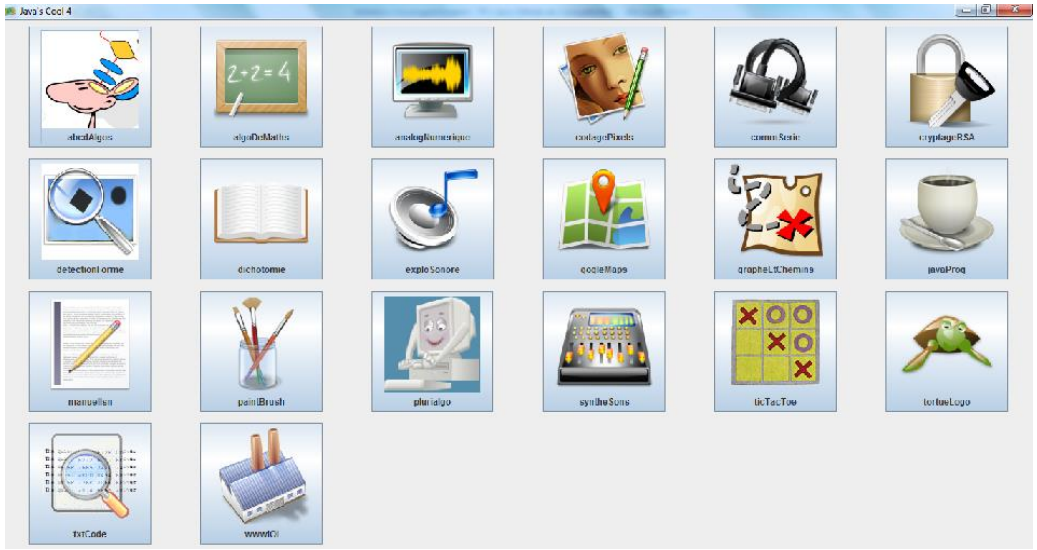
ISN : Initiation à la programmation TP 1

I) Premier programme avec Javascool

A) avec Javascool

- Sur votre lecteur réseau P : , créer un dossier que l'on nommera ISN
- Créer un sous dossier P :\JAVASCOOL et un sous dossier P :\TP1
- Télécharger le fichier Javascool-Proglets.jar : http ://javascool.gforge.inria.fr/ ?page=run
- Copier ce fichier dans P :\JAVASCOOL
- Lancer Javascool. On obtient la fenêtre ci-dessous :



L'environnement Java est un environnement de programmation en JAVA, simplifié par l'utilisation de Proglets et de la machine Java, développé par des chercheurs de l' INRIA.

Chaque icône correspond à une petite application (appelée une **proglet**) permettant de s'initier à la programmation en travaillant un point préçis.

Lancer la Proglet « abcdAlgos »

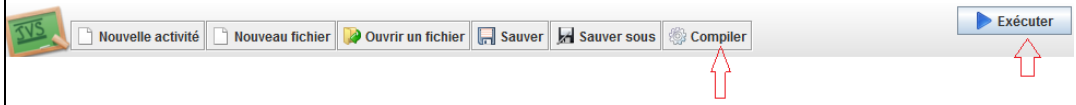
En cliquant sur « séquences d'instructions » on découvre le tutoriel « HelloWorld », qui montre le programme le plus simple que l'on peut écrire avec Javascool :

```
void main() {
    println("Hello World !");
}
```

En langage algorithmique :

```
Algorithme principal
Début
  Afficher("Hello World !")
Fin
```

Vous pouvez copier le code de ce programme Javascool dans l'éditeur à gauche (en le copiant avec Ctrl-C puis en le collant avec Ctrl-V), puis cliquer sur « Compiler », ce qui nécessite de sauvegarder votre premier fichier Javascool. Une fois l'emplacement du fichier choisi, la console doit afficher « Compilation réussie ! ».
Reste enfin à exécuter le programme (en cliquant sur « Exécuter » donc), pour voir s'afficher le texte voulu dans la console.



Exercice 1 :
Comme énoncé dans le tutoriel, modifier ce programme pour changer le texte qui s'affiche et ajouter de nouvelles phrases qui s'afficheront les unes après les autres.

Syntaxe à retenir

Structure générale d'un programme

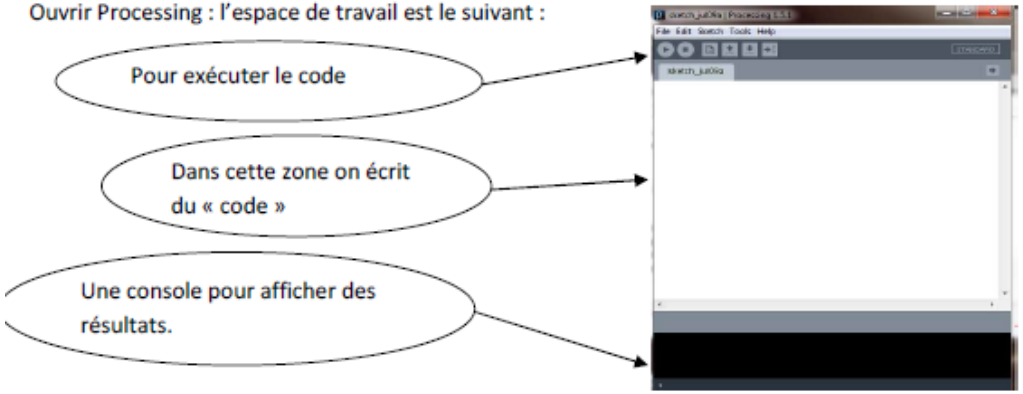
```
void main() {
        instruction 1 ;
        instruction 2 ;
        ..........
}
```

