Chapitre 5 (Partie I) : Sous-programmes

(NFA031 - Jour)

V. Aponte

Cnam

25 octobre 2017

Plan du cours

1. Qu'est-ce qu'un sous-programme ?.
2. Déclarer, utiliser : procédures, fonctions.
3. Exemples.

1. Qu'est-ce qu'un sous-programme ?

Qu'est-ce qu'un sous-programme ?

Sous-programme (méthode)

Instructions regroupée sous un nom, exécutées chaque fois que ce nom apparaît (invocation).

- invoqué, ou appelé (pour exécuter ses instructions),
- on peut lui fournir des entrées (arguments, paramètres),
- il peut retourner une valeur résultat, qu'on pourra utiliser après son exécution.
- 2 sortes de sous-programmes :
  - fonctions : retournent un résultat.
  - procédures : ne retournent pas de résultat.

Exemple d'utilisation : déterminer si une année est bissextile

Problème : lire une année, déterminer et afficher si elle est bissextile.

1. Lire une année an :
   - réalisé par un sous-programme
   - an doit être un entier strictement positif,
   - nous utiliserons une boucle pour demander à recommencer si nécessaire
2. Déterminer si an est bissextile (sous-programme)
3. Afficher un message avec ce résultat.

Lire une année strictement positive

Problème : lire une année, déterminer et afficher si elle est bissextile.

```
public static void main(....) {

    int  an = lirePos("Entrez une année: ");

    if (estBissextile(an)) {

        Terminal.ecrireStringln(an+" est bissextile.");

    } else {

        Terminal.ecrireString(an+" n'est pas bissextile." );

}
```

L'année est bissextile : true ou false ?

Problème : lire une année, déterminer et afficher si elle est bissextile.

```
public static void main(....) {

    int  an = lirePos("Entrez une année: ");

    if (estBissextile(an)) {

        Terminal.ecrireStringln(an+" est bissextile.");

    } else {

        Terminal.ecrireString(an+" n'est pas bissextile." );
}
```

Les sous-programmes sont indispensables

- entité syntaxique unique pour capturer plusieurs instructions ;
  - plus de recopie de code $\Rightarrow$ on invoque le sous-programme ;
  - programmes plus courts, concis, clairs,
  - modifications du code $\Rightarrow$ concentrés en un seul lieu ;
  - circonscrit changements/erreurs.
- essentiels pour découper/structurer les programmes ;
  - sous-programme devient "super-instruction",
  - programme main $\Rightarrow$ construit par appels à sous-programmes ;
  - aide à se concenter sur ce que le programme doit et non pas au "comment”.

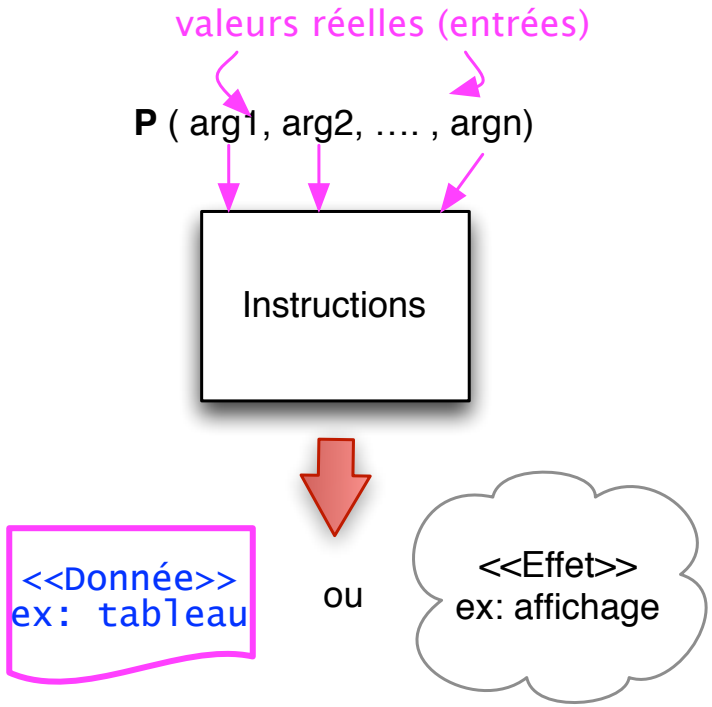
2. Déclarer et utiliser ses sous-programmes

