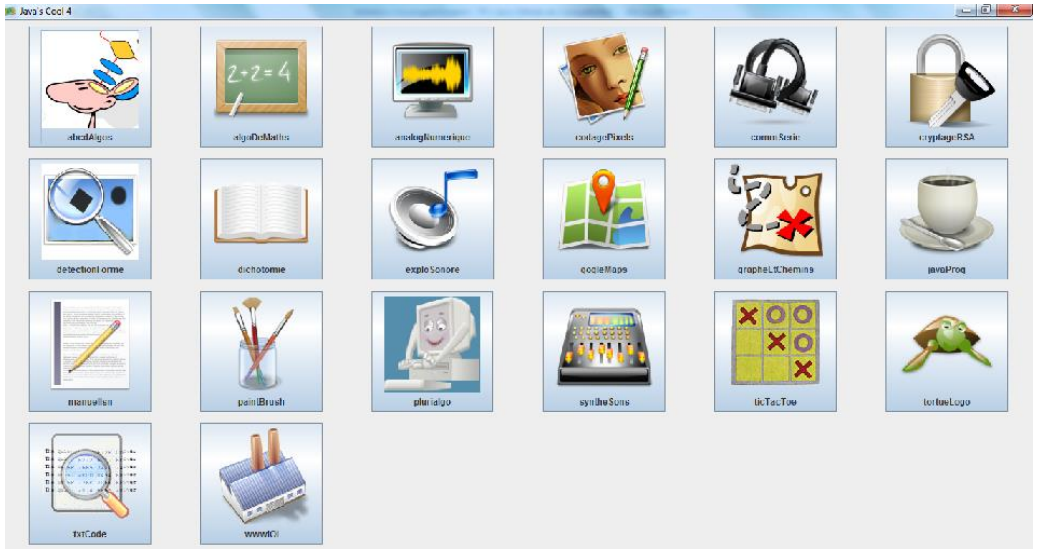
Langage de programmation: Initiation à java (Processing) ISN - TS2 19 janvier 2015

I) Premier programme avec Javascool

A) avec Javascool

- Créer un dossier que l'on nommera JAVA- ISN - TS2
- 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 »

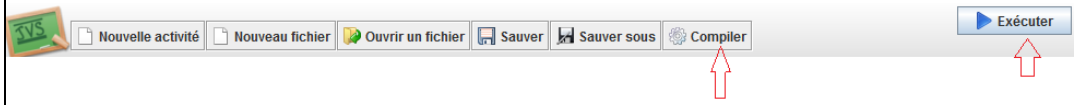
En cliquant sur « séquences d'instructions » on découvre le tutoriel « HelloWorld », qui montre le programme le plus simple que l'on peut écrire avec Javascool :

```
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**

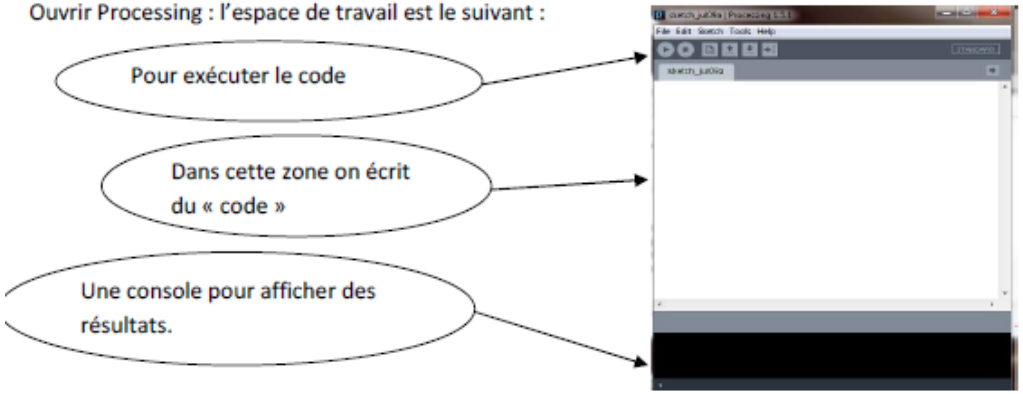
	void main() {
	instruction 1 ;
	instruction 2 ;
	……
	}

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) ;

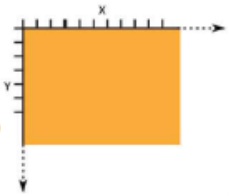
- Essayez avec : size(200,200) ;

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.

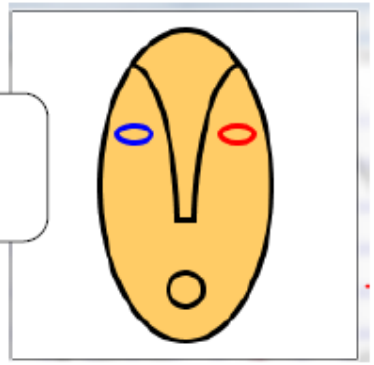
Ces valeurs x et y peuvent s'étendre théoriquement à l'infini, même si, en réalité, les contraintes de la taille de votre fenêtre vont délimiter la taille maximale d'une surface de création visible. C'est donc dans cet espace que nous allons dessiner.

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

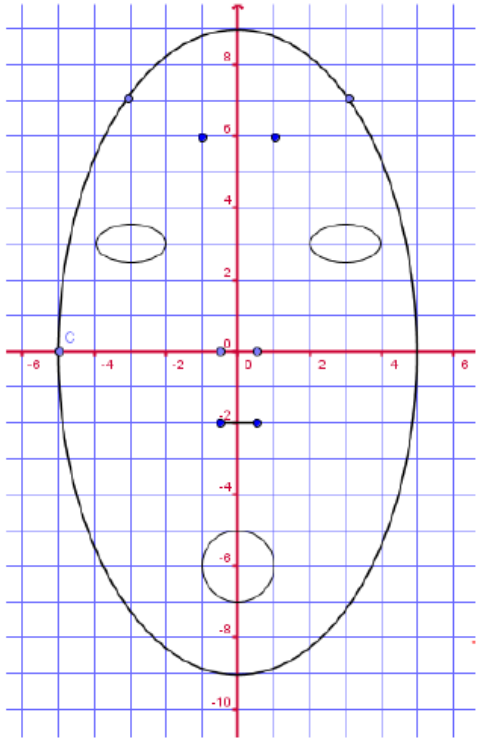
Début du programme