Ne pas oublier les points-virgules

B) avec Processing (https://processing.org/download/)

Processing est environnement de programmation Java qui permet d'exécuter des programmes dont le rendu est essentiellement graphique.

Ouvrir Processing : l'espace de travail est le suivant :



- Ecrire la ligne de code suivante :

println(" hello World") ;

et exécuter le programme.

En plus du « hello world » dans la console il est apparu une fenêtre de dessin (sans dessin) de dimension 100 pixels x 100 pixels.

La commande pour modifier cette dimension est :

size(largeur,hauteur) ;

- Essayez avec : size(200,200) ;

Le fond de la fenêtre est par défaut gris. On va le changer en blanc par la commande :

background(255,255,255) ;

Les codes couleurs sont à exprimer en RVB avec des entiers qui varient de 0 à 255. Ou en hexadécimal : p.ex. #FFFFFF correspond à blanc.

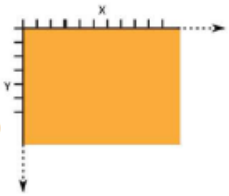
En noir avec : background(0,0,0) ; en rouge avec : background(255,0,0) ; etc.....

Quand on travaille en 2 dimensions (2D), on utilise deux axes de coordonnées x et y correspondant respectivement à la largeur (axe horizontal) et à la hauteur (axe vertical).

Par convention, le coin en haut à gauche correspond aux valeurs x=0 et y=0.

Les valeurs x sont croissantes vers la droite et les valeurs y sont croissantes vers le bas, c habitude du plan cartésien.

Ces valeurs x et y peuvent s'étendre théoriquement à l'infini, même si, en réalité, les contraintes de la taille de votre fenêtre vont délimiter la taille maximale d'une surface de création visible. C'est donc dans cet espace que nous allons dessiner.

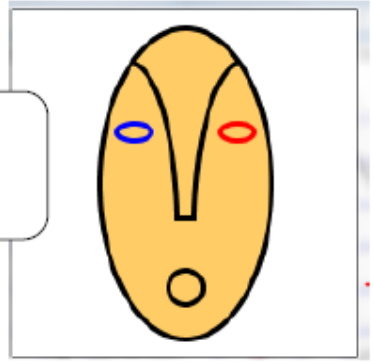


Dans Processing on peut écrire les lignes de code servant à dessiner un point, une ligne, un rectangle, une ellipse, un triangle, une courbe etc....

Le code pour réaliser un dessin se trouve sur l'**annexe 1** de ce TP.

- Il s'agit de réaliser dans une fenêtre de taille 200 x 200 pixels le dessin suivant :

- Il y a 4 ellipses
- Une ligne
- Deux courbes de Bézier.



- Le fond est blanc
- Le remplissage est jaune-orange
- Les contours sont lissés
- Les contours sont noirs
- Les yeux ont des contours bleu et rouge
- Les contours sont plus épais

Préambule : la couleur une synthèse additive des couleurs primaires R, G et B(vu en physique 1ère S) : http://dev.physicslab.org/asp/applets/additivecolors/default.asp

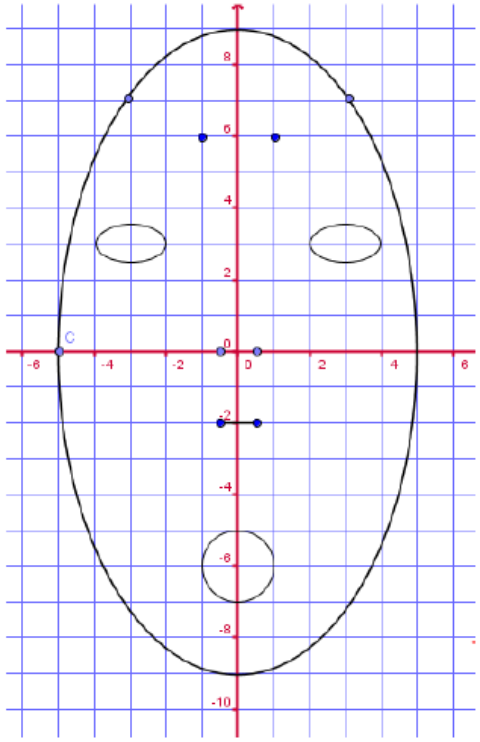
Début du programme

```
size(400,400) ;   // dimension de la fenêtre graphique
smooth() ; // lissage des traits
background(255,255,255) ; // on dessine un fond blanc sur la fenêtre graphique ;
stroke(0,0,0) ; // le contour de la fenêtre graphique sera noir
fill(255,204,102) ; // le remplissage sera jaune-orange
strokeWeight(3) ; // épaisseur des traits
translate(width/2,height/2) ; // le dessin sera translaté de 200 vers la droite et de 200 vers le bas
                              //width est la largeur de lz fenêtre et height en est la hauteur
```

La suite du programme est une liste d'instructions que l'orinateur exécutera chronologiquement

Voici le dessin réalisé avec pour origine O(0 ;0). Il suffira ensuite de le translater.