Sous-programmes en Java

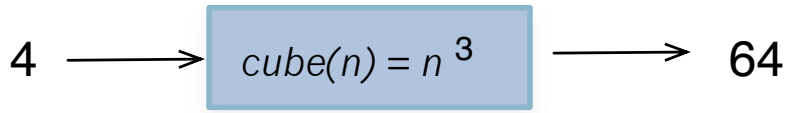
- Appelés méthodes.
- Deux sortes :
  - Statiques : (mot-clé `static`) $\Rightarrow$ non « attachées » à un objet.
    Nous utiliserons uniquement celles-ci !
  - Non statiques : ou « méthodes d'objets ». A voir plus tard (NFA032).
- beaucoup de méthodes prédéfinies :
  - `Math.random()`, `Math.abs()`, `Math.sqrt()`,
  - `System.out.print()`, `Integer.parseInt()`, etc.

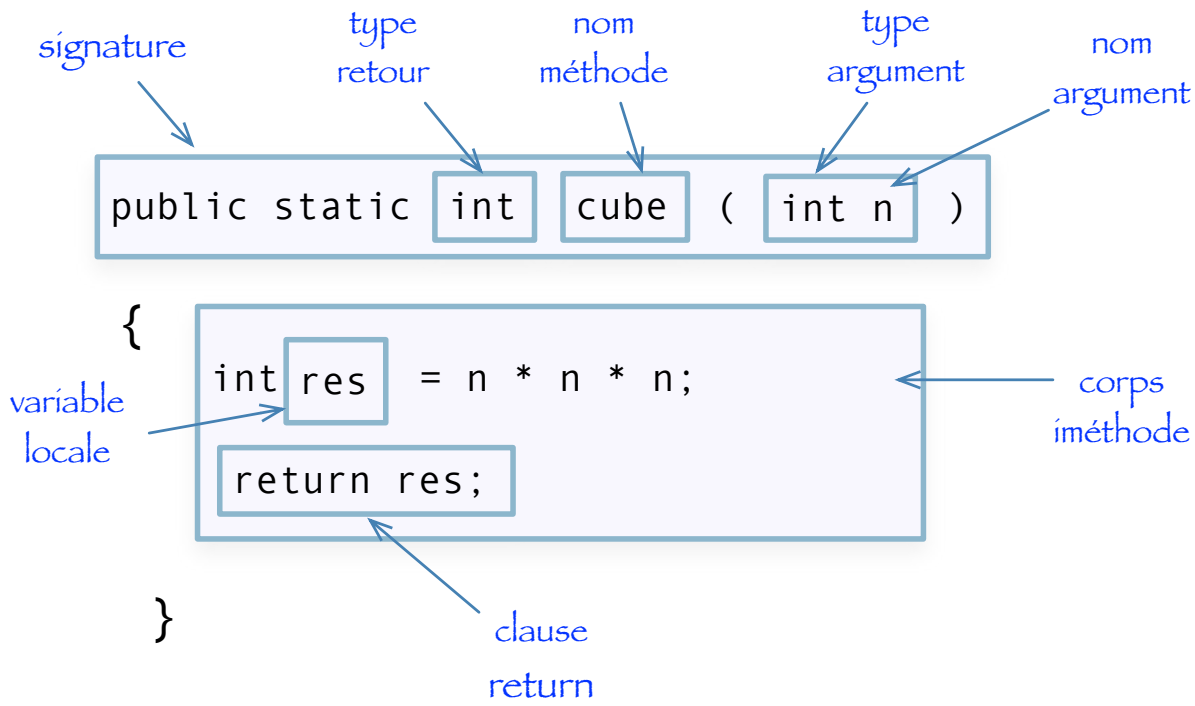
Sous-programme : entrées + exécution + sorties/retour

1. prend un ou plusieurs paramètres ;
2. si invoqué : exécute une suite d'instructions ;
3. produit un résultat ou un comportement.



Anatomie d'un sous-programme





Le rôle des paramètres

Paramètres

Ce sont les entrées du sous-programme : des valeurs passées en paramètre lors de chaque appel.

- nécessaires pour l'exécution du sous-programme.
- peuvent être différents à appel ;

```
Terminal.ecrireInt(5);
Terminal.ecrireInt(10);
```

- Certains sous-programme ne prennent pas d'arguments : `Terminal.lireDouble()`.

