Cours 1

I. Introduction aux variables

Soit le programme suivant :

Son exécution donne ce que vous distinguez dans la fenêtre inférieure noire.
Ce programme est composé de trois instructions : deux print et un println.

Je retiens que deux instructions sont destinées à afficher dans la console :
- print : qui affiche et laisse le curseur derrière la dernière lettre
- println : qui affiche et poursuit par un retour à la ligne.

1°) Les variables

Une variable est caractérisée par trois attributs :

- un nom : commence forcément par un caractère puis caractère ou chiffre. C fait la distinction entre minuscule et majuscule : mavariable et MaVariable sont 2 variables distinctes.
- un type : il définit le genre de valeurs que contiendra la variable par exemple : int, char, unsigned char, float, double et des types construits qui seront vus plus tard.
- une valeur qui est forcément du type défini plus haut.
- Une adresse qui sera vu plus tard.

2°) Déclaration des variables

L'utilisation d'une variable nécessite sa déclaration sous la forme : type nom_variable.
Cette déclaration doit être vue comme un contrat entre le programmeur et le compilateur. Comme tout contrat, il devra être respecté sinon le programmeur s'expose à une série de messages d'erreurs.

Exemple : int a,b,c; déclare 3 variables de type entières.

Une déclaration est un contrat !!!
Vous venez de dire, promis juré dans les variables nommées a,b,c vous ne mettrez que des entiers et que par conséquent jamais vous n'écrirez a=1.2; Ce sont les termes de votre contrat que vous devez respecter !!!

3°) Affectation d'une valeur dans une variable

Mettre une valeur dans une variable est ce que l'on appelle une affectation. Il existe plusieurs types d'affectation :

- directe avec le signe =
- à l'aide des opérateurs +=, -=, *=, /=, %=. Ces opérateurs seront utilisés plus loin.

Par exemple
a=5;
met 5 dans a.

Vous devez pensez à partir de maintenant l'opérateur d'affectation comme dissymétrique a=5 signifie que c'est 5 qui va dans a !!! et non pas l'inverse.

4°) Les expressions

Les expressions arithmétiques sont construites à l'aide des opérateurs '+', '-','*', '/'. On dispose aussi de l'opérateur modulo '%'.

Exercice 1

On rappelle que le code ASCII de x est 120, celui de '0' est 48.
On a la déclaration suivante : int i,j;

Que trouvera-t-on dans i avec les affectations :

```
i = 'x';
i = '0';
i = 3.3;
i = ('x'-'0')/3;
i = ('x'-'0')%3;
```

Le symbole « / » désigne la division. Celle-ci sera entière (c'est à dire sans reste) si elle est entourée par deux entiers. L'associativité de « * » et « / » est gauche vers la droite, ce qui signifie que l'on réalise ces opérations en commençant par la gauche. Que trouvera-t-on dans i avec les affectations:

```
i = 2 * j / 2; si la valeur dans j est 5
i = 2 * (j / 2); si la valeur dans j est 5
```

La division en C est par défaut une division entière. Ainsi 5/2 ne fait pas 2.5 mais 2 avec un reste de 1. Ce reste peut être trouvé avec l'opérateur modulo : %

Il existe en C un opérateur d'incrémentation : ++ et un opérateur de décrémentation : --

Exercice 2

Étant donné le programme :

```
void setup() {
  int i=1;
  print("i = ");println(i);
  print("i = ");println(++i);
  print("i = ");println(i++);
  print("i = ");println(i);
  print("i = ");println(i--);
  print("i = ");println(i);
  print("i = ");println(--i);
  print("i = ");println(i);
}
void draw() {
}
```

Que peut-on voir à l'écran lorsqu'on lance le programme ?

Les expressions booléennes

Une expression booléenne est une expression qui peut prendre deux valeurs : vraie et faux.

La notion d'expression booléenne n'a pas de véritable signification en C. Seule la notion d'expression existe. Ainsi remplacer l'une par l'autre ne pose aucun problème à un compilateur. Il en résulte que les deux valeurs d'une expression booléennes sont numériques : 1 (vrai) ou 0 (faux)

Le meilleur moyen de construire une expression booléenne est de partir des opérateurs de comparaison (ou opérateurs relationnels).

opérateurs relationnels	< <= > >=	Gauche -> Droite
opérateurs de comparaison	== !=	Gauche -> Droite
et logique	&&	Gauche -> Droite
ou logique	||	Gauche -> Droite

Exemples
a==4
b<18
(c>34) && (d!=89)

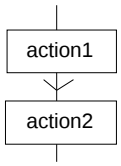
Exercice 3
Si les valeurs de a, b, c sont respectivement 12, 24, 56, quelles sont les valeurs des expressions booléennes :
a == 12
a != 12
b < 24
b >= 24
(c ==56) && (b>=23)
(c !=56) || (b>=23)

II.Structures de contrôle

Un programme peut être défini comme un enchaînement d'actions qui modifient en fonction de certaines conditions l'état de l'ensemble V des variables.
L'action est exécutée par un processeur, homme ou machine, qui comprend un certain langage ; elle doit par conséquent être décrite dans ce langage.
La séquence, le choix et l'itération sont à la base de toute l'algorithmique.

a) Enchaînement inconditionnel : la séquence

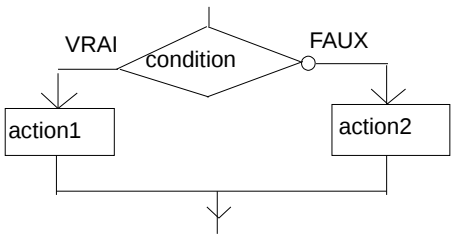
```
action1;
action2;
```



b) Enchaînement conditionnel : le choix

```
SI condition ALORS :
   action1
SINON :
   action2
FINSI
```

```
SI condition ALORS :
   action1
FINSI
```



En C ces structures de contrôles seront codées :

```
if (condition) {
  action1;
}
```

```
if (condition) {
  action1;
} else {
  action2;
}
```

où condition est en général une expression qui donne un résultat différent ou égal à 0. On reconnaît une **expression booléenne** (E.B.)