1 unité correspond à 20 et il faudra inverser les signes sur l'axe des Y



suite du programme

```
ellipse( ?, ?, ?, ?) ;// tête
ellipse( ?, ?, ?, ?) ;// bouche
stroke( ?, ?, ?) ; //couleur du contour de l'œil droit
ellipse( ?, ?, ?, ?) ;// œil droit
stroke( ?, ?, ?) ; couleur du contour de l'œil gauche
ellipse( ?, ?, ?, ?) ;// œil gauche
noFill() ; les prochaines figures n'auront pas de remplissage
stroke( ?, ?, ?) ; // leurs contours seront noir
bezier( ?, ?, ?, ?, ?, ?, ?, ?) ; //sourcil gauche
bezier( ?, ?, ?, ?, ?, ?, ?, ?) ; //sourcil droit
line( ?, ?, ?, ?) ; // nez
```

En fonction de l'avancée de vos travaux, vous pourrez rajouter une moustache, un chapeau, …..

Travail à rendre : A l'aide d'une feuille quadrillée, créer vous-même un dessin, par exemple un smiley ……

ANNEXE 1

La taille de la fenêtre est ici de 100 x 100 et le fond gris

LE POINT

Pour commencer à dessiner, nous allons partir d'un point. Sur l'écran, un point est l'équivalent d'un pixel localisé dans la fenêtre de visualisation par deux axes de coordonnées, x et y correspondant respectivement à la largeur (axe horizontal) et à la hauteur (axe vertical) de l'espace de dessin. En suivant ce principe, la création d'un point dans Processing s'effectue à l'aide de l'instruction point(x,y). Dans cet exemple, le point est très petit. Il est placé au centre de la fenêtre de visualisation.

Exemple `point(100, 100);`

LA LIGNE

Par définition, une ligne (AB) est constituée par une infinité de points entre un point de départ A et un point d'arrivée B. Pour la construire, nous allons nous intéresser uniquement aux coordonnées x et y de A et de B. Ainsi, si par exemple dans la fenêtre par défaut, le point A se situe dans la région en bas à gauche de votre fenêtre, et que le point B se situe en haut à droite, les instructions suivantes, peuvent dessiner cette ligne sous la forme line(xA,yA,xB,yB) :

par exemple : `line(15, 90, 95, 10);`

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 CO RNER) 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.

Exemple : `rect(10, 10, 80, 80);`

Comme les deux dernières valeurs (largeur et hauteur) sont identiques, on obtient un carré.
Amusez-vous à changer les valeurs et observez-en les résultats.
Pour que la première paire de valeurs corresponde au centre (le croisement des deux diagonales aux coordonnées 50, 50) du rectangle, il faut utiliser le mode CENTER comme suit :

```
rectM ode(CENTER);
rect(50, 50, 80, 40);
```

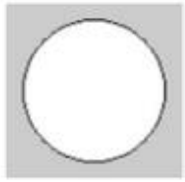
Cela donne le résultat identique à l'exemple précédent.

L'ELLIPSE

Comme pour le rectangle, l'ellipse se construit sous les modes CENTER (par défaut), et CORNER. Ainsi l'instruction suivante produit un cercle dont les coordonnées du centre sont les deux premières valeurs entre parenthèses. La troisième valeur correspond à la grandeur du diamètre sur l'axe horizontal (x) et la quatrième à la grandeur du diamètre sur l'axe vertical : notez que si les 3e et 4e valeurs sont identiques, on dessine un cercle et dans le cas contraire, une ellipse quelconque :

`ellipse(50, 50, 80, 80);`

Amusez-vous à faire varier les 3e et 4e valeurs et observez-en les résultats.



LE TRIANGLE

Le triangle est un plan constitué de trois points. L'invocation de triangle(x1,y1,x2,y2,x3,y3) définit les trois points de ce triangle :

`triangle(10, 90, 50, 10, 90, 90);`

L'ARC

Un arc ou section de cercle, peut se dessiner avec l'appel de arc(x, y, largeur, hauteur, début, fin), où la paire x, y définit le centre du cercle, la seconde paire ses dimensions et la troisième paire, le début et la fin de l'angle d'arc en radians :

`arc(50, 50, 90, 90, 0, PI);`

LE QUADRILATÈRE

Le quadrilatère se construit en spécifiant quatre paires de coordonnées x et y sous la forme quad(x1,y1,x2,y2,x3,y3,x4,y4) dans le sens horaire :

