

# Projet Algorithme Avancé

## Analyse

**Groupe 4 :**

*Justin Ferdinand*

*Maël Rhuin*

*Baptiste Risse*

# Sommaire



CDCF et Analyse descriptive de niveau 0 .....	p3<>p7
• CDCF.....	p3
• Première approche.....	p4
• Contraintes.....	p5
• Raisonnement.....	p6
• Fonctions principales.....	p7
Arbre Hiérarchique Fonctionnel.....	p8
Analyse descriptive de niveau 1.....	p9<>p12
Analyse descriptive de niveau 2.....	p13<>p17
Analyse descriptive de niveau 3.....	p18<>p19
Ordre de programmation.....	p20

# CDCF

---

**Sujet :** Transcrire en toutes lettres un nombre entre  
1 centimes et 1 milliards d'euros – 1 centime.

# Analyse descriptive de niveau 0

## Première approche

---

Le sujet porte sur la rédaction en toutes lettres d'un nombre entre 1 centime et 999 999 999,99 millions d'euros. Ce dernier requiert d'appliquer les règles du français mis à part les « contraintes principales données par le sujet :

- La gestion des traits d'unions n'est pas demandée
- Le pluriel d'euro est autorisé

Il faut gérer les divers nombres de façon intelligente donc veiller à une décomposition en conséquence pour pouvoir retranscrire en toutes lettres, c'est-à-dire :

- Les cas généraux : gérer les millions, les milliers, les centaines, les dizaines, les unités, ...
- Les cas particuliers : soixante et onze, quatre vingt dix, douze ...
- Les règles de français : quatre cents / quatre cent soixante, ...

# Contraintes

---

Ce sujet impose l'application des règles du français pour les nombres, à savoir :

-> La marque du pluriel pour 20 et 100 : seulement s'ils terminent le nombre ou multiplie les chiffres suivants sinon pour les autres il prenne la marque classique du pluriel « s ».

-> Le cas de 1000 : « mille » est invariable sauf pour une distance.

-> Le « et » : utilisé lorsque le nombre est > 20, vingt et un / trente et un ... (mis à part pour les nombres > 69\*)

-> Les cas de 1 : se dit onze lorsque le nombre est 11, 71 et 91 sinon se dit « un ».

-> Pour les nombres entre 70 et 90 : il faut gérer les « dix », « soixante » + « dix ».

-> Les nombres suivants : 11 12 13 14 15 16 17 18 19

-> Mai aussi : 20 30 40 50 60 80

\*2 derniers chiffres de la partie du nombre traitée.

Il faut aussi gérer des contraintes utilisateurs c'est-à-dire gérer et conditionner la saisie :

-> Il faut que ce soit un nombre de type réel compris en 0,01 et 999 999 999, 99 (euros).

La saisie doit donc ressembler à : pour 3 centimes -> 0,03 / pour 3 euros et 4 centimes -> 3,04

# Raisonnement

9	9	9	.	9	9	9	.	9	9	9	,	9	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---

Il faut vérifier le nombre => FP3

On décompose ensuite ce nombre en un tableau de 4 cases

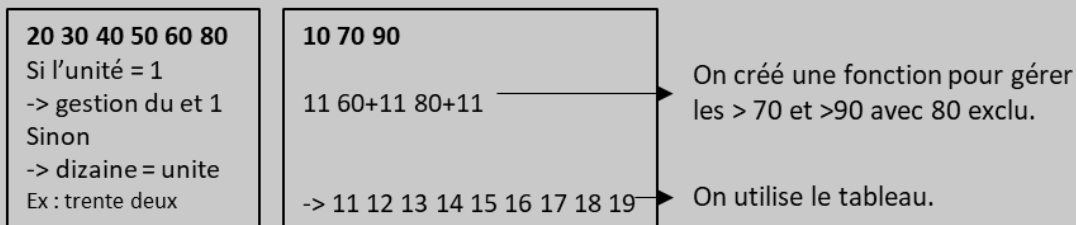
=> FP 1

[	999	;	999	;	999	;	99	]
---	-----	---	-----	---	-----	---	----	---

On traite et traduit ce tableau case par case => FP 2

Puis on redécompose chaque case : FS2.1 =>

999	=>	9	9	9
-----	----	---	---	---



Ensuite, on concatène les centaines avec les dizaines/unites

=> Si il n'y a rien après les centaines ou vingtaines, on met un « s »