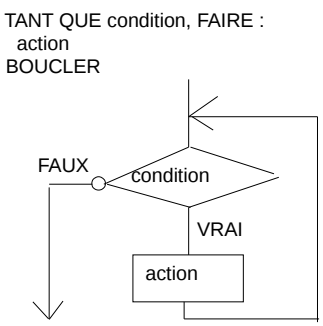
Exercice 4

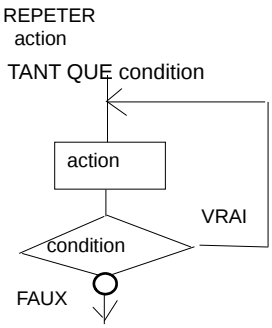
Qu'afficheront à l'écran les programmes suivants ?

```
void setup() {
  int a;
  a=7*5/6;
  if (a==5) {
    println("Bonjour");
  }
}
void draw() {
}
```

```
void setup() {
  int a,b;
  a=6/7;
  b = 12 + ++a;
  if (b!=13) {
    println("Salutations");
  } else {
    println("Bonjour a tous");
  }
}
void draw() {
}
```

c) Enchaînement répété : l'itération





En C ces structures de contrôles seront codées :

```
while (condition) {
   actions;
}
```

```
do {
  actions;
} while (condition);
```

Exemples

```
a=5;
while (a>0) {
   println("Bonjour");
   a--;
}
```

```
a=5;
do {
  println("Bonjour");
   a--;
} while (a>0);
```

Exercice 5

Qu'afficheront à l'écran les programmes suivants ?

```
void setup() {
  int a,i;
  a = 7 * 5 / 6;
  i = 0;
  while (i<a) {
    println("Bonjour");
    i++;
  }
}
void draw() {
}
```

```
void setup() {
  int a,b;
  a = 6 / 7;
  b = 12 + ++a;
  do  {
    println("Salutations");
    b++;
  } while (b <= 13);
}
void draw() {
}
```

Il existe aussi la structure :

```
for (debut ; condition de non fin ; passage à la suite) {
  actions;
}
```

Cette structure de boucle est très utilisée. Il vous faut la connaître par cœur :

```
// boucle qui s'exécute 10 fois
for (i=0 ; i<10 ; i++) { // i : compteur de boucle
  println("Salutations");
}
```

On a aussi :

```
while (true) { // while(1) en C mais pas en processing
  action1s
  if (condition) break;
  action2s;
}
```

d) Exercices Processing

Exercice 6

1°) Quelles erreurs ont été commises dans chacun des groupes d'instructions suivants :

a)
```
if (a<b) println("ascendant")
  else println("non ascendant");
```

b)
```
while a<24
      a++;
```

c)
```
do c=key while(c!='\n');//lecture du clavier par processing
```

d)
```
do while((c=key)!='\n');
```

e)
```
do {} while(1);
```

2°) Soit le petit programme suivant :

```
int a,b,c;
a=5;
b=a+5;
a+=b+a+5;
c=a;
while(true) { // while(1) en C mais pas en processing !
  print("Bonjour");
  if (b==10) break;
  print("Bye"); // Au revoir
}
```

Qu'affichera-t-il dans la console d'affichage ?

Exercice 7

Programme	Affichage (réponse)

`void setup() {`
`  int i,j;`
`  for(i=0;i<5;i++){`
`  for(j=i;j<5;j++)`
`  print("++");`
`  println();`
`  }`
`}`
 	
`void draw() {`
`}`	

On désire avoir l'affichage ci-contre sur l'écran. Modifier le programme ci-dessus pour cela en utilisant aussi une double boucle **Réponse** :	*+ **+ ***+ ****+ *****+

Exercice 8 (à faire en salle d'informatique)

Un programme pour faire l'acquisition d'un nombre (provenant du clavier) dans une variable, qui affiche la valeur pour vérification est donné ci-dessous.

```
int nb=0;
char Termine=0;

void setup() {
}

void draw() {
}

void keyPressed() {
 char ch;
 ch = key;
 if ((ch >='0')&&(ch <='9'))
   nb = 10*nb + ch-'0';
 else if(key==RETURN||key==ENTER) {
   println("La valeur entree est : ",nb);
```

```
   Termine=1; nb=0;
  }
}
```

Comme vous pouvez le remarquer les trois parties setup(), draw() et keypressed() sont obligatoires. keyPressed est appelé à chaque appui de touche au clavier. La touche sur laquelle on a appuyé se trouve dans la variable key.

1) On vous demande de le modifier pour prendre en compte une saisie de nombres négatifs. Comment modifier votre programme pour déplacer l'affichage dans draw() ?

2) Modifiez-le pour revenir aux nombres positifs mais hexadécimaux.

Cours 2

Cours sur les tableaux. Ils sont plus faciles à utiliser avec une initialisation.

I. Les listes (ou tableaux)

En C, un tableau à une dimension est déclaré ainsi :

type_des_éléments nom_du_tableau**[**nombre_éléments**]**;

Les éléments du tableau sont numérotés de 0 à nombre_éléments-1.