Où déclarer + utiliser des sous-programmes

Déclarer : nom, paramètres, corps ;

Utiliser : autant d'appels que nécessaire, à partir d'autres méthodes, et avec leurs arguments.

```
public class <nom-du-programme> {

   <déclaration-des-sousprogrammes> // ici !!!

   public static void main (String[] args) {

      <déclarations-instructions+appels>
   }
}
```

Exemple d'une fonction (+ démo)

```
public class ExempleFonctionCube {
  // la fonction cube
  public static int cube(int n){
      return n*n*n;
  }
  public static void main (String[] args){
    System.out.println("Un␣nombre?␣");
    int a = Terminal.lireInt();
    System.out.println("Le␣cube␣de␣"+a+"␣est␣"+ cube(a));
  }
}
```

Clause return et fonctions

Fonctions : doivent toujours finir leur exécution par un return

```
static int valeurAbsolue(int n){
    int res;
    if (n > 0) { res = n;
    } else if(n < 0){ return -n;
    } else { return 0;}
}
```

Si `n > 0`, pas d'instruction `return`.
⇒ erreur (compilation) :

```
> javac ValAbsFunc3.java
ValAbsFunc3.java:11: missing return statement
```

Exemple de procédure (+démo)

```
public static void afficheTriangleDecroissant(){
  for(int i=1; i<6; i++){
    for (int j=5; j>=i; j--){
        System.out.print(j + "␣");
    }
    System.out.println();
  }
}
```

- cette méthode ne produit pas de donnée ;
- produit un comportément (dessiner, afficher) ;
- type de retour $\Rightarrow$ `void` ;
- Autre particularités : pas d'arguments + invoque d'autres méthodes.

Fonctions et procédures

Selon qu'on veuille ou non obtenir une donnée résultat :

- Fonction :
  - retourne une valeur (à récupérer par l'appelant),
  - type de retour $\neq$ `void`
  - Exemple : `static int Math.min(int x, int y)`

    $\Rightarrow$ produit un int
- Procédure :
  - ne retourne pas de résultat.
  - type de retour = `void`
  - Exemple : `static void Terminal.ecrireInt(int x)`

    $\Rightarrow$ affiche un entier

Vocabulaire

- Arguments, paramètres : entrées d'un sous-programme ;
- Appel, invocation : exécution instructions du sous-programme sur des arguments donnés ;
- selon type de retour :
  - Fonction :
    - ★ type de retour $\neq$ `void`
    - ★ produit une valeur (à récupérer par l'appelant),
  - Procédure :
    - ★ produit un comportément (sans donnée obtenue).
    - ★ type de retour = `void`

Comprendre : méthode M invoque P(v_1,v_2)

