Parent, element, coercion

Author: Thierry Monteil
Author: Vincent Delecroix

Here is a short introduction about parent and elements, you can find some more details at this tutorial (sagenb live / jupyter live)

The elements (numbers, matrices, polynomials, ...) have parents (integer ring, rationnal field, matrix spaces, polynomial rings, ...):

```
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()
```

Parents are objects with their methods:

```
sage: a = 3/2

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

sage: q.

sage: alg = q.algebraic_closure()

sage: alg.
```

This allows for example to add numbers of different types, by transforming them into elements of a common parent:

```
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()
```

Here is a more subtle example where the result of the operation is a different parent:

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

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

Note that equality testing is done in a common parent:

```
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
```

Could you guess, before evaluating the following lines, how will `N` look like and what will be its parent ?

```
sage: N

sage: M.parent()

sage: a.parent()

sage: N.parent()
```

Here is an example showing the importance to know how are objects represented. We want to plot the sequence of points given by *sum_{k=0}^{n-1} z_n* where *z_n = exp(2 i pi u_n)* and *u_n = n log(n) sqrt(2)*.

Here is a first naive version:

```
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]
```

The computation is very slow since they are symbolic (i.e. there are done within the `Symbolic Ring` parent). To accelerate, we should use floating-point numbers instead:

```
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)
```

We can also visualize the points on the same graphic (note that graphics objects can be added):

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

You can compare the timings:

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

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