Enfin, on recompose pour afficher => FP4

M -> million €

m -> millier €

c -> centaine €

u -> centime €

<b>M</b>	M > 0 million(s)	<b>m</b>	m > 0 millier(s)	<b>c</b>	c > 0 euro(s)	<b>u</b>	u > 0 centime(s)
----------	---------------------	----------	---------------------	----------	------------------	----------	---------------------

Chaque case est affichée si sa valeur > 0 \*

On met « un » si la valeur >= 2

Sens d'affichage : gauche -> droite

\*on reprend le tableau issu de FP1

Pour afficher on reprend le tableau de FP2

# Les Fonctions Principales

---

Pour gérer toutes les contraintes vu précédemment, on répartit les tâches selon plusieurs fonctions principales :

-> Une fonction qui décompose (FP 1 : **decompose\_Nombre\_Initial( )**) en 4 parties :

millions                  milliers                  centaines          centimes

999 999 999 , 99

-> Une fonction qui traite (FP 2 : **traite\_Nombre\_Decompose( )**) le nombre précédemment décomposé :

Centaine                  Dizaine                  Unité

9 9 9

Cette fonction aura des fonctions secondaires pour traiter les différents cas de la langue française.

-> Une fonction qui vérifie la validité de la saisie (FP 3 : **saisie\_verification\_Nombre( )**) :

?> **974,52**                  <- saisie valide

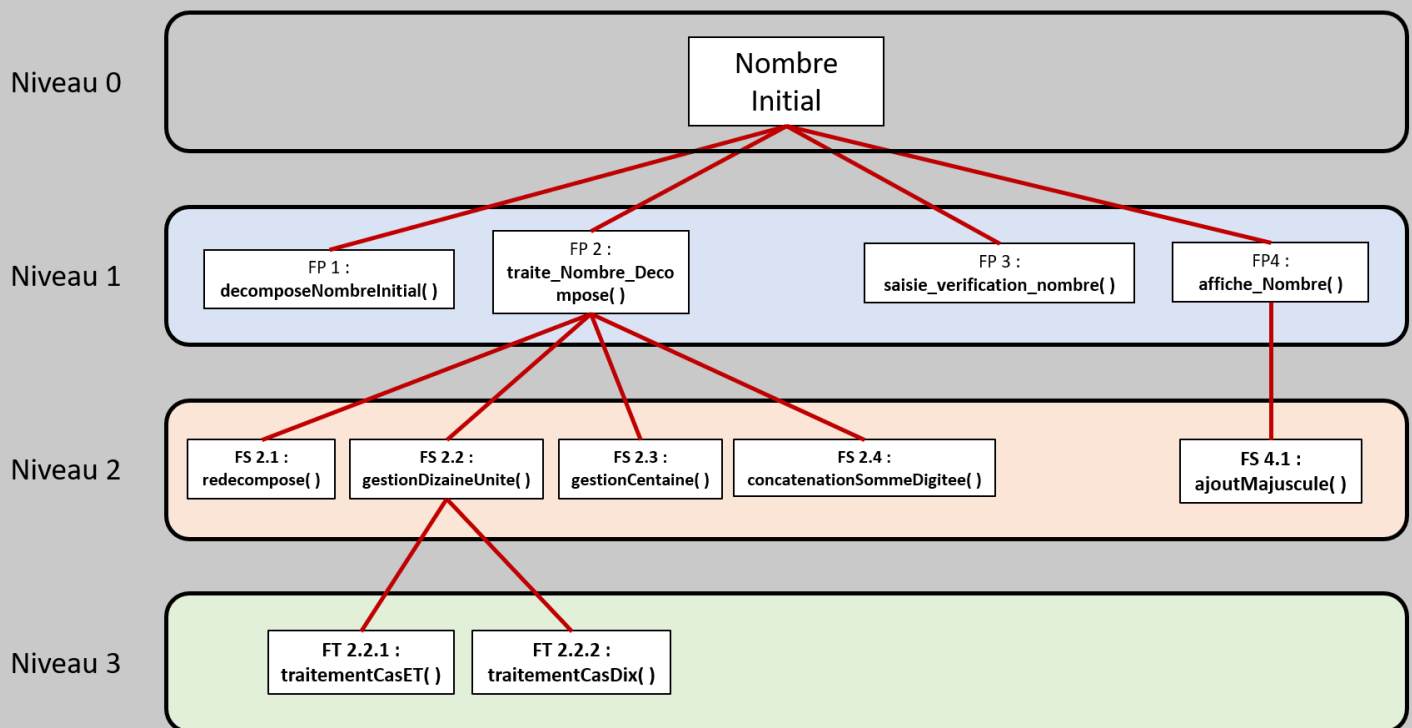
?> **paem,87**          <- saisie invalide