En processing le meilleur moyen de déclarer un tableau est de l'initialiser

Par exemple :

```
int[] tab={1,123,1210,4560,780,3200}; // en C : int tab[]={....};
```

déclare un tableau de 6 cases numérotées de 0 à 5.

La déclaration d'un tableau réalise en fait deux choses :
- réservation de place en mémoire pour les cases du tableau,
- le nom du tableau désigne désormais son adresse.

Par exemple avec la déclaration ci-dessus, vous venez de dire promis juré que désormais tab désigne l'adresse du tableau qui comporte 6 cases numérotées do 0 à 5. Ce sont les termes de votre contrat que vous devez respecter !!!
tab[6], par exemple ne vous est pas alloué !

Exercice1

1°) Soit le programme suivant :

```
void setup() {
  int[] tab=new int[3]; // declarer un tableau sans l'initialiser
// en C standard : int tab[3];
  tab[0]=5;
  tab[1]=tab[0]+5;
  tab[0]+=tab[1]+tab[0]+5;
  tab[2]=tab[0];
  while(true) {
    println("Bonjour");
    if (tab[1]==11) break;
    tab[1]++ ;
    println("Bye");
  }
}
void draw() {
}
```

Qu'affichera-t-il dans la console ?

2°) Quels résultats fournira le programme :

```
void setup() {
  int[] t={10,20,30};
  int i,j;
  for (i = 0; i < 3;i++) println(t[i]);
  for (i = 0,j = 0;i < 3;i++) t[i] = j++ + 1;
  for (i = 0; i < 3;i++) println(t[i]);
}
```

3°) Soit le tableau t déclaré ainsi : float[] t={1.5,5.7,-1.1};
Écrire les (seules) instructions permettant de calculer dans une variable nommée som, la somme des éléments de t.

4°) Réaliser un programme qui affiche le résultat de la conversion d'un nombre entier positif, exprimé en base 10, dans une base quelconque (comprise entre 2 et 10). Le nombre à convertir ainsi que la valeur de la base seront fournis en données.
i) Écrire l'algorithme réalisant ce programme.
ii) Coder cet algorithme en C, en utilisant un ou plusieurs tableaux.

Application : recherche de maximum dans un tableau

```
int[] tab={1,123,1210,4560,780,3200};
int i,max;
max = tab[0]; // initialisation de l'algorithme
for(i=1;i<6;i++) //pourquoi commencer à 1 ?
  if (max<tab[i])
    max = tab[i];
println("max=",max);
```

Exercice 2 : ajouter une recherche de minimum.

II. Les sous-programmes ou méthodes

Une action peut être une procédure (c'est un sous programme). Celle-ci a l'une des deux structures :

```
void  NomProcedure(void)
{
  Code;
}
```

```
void  NomProcedure()
{
  Code;
}
```

Elle s'appelle par "NomProcedure();" ou "NomProcedure(void);"
La structure de nos programmes est maintenant du type :

L'utilisation d'un sous-programme ou procédure mérite quelques explications :

J'écris le code	***J'appelle dans setup() ou ailleurs***

`void truc() {`
`....`
`}`
	`truc();`

Attention : L'utilisation d'un sous-programme nécessite une déclaration qui peut encore être considérée comme un contrat. Vous venez de dire promis juré que désormais votre sous-programme s'appelle truc qu'il n'a pas de paramètre (parenthèse vide) et qu'il ne retourne rien (void)

Introduction des variables locales et globales.

Exercice 3

1°) Soit le programme suivant :

```
void setup() {
  truc();
}
void truc(void) {
  int[] tab={0,0,0};
  tab[0]=5;
  tab[1]=tab[0]+6;
  tab[0]+=tab[1]+tab[0]+5;
  tab[2]=tab[0];
  while(true) {
    println("Bonjour");
    if (tab[1]==12) break;
    tab[1]++;
    println("Bye");
  }
}
```

Qu'affichera-t-il dans la fenêtre texte ?

2°) Qu'affichent les programmes suivants :

```
float nb;
void setup() {
  nb=2.0;
  nb = sqrt(nb);
  print("Racine = ");println(nb);
}

void draw() {
}
```

```
float nb;
void setup() {
  nb=2.5;
  nb = pow(nb,2);
  print("Calcul = ");println(nb);
}
```

III.Switch

Lorsque plusieurs alternatives sont emboîtées, il s'agit d'un choix multiple, ce qui dans un algorithme peut s'écrire :
AU CAS OU
condition 1, FAIRE : action 1
condition 2, FAIRE : action 2
condition 3, FAIRE : action 3
condition 4, FAIRE : action 4
DANS LES AUTRES CAS, FAIRE : action 0

Attention : L'ordre des conditions, dans un tel choix multiple, est essentiel.

En C :

```
switch (variable)  {          /* variable est de type int ou char
*/
case valeur_1 : action_1; break;
case valeur_2 : action_2; break;
...
case valeur_n : action_n; break;
default : action_0;
}
```

Les "break" imposent des conditions exclusives sur la variable. S'ils ne sont pas présents (car non obligatoires) l'algorithme exécuté peut se traduire :
AU CAS OU
condition 1, FAIRE : action_1 et toutes les suivantes
condition 2, FAIRE : action_2 et toutes les suivantes
condition 3, FAIRE : action_3 et toutes les suivantes
condition 4, FAIRE : action_4 et la suivante
DANS LES AUTRES CAS, FAIRE : action_0

Exercice 4 (salle informatique)
Pour un nombre fourni, donner à l'aide d'un menu le choix à l'utilisateur d'afficher : x^2, x^3, 1/x, $x^{1/2}$.
Suivant le choix traiter l'entrée : 1/x n'existe pas pour x=0, $x^{1/2}$ n'existe pas pour x négatif.
Écrire des procédures pour le traitement par variable locale. Revenir au menu initial si l'on veut recommencer.