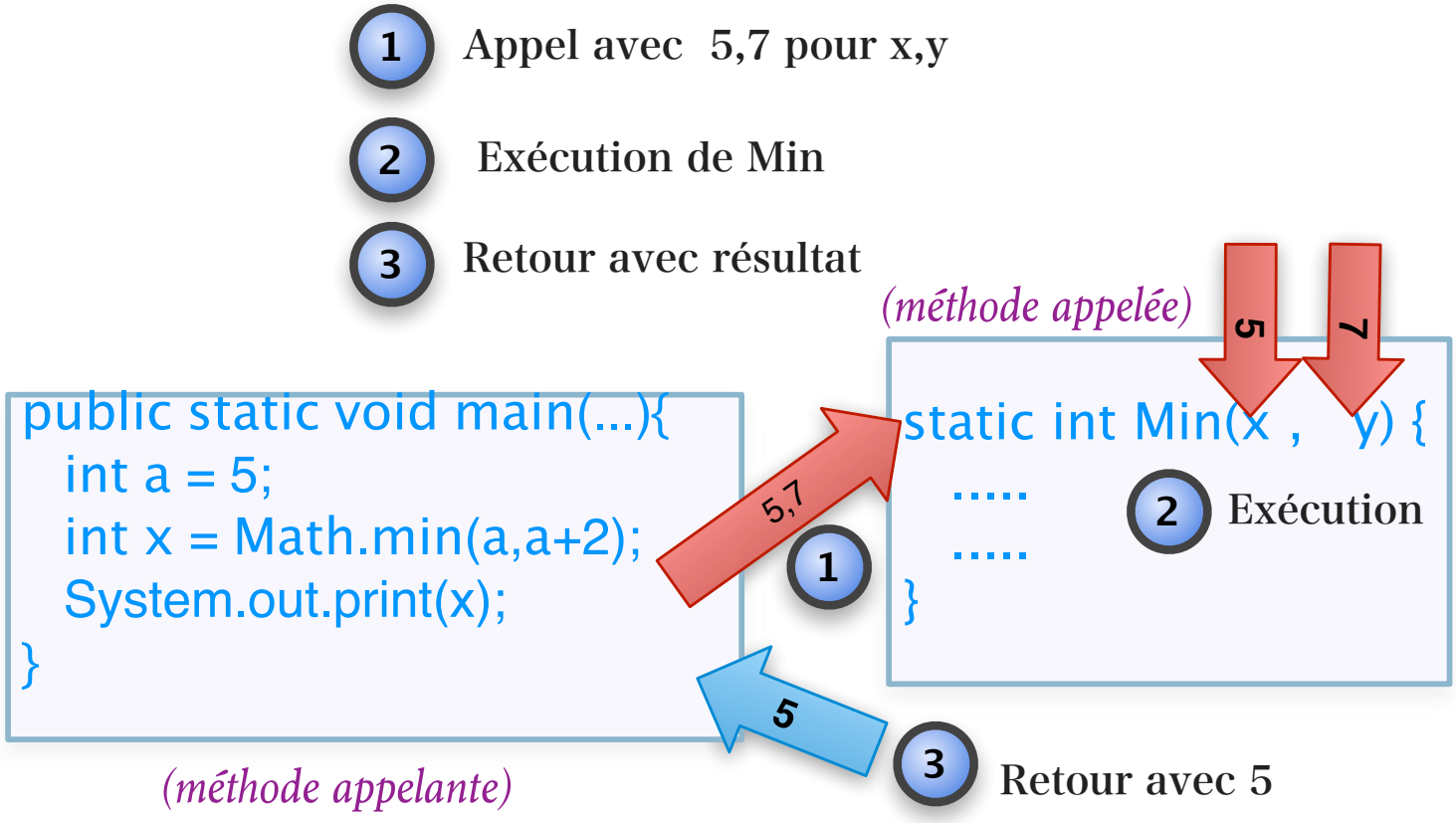
Vocabulaire :

- P(v_1,v_2) : appel à P(x,y) avec paramètres réels v_1, v_2 ;
- méthode appelante : celle qu'invoque le sous-programme P(x,y) ;

Exécution :

1. Interruption exécution de la méthode appelante M ;
2. Les paramètres réels v_1, v_2 + contrôle $\overset{\text{passés}}{\Longrightarrow}$ à P(x,y) ;
   - Arguments de P (x,y) $\overset{\text{affectés}}{\Longleftarrow}$ par v_1, v_2 ;
   - Exécution code de P ;
3. Retour $\overset{\text{(résultat)}}{\Longrightarrow}$ vers M ;
4. Reprise contrôle + exécution M (avec résultat obtenu).

Méthode appelle sous-programme Math.min



Appels imbriqués

Une méthode peut invoquer d'autres méthodes :

```
static boolean estMajuscule(char c){
    return ('A' <= c && c <= 'Z');
}
static boolean estMinuscule(char c){
    return ('a' <= c && c <= 'z');
}
static boolean estLettre(char c){
    return ( estMajuscule(c) || estMinuscule(c) );
}
public static void main(String[] args) {
    char c1 = 'a';
    if ( estLettre(c1) )
        System.out.println(c1+" est une lettre");
    else
        System.out.println(c1+" n'est est pas une lettre");
}
```

Visibilité des variables d'un sous-programme

Variables d'un sous-programme :

- ses paramètres,
- ses variables déclarées localement,

Visibles uniquement dans le sous-programme :

- un sous-programme ne peut utiliser que ses variables locales + paramètres ;
- toute autre variable est considérée inconnue (erreur compilation)

Variables méthode m $\approx$ visibles uniquement par m

Variables d'un sous-programme : exemple

```
static int plusUn(int x){
  int r = x+1;
  return r;
}
public static void main (String [] args){
  int x = 3;
  int y = plusUn(x*2);
  Terminal.ecrireStringln("Resultat: "+y);
  Terminal.ecrireStringln("valeur finale x: "+x);
```

- Quelles variables par méthode ?
- `x` est-elle commune aux deux méthodes ?
- Affichages ?

