```
 1: VARIABLES
 2: x EST_DU_TYPE NOMBRE
 3: y EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   LIRE ......
 6:   SI (........) ALORS
 7:     DEBUT_SI
 8:     ....... PREND_LA_VALEUR .......
 9:     AFFICHER .......
10:     FIN_SI
11: FIN_ALGORITHME
```

3 Boucles

Boucles POUR...DE...A

- Les boucles permettent de répéter des instructions autant de fois que l'on souhaite.
- Lorsqu'on connait par avance le nombre de fois que l'on veut répéter les instructions, on utilise une boucle du type POUR...DE...A dont la structure est la suivante :

```
POUR...ALLANT_DE...A...
  DEBUT_POUR
  ...
  FIN_POUR
```

- Exemple : l'algorithme ci-dessous permet d'afficher la racine carrée de tous les entiers de 1 jusqu'à 50.

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: racine EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   POUR n ALLANT_DE 1 A 50
 6:     DEBUT_POUR
 7:     racine PREND_LA_VALEUR sqrt(n)
 8:     AFFICHER racine
 9:     FIN_POUR
10: FIN_ALGORITHME
```

  La variable n est appelée « compteur de la boucle ».
- Remarques :
  - La variable servant de compteur pour la boucle doit être du type NOMBRE et doit être déclarée préalablement (comme toutes les variables).
  - Dans AlgoBox, cette variable est automatiquement augmentée de 1 à chaque fois.
  - On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais les instructions comprises entre DEBUT_POUR et FIN_POUR ne doivent en aucun cas modifier la valeur de la variable qui sert de compteur.

► Activité n°10

On cherche à concevoir un algorithme qui affiche, grâce à une boucle POUR...DE...A, les résultats des calculs suivants : 8*1 ; 8*2 ; 8*3 ; 8*4 ; ... jusqu'à 8*10 .
La variable n sert de compteur à la boucle et la variable produit sert à stocker et afficher les résultats. Compléter les lignes 5 et 7 dans l'algorithme ci-dessous pour qu'il réponde au problème :

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: produit EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   POUR n ALLANT_DE .... A .....
 6:     DEBUT_POUR
 7:     produit PREND_LA_VALEUR .......
 8:     AFFICHER produit
 9:     FIN_POUR
10: FIN_ALGORITHME
```

► Activité n°11

On considère l'algorithme suivant :

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: somme EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   somme PREND_LA_VALEUR 0
 6:   POUR n ALLANT_DE 1 A 100
 7:     DEBUT_POUR
 8:     somme PREND_LA_VALEUR somme+n
 9:     FIN_POUR
10:   AFFICHER somme
11: FIN_ALGORITHME
```

Compléter les phrases suivantes :

- Après exécution de la ligne 5, la variable somme contient la valeur : ☐
- Lorsque le compteur n de la boucle vaut 1 et après exécution du calcul ligne 8, la variable somme vaut : ☐
- Lorsque le compteur n de la boucle vaut 2 et après exécution du calcul ligne 8, la variable somme vaut : ☐
- Lorsque le compteur n de la boucle vaut 3 et après exécution du calcul ligne 8, la variable somme vaut : ☐

Que permet de calculer cet algorithme ?

☐

► Activité n°12

Compléter les lignes 6 et 8 de l'algorithme ci-dessous pour qu'il permette de calculer la somme $5^2 + 6^2 + 7^2 + \cdots + 24^2 + 25^2$.

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: somme EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5: | somme PREND_LA_VALEUR 0
 6: | POUR n ALLANT_DE .... A ......
 7: | | DEBUT_POUR
 8: | | somme PREND_LA_VALEUR somme+......
 9: | | FIN_POUR
10: | AFFICHER somme
11: FIN_ALGORITHME
```

Boucles TANT QUE...

– Il n'est pas toujours possible de connaitre par avance le nombre de répétitions nécessaires à un calcul. Dans ce cas là, il est possible d'avoir recours à la structure `TANT QUE...` qui se présente de la façon suivante :

```
TANT_QUE...FAIRE
  DEBUT_TANT_QUE
  ...
  FIN_TANT_QUE
```

Cette structure de boucle permet de répéter une série d'instructions (comprises entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE`) tant qu'*une certaine condition est vérifiée*.

– Exemple : Comment savoir ce qu'il reste si on enlève 25 autant de fois que l'on peut au nombre 583 ? Pour cela on utilise une variable `n`, qui contient 583 au début, à laquelle on enlève 25 tant que c'est possible, c'est à dire tant que n est supérieur ou égal à 25.

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: DEBUT_ALGORITHME
 4: | n PREND_LA_VALEUR 583
 5: | TANT_QUE (n>=25) FAIRE
 6: | | DEBUT_TANT_QUE
 7: | | n PREND_LA_VALEUR n-25
 8: | | FIN_TANT_QUE
 9: | AFFICHER n
10: FIN_ALGORITHME
```

– Remarques :
  – Si la condition du TANT QUE... est fausse dès le début, les instructions entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` ne sont jamais exécutées (la structure TANT QUE ne sert alors strictement à rien).
  – Il est indispensable de s'assurer que la condition du TANT QUE... finisse par être vérifiée (le code entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` doit rendre vraie la condition tôt ou tard), sans quoi l'algorithme ne pourra pas fonctionner.

► Activité n°13

On cherche à connaître le plus petit entier N tel que 2^N soit supérieur ou égal à 10000. Pour résoudre ce problème de façon algorithmique :
– On utilise une variable `N` à laquelle on donne au début la valeur 1.
– On augmente de 1 la valeur de `N` tant que 2^N n'est pas supérieur ou égal à 10000.
Une structure TANT QUE est particulièrement adaptée à ce genre de problème car on ne sait pas a priori combien de calculs seront nécessaires.
Compléter les lignes 5 et 7 de l'algorithme ci-dessous pour qu'il réponde au problème :
(remarque : `pow(2,N)` est le code AlgoBox pour calculer 2^N)

```
 1: VARIABLES
 2: N EST_DU_TYPE NOMBRE
 3: DEBUT_ALGORITHME
 4: | N PREND_LA_VALEUR 1
 5: | TANT_QUE (pow(2,N).......) FAIRE
 6: | | DEBUT_TANT_QUE
 7: | | N PREND_LA_VALEUR .......
 8: | | FIN_TANT_QUE
 9: | AFFICHER N
10: FIN_ALGORITHME
```

► Activité n°14

On considère le problème suivant :
– On lance une balle d'une hauteur initiale de 300 cm.
– On suppose qu'à chaque rebond, la balle perd 10% de sa hauteur (la hauteur est donc multipliée par 0.9 à chaque rebond).
– On cherche à savoir le nombre de rebonds nécessaire pour que la hauteur de la balle soit inférieure ou égale à 10 cm.
Compléter les lignes 7 et 10 de l'algorithme ci-dessous pour qu'il réponde au problème.

```
 1: VARIABLES
 2: nombre_rebonds EST_DU_TYPE NOMBRE
 3: hauteur EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5: | nombre_rebonds PREND_LA_VALEUR 0
 6: | hauteur PREND_LA_VALEUR 300
 7: | TANT_QUE (hauteur......) FAIRE
 8: | | DEBUT_TANT_QUE
 9: | | nombre_rebonds PREND_LA_VALEUR nombre_rebonds+1
10: | | hauteur PREND_LA_VALEUR ...........
11: | | FIN_TANT_QUE
12: | AFFICHER nombre_rebonds
13: FIN_ALGORITHME
```