Cours 3 : Dessiner avec Processing

Dans ce troisième cours, nous allons commencer à dessiner dans la fenêtre graphique de Processing.

I. L'espace de dessin

Vous avez certainement remarqué la petite fenêtre grise qui s'ouvre chaque fois que vous lancez un programme. C'est cette fenêtre qui est appelée espace de dessin. Elle est caractérisée par une taille et un repère 2D que nous allons détailler maintenant.

1°) Modifier la taille de l'espace de dessin

Par défaut la taille de la fenêtre affichée est 100x100 pixels.
Le changement se fait avec l'instruction size : size(largeur,hauteur)
size() ; sans paramètre reprendra les valeurs par défaut.

2°) Coordonnées dans l'espace de dessin

L'origine des coordonnées est comme toujours le point supérieur gauche de la fenêtre de dessin. L'axe horizontal est x et l'axe vertical est y.

II. Les formes

Nous allons présenter dans cette partie qu'un nombre limité de formes. Mais cela nous suffira grandement pour nos travaux futurs.

1°) Le point

Le sous-programme "point permet de dessiner un point unique à un endroit précis.
Voici un exemple :

```
void setup() {
  point(50, 50);
}

void draw() {
}
```

Exercice 1

Dessiner un cercle de centre (50,50) et de rayon 25 avec le sous-programme point... et une boucle.

Remarque : cet exercice a seulement pour but de travailler avec les fonctions sinus, cosinus mais certainement pas de montrer comment désormais on dessinera un cercle !

2°) La ligne

Le sous-programme responsable de ce tracé s'appelle tout simplement line.

Exercice 2

Tracer le résultat du programme suivant :

```
void setup() {
  line(15, 90, 95, 10);
}

void draw() {
}
```

3°) Le rectangle

Un rectangle se dessine par quatre valeurs en faisant l'appel de rect(x,y,largeur,hauteur). La première paire de valeurs x et y, par défaut (mode CORNER) correspond au coin supérieur gauche du rectangle, à l'instar du point. En revanche la seconde paire de valeurs ne va pas se référer à la position du coin inférieur droit, mais à la largeur (sur l'axe des x, horizontal) et à la hauteur (sur l'axe des y, vertical) de ce rectangle.

Exemples

Les deux exemples ci-dessous donnent le même résultat.

```
void setup() {
  rect(10, 10, 80, 80);
}

void draw() {
}
```

```
void setup() {
  rectMode(CENTER);
  rect(50, 50, 80, 80);
}

void draw() {
}
```

4°) L'ellipse

Comme pour le rectangle, l'ellipse se construit sous les modes CENTER (par défaut), et CORNER .

Exemples

Les deux exemples ci-dessous donnent le même résultat.

```
void setup() {
  ellipseMode(CORNER);
  ellipse(10, 10, 80, 80);
}

void draw() {
}
```

```
void setup() {
  ellipseMode(CENTER);
  ellipse(50, 50, 80, 80);
}

void draw() {
}
```

5°) Et le reste

Les primitives "triangle", "quad" (quadrilatère), "arc", "courbe"... sont aussi disponibles.

6°) Contour

Vous pouvez dessiner ou pas les contours. Sans contour se fait avec "noStroke()" tandis-que les contours se font avec "stroke()" (option par défaut).

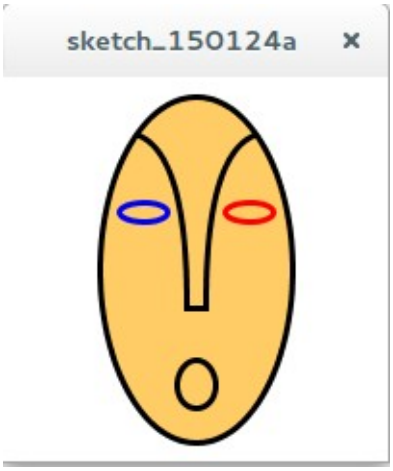
7°) Remplissage

L'option par défaut est le remplissage. Si vous ne voulez plus de remplissage appelez "noFill()" et si vous voulez revenir en arrière appelez "Fill()".

8°) Exemple complet

Un exemple complet présenté dans un document d'apprentissage de processing est le suivant.

La forme obtenue est présentée ci-contre tandis que le programme correspondant est présenté après.




```
void setup() {
  size(200, 200);
  smooth();
  background(255); // on dessine un fond blanc
  stroke(#000000); // le contour sera noir
  fill(#FFCC66); // le remplissage sera jaune
  strokeWeight(3);
  translate(width / 2, height / 2);
  ellipse(0, 0, 100, 180); // forme elliptique du masque
  ellipse(0, 60, 20, 25); // ellipse de la bouche
  stroke(255, 0, 0); // le contour sera rouge
  ellipse(28, -30, 25, 10); // ellipse de l'oeil droit
  stroke(0, 0, 255); // le contour sera bleu
  ellipse(-27, -30, 25, 10); // ellipse de l'oeil gauche
  noFill(); // les prochaines formes n'auront pas de remplissage
  stroke(#000000);
// le contour des prochaines formes sera noir
  bezier(-30, -70, -5, -60, -5, 0, -5, 20); // courbe du sourcil
droit
  bezier(30, -70, 5, -60, 5, 0, 5, 20); // courbe du sourcil gauche
  line(-5, 20, 5, 20); // ligne du nez pour joindre l'extrémité des
courbes
}

void draw() {
}
```

III.Exercices

Exercice 3

Refaire le masque ci-dessus en supprimant le translate du programme.

