

Méthodes, fonctions et variables globales

Valeur d'accueil et de reconversion en informatique (VARI1)

Daniel Porumbel (dp.cnam@gmail.com)

<http://cedric.cnam.fr/~porumbed/vari1/>

Rappel : méthodes `setup` et `draw`

Le code ci-dessous permet de tracer des lignes sans arrêt
(point de départ : le centre de la toile)

```
1 void setup () {  
2     println ( "Salut_toto" );  
3     size (600,600);  
4 }  
5 void draw () {  
6     line (300,300,random(600) ,random(600) );  
7     println ( "rebonjour_toto" );  
8 }
```

On peut dire qu'une méthode permet de donner un nom à plusieurs lignes de code

- 1 les lignes 2-3 : un bloc de code appelé `setup()`
- 2 les lignes 6-7 : un bloc de code appelé `draw()`

Rappel : méthodes `setup` et `draw`

Le code ci-dessous permet de tracer des lignes sans arrêt
(point de départ : le centre de la toile)

```
1 void setup() {  
2     println("Salut_toto");  
3     size(600,600);  
4 }  
5 void draw() {  
6     line(300,300,random(600),random(600));  
7     println("rebonjour_toto");  
8 }
```

On peut dire qu'une méthode permet de donner un nom à plusieurs lignes de code

- 1 les lignes 2-3 : un bloc de code appelé `setup()`
- 2 les lignes 6-7 : un bloc de code appelé `draw()`

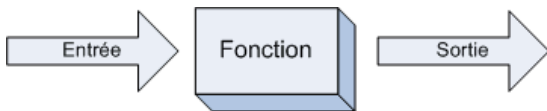
Fonctions

Une fonction comporte :

une entrée on fait « rentrer » des informations dans la fonction (on lui donne les données avec lesquelles travailler).

un traitement grâce aux informations reçues en entrée, la fonction exécute un traitement, elle fait quelque chose

une sortie à la fin la fonction **renvoie un résultat** via **return**.



méthode = une fonction qui ne renvoie rien (void)

- Une méthode n'a pas besoin de **return**
- **void**= pas de sortie

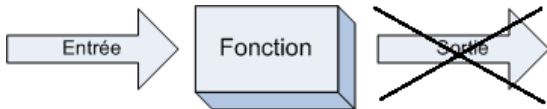
Fonctions

Une fonction comporte :

une entrée on fait « rentrer » des informations dans la fonction (on lui donne les données avec lesquelles travailler).

un traitement grâce aux informations reçues en entrée, la fonction exécute un traitement, elle fait quelque chose

une sortie à la fin la fonction **renvoie un résultat** via **return**.



méthode = une fonction qui ne renvoie rien (void)

- Une méthode n'a pas besoin de **return**
- **void** = pas de sortie

Exemple de définition de fonction :

```
int calculerCube (int x) {  
    return x*x*x;  
}
```

entrée : une variable entière

sortie : une variable entière

Exemple de définition de fonction qui ne renvoie rien :

```
void direBonjour (String nomPersonne) {  
    println ("Bonjour_" + nomPersonne);  
}
```

Exemples d'appels de fonctions

```
...  
int n = calculerCube(5); // n vaut 125  
direBonjour("Toto"); // Affiche "Bonjour Toto"  
...
```

Exemple de définition de fonction :

```
int calculerCube ( int x ) {  
    return x*x*x;  
}
```

entrée : une variable entière

sortie : une variable entière

Exemple de définition de fonction qui ne renvoie rien :

```
void direBonjour (String nomPersonne) {  
    println ( "Bonjour_" + nomPersonne );  
}
```

Exemples d'appels de fonctions

```
...  
int n = calculerCube (5); // n vaut 125  
direBonjour ("Toto"); // Affiche "Bonjour Toto"  
...
```

Exemple de définition de fonction :

```
int calculerCube (int x) {  
    return x*x*x;  
}
```

entrée : une variable entière

sortie : une variable entière

Exemple de définition de fonction qui ne renvoie rien :

```
void direBonjour (String nomPersonne) {  
    println ("Bonjour_" + nomPersonne);  
}
```

Exemples d'appels de fonctions

```
...  
int n = calculerCube (5); // n vaut 125  
direBonjour ("Toto"); // Affiche "Bonjour Toto"  
...
```


Définition de fonctions avec plusieurs entrées :

entrées : une chaîne de
caractères et une lettre

```
void direSalut(String nom, char initialePrenom){  
    println("Salut_" + initialePrenom + "_" + nom);  
}
```

