El objetivo de esta práctica es aprender a hacer pequeños programas en SAGE utilizando la instruccción `if` y los bucles `for` y `while`. Para evitar problemas en el uso de ciertos comando, comenzamos la sesión definiendo el anillo $\mathbb{Q}[x, y, z]$ con el orden lexicográfico:

```
R.<x,y,z>=PolynomialRing(QQ, 3, order='lex')
```

Ejercicio 1. Definir una función `is_in_I(f,F)` que reciba un polinomio `f` y una lista de polinomios `F`, todos en una variable, y devuelva `True` si `f` está en el ideal generado por los polinomios de `F` y `False` en caso contrario. Poner dos ejemplos con resultados distintos, en los que `f` tenga al menos grado 6 y las lista `F` tenga al menos tres polinomios.

La función `gcd(f,g)` definida en la primera sesión devuelve el máximo común divisor de dos polinomios. Si queremos calcular el máximo común divisor de los polinomios de una lista, posiblemente necesitemos usar un bucle `for`. El siguiente ejemplo, que calcula la suma de los enteros de una lista, puede ser útil:

```
def suma(L):
    sum=0
    for a in L:
        sum=a+sum
    return sum
suma([2,6,9,14,23,26,31,35])
| 146
```

También puede ser útil el comando `f.quo_rem(g)` que devuelve un par `(q,r)` donde `q` es el cociente de la división de `f` entre `g` y `r` es el resto.

```
(x^8-2*x^7+3* x^6-x^4+2*x^3+x-3).quo_rem(x^3-x+2)
|  (x^5 - 2*x^4 + 4*x^3 - 4*x^2 + 7*x - 10, 15*x^2 - 23*x + 17)
```

¡Ojo! Para SAGE la primera entrada de un par o de una lista es la 0:

```
(x^8-2*x^7+3* x^6-x^4+2*x^3+x-3).quo_rem(x^3-x+2)[0]
|  x^5 - 2*x^4 + 4*x^3 - 4*x^2 + 7*x - 10
(x^8-2*x^7+3* x^6-x^4+2*x^3+x-3).quo_rem(x^3-x+2)[1]
|  15*x^2 - 23*x + 17
```

Es posible que sea necesario usar la instrucción `if`, como se hace en el seguiente ejemplo que decide si la suma de los enteros de una lista es cero:

```
def suma_cero(L):
    if suma(L)==0:
        return True
    else:
        return False
suma_cero([2,6,9,14,23,26,31,35,-146])
|  True
```

Ejercicio 2. Definir una función `alg_div(f,F)` que reciba como entradas un polinomio `f` en tres variables y una lista `F` de polinomios, también en tres polinomios y, aplicando el algoritmo de división definido en la página 64 del libro, devuelva un par `[r,a]` donde `r` es el resto de la división y `a` es la lista **ordenada** de cocientes referentes a los polinomios de `F`. Aplicar la función a un caso en el que en `f` aparezcan las tres variables y tenga al menos cuatro monomios y `[F]` sea una lista de, al menos, tres polinomios. Volver a aplicar la función reordenando la lista `F`. Comprobar, en cada caso, que el algoritmo funciona, es decir, que

$$f = \sum_i a[i]F[i] + r.$$

El algoritmo de división, tal y como viene en el libro, es el siguiente

```
Input: f1, ..., fs, f
Output: a1, ..., as, r
a1 := 0; ... ; as := 0; r := 0
p := f
WHILE p <> 0 DO
     i := 1
     divisionoccurred := false
     WHILE i <= s AND divisionoccurred = false DO
          IF LT( fi ) divides LT(p) THEN
               ai := ai + LT(p)/LT( fi )
               p := p - (LT(p)/LT( fi )) fi
               divisionoccurred:= true
          ELSE
               i := i + 1
     IF divisionoccurred = false THEN
          r := r + LT(p)
          p := p - LT(p)
```

Como a priori no sabemos cuántos polinomios habrá en `F`, es posible que convenga crear una variable `a`, en lugar de las `ai`, que sea una lista. Inicialmente `a` es una lista de ceros. El comando `L.append(e)` añade el elemento `e` al final de la lista `L`. Por ejemplo

```
L=[]
for i in [1,..,7]:
    L.append(0)
L;
|  [0, 0, 0, 0, 0, 0, 0]
```

También puede ser útil la función `len(L)`, que devuelve la longitud de la lista `L`. El siguiente ejemplo, que calcula el factorial de un número, ayudará a comprender la sintaxis del bucle `while`:

```
def factorial(n):
    fact=1
    i=1
    while i<n+1:
        fact=i*fact
        i=i+1
    return fact
factorial(10)
|  3628800
```

Hay otras funciones que pueden ser necesarias. La función lógica `and` funciona como se espera. La función `R.monomial_divides(f,g)` devuelve `True` si el monomio `f` divide a `g`, y `False` en caso contrario. La función `R.monomial_quotient(f,g,coeff=True)` devuelve el cociente entre los monomios `f` y `g`. Para obtener el término líder de un polinomio `f` se usa la función `f.lt()`.

Se suponde que la última instrucción de la función será `return [r,a]`, donde `r` será el resto y `a` la lista de cocientes.

Tanto para el programa como para la posterior comprobación, se recuerda que para `SAGE` el primer elemento de cada lista está indexado por `0` y, si la longitud de la lista es `n`, el último elemento está indexado por `n-1`.

Al final de la sesión cada alumno la salvará y enviará su práctica a `miguelolalla@us.es`. El asunto del mensaje debe ser *Práctica 1 de SAGE* y en el cuerpo del mensaje debe aparecer el nombre completo de la alumna o el alumno.