Exercice 4

Écrire un programme qui dessine un échiquier vide 8x8 à l'écran. Vous avez l'autorisation d'utiliser l'instruction "`background(255);`" qui vous permettra de dessiner la moitié des cases.
1°) Réaliser cet échiquier avec une boucle par ligne. Il y aura donc 8 boucles.

2°) Réaliser ce même échiquier avec une double boucle.

Pour information : il est possible de faire le travail avec les translations. Mais celles-ci ne seront expliquées que dans le cours 5.

Exercice 5

Dessiner un soleil brillant dans un ciel bleu. Le soleil sera stylisé comme celui que dessinent les enfants, avec des rayons.

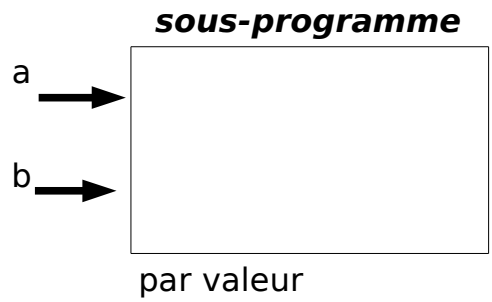
Cours 4 : retour sur les sous-programmes

I. Sous-programmes et paramètres

En C, la portée des variables, c'est à dire les endroits où elles sont connues, dépend de l'endroit où elles ont été déclarée. Il existe cependant des moyens de déroger à cette règle, mais nous ne les examinerons pas ici.

Les procédures permettent le passage de paramètres.

1°) Quand ?



2°) Comment ?

J'écris le code	*J'appelle*

`void truc(int a) {`
`....`
`}`
	`truc(5);`
	`truc(c);`

Attention : L'utilisation l'criture du code d'un sous-programme peut encore être considérée comme un contrat. Vous venez de dire promis juré que désormais votre sous-programme s'appelle truc qu'il a un seul paramètre par valeur (entier) et qu'il ne retourne rien (void)

Code = Contrat

```
void truc(int a);
```

Ce que je retourne	Comment je m'appelle	Ce que je passe comme paramètre

Respecter le contrat de la déclaration oblige le programmeur à respecter un certain nombre de règles, un peu comme un contrat de mariage.

- Quand j'écris le code la première ligne est identique à ma déclaration sauf que je remplace le « ; » par une accolade ouvrante : « { »

- Quand j'appelle, même si je le fais avec une variable je ne rappelle pas son type. La raison est qu'alors le compilateur prendrait cela comme une nouvelle déclaration.

Exercice 1

Soit le programme C suivant :

```
/******* DECLARATIONS ******/
float a,b,c; // c globale donc initialisée à 0.0

void setup() {
  a=3 ; b=5 ;
  println(a,b,c);
  Proc1(a);
  println(a,b);
  Proc2(a,b,c);
  println(a,b,c,i);
}

void Proc1(float x)
{ float y;
  println(x);
  x = x+10;
  y = x*4;
  println(x,y);
}

void Proc2(float x,float y,float z)
{ float i;
  i=0.0 ;
  println(x,y,i);
  x = x+10;
  i = x;
  y = y*4;
  z=i+y;
  println(x,y,z,i);
}
```

On demande :

de nommer les variables locales et les variables globales,
de trouver une erreur de portée de variable, erreur détectée à la compilation,
d'écrire les valeurs qui seront affichées sur l'écran au fur et à mesure de son déroulement.
Comment éviter les trois variables globales ?

II. Fonctions et paramètres

Notion de fonction. L'instruction return.

J'écris le code	***J'appelle***

`float truc(int a) {`
`....`
`return x;`
`}`
	`d=truc(5);`
	`d=truc(c);`

Attention : L'utilisation d'une fonction nécessite une déclaration qui peut encore être considérée comme un contrat. Vous venez de dire promis juré que désormais votre sous-programme s'appelle truc qu'il a un seul paramètre par valeur (de type int) et qu'il retourne un flottant (float)

code = Contrat

`float truc(int a)`

Ce que je retourne	Comment je m'appelle	Ce que je passe comme paramètre

Respecter le contrat de la déclaration oblige le programmeur à respecter un certain nombre de règles, un peu comme un contrat de mariage.
Quand j'écris le code de la fonction la première ligne est identique à ma déclaration sauf que je remplace le « ; » par une accolade ouvrante : « { »
Quand j'appelle, même si je le fais avec une variable je ne rappelle pas son type. La raison est qu'alors le compilateur prendrait cela comme une nouvelle déclaration.
Quand j'appelle une fonction je dois mettre la valeur retournée quelquepart, en général dans une variable prévue à cet effet.

III. Exercices

Exercice 2

Soit le programme C suivant :

```
/******* DECLARATIONS ******/
float a,b,c; // c globale donc initialisée à 0.0

void setup() {
  a=3 ; b=5 ;
  println(a,b,c);
  b=Proc2(a);
  println(a,b);
  a = Proc3(a,Proc2(b),c);
  println(a,b,c,i);
}

float Proc2(float x)
```

```
{ float y;
  println(x);
  x = x+10;
  y = x*4;
  println(x,y);
  return(y*2);
}

float Proc3(float x,float y,float z)
{ float i;
  i=0.0 ;
  println(x,y,i);
  x = x+10;
  i = x;
  y = y*4;
  z=i+y;
  println(x,y,z,i);
  return(z+x+y);
}
```

Attention : On note qu'il est possible avec processing d'écrire le setup avant les sous-programmes. En C standard il faudrait une déclaration supplémentaire.