`quad(10, 10, 30, 15, 90, 80, 20, 80);`

COURBE

Une courbe se dessine à l'aide de curve(x1, y1, x2, y2, x3, y3, x4, y4), où x1 et y1 définissent le premier point de contrôle, x4 et y4 le second point de contrôle , x2 et y2 le point de départ de la courbe et x3, y3 le point d'arrivée de la courbe :

```
curve(0, 300, 10, 60, 90, 60, 200, 100);
```

COURBE BÉZIER

La courbe de type Bézier se construit à l'aide de bezier(x1,y1,x2,y2,x3,y3,x4,y4)

```
bezier(10, 10, 70, 30, 30, 70, 90, 90);
```

COURBE LISSÉE

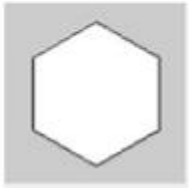
L'appel de curveVertex() dessine plusieurs paires de coordonnées x et y, entre deux points de contrôle, sous la forme
curveVertex(point de contrôle initial,xN,yN,xN,yN,xN,yN, point de contrôle final) ce qui permet de construire des courbes lissées :

```
beginShape();
curveVertex(0, 100);
curveVertex(10, 90);
curveVertex(25, 70);
curveVertex(50, 10);
curveVertex(75, 70);
curveVertex(90, 90);
curveVertex(100, 100);
endShape();
```

FORMES LIBRES

Plusieurs formes libres peuvent être dessinés par une succession de points en utilisant la suite d'instructions beginShape(),vertex(x,y),...,endShape(). Chaque point se construit par ses coordonnées x et y. La fonction CLO SE dans endShape(CLO SE) indique que la figure sera fermée, c'est-à-dire que le dernier point sera relié au premier, comme dans l'exemple ci-dessous de dessin d'un hexagone :

```
beginShape();
vertex(50, 10);
vertex(85, 30);
vertex(85, 70);
vertex(50, 90);
vertex(15, 70);
vertex(15, 30);
endShape(CLO SE);
```



CONTOURS

Vous avez remarqué que jusqu'à présent, toutes les figures données en exemple comportent un contour, ainsi qu'une surface de remplissage. Si vous voulez rendre invisible le contour, utilisez noStroke() en faisant bien attention de le placer avant la forme à dessiner :

```
noStroke();
quad(10, 10, 30, 15, 90, 80, 20, 80);
```

REMPLISSAGE

De la même manière, il est possible de dessiner des formes sans surface de remplissage avec l'instruction noFill() :

```
noFill();
quad(10, 10, 30, 15, 90, 80, 20, 80);
```

Par défaut, le-fond de la fenêtre de visualisation (l'espace de dessin) est gris neutre, les contours des figures sont noirs, et la surface de remplissage est blanche.

ISN : Initiation à la programmation TP 2
Variables et instructions conditionnelles

I) Les variables.

Pour le moment, l'utilisateur ne peut interagir avec la machine qu'au travers du code.
Le but de la suite est d'introduire la notion de variables qui permettra d'instaurer un « dialogue » à l'aide des variables.

Ouvrir Javascool
Lancer la Proglet « abcdAlgos »
Aller sur le "Parcours d'initiation", cliquer sur "Page initiale" puis "variables".
On trouve ici la traduction de l'algorithme suivant :

Variable : texte de type chaîne de caractères
Traitement :
Afficher("Bonjour, quel est ton nom ? ")
Saisir(texte)
Afficher("Enchanté ")
Afficher(texte)
Afficher(" , et … à bientôt ! ")
Sortie : affichage

qui s'écrit en langage javascool :

```
void main() {
   String texte ;
   println("Bonjour, quel est ton nom ? ") ;
    texte = readString() ;
    println("Enchanté  " + texte + " , et … à bientôt ! ") ;
}
```

ou de façon plus synthétique

```
void main() {
   println("Bonjour, quel est ton nom ? ") ;
    String texte = readString() ;
    println("Enchanté  " + texte + " , et … à bientôt ! ") ;
}
```

La " variable" dont il est question dans ce tutoriel s'appelle ici texte.
Comme on le voit en compilant puis en exécutant ce programme, une petite fenêtre " Entrée au clavier" s'ouvre pour permettre la saisie de texte, qui est utilisée ensuite pour l'affichage.

Cet affichage avec la fonction **println** implique un retour à la ligne. L'opérateur " +", quand il s'applique à un affichage (une ou plusieurs variables ou bien une phrase entourée par des guillemets) sert à créer une seule chaîne de caractères à partir de ces données : on parle alors de **concaténation**.

Remarque : en utilisant "Reformater le code", on décale les différents blocs rendant plus lisible le code N'hésitez pas à l'utiliser !

Exercice 1 :
Réaliser les exercices proposés dans le tutoriel pour bien comprendre qu'il est possible de donner à texte un nom différent, et qu'il existe des types de variables autres que string (ou chaîne de caractères en langage algorithmique) : les types numériques int et double.

