Introduction

Pour avoir du style, il faut éviter les redites.

Dans ce chapitre, nous introduisons une nouvelle construction : la définition de **fonction**, qui permet d'isoler une instruction qui revient plusieurs fois dans un programme.

Une fonction est définie par un **nom**, par ses **arguments** qui porteront les valeurs communiquées par le programme principal à la fonction au moment de son appel et éventuellement une valeur de retour communiquée au programme par la fonction en fin d'exécution.

John McCarthy (1927 - 2011) est l'auteur du langage Lisp (1958), dont la principale construction est la définition de fonctions.
Il est aussi l'un des inventeurs de la notion de temps partagé, qui permet à plusieurs personnes d'utiliser un même ordinateur en même temps.
Il a écrit l'un des premiers programmes jouant aux échecs et inventé pour cela un algorithme, la méthode alpha-bêta, qui permet de jouer non seulement aux échecs, mais aussi à de nombreux autres jeux.

Isoler une instruction

Nous revenons dans ce chapitre sur la question de la portée des variables dans le cas des programmes qui comportent des fonctions.

Nous introduisons aussi des variables ***globales*** dont la portée est le programme tout entier.

Isoler une instruction

Au cours des chapitres précédents, il nous est arrivé d'écrire des programmes dans lesquels certaines instructions revenaient plusieurs fois, parce que nous voulions faire plusieurs fois la même chose.
Un exemple de programme présentant des répétitions est le suivant :

```
print("Le vol  en direction de ");
print("Tokyo");
print(" décollera à ");
print("9h00");
println();
println("---------------------------------------");
println();
print("Le vol  en direction de ");
print("Sydney");
print(" décollera à ");
print("9h30");
println();
println("---------------------------------------");
println();
```

Isoler une instruction

```
print("Le vol  en direction de ");
print("Toulouse");
print(" décollera à ");
print("9h45");
println();
println("-------------------------------------");
println();
```

dans lequel les lignes suivantes :

```
println();
println("-------------------------------------");
println();
```

sont répétées 3 fois.

Isoler une instruction

Au lieu de répéter la totalité de ces trois lignes, on peut leur donner un nom, par exemple tirer_un_trait, puis les remplacer par ce nom dans le programme principal à chaque fois qu'elles sont utilisées.

Pour définir la fonction tirer_un_trait, on procède de la façon suivante :

```
void tiret_un_trait()
{
        println();
        println("----------------------------------------");
        println();
}
```

Isoler une instruction

Au lieu de répéter la totalité de ces trois lignes, on peut leur donner un nom, par exemple tirer_un_trait, puis les remplacer par ce nom dans le programme principal à chaque fois qu'elles sont utilisées.

Pour définir la fonction tirer_un_trait, on procède de la façon suivante :

```
void tiret_un_trait()
{
        println();
        println("----------------------------------------");
        println();
}
```

Isoler une instruction

et on utilise ensuite cette fonction dans le programme principal, comme si c'était une instruction du langage :

```
print("Le vol  en direction de ");
print("Tokyo");
print(" décollera à ");
print("9h00");
tirerUnTrait();

print("Le vol  en direction de ");
print("Sydney");
print(" décollera à ");
print("9h30");
tirerUnTrait();

print("Le vol  en direction de ");
print("Toulouse");
print(" décollera à ");
print("9h45");
tirerUnTrait();
```

Isoler une instruction

Utiliser des fonctions évite les **répétitions** dans les programmes et rend donc ces derniers plus courts et, surtout, plus faciles à lire et à comprendre : pour comprendre le programme ci-dessus, il n'est pas nécessaire de savoir comment la fonction tirerUnTrait est définie, il suffit de savoir ce qu'elle fait.

Utiliser des fonctions permet aussi d'organiser le **travail de développement** : on peut décider d'écrire le programme principal un jour et d'écrire la fonction tirerUnTrait le lendemain.
On peut aussi décider de confier l'écriture du programme principal à un programmeur et l'écriture de la fonction tirerUnTrait à un autre.

Enfin, si l'on veut modifier la longueur du trait à tirer ou le nombre de lignes sautées au-dessus et en-dessous de ce trait, il suffit de modifier le corps de la fonction et non le programme principal à chacun des endroits concernés.

Passer des arguments

Le programme précédent est formé de trois blocs qui annoncent chacun l'horaire d'un vol.

On peut vouloir aller plus loin dans l'organisation de ce programme et écrire une fonction **annoncerUnVol**, qu'il suffirait d'appeler trois fois dans le programme principal.

Cependant, contrairement à l'exemple de la fonction tirerUnTrait, ces trois blocs ne sont pas absolument identiques : la **destination** et l'**horaire du vol** différent d'un cas à l'autre.