On demande :
- de nommer les variables locales et les variables globales,
- de trouver une erreur de portée de variable, erreur détectée à la compilation,
- d'écrire les valeurs qui seront affichées sur l'écran au fur et à mesure de son déroulement.

Peut-on éviter les trois variables globales ?

Exercice 3

Écrire une fonction mini() qui calcule le minimum d'un tableau qui est passé en paramètre. On pourra s'inspirer du code suivant

```
int[] tab={1,123,1210,4560,780,3200};
int i,max;

int maxi(int[] table){
  int i;
  max = table[0]; // initialisation de l'algorithme
  for(i=1;i<6;i++) //pourquoi commencer à 1
    if (max<table[i])
      max = table[i];
  return max;
}

void setup(){
  println("max=",maxi(tab));;
}
```

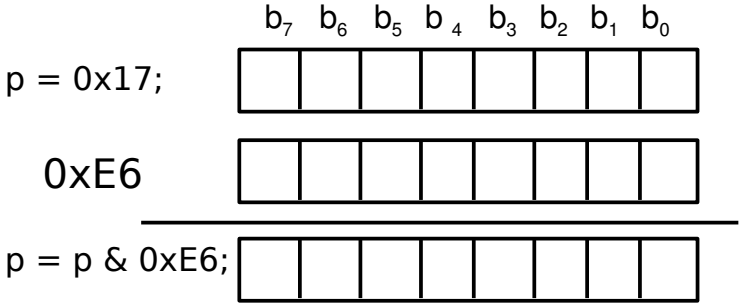
Exercice 4

Écrire un programme qui utilise une fonction factorielle (avec calcul itératif).

IV. Opérateurs complémentaires importants

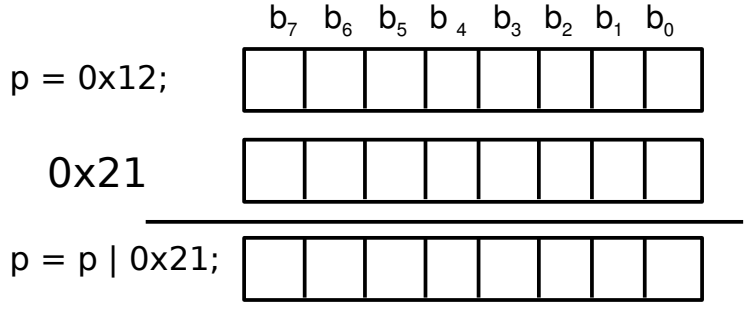
Propriété du ET logique :
x.0 = 0
x.1 = x
Ces propriétés servent à mettre un 0 dans un bit particulier. La constante utilisée pour cela est appelée **masque**.

Exercice 5
Compléter et marquer les bits qui ont été mis à 0



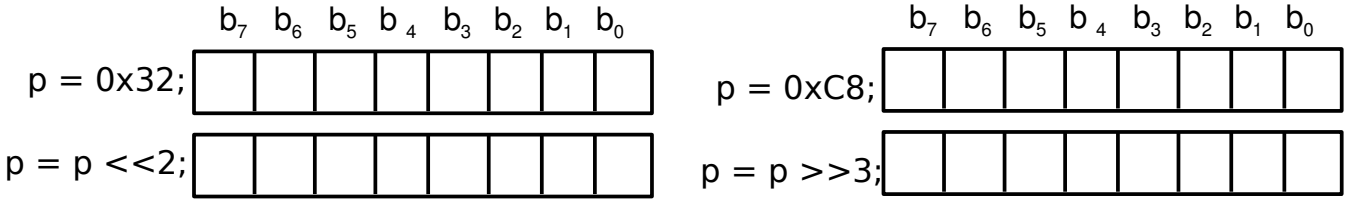
Propriété du OU logique :
x+0 = x
x+1 = 1
Ces propriétés servent à mettre un 1 dans un bit particulier.

Exercice 6
Compléter et marquer les bits qui ont été mis à 1



Opérateurs de décalage << et >> :

Exercice 7
Compléter en donnant les valeurs décimales correspondantes :



Cours 5 : Plus loin avec le dessin

Architecture de programme avec setup() et draw(). Gestion fine des appels de draw() avec frameRate().

I. Utiliser des fonctions et sous-programmes pour le dessin

Dessiner une forme complexe peut être compliqué. Vous pouvez le réaliser avec des sous-programmes et des paramètres.

Exercice 1

Réaliser votre propre sous-programme "myRect" qui trace un rectangle avec la primitive "line" qui utilise les paramètres passés avec myRect.

II.Transformations

Un certain nombre de transformations sont disponibles dans Processing. Il s'agit de déplacer, trourner, mettre à l'échelle. Nous allons explorer les deux premières.

1°) Déplacer

La primitive correspondante s'appelle "translate" et nécessite deux paramètres :

- translation suivant l'axe des x

- translation suivant l'axe des y

Voici un exemple :

```
void setup() {
  translate(width/2,height/2);
  ellipseMode(CENTER);
  ellipse(10, 10, 40, 40);
}

void draw() {
}
```

Exercice 2

Reprendre l'exercice sur l'échiquier avec translate (exercice 3 du cours 3).

