Les algorithmes classiques

La somme des termes d'une liste

---

algorithme *somme*(L)
  L : liste de nombres
  Résultat : la somme des éléments de L (convention : 0 si L est vide)
début algorithme
  $n \longleftarrow$ *taille*(L)
  $s \longleftarrow 0$
  pour i **allant de** 0 **à** $n-1$ **faire**
    $s \longleftarrow s + L[i]$
  fin pour
  renvoyer s
fin algorithme

```
def somme(L):
    n = len(L)
    s = 0
    for i in range(0,n):
        # 0<=i<=n-1
        s = s+L[i]
    return s
```

◇

```
def somme(L):
    s = 0
    for x in L:
        s = s+x
    return s
```

◇

```
def somme(L):
    n = len(L)
    s = 0
    i = 0
    while i<n:
        s = s+L[i]
        i = i+1
    return s
```

◇

La moyenne et la variance d'une liste

---

algorithme *moyenne*(*L*)
  L : liste non vide de nombres
  Résultat : la moyenne des éléments de *L*
début algorithme
  renvoyer *somme*(*L*)/*taille*(*L*)
fin algorithme

```
def moyenne(L):
    return somme(L)/len(L)
```

◇

◇ La variance de la liste L est définie par :

$$\text{variance}(L) = \frac{1}{n}\sum_{k=0}^{n-1}(L[k]-m)^2$$

où n est la longueur de la liste L et m sa moyenne.

---

algorithme *variance*(*L*)
  L : liste non vide de nombres
  Résultat : la variance des éléments de *L*
début algorithme
  $n \longleftarrow$ *taille*(*L*)
  $s \longleftarrow 0$
  pour i **allant de** 0 **à** $n-1$ **faire**
    $s \longleftarrow s + (L[i] - moyenne(L))^2$
  fin pour
  renvoyer s/n
fin algorithme

```
def variance(L):
    n = len(L)
    s = 0
    for i in range(0,n):
        # 0<=i<=n-1
        s = s+(L[i]-moyenne(L))**2
    return s/n
```

◇

---

algorithme $variance(L)$
L : liste non vide de nombres
Résultat : la variance des éléments de L
début algorithme
$n \longleftarrow taille(L)$
$m \longleftarrow moyenne(L)$
$s \longleftarrow 0$
pour i **allant de** 0 **à** $n-1$ **faire**
$s \longleftarrow s+(L[i]-m)^2$
fin pour
renvoyer s/n
fin algorithme

```
def variance(L):
    n = len(L)
    m = moyenne(L)
    s = 0
    for i in range(0,n):
        # 0<=i<=n-1
        s = s+(L[i]-m)**2
    return s/n
```

◇

```
def variance(L):
    n = len(L)
    m = moyenne(L)
    s = 0
    i = 0
    while i<n:
        s = s+(L[i]-m)**2
        i = i+1
    return s/n
```

◇

Le maximum d'une liste

---

algorithme *maximum*(L)
  L : liste non vide de nombres réels (ou entiers)
  Résultat : le maximum des éléments de L
début algorithme
  $n \longleftarrow$ *taille*(L)
  $m \longleftarrow L[0]$
  pour i **allant de** 1 **à** $n-1$ **faire**
    si $L[i] > m$ **alors**
      $m \longleftarrow L[i]$
    fin si
  fin pour
  renvoyer m
fin algorithme

```
def maximum(L):
    n = len(L)
    m = L[0]
    for i in range(1,n):
        # 1<=i<=n-1
        if L[i]>m:
            m = L[i]
    return m
```

◇

```
def maximum(L):
    n = len(L)
    m = L[0]
    i = 0
    while i<n:
        if L[i]>m:
            m = L[i]
        i = i+1
    return m
```

◇

Nombre d'apparitions d'un élément dans une liste

---

algorithme *compte*(x, L)
  x : élément quelconque
  L : liste
  Résultat : le nombre de fois (éventuellement 0) où x apparait dans L
début algorithme
  $n \longleftarrow$ *taille*(L)
  compteur $\longleftarrow 0$
  pour i **allant de** 0 **à** $n-1$ **faire**
    si $x = L[i]$ **alors**
      compteur $\longleftarrow$ *compteur* $+1$
    fin si
  fin pour
  renvoyer *compteur*
fin algorithme

```
def compte(x,L):
    n = len(L)
    compteur = 0
    for i in range(0,n):
        # 0<=i<=n-1
        if L[i]==x:
            compteur = compteur+1
    return compteur
```

◇

```
def compte(x,L):
    compteur = 0
    for y in L:
        if y==x:
            compteur = compteur+1
    return compteur
```

◇

```
def compte(x,L):
    n = len(L)
    compteur = 0
    i = 0
    while i<n:
        if L[i]==x:
            compteur = compteur+1
        i = i+1
    return compteur
```

◇

Appartenance d'un élément à une liste

---

algorithme *appartient*(x, L)
  x : élément quelconque
  L : liste
  Résultat : *vrai* si x apparait dans L, *faux* sinon
début algorithme
  $n \longleftarrow$ *taille*(L)
  trouvé $\longleftarrow$ *faux*
  $i \longleftarrow 0$
  tant que $i < n$ **et** *trouvé* $=$ *faux* **faire**
    si $x = L[i]$ **alors**
      trouvé $\longleftarrow$ *vrai*
    fin si
  fin tant que
  renvoyer *trouvé*
fin algorithme

```
def appartient(x,L):
    n = len(L)
    trouvé = False
    i = 0
    while i<n and (not trouvé):
        if L[i]==x:
            trouvé = True
    # À ce point, on a :
    # - soit trouvé=True, dans ce cas x apparait dans L
    # - soit trouvé=False, dans ce cas i=n et x n'apparait pas dans L
    return trouvé
```

◇

```
def appartient(x,L):
    n = len(L)
    trouvé = False
    for i in range(0,n):
        if L[i]==x:
            trouvé = True
    return trouvé
```

◇

```python
def appartient(x,L):
    n = len(L)
    for i in range(0,n):
        if L[i]==x:
            return True
            # Si on a trouvé x, la boucle s'arrête ici
    # fin de la boucle for
    # À ce point, on a parcouru toute la liste sans renvoyer True
    # C'est donc que x n'apparait pas dans L
    return False
```

◇

Calcul « naïf » de x^n (sans utiliser **)

---

algorithme *puissance*(x, N)
  x : nombre
  N : nombre entier positif
  Résultat : x^N (convention : $x^0 = 1$)
début algorithme
  $a \longleftarrow 1$
  pour i **allant de** 0 **à** $n-1$ **faire**
    $a \longleftarrow a \times x$
  fin pour
  renvoyer a
fin algorithme

```python
def puissance(x,N):
    a = 1
    for i in range(0,n):
        # 0<=i<=n-1
        a = a*x
    return a
```

◇

```python
def puissance(x,N):
    a = 1
    i = 0
    while i<n:
        a = a*x
    return a
```

◇