Il faut donc **paramétrer** l'instruction que l'on isole pour pouvoir choisir la destination et l'horaire du vol.

Dans notre exemple, les arguments doivent représenter la destination, que l'on nomme destination, et l'horaire de vol, que l'on nomme horaire.

Passer des arguments

On définit alors cette fonction de la manière suivante :

```
void  annoncerUnVol (String destination, String horaire)
{
      print("Le vol  en direction de ");
      print(destination);
      print(" décollera à ");
      print(horaire);
      println();
      println("----------------------------------------");
      println();
}
```

Passer des arguments

Et le programme principal devient :

```
void main()
{
    annoncerUnVol("Tokyo","9h00");
    annoncerUnVol("Sydney","9h30");
    annoncerUnVol("Toulouse","9h45");
}
```

Récupérer une valeur

Le passage d'arguments permet donc de communiquer des informations depuis le programme principal vers une fonction. On veut aussi souvent communiquer des informations dans l'autre sens : depuis une fonction, vers le programme principal. Par exemple, si l'on veut isoler, dans une fonction, l'instruction suivante qui calcule le nombre n de fois que le caractère a apparaît dans une chaîne s :

```
int i, n;
n = 0;
for (i = 0; i < s.length(); i = i + 1)
{
      if (s.charAt(i) == 'a')
      {
          n = n + 1;
      }
}
```

Récupérer une valeur

On veut non seulement que le programme principal puisse communiquer la chaîne de caractères s à la fonction, mais aussi que la fonction puisse communiquer le nombre n au programme principal.

Cela mène à écrire la fonction suivante :

```
int nombreDea (String  s)
{
      int i, n;
      n =  0;
      for (i =  0; i < s.length(); i =  i +  1)
      {
            if (s.charAt(i) == 'a')
            {
               n =  n +  1;
            }
      }
     return n;
}
```

Récupérer une valeur

L'exécution de l'instruction return n; a pour effet d'interrompre l'exécution du corps de la fonction et de renvoyer la valeur de l'expression n au programme principal.

Comme la fonction **nombreDea** renvoie une valeur, l'appel nombreDea("abracadabra"), dans le programme principal, n'est pas une instruction, mais une expression, qui a la valeur 5.

On peut l'utiliser, par exemple, dans une affectation

x = nombreDea("abracadabra");.

Si une fonction ne renvoie pas de valeur, par exemple si elle ne fait qu'afficher des messages à l'écran, on fait précéder sa définition du mot-clé **void**, sinon, on la fait précéder du type de sa valeur de retour.

Ecrire l'entête d'une fonction

SAVOIR-FAIRE Écrire l'en-tête d'une fonction

1. Choisir un nom qui indique clairement ce que fait la fonction.
2. Identifier les arguments qui varient lors des différents appels de la fonction dans le programme principal. Donner un nom à chacun de ces arguments.
3. Identifier un type approprié pour chacun de ces arguments.
4. Identifier si la fonction renvoie une valeur et, si oui, le type de cette valeur.

Exercice 1

Écrire l'en-tête d'une fonction qui calcule la vitesse moyenne d'un mobile connaissant son temps de parcours et la distance parcourue.

Ecrire l'entête d'une fonction

Exercice 2

Écrire l'en-tête des fonctions suivantes :

- Une fonction qui indique s'il est possible de construire un triangle avec trois segments de mesures données.
- Une fonction qui calcule le plus grand diviseur commun (PGCD) de deux nombres entiers.
- Une fonction qui trace à l'écran un segment entre deux points.
- Une fonction qui écrit à l'écran les initiales d'une personne dont on donne le nom complet.

Ecrire une fonction

SAVOIR-FAIRE Écrire une fonction

1. Écrire l'en-tête de la fonction.
2. Écrire le corps de la fonction comme si les arguments étaient déjà remplis par des valeurs.
3. Ne pas oublier l'instruction return, le cas échéant.
4. Prévoir une exécution correcte de la fonction quelle que soit la valeur donnée à chacun des arguments, y compris dans des cas que l'on n'a pas forcément anticipés dans le cours normal du programme principal.

Ecrire une fonction

Exercice 3

Écrire les fonctions suivantes.

- Une fonction qui renvoie la plus grande de deux valeurs de type int.
- Une fonction qui répète un même mot un certain nombre de fois au choix.
- Une fonction, construite à partir de la fonction Math.random, qui tire au sort un nombre entier entre deux bornes données en arguments.
- Une fonction qui décide s'il est possible de construire un triangle avec trois segments de mesures données.