-> Une fonction qui affiche le résultat du nombre qui est dès lors écrit en toutes lettres (FP4 : **affiche\_Nombre( )**):

?> **89,04**

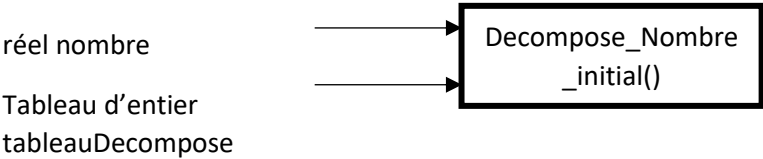
?> **Quatre vingt neuf euros et quatre centimes.**

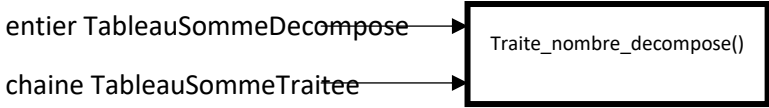
# Arbre Hiérarchique fonctionnel

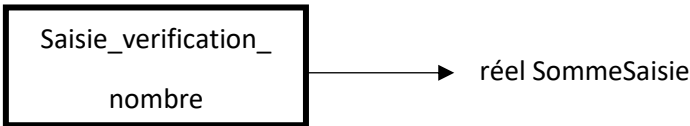




# Analyse descriptive de niveau 1

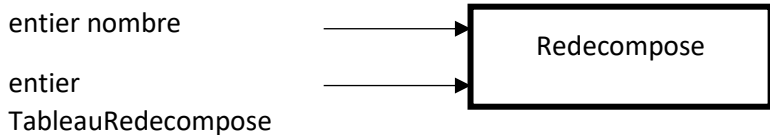
<p><b>FP1</b> <b>decompose_Nombre_Initial()</b></p>	<p><b>21/11/2021</b> <b>Justin FERDINAND</b></p>
<p><b>Valeur Ajoutée</b></p> <p>Ce service permet de décomposer le nombre initial dans un tableau d'entiers de 4 cases qui représente les millions, milliers, unités et centimes.</p>	
<p><b>INPUT</b></p> <p>Val réel nombre, adr entier tableauDecompose[4]</p>	<p><b>OUTPUT</b></p> <p>-----</p>
<p><b>Logigramme Fonctionnel</b></p>  <pre> graph LR     A[réel nombre] --&gt; C[Decompose_Nombre_initial()]     B[Tableau d'entier tableauDecompose] --&gt; C     style C stroke-width:4px     </pre>	
<p><b>Service Fonctionnel</b></p> <pre> Procédure decomposeNombreInitial(val réel nombre, val entier TableauDecompose[4]) {   entier partie_million_millier &lt;- nombre / 1000   entier partie_centaine_centime &lt;- ((nombre - (partie_million_millier*1000)) * 100) + 0.1   entier partie_centaine_centime &lt;- (int)(inter)   entier indiceActuel &lt;- 0   Pour(indiceActuel allant de 0 à 3) TableauDecompose[indiceActuel] &lt;- 0    TableauDecompose[0] &lt;- (partie_million_millier / 1000)   TableauDecompose[1] &lt;- (partie_million_millier % 1000)    TableauDecompose[2] &lt;- (partie_centaine_centime / 100)   TableauDecompose[3] &lt;- (partie_centaine_centime % 100) } </pre>	
<p><b>Commentaire</b></p> <p>Le nombre est divisé en deux parties par soucis d'erreur de manipulation des grands nombres avec les opérateurs numériques comme % modulo.</p>	