Moyenne de 3 entiers :

sortie : float pas int

```
float calculerMoyenne(int a, int b, int c){  
    return (float)(a+b+c)/3; // conversion float  
}
```

Appeler ces fonctions :

apostrophes pour char

```
...  
direSalut("Toto", 'T'); // => "Salut T Toto"  
direSalut("Titi", 'S'); // => "Salut S Titi"  
float m = calculerMoyenne(7,7,8); // m=7.33  
...
```

Définition de fonctions
avec plusieurs entrées :

entrées : une chaîne de
caractères et une lettre

```
void direSalut(String nom, char initialePrenom){  
    println("Salut_" + initialePrenom + "_" + nom);  
}
```

Moyenne de 3 entiers :

sortie : float pas int

```
float calculerMoyenne(int a, int b, int c){  
    return (float)(a+b+c)/3; //conversion float  
}
```

Appeler ces fonctions :

```
...  
direSalut("Toto", 'T'); //=> "Salut T Toto"  
direSalut("Titi", 'S'); //=> "Salut S Titi"  
float m = calculerMoyenne(7,7,8); //m=7.33  
...
```

apostrophes pour char

Définition de fonctions
avec plusieurs entrées :

entrées : une chaîne de
caractères et une lettre

```
void direSalut(String nom, char initialePrenom){  
    println("Salut_" + initialePrenom + "_" + nom);  
}
```

Moyenne de 3 entiers :

sortie : float pas int

```
float calculerMoyenne(int a, int b, int c){  
    return (float)(a+b+c)/3; //conversion float  
}
```

Appeler ces fonctions :

apostrophes pour char

```
...  
direSalut("Toto", 'T'); //=> "Salut T Toto"  
direSalut("Titi", 'S'); //=> "Salut S Titi"  
float m = calculerMoyenne(7,7,8); //m=7.33  
...
```

Processing exécute `setup()` en premier

La première fonction exécutée est :

`setup()` **SOUS** Processing

`main()` **SOUS** C/C++/Java

Elle peut appeler d'autres fonctions

- fonctions prédéfinies : `size(...)`, `ellipse(...)`, etc.
- fonction `calculerMax(...)` définie par nous

Processing exécute `setup()` en premier

La première fonction exécutée est :

`setup()` **SOUS** Processing

`main()` **SOUS** C/C++/Java

```
int calculerMin(int a, int b){
    if (a<b)
        return a;
    else
        return b;
}
void setup(){
    int x = (int)random(200),y=(int)random(300);
    int rayon = calculerMin(x,y);
    size(600,600);
    ellipse(x,y,2*rayon,2*rayon);
}
//2*rayon=diamètre=>toucher bord
```

La fonction `draw()`

- appelée **de manière répétitive** par Processing
 - Rappel : `setup()` est appelée une seule fois au début
- le nombre d'appels par seconde peut être modifié par un appel `frameRate(nombre)`
- Pour faire une animation : tracer une ligne à des coordonnées aléatoires à l'intérieur du `draw()`

```
void setup(){
  size(600,600);
  frameRate(1000);
}
void draw(){
  //tracer une ligne du pixel x au pixel y
  //les coordonnées x et y: aléatoires <600
  line(random(600),random(600), //pixel x
        random(600),random(600)); //pixel y
}
```

La fonction `draw()`

- appelée **de manière répétitive** par Processing
 - Rappel : `setup()` est appelée une seule fois au début
- le nombre d'appels par seconde peut être modifié par un appel `frameRate(nombre)`
- Pour faire une animation : tracer une ligne à des coordonnées aléatoires à l'intérieur du `draw()`

```
void setup() {  
  size(600,600);  
  frameRate(1000);  
}  
void draw() {  
  //tracer une ligne du pixel x au pixel y  
  //les coordonnées x et y: aléatoires <600  
  line(random(600),random(600), //pixel x  
        random(600),random(600)); //pixel y  
}
```

Mode actif et mode statique

Processing connaît deux modes :

mode actif On déclare des fonctions et toutes les calculs/affichages sont écrit dans les (corps des) fonctions

mode statique On ne déclare pas de fonction et on écrit une séquence d'instruction, comme jusqu'à aujourd'hui

Impossible de mélanger les deux modes : comment exécuter le programme ci-après ?

```
...  
println ("Toto");  
void setup() {  
    println ("Dans_le_corps_de_la_fonct._setup");  
}  
...
```


Mode actif et mode statique

Processing connaît deux modes :

mode actif On déclare des fonctions et toutes les calculs/affichages sont écrit dans les (corps des) fonctions

mode statique On ne déclare pas de fonction et on écrit une séquence d'instruction, comme jusqu'à aujourd'hui

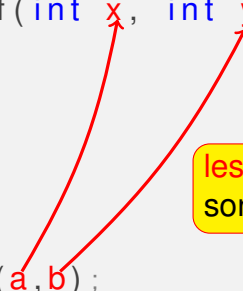
Impossible de mélanger les deux modes : comment exécuter le programme ci-après ?

```
..  
println("Toto");  
void setup(){  
  println("Dans le corps de la fonct. setup");  
}  
...
```

Impossible de savoir quand afficher Toto :
avant ou après l'appel à `setup()` ?