Exercice 2 :
Avec les types numériques, on peut utiliser les opérateurs arithmétiques usuels notés +, -, * et /.
En utilisant uniquement ces opérateurs, écrire le programme équivalent à l'algorithme ci-contre.
(Attention, pour calculer le carré d'un nombre vous devez le multiplier par lui-même ou utiliser par la fonction pow, décrite dans l'onglet « Mémo »)

Algorithme principal

```
Entrées : rayon, perim, aires de type réels
Initialisation : affecter à Pi la valeur 3.14159
Traitement :
      Afficher("Entrer la rayon : ")
      Saisir (rayon)
      Affecter à perim la valeur 2*rayon*Pi
      Affecter à aire la valeur Pi*rayon^2
Sortie :
      Afficher("Périmètre du cercle de rayon ", rayon , ": ")
      Afficher(perim)
      Afficher("Aire : ")
      Afficher(aire)
```

Syntaxe à retenir

Déclaration des variables et des constantes

```
type_1 nom_var1; (par exemple: int x;)
type_2 nom_var2, nom_var3; (par exemple: double y, z;)
…
final type_1 nom_const_1 = valeur_constante;
      (par exemple: final double pi = 3.14159;)
```

Types

```
int, double, boolean, char, String
```

Rq : Les caractères sont encadrés par des simples quotes : 'o'
Les chaînes sont délimitées par des doubles quotes : `"chaîne"`

Séquence

```
{
  Actions_1 ;
  Actions_2 ;
}
```

Action de lecture/écriture

```
variable = readInt(); (ou readDouble(), readString(), etc.)
println(liste des éléments à afficher);
```

Affectation

```
variable = expression;
```

II) Faire des choix

Consulter le tutoriel Découvrir l'instruction conditionnelle qui résume la traduction des "SI ... ALORS ... SINON" du langage algorithmique par des if ... then ... else en langage Javascool.

Il est important de remarquer que les conditions d'égalité stricte ont des formes particulières, d'abord l'égalité entre deux nombres :

Langage algorithmique	**Langage Java**
"Si (x = 2) Alors ..."	if (x==2) ....

Pour l'égalité entre chaînes de caractères :

Langage algorithmique	**Langage Java**
"Si (texte="Bonjour") Alors ..."	if (equal(texte,"Bonjour")) ....
	Langage Processing
	if(texte.equals("Bonjour"))….

La dernière partie du tutoriel aborde la notion de variable booléenne, qui ne peut prendre comme valeur que true ou false.

Syntaxe à retenir	Action conditionnelle	

	Si (condition vraie) alors
	début
	Action1
	Action2 ...
	finSi
	sinon
	début
	Action3 ...
	finSinon
		if(temperature < 15) {
		println("allumer chauffage");
		} else {
		println("ne rien faire");
		}

Exercice 3 :

Ecrire un programme permettant de tester si un nombre entier saisi par l'utilisateur est pair avec Processing.

En java, pour calculer le reste de la division de *a* par *b*, on utilise l'instruction ;

reste = *a*%*b* ;

Comment affecter à une variable *nombre* l'entier saisi au clavier avec Processing ????

- On crée d'abord une boîte de dialogue :

Pour cela, on utilise une bibliothèque spéciale de Java ; **En tout début de programme**, on écrit :

```
import javax.swing.* ;   // cela importe la bibliothèque « javax.swing » des boites de dialogues
```

- On affiche la boite de dialogue demandant une entrée à l'utilisateur ;

```
String s =(String)JOptionPane.showInputDialog(null , "Saisir un entier :" , "Dialog" , JOptionPane.PLAIN_MESSAGE);
```

s contiendra la valeur entrée par l'utilisateur ; Cette entrée est toujours lue comme un « String » or cette entrée doit être comprise comme un entier. Pour cela, utiliser l'instruction :

```
int nombre =Integer.parseInt(s) ;  // On déclare la variable « nombre » et on y met la valeur de s
                                   //en tant que nombre …….
```

- Deux modes d'affichage du résultat, soit dans la console, soit dans la fenêtre :
  - Dans la console : println(……) ;
  - Dans la fenêtre : textSize(14) ; // la taille du texte
    text("Pair",40,40) ; // affiche à partir du point de coordonnées(40 ;40).

Travail personnel :

condition vraie ou fausse ??

Un lien cliquable mécanismes logiques après le dernier exercice du tutoriel permet d'en savoir plus.

Enfin, on peut noter qu'il manque, dans Javascool, la définition de la structure switch (équivalent à SELON en algorithmique), qui permet de ne pas avoir à enchaîner les tests, comme dans l'exemple suivant :

```
void main() {
 println("Entrez un numéro de mois:");
 int n = readInt();
 switch (n) {
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12:
   println("Mois à 31 jours");
   break;
  case 4:
  case 6:
  case 9:
  case 11:
   println("Mois à 30 jours");
   break;
  case 2: println("Mois à 28 ou 29 jours");
   break;
  default: println("Erreur de saisie");
 }
}
```

*Le mot-clef **break** permet de séparer les cas, comme ici pour traiter de façon unique les mois à 30 ou 31 jours.*

Exercice 4 : simulation du jet d'un dé

Dans Processing, Pour générer **un réel** aléatoire entre 1 et 6, on utilise la méthode :

random(lower,upper).