<b>FP2</b> <b>traite_Nombre_Decompose</b>	<b>21/11/2021</b> <b>Baptiste RISSE</b>
<b>Valeur Ajoutée</b>  Ce service permet de traiter le nombre saisi de l'utilisateur une fois que celui-ci a été décomposé. Il retranscrit en toute lettre les valeurs du tableau passé en paramètre dans un tableau de string à 4 cases.	
<b>INPUT</b>  adr entier TableauSommeDecompose[4],adr chaine TableauSommeTraitee[4]	<b>OUTPUT</b>  -----
<b>Logigramme Fonctionnel</b>   <pre> graph LR     A[entier TableauSommeDecompose] --&gt; C[Traite_nombre_decompose()]     B[chaine TableauSommeTraitee] --&gt; C           </pre>	
<b>Service Fonctionnel</b>  <pre> Procédure traiteNombreDecompose(val entier TableauSommeDecompose[4],val chaine TableauSommeTraitee[4]) {   Entier indiceActuel &lt;- 0   Pour indiceActuel allant de 0 à 3 {     entier TableauRedecomposeInter[3]     redecoupe(TableauSommeDecompose[indiceActuel], TableauRedecomposeInter)      chaine TableauTraiteeInter[3]     TableauTraiteeInter[0] &lt;- gestionCentaine(TableauRedecomposeInter[0])      chaine TableauGestionDizaineUnite[2]     gestionDizaineUnite(TableauRedecomposeInter, TableauGestionDizaineUnite)      TableauTraiteeInter[1] &lt;- TableauGestionDizaineUnite[0]     TableauTraiteeInter[2] &lt;- TableauGestionDizaineUnite[1]      Chaine SommeTraiteeDigitée = concatenationSommeDigitée(TableauTraiteeInter, TableauRedecomposeInter);     TableauSommeTraitee[indiceActuel] &lt;- SommeTraiteeDigitée   } }           </pre>	
<b>Commentaire</b>  -----	

<b>FP3</b> <b>SaisieNombreEtVerification()</b>	<b>21/11/2021</b> <b>Justin FERDINAND</b>
<b>Valeur Ajoutée</b>  Ce service permet à l'utilisateur de saisir un nombre et de vérifier que le nombre saisi soit bien compris entre 0.01 et 999 999 999, 99.	
<b>INPUT</b>  -----	<b>OUTPUT</b>  réel sommeSaisie
<b>Logigramme Fonctionnel</b>   <pre> graph LR     A[Saisie_verification_nombre] --&gt; B[réel SommeSaisie]           </pre>	
<b>Service Fonctionnel</b>  <pre> Fonction saisieNombreEtVerification() {   chaine message &lt;- "Somme saisie invalide, veuillez retaper dans le bon format votre somme : "   réel sommeSaisie   Afficher "Saisissez une somme comprise entre 1 centime et 1 milliard d'euros - 1 centime : "   Tant que ( !(demande supérieur à sommeSaisie) ou sommeSaisie inférieur à 0.01 ou sommeSaisie supérieur ou égal à 1000000000 ) {     si demande vide alors retourner faux     sinon       si la demande rate {         afficher message         vider demander         ignorer l'erreur       } sinon {         Afficher message       }     }   }   retourner sommeSaisie }           </pre>	
<b>Commentaire</b> Cin permet de demander Clear permet de vider le contenu de cin Ignore supprime la ligne éronnée	