2°) Tourner

Deux systèmes de mesure existent pour mesurer un angle : les radians et les degrés. Par défaut, Processing travaille en radians mais pour nous il est d'habitude plus facile de raisonner en degrés. Par exemple tourner de 180°, c'est faire un demi-tour. Processing permet de passer de transformer une unité en une autre grâce aux fonctions radians() et degrees() .

float d = degrees(PI/4); // transforme des radians en degrés

float r = radians(180.0); // transforme des degrés en radians

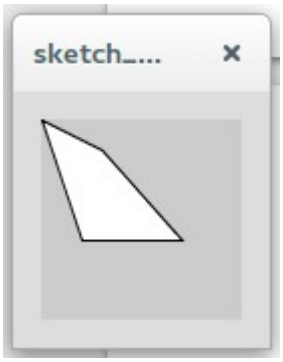
Illustrons la fonction rotate() par un exemple simple. Nous allons faire pivoter un carré autour de l'origine.

```
void setup() {
  size(200, 200);
  noStroke();
  fill(0);
  rotate(PI/4);
  rect(0, 0, 100, 100);
}

void draw() {
}
```

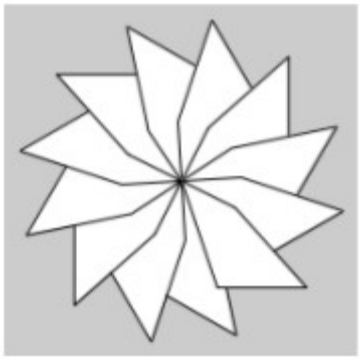
Exercice 2

On donne le programme ci-dessous et sa figure correspondante :

```
void setup() {
  squad(0, 0, 30, 15, 70, 60, 20, 60);
}

void draw() {
}
```



Pouvez-vous modifier le programme pour qu'il dessine la figure ci-dessous :



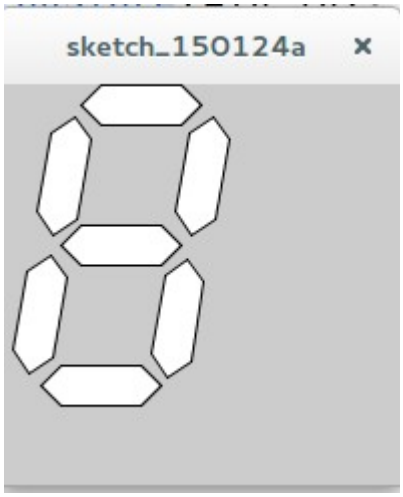
Exercice 3

Réaliser un afficheur 7 segments.

1°) Modifier le sous-programme "segment()" ci-dessous pour lui permettre d'allumer ou d'éteindre le segment. Choisir convenablement les couleurs allumées/éteintes. Tester.

2°) Modifier maintenant le sous-programme "digit7segs()" pour qu'il puisse allumer individuellement chacun des segments à partir d'une variable de type int passée comme paramètre. Le segment "a" sera choisi, à votre choix, comme poids faible ou poids fort.

```
void setup() {
  size(200,200);
  digit7segs();
}
void digit7segs(){
  translate(40,0);
  segment(); // segment a
  translate(-10,70);
  segment(); // segment g
  translate(-10,70);
  segment(); // segment d
  translate(15,-53);
  rotate(radians(100));
  segment(); // segment e
  translate(-70,0);
  segment(); // segment f
  translate(-12,-68);
  segment(); // segment b
  translate(72,0);
  segment(); // segment c
}

void segment() {
  beginShape();
  vertex(0,10);
  vertex(10,0);
  vertex(50,0);
  vertex(60,10);
  vertex(50,20);
  vertex(10,20);
 endShape(CLOSE);
}
void draw() {
}
```



Cours 1

Exercice 8

1°)

```
int nb=0;
char Termine=0;
char Neg=0;
void setup() {
}
void draw() {
}
void keyPressed() {
  char ch;
  ch = key;
  if (ch == '-')
    Neg=1;
  else if ((ch >= '0') && (ch <= '9'))
    nb = 10*nb + ch -'0';
  else if (key==RETURN || key==ENTER) {
    print("La valeur entree est : ");
    if (Neg==1) println(-nb); else println(nb);
    Termine=1; nb=0;
  }
}
```

Cet réponse ne fait pas d'analyse syntaxique. Si vous entrez 23-6 il agira comme si on avait entré -236 !

Pour déporter l'affichage dans draw() il faut déplacer bien sûr les « print » mais aussi le nb=0 sans oublier de mettre « Termine » à 0 !

2°)

```
int nb=0;
char Termine=0;
void setup() {
}
void draw() {
}
void keyPressed() {
  char ch;
  ch = key;
  if  ((ch >= 'A') && (ch <= 'F'))
    nb = 16*nb + ch - 'A' +10;
  else if ((ch >= '0') && (ch <= '9'))
    nb = 16*nb + ch -'0';
  else if (key==RETURN || key==ENTER) {
    print("La valeur entree est : ");
    println(nb);
    Termine=1; nb=0;
  }
}
```

Mais vous ne pouvez plus entrer que des nombres hexadécimaux !

Cours 3

Exercice 3

```
void setup() {
  size(200, 200);
  smooth();
  background(255); // on dessine un fond blanc
  stroke(#000000); // le contour sera noir
  fill(#FFCC66); // le remplissage sera jaune
  strokeWeight(3);
  //translate(width / 2, height / 2);
  ellipse(100, 100, 100, 180); // forme elliptique du masque
  ellipse(100, 160, 20, 25); // ellipse de la bouche
  stroke(255, 0, 0); // le contour sera rouge
  ellipse(128, 70, 25, 10); // ellipse de l'oeil droit
  stroke(0, 0, 255); // le contour sera bleu
  ellipse(73, 70, 25, 10); // ellipse de l'oeil gauche
  noFill(); // les prochaines formes n'auront pas de remplissage
  stroke(#000000);
// le contour des prochaines formes sera noir
  bezier(70, 30, 95, 40, 95, 100, 95, 120); // courbe du sourcil droit
  bezier(130, 30, 105, 40, 105, 100, 105, 120); // courbe du sourcil gauche
  line(95, 120, 105, 120); // ligne du nez pour joindre l'extrémité des courbes
}

void draw() {
}
```