Elle génère un réel entre lower(compris) et upper (non compris).

Pour ne retenir que la partie entière de ce nombre aléatoire, on utilise la fonction

int(random(lower,upper)) ;

Pour obtenir, un entier aléatoire entre 1 et 6 (compris), il faudra donc écrire :

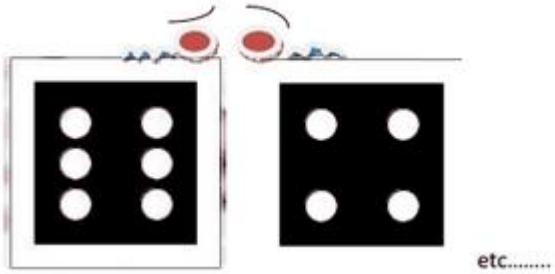
int(random(1,7)) ;

Dans Javascool, random(lower,upper) génère **un entier** entre lower(compris) et upper (non compris).

Ecrire, dans Javascool, un programme qui simule le lancer d'un dé et qui affiche le résultat Obtenu à l'aide de la structure SWITCH

Travail 1 à rendre :

Ecrire, dans Processing, un programme qui simule le lancer d'un dé et qui affiche le résultat du dé dans la fenêtre comme ci-dessous :



Une fois terminé, mettre tout votre programme dans les instructions suivantes :

```
void draw(){
frameRate(15) ;
if(mousePressed){
.....
.....
}
```

Votre programme

La méthode draw est spécifique à Processing et permet de faire tourner le programme en boucle et ici cela ne se fait que si l'on « clique » sur la souris dans la fenêtre d'exécution

La fréquence de répétition par défaut est de 30 fois/seconde . Quand on veut la modifier, il faut utiliser la méthode frameRate(...) ;

Travail 2 à rendre :

La méthode setup() {.....} s'utilise pour exécuter certaines instructions en début de programme et qu'une seule fois.

On utilise la méthode draw() {....} pour faire « répéter le programme »

Voici le code d'un programme :

La variable x est déclarée avant les méthodes, car si on la déclare dans une méthode, elle n'existe que dans la méthode.

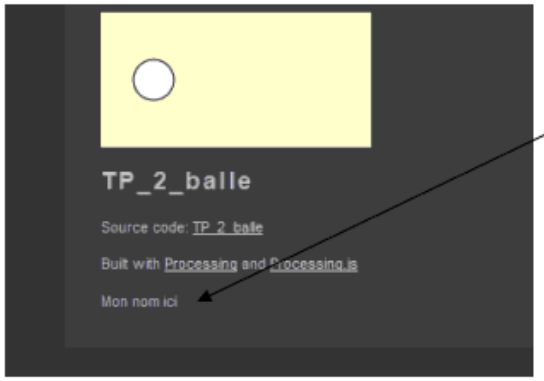
```
int x=0;

void setup(){
  size(200,100);
 }

void draw(){
    background(255,255,204);
    ellipse(x,50,30,30);
    x=x+1;
}
```

Taper ce programme et exécuter-le.

1°) Modifier ce programme pour que « la balle » s'arrête au bord de la fenêtre.
2°) Modifier ce programme pour que « la balle » rebondisse d'un bord à l'autre.

Rendre ces deux derniers programmes sous forme de pages web à l'aide du mode Javascript générées par Processing et modifiées par vos soins …. (votre nom au minimm)

Pour modifier la page web par Processing, il faut ouvrir le dossier webexport puis ouvrir avec Notepad le fichier index.html et rajouter des lignes de texte là où vous le souhaitez …..




```
                </noscript>
            </div>
                <h1>TP_2_balle</h1>
            <p id="description"></p>
                <p id="sources">Source code:
_balle.pde">TP_2_balle</a> </p>
                <p>
                Built with <a href="http://p
title="Processing">Processing</a>
                and <a href="http://processi
title="Processing.js">Processing.js</a>
                </p>
<p>  Mon nom ici  </p>
            </div>
        </body>
</html>
```

ISN : Initiation à la programmation TP 3
Fonctions - Instructions répétitives

I) Fonctions

Avec Javascool

Lancer la Proglet « abcdAlgos »

Le tutoriel "Fonctions" de Javascool permet de bien comprendre la notion de structuration d'un programme en différentes fonctions, pour le rendre plus lisible et plus facilement ré-utilisable. Il est notamment important de noter que nous avons en fait déjà utilisé des fonctions prédéfinies de Java : println, pow, etc.