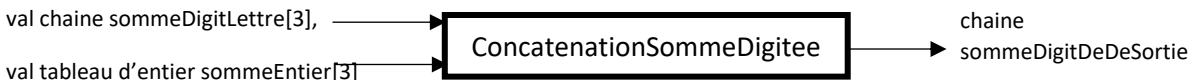
<b>FP4</b> <b>Affiche_nombre</b>	<b>21/11/2021</b> <b>Maël Rhuin</b>
<b>Valeur Ajoutée</b>  Ce service permet d'afficher le nombre en toute lettre saisi par l'utilisateur après une re manipulation du tableau de string de FP2.	
<b>INPUT</b>  Adr chaine TableauSommeEnLettre[4], Adr entier TableaulInitial[4], val entier sommeInitial	<b>OUTPUT</b>  -----
<b>Logigramme Fonctionnel</b>  <div> <div> chaîne tableauSommetEnLettre  entier tableaulInitial[4]  entier sommeInitial </div> <div> <div></div> <div></div> <div></div> </div> <div> Affiche_nombre </div> </div>	
<b>Service Fonctionnel</b>  <pre> procédure affichage(chaine TableauSommeEnLettre[4], entier TableaulInitial[4], réel sommeInitial) {    ajoutMajuscule(TableauSommeEnLettre, TableaulInitial)   chaîne sommeFinalEnTouteLettre = ""   si (la longueur du TableauSommeEnLettre[0] est supérieure à 0) {     sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre + TableauSommeEnLettre[0] +     ((TableaulInitial[0] supérieur à 1) ? "millions" : "million") + " "     si ((TableaulInitial[1] égal à 0) et (TableaulInitial[2] égal à 0))       sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre + "d"   }   si (la longueur du TableauSommeEnLettre[1] est supérieure à 0)     sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre +TableauSommeEnLettre[1] + "mille" + " "   si (la longueur du TableauSommeEnLettre[2] est supérieur à 0)     sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre + TableauSommeEnLettre[2]   si (sommeInitial supérieur ou égal à 1) {     sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre + si (sommeInitial égal à 1)     alors "euro" sinon "euros"   }   si (la longueur du TableauSommeEnLettre[3] supérieur à 0)     sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre + " et " + TableauSommeEnLettre[3] +   si ((TableaulInitial[3] supérieur à 1) alors "centimes." sinon "centime.")     sinon sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre + "."   }   si (sommeInitial inférieur à 1 et la longueur du TableauSommeEnLettre[3] est supérieure à 0) alors     sommeFinalEnTouteLettre &lt;- sommeFinalEnTouteLettre + TableauSommeEnLettre[3] +   si((TableaulInitial[3] supérieur à 1) alors "centimes d'euro." sinon "centime d'euro.")    afficher "Somme retranscrite : " + sommeFinalEnTouteLettre } </pre>	
<b>Commentaire</b>  -----	

# Analyse descriptive de niveau 2

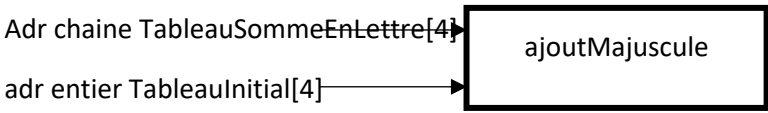
FS2.1 redecompose()	21/11/2021 Justin FERDINAND
<p><b>Valeur Ajoutée</b></p> <p>Ce service permet de décomposer chaque case du tableau issu de FP1 en un sous tableau de 3 cases comme suit : centaine – dizaine – unité.</p>	
<p><b>INPUT</b></p> <p>Val entier nombre, adr entierTableauRedecompose[3]</p>	<p><b>OUTPUT</b></p>
<p><b>Logigramme Fonctionnel</b></p>  <pre> graph LR     A[entier nombre] --&gt; C[Redecompose]     B[entier TableauRedecompose] --&gt; C     </pre>	
<p><b>Service Fonctionnel</b></p> <pre> Procédure redecompose(val entier nombre, adr entier TableauRedecompose[3]) {   TableauRedecompose[2] &lt;- nombre modulo 10   TableauRedecompose[1] &lt;- (nombre / 10) modulo 10   TableauRedecompose[0] &lt;- partieDecimale(nombre / 100) } </pre>	
<p><b>Commentaire</b></p> <p>La fonction partieDecimale permet de retourner la partie décimale d'un nombre.</p>	

<b>FS2.2</b> <b>gestionDizaineUnite()</b>	<b>21/11/2021</b> <b>Baptiste Risse</b>
<b>Valeur Ajoutée</b> Ce service permet de traduire les dizaines et unités en toute lettre dans un tableau de string de 2 cases.	
<b>INPUT</b>  Adr entier nombre[3], adr chaine dizaine_unite_lettre[2]	<b>OUTPUT</b>
<b>Logigramme Fonctionnel</b>  <pre> graph LR     A[Tableau d'entier nombre[3]] --&gt; C[gestionDizaineUnite]     B[chaine dizaine_unite_lettre[2]] --&gt; C           </pre>	
<b>Service Fonctionnel</b>  <pre> Procédure gestionDizaineUnite(adr entier nombre[3], adr chaine dizaine_unite_lettre[2]) {   chaine dizaine_unite_lettre[3];    Si(nombre[1] supérieur ou égal à 2 et nombre[1] inférieur ou égal à 7){     dizaine_unite_lettre[1] &lt;- traitementCasEt(nombre);   }   Sinon Si(nombre[1] égal à 0 ou nombre[1] égal à 8 ou nombre[1] égal à 9){     dizaine_unite_lettre[1] &lt;- traitementCasDix(nombre[1])   }    Si(nombre[1] égal à 1 ou nombre[1] égal à 7 ou nombre[1] égal à 9){     pour indiceActuel allant de 0 à 9{       Si(nombre[2] égal à indiceActuel)         dizaine_unite_lettre[2] &lt;- tableau_onze[indiceActuel]     }   }   Sinon Si(nombre[1] égal à 0 ou nombre[1] supérieur ou égal à 2 et nombre[1] inférieur ou égal à 6 ou nombre[1] égal à 8){     pour indiceActuel allant de 0 à 8 {       Si(nombre[2] - 1 égal à indiceActuel)         dizaine_unite_lettre[2] &lt;- tableau_unite[indiceActuel]     }   } }           </pre>	
<b>Commentaire</b>  -----	

