TP 12- Proposition de solutions

Solution 1 Définition de polynômes :

```
Q=[1,0,1,3,-1]
R=[0,1,1,-1]
S=[0]*k+[1]
```

Solution 2 Fonction coeff

```
def coeff(P,i):
    if i<len(P):
        return P[i]
    else:
        return 0
```

Les liste en PYTHON est finie. Appeler un coefficient de rang au delà engendre un message d'erreur :

```
IndexError: list index out of range
```

Solution 3 Fonction degre

```
def degre(P):
    while len(P)>0 and abs(P[-1])<1e-15:
        P.pop()
    return len(P)-1
```

Par exemple :

```
>>>degre(Q)
4
>>>degre([]) # poly nul
-1
>>>degre([0,0,0]) # poly nul
-1
>>>degre([0,0,1,0])
2
```

Solution 4 Somme de polynômes

```
def somme(P,Q):
    P=P+[0]*(len(Q)-len(P))
    Q=Q+[0]*(len(P)-len(Q))
    S=[P[i]+Q[i] for i in range(len(P))]
    return S
```

ou

```
def somme(P,Q):
    S=[coeff(P,i)+coeff(Q,i) for i in
                range(max(len(P),len(Q)))]
    return S
```

Par exemple :

```
>>>somme(Q,R)
[1, 1, 2, 2, -1]
```

Solution 5 Produit de polynômes

```
def produit(P,Q):
    R=[]
    for i in range(len(P)+len(Q)-1):
        S=0
        for k in range(i+1):
            S=S+coeff(P,k)*coeff(Q,i-k)
        R.append(S)
    return R
```

Solution 6 Coefficient binomial

```
def puissance(P,n):
    R=[1]
    for i in range(n):
        R=produit(R,P)
    return R
```

Ce qui permet de récupérer les coefficients binomiaux en développant $(1+X)^n$: $\binom{n}{k}$ est le coefficient de degré k.

```
def parmi(k,n):
    P=[1,1]
    return puissance(P,n)[k]
```

En particulier :

```
>>>parmi(3,7)
35
```

Solution 7 Dérivée et primitive d'un polynôme

Dérivée d'un polynôme

```
def derive(P):
    R=[]
    for i in range(1,len(P)):
        R.append(i*P[i])
    return R
```

Par exemple :

```
>>>derive(Q)
[0, 2, 9, -4]
```

Primitive d'un polynôme

```
def integre(P):
    R=[0]
    for i in range(len(P)):
        R.append(P[i]/(i+1))
    return R
```

En particulier :

```
>>>integre(Q)
[0, 1.0, 0.5, -0.66..., 0.0, 0.6]
```

Solution 8 Évaluation basique :

```
def eval(P,x)
    e=0
    for i in range(0,len(P)):
        e=e+P[i]*x**i
    return e

P=[1,1,-2,0,3]
x=input('Donner x : ')
disp(eval1(P,x),'eval1')
```

➜ Soit n le degré de P, alors P a $n+1$ coefficients : `len(P)` vaut $n+1$:

- Nombre de multiplications : $2+\sum_{i=2}^{n} 1+i-1 = \frac{n(n+1)}{2}+1$
- Nombre d'additions : $\sum_{i=0}^{n} 1 = n+1$.
- Nombre d'affectations :$1+\sum_{i=0}^{n} 1 = n+2$

Solution 9 Optimisation du calcul de la puissance

```
def eval2(P,x):
    e,m=0,1
    for i in range(0,len(P)):
        e=e+P[i]*m
        m=m*x
    return e
```

➜ Soit n le degré de P, alors P a $n+1$ coefficents : `length(P)` vaut $n+1$:

- Nombre de multiplications : $\sum_{i=0}^{n} 2 = 2n+2$
- Nombre d'additions : $\sum_{i=0}^{n} 1 = n+1$.
- Nombre d'affectations : $2+\sum_{i=0}^{n} 2 = 2n+4$.

Solution 10 Algorithme de Hörner

```
def eval3(P,x):
    if degre(P)==-1:
        return 0
    s=P[-1]
    for a in P[-2::-1]:
        s=s*x+a
    return s
```

➜ Soit n le degré de P, alors P a $n+1$ coefficients : `length(P)` vaut $n+1$:

- Nombre de multiplications : $\sum_{i=0}^{n-1} 1 = n$
- Nombre d'additions : $\sum_{i=0}^{n-1} 1 = n$.
- Nombre d'affectations : $1+\sum_{i=0}^{n-1} 1 = n+1$.