Nous insistons ici encore une fois sur la traduction d'une fonction écrite en langage algorithmique, en reprenant la fonction int abs(int x) décrite au début du tutoriel. L'algorithme suivant :

Langage algorithmique

```
Fonction abs
Entrée: x: entier
Sortie: entier
Début
 Si (x > 0) Alors Retourner x
 Sinon Retourner -x
Fin_abs

Procédure affichage_abs
Entrée: x: entier
Début
 Afficher("Valeur absolue: ", abs(x))
Fin_affichage_abs

Algorithme principal
Variable y: entier
Début
 Afficher("Entrer y: ")
 Saisir(y)
 affichage_abs(y)
Fin
```

Langage java

```
int abs(int x) {
   if (x > 0) {
      return  x;
   } else {
      return - x;
   }
}

void affichage_abs(int x) {
   println("Valeur absolue: " + abs(x));
}
void main() {
   println("Entrer y: ");
   int y = readInteger();
   affichage_abs(y);
}
```

On voit dans cet exemple plusieurs éléments remarquables :

- une même variable x peut être utilisée dans deux fonctions différentes (ici `abs` et `affichage_abs`), ce sont deux variables différentes qui n'ont aucun rapport entre elles pour l'ordinateur : on dit que ces variables sont ***locales*** à la fonction qui les déclare.
- Décrivons chaque fonction :

  int abs(int x) { action(s) }

  le type de la sortie. La fonction devra renvoyer l'entier en sortie avec return

  int abs (int x){ actions }

  le nom de la fonction suivie prenant en entrée (en argument) l'entier x

  le(s) variable(s) d'entrée(s) sont placée(s) entre parenthèses après le nom de la fonction, en précisant leur type : `(int x)`. Cette fonction renvoie **une** sortie (x ou – x)

Remarque : une fonction peut prendre plusieurs arguments en entrée.

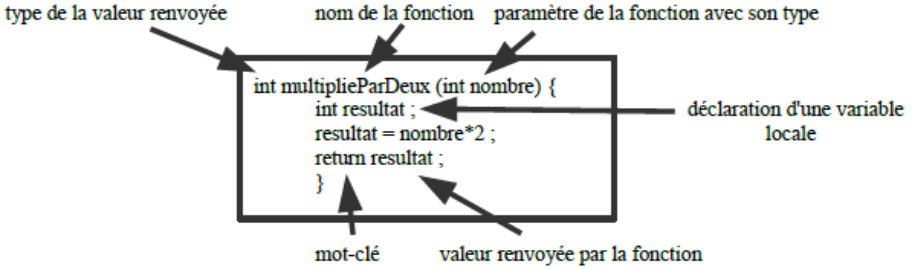
void affichage_abs (int x){ action }

on parle généralement en algorithmique de ***procédure*** lorsqu'une fonction ne retourne pas de valeur en sortie, ce qui se traduit par le mot-clef **void** placé devant le nom de la fonction.

- il peut y avoir zéro, une ou plusieurs entrées ; en revanche il y a zéro ou une sortie au maximum.
- les variables d'entrée sont placées entre parenthèses après le nom de la fonction, en précisant leur type : `(int x)`
- comme indiqué dans le tutoriel, l'instruction void main() traduit par conséquent le fait que l'algorithme principal ne prend aucune entrée, et qu'il ne retourne aucune valeur en sortie.

Résumé :

	Fonctions programmées par l'utilisateur

Les fonctions	void NomDeLaFonction(type param1, type param2, …..) {
	……..
	Actions ;
	……
	}
	void est un mot-clé qui indique l'absence de valeur renvoyée
	 
	
	L'exécution de l'instruction **return resultat** permet d'interrompre l'exécution du corps de la fonction et de renvoyer la valeur de la variable **résultat** au programme principal

	Fonctions prédéfinies
Les fonctions de lecture au clavier / d'écriture à l'écran	• **clear();** Efface l'affichage. • **String prenom = readString();** Lit une chaîne de caratères dans la ligne d'entrée au clavier. • **int age = readInteger();** Lit un nombre entier dans la ligne d'entrée au clavier. • **double taille = readDouble();** Lit un nombre décimal dans la ligne d'entrée au clavier. • **println("Oh");** Ecrit «Oh» dans la fenêtre de sortie. • **println("Oh," + age + " ans , est un bel âge !");** Ecrit « Oh, 20 ans, est un bel âge ! » (quand age=20).
fonctions utilisées	• **sleep(1000);** Arrête l'exécution du programme pendant 1000 milli-secondes. • **double x = random();** Renvoie dans la variable **x** un nombre aléatoire décimal compris entre 0 et 1. • **int n = random(1, 6);** Renvoie dans la variable **n** un nombre aléatoire entier entre 1 et **5** • **boolean y = equal("Dupond", "Dupont");** Compare les deux chaînes de caratères « Dupond » et « Dupont » et renvoie dans la variable **y** la valeur **false**. • **double z = pow(x, 2);** Renvoie dans la variable **y** le carré x^2. • **double x = sqrt(x);** Renvoie dans la variable **x** la racine de **x**.

Un des intérêts de la programmation de ces fonctions est de rendre plus lisible la lecture du programme principal en sous-traitant certaines actions à des fonctions.
Un métaphore peut-être le moteur d'une voiture :

- l'ensemble des pièces constitue le moteur (chaque fonction constitue le programme)
- Le programme principal est de mettre la clé et de la tourner pour démarrer (void main())

Pour déceler une panne, le mécanicien teste les pièces. Il peut en être de même avec chaque fonction pour déceler une erreur de codage, notamment lorsque le nombre de lignes de codes est très important.