Cours 4

Exercice 1 :

3.0 5.0 0.0
3.0
13.0 52.0
3.0 5.0
3.0 5.0 0.0
13.0 20.0 33.0 13.0
3.0 5.0 0.0

Exercice 2 :

3.0 5.0 0.0
3.0
13.0 52.0
3.0 104.0
104.0
114.0 456.0
3.0 912.0 0.0
13.0 3648.0 3661.0 13.0
7322.0 104.0 0.0

Cours 5

Exercice 2 (corrigé)

```
void setup() {
  size(200,200);
  smooth();
  translate(width/2, height/2);
  for (int i=0;i<360;i+=30){
    rotate(radians(30));
    quad(0, 0, 30, 15, 70, 60, 20, 60);
  }
}

void draw() {
}
```

Exercice 3

Voici un compteur qui est affiché pour ses 4 bits de poids faible en hexadécimal

```
int[] val={63,6,91,79,102,109,125,7,127,111,119,124,57,94,121,113};
int cmpt = 0;

void setup() {
  size(200,200);
  frameRate(1);
}
void digit7segs(){
  translate(40,0);
  segment();
  translate(-10,70);
  segment();
  translate(-10,70);
  segment();
  translate(15,-53);
  rotate(radians(100));
  segment();
  translate(-70,0);
```

```
  segment();
  translate(-12,-68);
  segment();
  translate(72,0);
  segment();
}

void segment() {
  beginShape();
  vertex(0,10);
  vertex(10,0);
  vertex(50,0);
  vertex(60,10);
  vertex(50,20);
  vertex(10,20);
 endShape(CLOSE);
}

void Affiche7segs(int val) {
  noStroke();
  translate(40,0);
  if ((val & 1) == 1) fill(255,0,0); else fill(255);
  segment(); // segment a
  translate(-10,70);
  if ((val & 64) == 64) fill(255,0,0); else fill(255);
  segment(); // segment g
  translate(-10,70);
  if ((val & 8) == 8) fill(255,0,0); else fill(255);
  segment(); // segment d
  translate(15,-53);
  rotate(radians(100));
  if ((val & 16) == 16) fill(255,0,0); else fill(255);
  segment(); // segment e
  translate(-70,0);
  if ((val & 32) == 32) fill(255,0,0); else fill(255);
  segment(); // segtment f
  translate(-12,-68);
  if ((val & 2) == 2) fill(255,0,0); else fill(255);
  segment(); // segment b
  translate(72,0);
  if ((val & 4) == 4) fill(255,0,0); else fill(255);
  segment(); // segment c
}
void draw() {
  Affiche7segs(val[cmpt & 15]);
 cmpt++;
}
```

Exercice 4 (corrigé) : affichage de 16 leds en forme de cercle alternées éteintes allumées.

```
int[] tabx={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
taby={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
void setup() {
  int i;
  size(300, 300);
  noStroke();  // Pas de bordure
```

```
  frameRate(2);  // Run "draw()" 2 times per second,
                 // if possible.
   for (i=0;i<16;i++) {
     tabx[i]= (int) (150+100*cos(i*TWO_PI/16));
     taby[i]= (int) (150+100*sin(i*TWO_PI/16));
   }
   ellipseMode(CENTER);
}

void draw() {
  char[] leds={128,255,128,255,128,255,128,255,128,255,128,255,128,255,128,255};
  int i;
  for(i=0;i<16;i++) {
    fill(leds[i],0,0);
    ellipse(tabx[i],taby[i],10,10);
  }
}
```

Expliquer les formules trigonométriques.

3°) **Exercice 5 :** affichage des 16 leds de la question précédente mais alternativement allumées et éteintes.

L'idée est de dessiner les leds puis de tester si la valeur est à 128 de la passer à 255 et inversement...

Ne pas oublier de sortir le tableau leds en variable générale (Pourquoi?)

Correction de l'exercice 5 :

```
int[] tabx={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},taby={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
char[] leds={128,255,128,255,128,255,128,255,128,255,128,255,128,255,128,255};
void setup() {
  int i;
  size(300, 300);
  noStroke();
  frameRate(2);  // Run "draw()" 2 times per second,
                 // if possible.
   for (i=0;i<16;i++) {
     tabx[i]= (int) (150+100*cos(i*TWO_PI/16));
     taby[i]= (int) (150+100*sin(i*TWO_PI/16));
   }
   ellipseMode(CENTER);
}

void draw() {

  int i;
  for(i=0;i<16;i++) {
    fill(leds[i],0,0);
    ellipse(tabx[i],taby[i],10,10);
    if (leds[i]==128)
      leds[i]=255;
    else
      leds[i]=128;
  }
}
```

4°) **Exercice 6 :** Réaliser une LED allumée qui fait le tour du cercle sans arrêt.

Correction

Le setup() est identique à l'exercice précédent. Seul le code de draw() est donc donné.

```
void draw() {
  int i;
  for(i=0;i<16;i++) {
    fill(leds[i],0,0);
    ellipse(tabx[i],taby[i],10,10);
  }
  // recherche de la led allumée
  for (i=0;i<16;i++)
    if (leds[i]==255) break;
  leds[i]=128; //on éteint la led trouvée
  if (i < 15)
    leds[i+1]=255; // allume la suivante si possible
  else
    leds[0]=255; // autrement suivante en zéro
}
```