```
size(200,200) ;   // 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 100 vers la droite et de 100 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 ……

En utilisant ce que vous avez appris aujourd'hui, réalisez une image **personnelle** simple d'un petit extraterrestre qui nous servira de mascotte.

Annexe 1 du TP (Processing)

Les formes de bases :

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

LE POINT

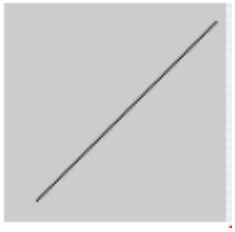
Pour commencer à dessiner, nous allons partir d'un point. Sur l'écran, un point est l'équivalent d'un pixel localisé dans la fenêtre de visualisation par deux axes de coordonnées, x et y correspondant respectivement à la largeur (axe horizontal) et à la hauteur (axe vertical) de l'espace de dessin. En 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)** :

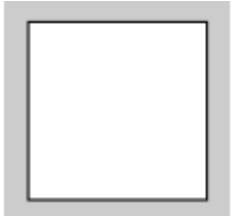
par exemple : **line(15, 90, 95, 10);**



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.

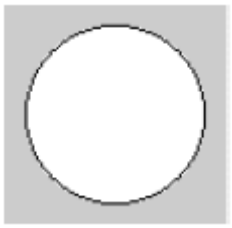
Annexe 1 du TP1 (Processing)

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);

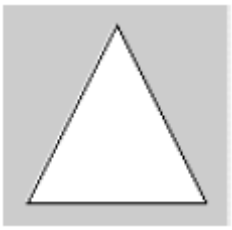
Amusez-vous à faire varier les 3e et 4e valeurs et observez-en les résultats.



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

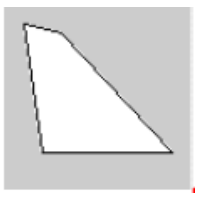
Un arc ou section de cercle, peut se dessiner avec l'appel de arc(x, y, largeur, hauteur, début, fin), où la paire x, y définit le centre du cercle, la seconde paire ses dimensions et la troisième paire, le début et la fin de l'angle d'arc en radians :

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);



Annexe 1 du TP (Processing)

COURBE

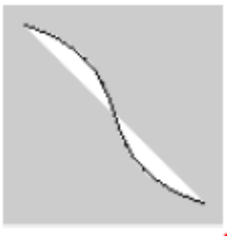
Une courbe se dessine à l'aide de **curve(x1, y1, x2, y2, x3, y3, x4, y4)**, où x1 et y1 définissent le premier point de contrôle, x4 et y4 le second point de contrôle , x2 et y2 le point de départ de la courbe et x3, y3 le point d'arrivée de la courbe :

```
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 :

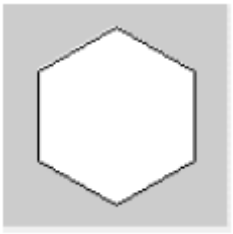
```
beginShape();
curveVertex(0, 100);
curveVertex(10, 90);
curveVertex(25, 70);
curveVertex(50, 10);
curveVertex(75, 70);
curveVertex(90, 90);
curveVertex(100, 100);
endShape();
```

Annexe 1 du TP (Processing)

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

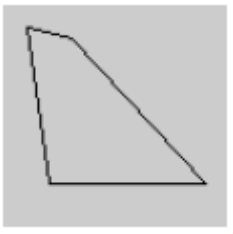
Vous avez remarqué que jusqu'à présent, toutes les figures données en exemple comportent un contour, ainsi qu'une surface de remplissage. Si vous voulez rendre invisible le contour, utilisez **noStroke()** en faisant bien attention de le placer avant la forme à dessiner :

```
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.