Tester le passage des paramètres par valeur

```
void testModif(int x, int y){  
    x = 9;  
    y = 12;  
}  
void setup(){  
    int a=4;  
    int b=5;  
    testModif(a, b);  
    println(a);  
    println(b);  
}
```



les valeurs de a et b
sont affectées à x et y

Qu'affiche ce programme ?

Tester le passage des paramètres par valeur

```
void testModif(int x, int y){  
    x = 9;  
    y = 12;  
}  
void setup(){  
    int a=4;  
    int b=5;  
    testModif(a, b);  
    println(a);  
    println(b);  
}
```

les valeurs de a et b
sont affectées à x et y

Qu'affiche ce programme ?

Exceptions à la règle du passage par valeur : les tableaux et les objets (à voir plus tard).

Une fonction avec un tableau

```
int sommeTab(int [] tab){
    //tab: visible que dans sommeTab();
    return tab[0]+tab[1]+tab[2];
}
void setup(){
    //t: visible que dans setup()
    int [] t = new int[3];
    t[0] = 1;
    t[1] = 2;
    t[2] = 3;
    println(sommeTab(t));
}
```

Une autre variable locale **s**

```
int sommeTabMult2(int [] tab){
    int s = 2;
    //s visible que dans sommeTabMult2 (...) !!!
    return s*(tab[0]+tab[1]+tab[2]);
}

void setup(){
    //t: visible que dans setup()
    int [] t = new int[3];
    t[0] = 1;
    t[1] = 2;
    t[2] = 3;
    println(sommeTabMult2(t));
}
```

Enchaîner l'appel de fonctions

```
int doubler(int x){  
    return 2*x;  
}
```

```
}
```

```
void setup(){
```

```
    println(doubler(doubler(doubler(3))));
```

```
}
```

3 appels imbriqués



Enchaîner l'appel de fonctions

```
int doubler(int x){  
    return 2*x;  
}  
void setup(){  
    println(doubler(doubler(doubler(3))));  
}
```

3 appels imbriqués



On suppose que `int detMin(int, int)` est déjà écrite

- `detMin(...)` renvoie la valeur minimale et ne touche pas aux variables globales ! (travail en équipe)

```
int [] x={7,2,19,11};  
void setup(){  
    int minDesQuatres=detMin(detMin(x[0],x[1]),  
                             detMin(x[2],x[3]));  
    println(minDesQuatres);  
}
```

Enchaîner l'appel de fonctions

```
int doubler(int x){  
    return 2*x;  
}  
void setup(){  
    println(doubler(doubler(doubler(3))));  
}
```

3 appels imbriqués



Ces deux fonctionns renvoient le même résultat :

```
int sommeTabMult2(int [] tab){  
    int s = 2;  
    return s*(tab[0]+tab[1]+tab[2]);  
}  
int sommeTabMult2Bis(int [] tab){  
    return doubler(sommeTab(tab));  
}
```


Variables : globales, locales ou arguments

Comment corriger et lancer ce programme ?

```
int x; //x=variable globale
int detMin(int a, int b) { //a et b: arguments
    println(x); //x visible partout
    if (a<=b)
        return a;
    return b; //on arrive ici si b<a
    println("Toto??"); //on arrive jamais ici
}
void setup() {
    x = 8;
    int y = 9; //y: variable locale
    y = detMin(x,y);
    println(y);
    println(a); // erreur: a n'est pas
                //déclaré ici!
}
```

Variables : globales, locales ou arguments

Comment corriger et lancer ce programme ?

```
int x; //x=variable globale
int detMin(int a, int b){ //a et b: arguments
    println(x); //x visible partout
    if(a<=b)
        return a;
    return b; //on arrive ici si b<a
println("Toto ???"); //on arrive jamais ici
}
void setup(){
    x = 8;
    int y = 9; //y: variable locale
    y = detMin(x,y);
    println(y);
println(a); // erreur: a n'est pas
}
```

unreachable code

a can not be resolved to a variable

Attention aux variables globales !!!