<p><b>FS2.3</b> <b>gestionCentaine()</b></p>	<p><b>21/11/2021</b> <b>Baptiste Risse</b></p>
<p><b>Valeur Ajoutée</b></p> <p>Ce service permet de traduire le chiffre des centaines en toute lettre et renvoie le résultat au type string.</p>	
<p><b>INPUT</b></p> <p>Val entier centaine</p>	<p><b>OUTPUT</b></p> <p>Val chaine centaine_lettre</p>
<p><b>Logigramme Fonctionnel</b></p> <pre> graph LR     A[entier centaine] --&gt; B[gestionCentaine]     B --&gt; C[chaine centaine_lettre]             </pre>	
<p><b>Service Fonctionnel</b></p> <pre> Fonction gestionCentaine(val entier centaine) : String {     chaine centaine_lettre     entier indiceActuel &lt;- 0     pour indiceActuel allant de 0 à 8{         Si(centaine - 1 égal à indiceActuel){             Si(centaine égal à 1)                 centaine_lettre &lt;- "cent"             Sinon                 centaine_lettre &lt;- tableau_unite[indiceActuel] + " cent"         }     }      retourner centaine_lettre }             </pre>	
<p><b>Commentaire</b></p> <p>-----</p>	


<p><b>FS2.4</b></p> <p><b>concatenationSommeDigitée()</b></p>	<p><b>21/11/2021</b></p> <p><b>Maël Rhuin</b></p>
<p><b>Valeur Ajoutée</b></p> <p>Ce service permet de concaténer les strings de FS2.2 avec celles de FS2.3 tout en gérant le pluriel selon les contraintes posées. Elle renvoie le résultat sous forme d'un string.</p>	
<p><b>INPUT</b></p> <p>val chaine sommeDigitLettre[3], val entier sommeEntier[3]</p>	<p><b>OUTPUT</b></p> <p>Chaine sommeDigitDeSortie</p>
<p><b>Logigramme Fonctionnel</b></p>  <pre> graph LR     A["val chaine sommeDigitLettre[3]"] --&gt; B["ConcatenationSommeDigitée"]     C["val tableau d'entier sommeEntier[3]"] --&gt; B     B --&gt; D["chaine sommeDigitDeSortie"]     </pre>	
<p><b>Service Fonctionnel</b></p> <pre> Fonction concatenationSommeDigitée(val chaine sommeDigitLettre[3], val entier sommeEntier[3]) {   Entier indiceActuel &lt;- 0   //pluriel   Si ((sommeEntier[0] supérieur à 1) et (sommeEntier[1] égal à 0 et sommeEntier[2] égal à 0))     sommeDigitLettre[0] &lt;- sommeDigitLettre[0] + "s"    Si ((sommeEntier[1] égal à 2 ou sommeEntier[1] égal à 8) et (sommeEntier[2] égal à 0))     sommeDigitLettre[1] &lt;- sommeDigitLettre[1] + "s"    //Concaténationv   chaine sommeDigitDeSortie &lt;- ""   pour (indiceActuel allant de 0 à 2)     si (la longueur de sommeDigitLettre[indiceActuel] est supérieure à 0)       sommeDigitDeSortie &lt;- sommeDigitDeSortie + sommeDigitLettre[indiceActuel] + " "   retourner sommeDigitDeSortie }         </pre>	
<p><b>Commentaire</b></p> <p>-----</p>	



