Numériques et Sciences de l’Informatique

DST d’NSI

Appréciations :

Note :

\_\_\_\_\_\_\_\_\_\_\_\_

Signature :

# PARTIE A :

## Exercice 1 :

1)

|  |  |  |
| --- | --- | --- |
| Index | Code | Description |
| 1 | def rendu\_monnaie(pieces,s) : | #On initialise la fonction |
| 2 | """renvoie le nombre minimal de pièces pour faire la somme s avec le système pièces""" | #On indique ajoute une légende pour la fonction. |
| 3 | if s==0: | #Si la somme initiale est 0… |
| 4 | return 0 | #On retourne la valeur 0 car il n'y a rien à rendre |
| 5 | r=s #s=1+1+...+1 dans le pire des cas | #En revanche, si la valeur est non nulle on attribue la valeur s à la variable r (si l'on reprends les valeurs dans l'exemple plus haut, r sera égal à 3) |
| 6 | for p in pieces: | #Pour chaque valeur dans la liste pieces (ici: [1,2]) on attribut l'une de ces valeur, au premier tour p sera égal à 1, au deuxième tour, p sera égal à 2. |
| 7 | if p<=s: | #on vérifie que la somme à calculer est bien supérieur à la valeur choisie dans la liste. |
| 8 | r = min ( r , 1 + rendu\_monnaie (pieces,s-p)) | #on initialise r avec la plus petite valeur de la récursive du programme, c'est-à-dire, la valeur actuelle de r et la fonction actuelle avec la liste pieces et la somme initiale à laquelle on a soustrait la valeur à tester. |
| 9 | return r | #on retourne le résultat final (r) |
| 10 | #L’appel |  |
| 11 | rendu\_monnaie(pieces,s) | #ligne rentrée pour lancer la fonction rendu\_monnaie(pieces,s)#ici pieces=[1,2] et s=3 |

Cf : Tableau Excel, « PARTIE\_A,Exercice 1,1,Arbre\_Binaire »

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Branche 0 |  |  |  |  | 3 |  |  |  |
|  |  |  | "3-1" |  |  |  | "3-2" |  |
| Branche 1 |  |  | 2 |  |  |  | 1 |  |
|  |  | "2-1" |  | "2-2" |  | "1-1" |  | "1-2" |
| Branche 2 |  | 1 |  | 0 |  | 0 |  | -1 |
|  | "1-1" |  |  |  |  |  |  |  |
| Branche 3 | 0 |  |  |  |  |  |  |  |

2)

Cf : Excel, « PARTIE\_A,Exercice 1,2,Calculs\_redondants »

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Branche 0 |  |  |  |  | 3 |  |  |  |
|  |  |  | "3-1" |  |  |  | "3-2" |  |
| Branche 1 |  |  | 2 |  |  |  | 1 |  |
|  |  | "2-1" |  | "2-2" |  | "1-1" |  | "1-2" |
| Branche 2 |  | 1 |  | 0 |  | 0 |  | -1 |
|  | "1-1" |  |  |  |  |  |  |  |
| Branche 3 | 0 |  |  |  |  |  |  |  |

Il a appelé deux fois s = 1 et trois fois s = 0

## Exercice 2 :

1)

|  |  |  |
| --- | --- | --- |
| Index | Code | Description |
| 1 | def rendu\_monnaie(pieces,s) : | #On initialise la fonction |
| 2 | """renvoie le nombre minimal de pièces pour faire la somme s avec le système pièces""" | #On indique ajoute une légende pour la fonction. |
| 3 | nb=[0]\*(s+1) | # on initialise la liste nb avec s+1 0. |
| 4 | for n in range(1,s+1): | #on boucle n fois s+1 en partant de 1, (au lieu de 0 en par défaut) |
| 5 | nb[n]=n #n=1+1+1+...+1 dans le pire des cas | #on écrase la valeur n de la liste avec le résultat n |
| 6 | for p in pieces: | #Pour chaque valeur dans la liste pieces (ici: [1,6,10]) on attribut l'une de ces valeur, au premier tour p sera égal à 1, au deuxième tour, p sera égal à 6, et au troisième tour, p sera égal à 10. |
| 7 | if p <=n: | #on vérifie que la somme à calculer est bien supérieur à la valeur choisie dans la liste. |
| 8 | nb[n]=min(nb[n],1+nb[n-p]) | #on écrase la valeur de la liste à la position n par la plus petite valeur du calcule 1+nb[n-p] |
| 9 | return nb[s] | #on retourne le résultat final nb[s] |
| 10 | rendu\_monnaie(pieces,s) | #Pour une liste de possibilité de pièces (Par exemple: [1,6,10]) et une somme initial à calculer (Par exemple:12) |

Le programme retourne le tableau : [0, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1]

Comme le dernier membre de la liste est celui qui sera retourné par le programme, ce dernier retourneras donc 1.