Quel est le résultat/affichage de ce programme ???

```
int x; //x=variable globale
int detMin(int a, int b) { //a et b: arguments
    x = x-1; //x visible partout
    if (a<=b)
        return a;
    return b; //on arrive ici si b<a
}
void setup() {
    x = 8;
    int y = 9; //y: variable locale
    y = detMin(x,y); //l'appel modifie x!!!
    y = detMin(x,y);
    println(y);
}
```

Le passage des paramètres aux fonctions

```
int f(int x, int y) {  
    return x+y;  
}  
void setup() {  
    int somme;  
    somme = f(2,3); //somme devient 5  
    int a=4, b=5;  
    somme = f(a,b); //somme devient 9  
    println(somme);  
}
```

- `x` et `y` sont visibles/connus uniquement à l'intérieur (du corps) de la fonction `f(...)`
- `a` et `b` sont visibles uniquement à l'intérieur (du corps) de la fonction `setup()`

Le passage des paramètres aux fonctions

```
int f(int x, int y) {  
    return x+y;  
}  
void setup() {  
    int somme;  
    somme = f(2,3);  
    int a=4, b=5;  
    somme = f(a,b);  
    println(somme);  
}
```

les valeurs de **a** et **b**
sont affectées à **x** et **y**

//somme devient 5

//somme devient 9

- **x** et **y** sont visibles/connus uniquement à l'intérieur (du corps) de la fonction **f(...)**
- **a** et **b** sont visibles uniquement à l'intérieur (du corps) de la fonction **setup()**

Arrêter l'appel à draw() après 50 itérations

Solution : on utilise une variable globale **compteur**

```
int compteur = 0;
void setup() {
  size(600,600);
}
void draw() {
  compteur = compteur + 1;
  if (compteur < 50) {
    line(random(600), random(600), // pixel x
         random(600), random(600)); // pixel y
  }
}
```

Comment faire d'autres dessins après l'itération 50 ?

Programmation risquée

Soit le même programme en Processing ou C : qu'est ce qu'il affiche ?

Processing

```
int fortune = 100000;
void direSiJeSuisRiche () {
    if (fortune >= 100000)
        println ("Riche");
}
void setup () {
    int fortune = 100;
    direSiJeSuisRiche ();
}
```

C

```
int fortune = 100000;
void direSiJeSuisRiche () {
    if (fortune >= 100000)
        printf ("Riche");
}
void main () {
    int fortune = 100;
    direSiJeSuisRiche ();
}
```

Réponse : fortune \neq fortune

- la première est une variable globale, la deuxième est locale

Programmation risquée

Soit le même programme en Processing ou C : qu'est ce qu'il affiche ?

Processing

```
int fortune = 100000;
void direSiJeSuisRiche () {
    if (fortune >= 100000)
        println ("Riche");
}
void setup () {
    int fortune = 100;
    direSiJeSuisRiche ();
}
```

C

```
int fortune = 100000;
void direSiJeSuisRiche () {
    if (fortune >= 100000)
        printf ("Riche");
}
void main () {
    int fortune = 100;
    direSiJeSuisRiche ();
}
```

Réponse : ~~fortune~~ ≠ fortune

- la première est une variable globale, la deuxième est locale

Programmation ultra-riquée

```
int fortune;  
void f(int fortune){  
    fortune=9;  
}  
void g(){  
    int fortune;  
    fortune = 15;  
}  
void h(){  
    fortune = 9;  
}  
void setup(){  
    fortune = 12;  
    f(fortune);  
    println(fortune);  
    g();  
    println(fortune);  
    h();  
    println(fortune);  
}
```

Que affiche ce programme ?

Programmation ultra-risquée

```
int fortune;  
void f(int fortune) {  
    fortune=9;  
}  
void g() {  
    int fortune;  
    fortune = 15;  
}  
void h() {  
    fortune = 9;  
}  
void setup() {  
    fortune = 12;  
    f(fortune);  
    println(fortune);  
    g();  
    println(fortune);  
    h();  
    println(fortune);  
}
```

Attention :

fortune \neq fortune \neq fortune

- la première est une variable globale
 - visible partout
- la deuxième est une variable locale, un argument de la fonction `f()`
- la troisième est une variable locale dans la fonction `g()`

Programmation ultra-risquée

```
int fortune;  
void f(int fortuneArg){  
    fortuneArg=9;  
}  
void g(){  
    int fortuneLocal;  
    fortuneLocal = 15;  
}  
void h(){  
    fortune = 9;  
}  
void setup(){  
    fortune = 12;  
    f(fortune);  
    println(fortune);  
    g();  
    println(fortune);  
    h();  
    println(fortune);  
}
```

Solution : utiliser des noms de variables différents!!