, polynomes,...) ont des parents (corps des nombres rationnels, espaces de matrices, anneau de polynomes,...).

```
sage: 3.parent()

sage: 3.parent() == ZZ

sage: m = matrix([[1,2,3],[4,5,6]])
sage: m.parent()
sage: m.parent() == MatrixSpace(ZZ,2,3)

sage: RDF.an_element()

sage: RDF.random_element().parent()
```

Les parents aussi sont des objets avec leurs méthodes.

```
sage: a = 3/2

sage: q = a.parent()
sage: q

sage: q.

sage: alg = q.algebraic_closure()

sage: alg.
```

Cela permet par exemple d'aditionner des nombres de types différents, en les transformant en éléments d'un parent commun.

```
sage: from sage.structure.element import get_coercion_model
sage: cm = get_coercion_model()
sage: K = RDF
sage: L = RealField(2)
sage: M = cm.common_parent(K, L)
sage: M

sage: (K.an_element() + L.an_element()).parent()
```

Voici un exemple plus subtil où le résultat de l'opération est un parent différent.

```
sage: R = ZZ['x']
sage: cm.common_parent(R, QQ)
```

```
sage: (R.an_element() + QQ.an_element()).parent()
```

L'égalité aussi est effectuée dans un parent commun :

```
sage: a = RDF(pi)
sage: a

sage: b = RealField(2)(3)
sage: b

sage: a == b
```

Exercice:

```
sage: M = random_matrix(QQbar,2)
sage: M

sage: a = 0.2

sage: N = M + a
```

Sauriez-vous deviner sans évaluer les lignes suivantes à quoi ressemble `N` et quel est son parent ?

```
sage: N

sage: M.parent()

sage: a.parent()

sage: N.parent()
```

Voici un exemple montrant l'importance de savoir comment sont représentés les objets. Nous voulons tracer la suite des points *sum_{k=0}^{n-1} z_n* où la suite *z_n = exp(2 i pi u_n)* et *u_n = n log(n) sqrt(2)*.

Voici une première version naïve :

```
sage: u = lambda n: n * log(n) * sqrt(2)
sage: z = lambda n: exp(2 * I * pi * u(n))

sage: z(5)

sage: vertices = [0]
sage: for n in range(1,20):
....:     vertices.append(vertices[-1]+z(n))

sage: vertices[7]
```

Les calculs sont vraiment lents car symboliques (i.e. dans le parent `Symbolic Ring`). Pour améliorer les performances, ils faut utiliser des nombres flottants :

```
sage: pi_approx = pi.numerical_approx()
sage: sqrt2_approx = (2.0).sqrt()
sage: u_float = lambda n: n * (1.0*n).log() * sqrt2_approx
sage: z_float = lambda n: (2.0 * CDF(0,1) * pi_approx * u_float(n)).exp()
sage: u_float(5)

sage: z_float(5)

sage: vertices = [CDF(0)]
sage: for n in range(1,10000):
....:     vertices.append(vertices[-1]+z_float(n))
```

```
sage: vertices[7]
```

```
sage: line2d(vertices)
```

On peut aussi visualiser les points sur le même graphique (les objets graphiques peuvent être additionnés) :

```
sage: line2d(vertices) + point2d(vertices, color='red')
```

Pour comparer les temps de calcul :

```
sage: timeit("sum(z(n) for n in range(1,100))")
```

```
sage: timeit("sum(z_float(n) for n in range(1,100))")
```