<p><b>FS4.1</b></p> <p><b>ajoutMajuscule()</b></p>	<p><b>21/11/2021</b></p> <p><b>Maël Rhuin</b></p>
<p><b>Valeur Ajoutée</b></p> <p>Ce service permet d'ajouter une majuscule en début de proposition.</p>	
<p><b>INPUT</b></p> <p>Adr chaine TableauSommeEnLettre[4] adr entier TableauInitial[4]</p>	<p><b>OUTPUT</b></p> <p>-----</p>
<p><b>Logigramme Fonctionnel</b></p>  <pre> graph LR     A[Adr chaine TableauSommeEnLettre[4]] --&gt; C[ajoutMajuscule]     B[adr entier TableauInitial[4]] --&gt; C     style C stroke-width:2px     </pre>	
<p><b>Service Fonctionnel</b></p> <pre> Procédure ajoutMajuscule(adr string TableauSommeEnLettre[4],adrI int TableauInitial[4]) {   entier indiceMajuscule &lt;- 0   entier indiceActuel &lt;- 0   pour indiceActuel allant de 0 à 3 {     Si (TableauInitial[indiceActuel] supérieur à 0) quitter la boucle;     indiceMajuscule &lt;- indiceMajuscule + 1   }    // Attribution de la majuscule   chaine inter &lt;- TableauSommeEnLettre[indiceMajuscule]   inter[0] &lt;- majuscule(inter[0])    TableauSommeEnLettre[indiceMajuscule] &lt;- inter } </pre>	
<p><b>Commentaire</b></p> <p>La fonction majuscule retourne le caractère en majuscule.</p>	

# Analyse descriptive de niveau 3

FT 2.2.1 traitementCasEt()	21/11/2021 Baptiste Risse
<p align="center"><b>Valeur Ajoutée</b></p> <p>Ce service permet de traiter les nombre composé de « et » comme « vingt et un » ou « soixante et un »... Il renvoie le résultat sous forme de string.</p>	
<p align="center"><b>INPUT</b></p> <p align="center">Adr entier nombre[3]</p>	<p align="center"><b>OUTPUT</b></p> <p align="center">Val chaine dizaine_lettre</p>
<p align="center"><b>Logigramme Fonctionnel</b></p> <pre> graph LR     A[Tableau d'entier[3]] --&gt; B[traitementCasEt]     B --&gt; C[chaine centaine_lettre]             </pre>	
<p align="center"><b>Service Fonctionnel</b></p> <pre> Fonction traitementCasEt(adr entier nombre[3]) {   chaine dizaineEnLettre   entier indiceActuel &lt;- 0   pour indiceActuel allant de 0 à 5 {     si (nombre[1] - 2 égal à indiceActuel) {       dizaineEnLettre &lt;- tableau_dizaine[indiceActuel];       si (nombre[2] égal à 1) dizaineEnLettre &lt;- dizaineEnLettre + " et"     }   }   si (nombre[1] égal à 7) {     dizaineEnLettre &lt;- tableau_dizaine[4]     si (nombre[2] égal à 1) dizaineEnLettre &lt;- dizaineEnLettre + " et"   }   retourner dizaineEnLettre; }             </pre>	
<p align="center"><b>Commentaire</b></p> <p align="center">-----</p>	

<b>FT 2.2.2</b> <b>traitementCasDix()</b>	<b>21/11/2021</b> <b>Baptiste RISSE</b>
<p align="center"><b>Valeur Ajoutée</b></p> <p>Ce service permet de gérer les cas où le nombre écrit fini par « dix » comme « soixante dix » et « quatre vingt dix » ... Il renvoie le résultat sous forme de string.</p>	
<p align="center"><b>INPUT</b></p> <p align="center">Val entier dizaine</p>	<p align="center"><b>OUTPUT</b></p> <p align="center">Val chaine dizaine_lettre</p>
<p align="center"><b>Logigramme Fonctionnel</b></p>  <pre> graph LR     A[entier dizaine] --&gt; B[traitementCasDix]     B --&gt; C[chaine dizaine_lettre]     </pre>	
<p align="center"><b>Service Fonctionnel</b></p> <pre> Fonction traitementCasDix(val entier dizaine) : chaine {   chaine dizaine_lettre   Si(dizaine &lt;- 1)     dizaine_lettre &lt;- "";   Sinon Si(dizaine &lt;- 0)     dizaine_lettre &lt;- "";   Sinon Si(dizaine égal à 8 ou dizaine égal à 9)     dizaine_lettre &lt;- tableau_dizaine[5]    retourner dizaine_lettre }         </pre>	
<p align="center"><b>Commentaire</b></p> <p align="center">-----</p>	

# Ordre de programmation

