

```

# La boucle while dans une liste ou un tableau.py

001 | # Le cas du parcours complet d'une liste, i.e. équivalent à for
002 | def somme(t):
003 |     somme = 0
004 |     i = 0 # initialisation de l'indice
005 |     while i < len(t): # inégalité stricte
006 |         somme = somme + t[i]
007 |         i = i + 1 # incrémentation de l'indice
008 |     return somme
009 |
010 | assert somme([1,2,3,4]) == 10
011 |
012 | def somme_dec(t):
013 |     somme = 0
014 |     i = len(t)-1 # initialisation de l'indice
015 |     while i > -1:
016 |         somme = somme + t[i]
017 |         i = i - 1 # décrémentation de l'indice
018 |     return somme
019 |
020 | assert somme_dec([1,2,3,4]) == 10
021 |
022 | ## Le cas du parcours avec arrêt sous condition, retour booléen
023 | def est_element(x,t):
024 |     i = 0
025 |     while i < len(t) and t[i] != x: # évaluation paresseuse
026 |         i = i + 1
027 |     return i < len(t) # t[i] == x donne un out of range !
028 |
029 | assert est_element(3,[1,2,3,4]) == True
030 | assert est_element(5,[1,2,3,4]) == False
031 |
032 | def est_element(x,t):
033 |     i = len(t)-1 # initialisation de l'indice
034 |     while i > -1 and t[i] != x: # évaluation paresseuse
035 |         i = i - 1
036 |     return i > -1 # t[i] == x donne un out of range !
037 |
038 | assert est_element(3,[1,2,3,4]) == True
039 | assert est_element(5,[1,2,3,4]) == False
040 |
041 | ## Le cas du parcours avec arrêt sous condition, retour valeur
042 | def position(x,t):
043 |     i = 0 # initialisation de l'indice
044 |     while i < len(t) and t[i] != x: # évaluation paresseuse
045 |         i = i + 1 # incrémentation de l'indice
046 |     if i < len(t):
047 |         return i # cas où la recherche aboutit
048 |     return -1 # cas où la recherche n'aboutit pas
049 |
050 | assert position(3,[1,2,3,4]) == 2
051 | assert position(5,[1,2,3,4]) == -1
052 |
053 | def position(x,t):
054 |     i = len(t)-1 # initialisation de l'indice
055 |     while i > -1 and t[i] != x: # évaluation paresseuse
056 |         i = i - 1 # incrémentation de l'indice
057 |     if i > -1:
058 |         return i # cas où la recherche aboutit
059 |     return -1 # cas où la recherche n'aboutit pas
060 |
061 | assert position(3,[1,2,3,4]) == 2
062 | assert position(5,[1,2,3,4]) == -1
063 |
064 | ## Le cas du parcours avec arrêt sous condition, décompte nombre de zéros à droite
065 | # Exemple 1
066 | def nombre_zeros_droite(t,i):
067 |     c = 0
068 |     j = i # initialisation de l'indice
069 |     while j < len(t) and t[j] == 0: # évaluation paresseuse
070 |         c = c + 1
071 |         j = j + 1 # incrémentation de l'indice
072 |     return c
073 |
074 | # version ultra-optimisée

```

```

075 | # c et j-i varient de la même façon: on pose c = j-i càd. j = i+c
076 | def nombre_zeros_droite(t,i):
077 |     c = 0 # initialisation du compteur
078 |     while i+c < len(t) and t[i+c] == 0: # évaluation paresseuse
079 |         c = c + 1 # incrémentation du compteur
080 |     return c
081 |
082 | #ou
083 | def nombre_zeros_droite(t,i):
084 |     c = i # initialisation du compteur
085 |     while c < len(t) and t[c] == 0: # évaluation paresseuse
086 |         c = c + 1 # incrémentation du compteur
087 |     return c - i
088 |
089 | liste = [0,0,1,1,0,0,0,1,0,0,0,1,1]
090 | assert nombre_zeros_droite(liste,3) == 0
091 | assert nombre_zeros_droite(liste,5) == 2
092 | assert nombre_zeros_droite(liste,20) == 0
093 |
094 | # Exemple 2
095 | def calculeL(hist,i):
096 |     j = i
097 |     while j > -1 and hist[j] >= hist[i]:
098 |         j = j - 1
099 |     return j + 1
100 |
101 | hist = [3,1,2,4,2,2,0,1]
102 | assert calculeL_indice(hist,2) == 2
103 | assert calculeL_indice(hist,5) == 2
104 |
105 | #ou
106 | def calculeL(hist,i):
107 |     j = i
108 |     while j > 0 and hist[j-1] >= hist[i]:
109 |         j = j - 1
110 |     return j
111 |
112 | hist = [3,1,2,4,2,2,0,1]
113 | assert calculeL_indice(hist,2) == 2
114 | assert calculeL_indice(hist,5) == 2
115 |

```