Premiers pas avec Sage

1 Polynômes

1.1 Généralités

On commence par définir l'anneau de polynôme dans lequel on travaille :

```
sage: R.< indets > = PolynomialRing( base_ring )
```

où `indets` la liste des indéterminées, séparées par des virgules et `base_ring` l'anneau (ou le corps) de base (`ZZ` pour $\mathbb{Z}$, `QQ` pour $\mathbb{Q}$, `QQbar` pour $\overline{\mathbb{Q}}$, `RR` pour $\mathbb{R}$, `CC` pour $\mathbb{C}$, `IntegerModRing(n)` pour $\mathbb{Z}/n\mathbb{Z}$, `GF(p)` pour $\mathbb{F}_p, \dots$). Evidemment tout ce qui touche à la factorisation ou aux racines va dépendre de l'anneau de base. Par exemple pour travailler dans $\mathbb{F}_3[x, y]$ on envoie

```
sage: R.< x,y > = PolynomialRing( GF(3) )
```

Plus rapidement, on peut aussi taper l'une ou l'autre des commandes suivantes

```
sage: R.< x,y > = GF(3) ['x,y']  
sage: R.< x,y > = GF(3) []
```

Pour spécifier l'ordre monomial que l'on veut, par exemple `lex` (`lex` pour lexicographique, `deglex` pour lexicographique gradué et `degrevlex` pour lexicographique gradué inverse, avec `x>y` à chaque fois), on peut définir $\mathbb{F}_3[x, y]$ de la sorte :

```
sage: R.< x,y > = PolynomialRing( GF(3), ordre='lex' )
```

On vérifie que l'on a bien ce que l'on veut en évaluant `R` :

```
sage: R  
Multivariate Polynomial Ring in x, y over Finite Field of size 3
```

On peut aussi utiliser `print` ou `show` (plus joli) pour afficher `R` :

```
sage: print (R)  
Multivariate Polynomial Ring in x, y over Finite Field of size 3  
sage: show (R)  
 $\mathbb{F}_3[x, y]$ 
```

On peut ensuite définir des polynômes, par exemple $P(x, y) = 4x^2y - yx^2 = x^2y - yx^2 \in \mathbb{F}_3[x, y]$ et utiliser `print` ou `show` pour l'afficher.

```
sage: P = 4*x^2*y-x*y^2; print(P)
4*x^2*y - x*y^2
sage: show(P)
 $4x^2y - xy^2$ 
```

La substitution est naturelle :

```
sage: R.<x,y> = PolynomialRing(QQ)
      P = x^2+y+1
      P(2,1)
6
```

On peut considérer un polynôme de $k[x, y]$ comme un polynôme de $k[y][x]$ et inversement :

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: P = 4*x^2*y-x*y^2
      show(P)
      show(P.polynomial(x))
      show(P.polynomial(y))
 $4x^2y - xy^2$ 
 $4yx^2 - y^2x$ 
 $xy^2 + 4x^2y$ 
```

Pour connaître le degré du polynôme P :

```
sage: P.degree()
3
```

De la même manière, sur un anneau de polynômes à une indéterminée,

- connaître l'indéterminée : `P.variables()`
- coefficient de x^k : `P[k]`
- coefficient dominant : `P.leading_coefficient()`
- liste des coefficients : `P.coeffs()`
- liste des coefficients non nuls : `P.coefficients()`
- dérivée : `P.derivative()` ou `diff(P)`
- test d'irréductibilité : `P.is_irreducible()`
- factorisation sur le corps de base : `P.factor()`
- développement : `P.expand()` (attention par défaut, le polynôme est stocké sous forme développée et Sage renvoie une erreur si on lui demande de développer un polynôme déjà développé).
- racines dans le corps de base : `P.roots()`
- racines dans une extension (ici $\mathbb{C}$) : `P.roots(CC)`

1.2 Division euclidienne, pgcd...

Dans cette partie, on suppose que l'on a défini des polynômes A , B et C dans $k[x]$ où k est un corps commutatif.

- test de divisibilité : `A.divides(B)`
- division euclidienne de A par B : `P//Q, q,r = A.quo_rem(B)`
- A modulo B : `A%B`
- pgcd(A, B) : `A.gcd(B), gcd(A,B)`

- $\text{ppcm}(A, B) : A.\text{lcm}(B), \text{lcm}(A, B)$
- $\text{pgcd}(A, B, C), \text{ppcm}(A, B, C) : \text{gcd}([A, B, C]), \text{lcm}([A, B, C])$

2 Boucles, tests, fonctions

2.1 Comparaisons et opérateurs booléens basiques

- $x == y$: renvoie `true` si les objets x et y sont égaux, `false` sinon.
- $x <> y$ ou $x != y$: renvoie `true` si les objets x et y sont différents, `false` sinon.
- $x < y, x > y, x <= y, x >= y$: à moins d'être vraiment endormi, on comprend assez bien le principe...
- `and` : ET logique.
- `or` : OU logique. Si si.
- `not` : devine...

2.2 Boucles

La syntaxe générale est la suivante :

```
sage: while <expression> :
      <instruction>
      <instruction>
      .
      .
      .
```

L'indentation se fait avec la touche Tab ou avec des espaces. Attention, les instructions à l'intérieur de la boucle doivent être indentées de la même manière. Par exemple,

```
sage: i = 1
      while i < 10 :
          i = i+1
          print(i)
```

va renvoyer les entiers de 1 à 10, alors que

```
sage: i = 1
      while i < 10 :
          i = i+1
      print(i)
```

va renvoyer 10 et

```
sage: i = 1
      while i < 10 :
          i = i+1
          print(i)
```

va renvoyer une erreur.

2.3 Tests

Syntaxe générale :

```
sage: if <cond1> :  
      <inst1>  
      elif <cond2> :  
          <inst2>  
      elif <cond3> :  
          <inst3>  
      else :  
          <instn>
```

Comme pour les boucles, l'indentation doit être rigoureusement la même pour toutes les instructions de la branche.

2.4 Fonctions

Syntaxe générale, où bien entendu, `nom_fonction` est le nom de la fonction et `args` les arguments de la fonction, séparés par des virgules.

```
sage: def nom_fct (args) :  
      <instruction>  
      .  
      .  
      .  
      <instruction>  
      return <resultat>
```

Encore une fois, l'indentation est importante...