Variables d'un sous-programme : exemple (2)

- Variables :
  - méthode `plusUn` ⇒`x` et `r` ;
  - méthode `main`⇒ `x` , `y` et `args` ;
- Chaque x est locale à "sa" méthode ⇒2 variables x différentes.
- Affichages :

```
Resultat : 7
valeur finale x : 3
```

3. Exemples

Exemples : fonctions et procédures

Ré-visitons le problème de départ : lecture d'une année strictement positive, test de bissextile, affichage.

1. Lire un nombre (année) strictement positif (boucle).
2. Tester si bissextile
3. Afficher résultat

Exemple : fonction de lecture d'un entier positif

Problème : lire un entier strictement positif
Solution : fonction avec boucle de lecture + validation
retourne : l'entier (valide)

```
static int lirePos(String message) {
  int n = 0;
  while (n<=0)  {
    Terminal.ecrireString(message);
    n = Terminal.lireInt();
    if (n<=0) {
      Terminal.ecrireStringln("Le nombre doit etre positif"
    }
   }
   return n;
}
```

A quoi correspond le paramètre `message` ?

Fonction de lecture d'un entier positif : appel

```
/* Lit un entier strictement positif en affichant
 * message pour inviter  a la saisie  */
static int lirePos(String message) {
  int n = 0;
  while (n<=0)  {
    Terminal.ecrireString(message);
    n = Terminal.lireInt();
    if (n<=0) {
      Terminal.ecrireStringln("Le nombre doit etre positif"
    }
   }
   return n;
}
public static void main (String args[]) {
  int an = lirePos("Une annee? ");
  ....
```

Fonction qui teste si bissextile

Problème : tester si une année est bissextile.

```
 /* Retourne true si a est une annee bissextile.
  */
static boolean estBissextile (int a) {
   boolean d4 = (a%4 == 0);
   boolean d100 = (a%100 == 0);
   boolean d400 = (a%400 == 0);
   return  (d4 && !d100) || d400;
}
```

Fonctions ⇒ ni saisie ni affichage

Principe de programmation : sauf fonctions de lecture, une fonction :

- ne fait pas de lecture ⇒ ses entrées = paramètres ;
- ne fait pas d'affichage ⇒ ses sorties = valeur de retour (return) ;

```
static boolean estBissextile (int a) {
   boolean d4 = (a%4 == 0);
   boolean d100 = (a%100 == 0);
   boolean d400 = (a%400 == 0);
   return  (d4 && !d100) || d400;
}
```

Ne lit pas l'année ⇒ la prend en paramètre ;
N'affiche pas le résultat ⇒ la retourne à celui qui l'appelle !

Méthode main

```
public static void main (String[] args){
    // lecture de l'annee
    int a = lirePos("Entrez l'annee à tester: ");
    // affichage du résultat
    if (estBissextile(a)){
      System.out.println("L'année "+ a+" est bissextile");
    }else {
      System.out.println("L'année "+ a+" n'est pas bissexti
    }
  }
```

Programme complet

```
public class EstBissextileMeth {
  public static boolean estBissextile(int a){
    boolean d4 = (a%4 == 0);
    boolean d100 = (a%100 == 0);
    boolean d400 = (a%400 == 0);
    return (d4 && !d100) || d400;
  }

  static int lirePos(String message) {
    int n = 0;
    while (n<=0)  {
      Terminal.ecrireString(message);
      n = Terminal.lireInt();
      if (n<=0) {
        Terminal.ecrireStringln("Le nombre doit etre positif");
      }
    }
    return n;
  }

  public static void main (String[] args){
    // lecture de l'annee
    int a = lirePos("Entrez l'annee à tester: ");
    // affichage du résultat
    if (estBissextile(a)){
      System.out.println("L'année "+ a+" est bissextile");
    }else {
      System.out.println("L'année "+ a+" n'est pas bissextile");
    }
  }
}
```

Lire des notes comprises entre 0 et 20

Problème : valider les notes lues (comprises entre 0 et 20)
Solution : boucle de lecture+ validation d'un nombre
params : message affiché + bornes inférieur, supérieur ;
valeur retour : nombre compris entre inf et sup

```
static double lireNote(String me,double inf,double sup) {
    while (true)  {
        Terminal.ecrireString(me);
        double n = Terminal.lireDouble();
        if (n<inf || n> sup) {
          Terminal.ecrireStringln("Entree invalide");
        } else return n;
    }
 }
```

Que faut-il modifier pour utiliser cette méthode ?