4)

|  |  |
| --- | --- |
| Index | Program |
| 1256789101112 | def rendu\_monnaie(pieces,s) :"""renvoie le nombre minimal de pièces pour faire la somme s avec le système pièces"""nb=[0]\*(s+1)for n in range(1,s+1):nb[n]=n #n=1+1+1+...+1 dans le pire des casfor p in pieces:if p <=n:nb[n]=max(nb[n],1+nb[n-p])return nb[s]rendu\_monnaie(pieces,s) |

5)

6)

|  |  |
| --- | --- |
| Index | Program |
| 12345678910111213141516171819202122232425 | def aligne\_sequence(s1,s2):"""Le score du meilleur alignement de s1 et s2"""GE=[" ","G","E","N","O","M","E"]n1,n2=len(s1),len(s2)sc=[[0]\*(n2+1) for \_ in range(n1+1)]for i in range(1,n1+1):sc[i][0]=-ifor j in range(1,n2+1):sc[0][j]=-jfor i in range(1,n1+1):for j in range(1,n2+1):s=max(-1+sc[i-1][j],-1+sc[i][j-1])if s1[i-1]==s2[j-1]: sc[i][j]=max(s,1+sc[i-1][j-1])else: sc[i][j]=max(s,-1+sc[i-1][j-1])print("E N O R M E")for b in range(len(sc)):for f in range(len(sc[b])):length=len("{}".format(sc[b][f]))if length==1: print("| {}".format(sc[b][f]),end=" ")else: print("|{}".format(sc[b][f]),end=" ")print("|",end="")print("{}".format(GE[b]))return sc[n1][n2]word1="ENORME"word2="GENOME"print(aligne\_sequence(word2,word1)) |

7)

|  |  |
| --- | --- |
| Index | Program |
| 1234567891011121314151617181920212223 | def aligne\_sequence(s1,s2):"""Le score du meilleur alignement de s1 et s2"""GE=[" ","G","E","N","O","M","E"]n1,n2=len(s1),len(s2)sc=[[0]\*(n2+1) for \_ in range(n1+1)]for i in range(1,n1+1):sc[i][0]=-ifor j in range(1,n2+1):sc[0][j]=-jprint("E N O R M E")for i in range(1,n1+1):print("{}".format(GE[b]),end="")for j in range(1,n2+1):s=max(-1+sc[i-1][j],-1+sc[i][j-1])if s1[i-1]==s2[j-1]: sc[i][j]=max(s,1+sc[i-1][j-1])else: sc[i][j]=max(s,-1+sc[i-1][j-1])print("|{:>3}".format(sc[i][j]),end=" ")print()return sc[n1][n2]word1="-ENORME"word2="GENO-ME"print(aligne\_sequence(word2,word1)) |

# PARTIE C :

## Exercice 1 :

1)

|  |  |
| --- | --- |
| Index | Program |
| 123456789101112131415161718 | def fibod(n): N1=[0] N2=[0] while n>0:  if n==0 or n==1:   return n  else:   if n not in N1 and n not in N2:    n1=(n-1)    N1.append(n1)    n2=(n-2)    N2.append(n2)    n=n1+n2    print(n)    return (fibod(n-1)+fibod(n-2))   else:    return nprint(fibod(7)) |

1.1) autre façon possible pour le premier

|  |  |
| --- | --- |
| Index | Program |
| 12345678910 | def fibo(n):fist=[0]if n==0 or n==1: return nelse:for i in range(len(fist)):if ((n-1)+(n-2))==fist[i]:continueelse:fist.append((n-1)+(n-2))return fibo(n-1)+fibo(n-2)print(fibod(7)) |

2) autre façon possible pour le premier

|  |  |
| --- | --- |
| Index | Program |
| 12345678910 | def fibo(n):fist=[0]if n==0 or n==1: return nelse:for i in range(len(fist)):if ((n-1)+(n-2))==fist[i]:continueelse:fist.append((n-1)+(n-2))return fibo(n-1)+fibo(n-2)print(fibod(7)) |