Exercice :
Réaliser les exercices proposés dans le tutoriel pour bien comprendre les mécanismes décrits précédemment.

II) Les boucles

Avec Processing

Voici l'affichage obtenu lors de l'exécution du programme

```
line(0, 0, 100, 0);
line(0, 10, 100, 10);
line(0, 20, 100, 20);
line(0, 30, 100, 30);
line(0, 40, 100, 40);
line(0, 50, 100, 50);
line(0, 60, 100, 60);
line(0, 70, 100, 70);
line(0, 80, 100, 80);
line(0, 90, 100, 90);
```

Dans Processing

Ce code contient dix fois l'instruction line().
Il est possible de réduire le nombre de ligne de code en utilisant une structure de boucle :

LA BOUCLE FOR

```
for (int i = 0; i < 100; i = i + 10) {
line(0, i, 100, i);
}
```

La forme **for** équivalente à **Pour** en langage algorithmique.
Elle permet de répéter une série d'instructions un nombre de fois **défini**. Elle incorpore une variable i avec la valeur de départ 0 , la valeur de fin 90 en s'incrémentant ici de 10 en 10. Au fil des passages dans la boucle, elle prendra donc les valeurs suivantes :
i = 0 , i = 10 , i = 20 , i = 30 , i = 40 , i =50 , i = 60 , i = 70 , i = 80 , i = 90

Remarque :

- l'incrémentation `n++` est équivalente en Java à `n=n+1`
- l'incrémentation `n--` est équivalente en Java à `n=n-1`

Syntaxe à retenir	**Actions itératives**
	for(int i = NombreDépart ; i <=NombreMaximal ; i = i + increment) {
	actions;
	}

Tapez ce programme et l'exécuter

```
size(255,255);
for(int i=0;i<=255;i=i+1){
  stroke(0,0,i);//couleur de la ligne
  line(0,i,255,i);
}
```

Remarque : La variable entière i est déclarée dans la boucle for. On aurait pu la déclarer au début du programme.

Il permet d'afficher un dégradé de bleus dans une fenêtre 255×255.

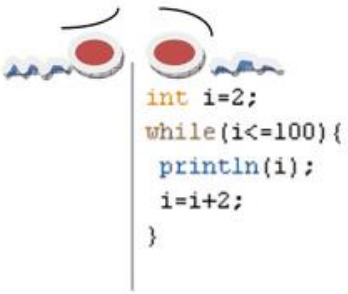
Modifier ce programme pour obtenir les résultats ci-dessous :



LA BOUCLE while

Elle permet de répéter une série d'actions tant qu'une condition est vraie.
La forme **while** équivalente à **Tant_que** en langage algorithmique.

Le programme ci-contre permet d'afficher dans le console les nombres pairs jusqu'à 100.



```
int i=2;
while(i<=100){
 println(i);
 i=i+2;
}
```

Syntaxe à retenir	**Actions itératives**
	while (condition) {
	actions;
	incrémentation;
	}

Exercice :

Avec Javascool (Proglet « abcdAlgos »)

1°) Ecrire une fonction **Boucle1** qui prend en paramètre un entier *n*, et écrit les entiers de 0 à n-1 sur le terminal.

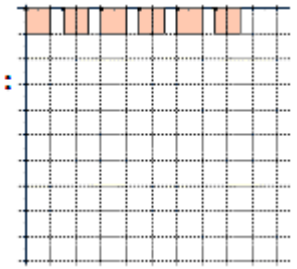
2°) Implémenter une fonction **BoucleInverse** qui prend en paramètre un entier *n*, et affiche les entiers de *n*-1 à 0 sur le terminal.

3°) Rédiger une fonction **BouclePair** qui prend en paramètre un entier *n*, et écrit les nombres pairs compris entre 0 et *n* – 1.

4°) Ecrire un algorithme qui demande un entier de départ, et qui ensuite affiche les dix entiers suivants. Par exemple : en entrant 17, l'ordinateur affichera les entiers de 18 à 27. Implémenter une fonction **DixNombres** réalisant cet algorithme.

Les boucles imbriquées

Voici ci-contre une fenêtre 100×100 contenant 6 carrés.
Les coordonnées des points sont :
(0;0) , (15;0) , (30,0) , (45;0) , (60;0) , (75,0).



1°) Ecrire un programme qui réalise ce dessin.
2°) Pour des raisons « d'esthétique », on rajoutera en début de programme l'instruction : translate(7,7).
Résultat à obtenir :

Rajouter l'instruction permettant d'obtenir des carrés colorisés aléatoirement comme ci-contre.	

L'objectif maintenant est d'obtenir un quadrillage de tels carrés
En hauteur (Ou largeur) : `for(.... ;..... ;.....){`
En largeur (ou hauteur) : `for((.... ;..... ;.....){`
`fill(random(255),random(255),random(255));`
`rect(? , ? , 10, 10);`
`}`
`}`
	

Cette notion de boucles imbriquées sera employée pour parcourir les pixels d'une image.