La portée des variables

SAVOIR-FAIRE : Identifier la portée des variables dans un programme comportant des fonctions

- Si une variable est déclarée en dehors de toute fonction, alors elle est globale.
- Si une variable est déclarée à l'intérieur d'une fonction, y compris le programme principal main, alors sa portée est limitée à cette fonction.
- Si une variable fait partie des arguments d'une fonction, alors sa portée est limitée à cette fonction.
- Si deux variables de même nom sont déclarées dans deux fonctions différentes, alors elles représentent en réalité deux boîtes distinctes et la portée de chacune est limitée à la fonction correspondante. Dans ce cas, elles peuvent même avoir des types différents.
- Il faut éviter d'utiliser le même nom pour une variable globale et une variable locale. Cependant, si cela se produit, à l'intérieur de la portée de la variable locale, c'est celle- ci qui est visible et non plus la variable globale, qui ne sera à nouveau accessible que quand on sera sorti de la portée de la variable locale.

La portée des variables

Exercice 4

Déterminer la portée de chaque variable dans le programme suivant. L'utilisation qui est faite de ces variables est-elle cohérente avec cette portée ? Si non, comment corriger ce programme ?

```
int z, y;

void v (double x)
{
        double u;
        u = x * x;
        z = (int) x;
}

void main ()
{
        double t;
        y = 4;
        t = 1 / (double) y;
        v(t);
        println(u);
}
```

La portée des variables

Exercice 5

Trouver les erreurs de portée dans le programme suivant.

```
int i;

int h (int j) {
        int k;
        j = j + 1;
        println(i); println(j);
        k = j + i;
        m = m - 1;
        i = 5;
        return k;
}

void main () {
        int m, n;
        m = 1;
        i = 10;
        println(m);
        n = h(m);
        println(m); println(k);
        println(i + j);
}
```

Le passage par valeurs

Dans le programme suivant :

```
int a,b;
void  main ()  {
        int c;
        a =  4;
        b =  7;
        c =  a;
        a =  b;
        b =  c;
        print(a + "   " + b);
}
```

l'exécution de l'instruction : c = a; a = b; b = c;

a pour effet d'échanger le contenu des boîtes a et b.

Le contenu initial de la boîte a est 4 et celui de la boîte b est 7 ; après l'exécution de cette instruction, le contenu de la boîte a est 7 et celui de la boîte b est 4.

Ainsi le programme affiche : 7 4.

Le passage par valeurs

Cette opération d'échange du contenu de deux boîtes étant souvent utilisée, on peut vouloir l'isoler dans une fonction.

```
int a,b;

void  echange (int x,  int y) {
        int z;
        z =  x;
        x =  y;
        y =  z;
}
void  main ()  {
        a =  4;
        b =  7;
        echange(a,b);
        print(a + "  " + b);
}
```

Toutefois, contrairement au précédent, ce programme affiche :
4 7 et non : 7 4

Le passage par valeurs

En effet, l'appel de la fonction echange(a,b); dans l'état 1 crée l'état 2, crée une boîte z, échange le contenu des boîtes x et y en utilisant la boîte z (3) et supprime la boîte z, puis les boîtes x et y (4).

Le contenu des boîtes a et b n'a donc pas changé.

1

2

3

4

La portée des variables

SAVOIR-FAIRE : **Choisir entre un passage par valeur et une variable globale**

Si une fonction doit utiliser une variable a du programme principal, deux cas sont à distinguer :

- Si la fonction n'utilise que la valeur contenue dans a sans jamais la modifier, alors cette valeur peut être passée en argument à la fonction et la variable a peut être locale au programme principal.
- Si la fonction doit modifier la valeur contenue dans a, alors la variable a doit être globale.

La portée des variables

Exercice 6

Dans le programme suivant, quelles sont les expressions passées par valeur ? Qu'affiche ce programme lorsqu'on l'exécute ?

```
int i;

int h (int j) {
        int k;
        j = j + 1;
        println(i); println(j);
        k = j + i;
        i = 5;
        return k;
 }

void main () {
        int m, n;
        m = 1; i = 10;
        println(m);
        n = h(m);
        println(m); println(n); println(i);
}
```

Le passage par valeurs et les tableaux

Contrairement à celui de la section précédente, le programme suivant, qui utilise des tableaux, affiche bien 7 4.

```
int [] a,b;

void  echange (int [] x,  int [] y) {
      int z;
      z =  x[0];
      x[0]  =  y[0];
      y[0] =  z;
}

void  main ()  {
      a =  new int [2];
      b =  new int [2];
      a[0] =  4;
      b[0]  =  7;
      echange(a,b);
      print(a[0] + "    " +b[0]);
}
```

Le passage par valeurs et les tableaux

Le contenu des boîtes de noms a et b est bien recopié dans les boîtes de noms x et y au moment de l'appel.

Néanmoins, les tableaux eux-mêmes ne sont pas recopiés, car les contenus des boîtes de noms a et b ne sont pas des tableaux mais des références de tableaux.