Erreurs fréquentes (1)

Un sous-programme qui reçoit un paramètre, ne doit pas mofifier ce paramètre avant de l'utiliser (contradiction car paramètre = entrée).

```
  static int valeurAbsolue(int n){
     n = 7;    // Non!!
     if (n > 0) { return n;
     } else if (n < 0) { return -n;
     } else { return 0;
     }
  }

  public static void main (String args[]) {
     int x = -3;
     Terminal.ecrireString("Valeur absolue de "+x +":");
     Terminal.ecrireIntln(valeurAbsolue(x));
  }
}
```

Erreurs fréquentes (2)

Un sous-programme qui reçoit un paramètre, ne doit pas saisir sa valeur (contradiction, car le paramètre est déjà l'entrée).

```
  static int valeurAbsolue(int n){
     Terminal.ecrireString("Donnez un entier ");
     n = Terminal.lireInt(); // Non!!
     if (n > 0) { return n;
     } else if (n < 0) { return -n;
     } else { return 0;
     }
  }

  public static void main (String args[]) {
     int x = -3;
     Terminal.ecrireString("Valeur absolue de "+x +":");
     Terminal.ecrireIntln(valeurAbsolue(x));
  }
}
```

Erreurs fréquentes (3)

Une fonction retourne un résultat et ne doit pas s'occuper de son affichage.

```
static int valeurAbsolue(int n){
    if (n > 0) { Terminal.ecrireInt(n);// Non!!
   } else if (n < 0) { return -n;
                       Terminal.ecrireInt(-n);// Non!!
   } else {Terminal.ecrireInt(0); return 0;// Non!!
   }
}
public static void main (String args[]) {
   int x = -3;
   Terminal.ecrireString("Valeur␣absolue␣de␣"+x +":");
   Terminal.ecrireIntln(valeurAbsolue(x));
}
```

Ce programme ne se comporte pas correctement. Pourquoi ?

Erreurs fréquentes (4)

Penser qu'il y a un lien entre le nom donné au paramètre et celui de la variables de l'appel (ici x).

```
  static int valeurAbsolue(int x){
      if (x > 0) { return x;
    } else if (x < 0) { return -x;
    } else {  return 0;
    }
  }
  public static void main (String args[]) {
    int x = -3;
    Terminal.ecrireString("Valeur absolue de "+x +":");
    Terminal.ecrireIntln(valeurAbsolue(x));
  }
}
```

le paramètre x du sous-programme, et la variable x du main, sont de variables locales distintes et indépendantes.

Erreurs fréquentes (resumé)

- Un sous-programme ne saisit pas les valeurs de ses paramètres.
- Un sous-programme ne modifie pas les valeurs de ses paramètres (avant de s'en servir).
- Une fonction n'affiche pas son résultat. Elle le renvoie.
- Une fonction ne modifie jamais ses paramètres.
- Les paramètres formels (ceux déclarés dans la méthode) sont des variables locales et donc, sont indépendantes des variables utilisées pour un appel.

Procédure ou Fonction ?

Si l'on veut écrire un sous-programme, comment savoir s'il doit correspondre à une procédure ou à une fonction ?
Comment savoir quels sont les arguments ?

- Le sous-programme doit-il calculer un résultat ? Quel est son type ? (fonction + type de retour)
- Doit-il faire des affichages sans calculs ? (procédure)
- Quelles sont les données nécessaires à son exécution ? Quels sont leurs types ? (paramètres)
- Si un sous-programme doit réaliser des entrées/sorties et des calculs, il vaut mieux le couper en autant de morceaux que nécessaire afin de séparer clairement les fonctions des procédures.

Programmez avec des sous-programmes !

Les sous-programmes servent à structurer les programmes.

- A chaque fois que cela est utile : utilisez des sous-programmes :
  1. Identifier une sous-tâche ;
  2. Identifier ses entrées/sorties + écrire son code ;
  3. invoquer partout où cette sous-tâche est nécessaire ;
  4. ré-utiliser le sous-programme dans d'autres programmes...
- Sous-programmes prédéfinis : dans les « bibliothèques » d'un langage.
  - à utiliser autant que possible !